А. И. КИТОВ

Программирование экономических и управленческих задач

Издательство «Советское радио»
Москва—1971

УДК 681.142.2

К и т о в А. И. **Программирование экономических и управленческих задач.** М. Изд-во «Советское радио», 1971, 372 стр., т. 19 700 экз., ц. 1 р. 30 к.

Настоящая книга посвящена вопросам подготовки и программирования задач обработки экономической информации и управления в автоматизированных системах.

Рассматривается состав, содержание и особенности решения типовых задач в автоматизированных системах управления экономического назначения. Описывается структура подобных систем и методика разработки математического обеспечения для них. Излагается система автоматизации программирования экономических и математических задач АЛГЭМ, получившая практическое применение во многих вычислительных центрах. Рассматривается методика ассоциативного (спискового) программирования, применяемого при обработке больших объемов информации, имеющей изменяющуюся сложную структуру.

Описываются принципы построения и алгоритмы работы информационно-поисковых систем документального (библиографического) и фактографического типов и возможности их использования в экономических и научно-информационных областях. Большинство описанных в книге систем реализованы практически.

Книга предназначена для алгоритмистов, программистов, инженеров и экономистов, работающих в области автоматизации процессов управления и обработки экономической информации, и студентов высших учебных заведений соответствующих специальностей.

3-3-14
75-71

Оглавление

Предисловие

Программой КПСС и решениями XXIV съезда КПСС намечены широкие перспективы внедрения ЭВМ в сферу управления экономикой. В различных отраслях промышленности, на предприятиях, в научно-исследовательских учреждениях накоплен значительный опыт применения ЭВМ и математических методов для решения различных экономических задач. Этот опыт нуждается в критическом анализе и обобщении. Необходимо, чтобы выполненные работы сделались достоянием широких кругов специалистов, занятых в аналогичных разработках. Это позволит шире использовать полученные положительные результаты, сократить дублирование разработок, избежать ошибочных решений и будет способствовать выработке единых взглядов и исходных принципов на построение экономических автоматизированных систем управления в нашей стране.

В настоящей книге изложены результаты работ в указанной области, в которых автор принимал участие. Материал первой главы построен на основе анализа разработки типовой отраслевой автоматизированной системы управления (ОАСУ) для группы машиностроительных министерств. Последующие главы содержат материал разработок, выполненных аспирантами автора (ныне кандидатами наук). Так, система автоматизации программирования АЛГЭМ, получившая достаточно широкое практическое применение, разработана канд. физ-мат. наук Шиллер Ф. Ф., канд. техн. наук Кирюхиным Ю. П., Бородулиной Н. Г., Зверевой А. К., Жадан Н. В., Мазеевым М. Я. В разработке языка АЛГЭМ-2 принимали участие Шиллер Ф. Ф. и Мазеев М. Я. Автоматизированная фактографическая информационно-поисковая система (ИПС) для учета и планирования оборудования в ОАСУ разработана и внедрена в практику канд. техн. наук Ерлихманом С. Г.

Библиографическая ИПС «Сетка-5» построена и внедрена в ряде мест канд. техн. наук Гороховым С. А. Развитие этой системы и внедрение ИПС для учета определенного контингента кадров в Министерстве высшего и среднего специального образования СССР выполнено канд. техн. наук Цветаевой И. Л.

В разработке и программировании задач и алгоритмов автоматизированной системы научной медицинской информации, описанной в § 21, принимали участие канд. техн. наук Вуль Ю. Е., аспирантка Грачева Е. К., программисты Инякина Т. И., Хлебнова Л. Н., Сошникова Р. И. На базе ассоциативного словаря понятий, описанного в последнем параграфе, канд. техн. наук Волковым Б. Г. создана специализированная ИПС для опознавания объектов, в которой реализован «диалоговый» способ взаимодействия между ЭВМ и человеком. Канд. техн. наук Керимовым С. К. разработана и внедрена в практику ассоциативно-адресная ИПС с широко развитой системой грамматических отношений между дескрипторами, приспособленная для поиска химической информации. Канд. техн. наук Арнаудовым Д. (Народная республика Болгария) разработана методика использования системы АЛГЭМ для программирования информационно-поисковых задач. Результаты всех перечисленных работ изложены в настоящей книге.

Материал главы 5 «Ассоциативное программирование» взят из предыдущей книги автора «Программирование информационно-логических задач» («Советское радио», 1967 г.)

Автор выражает благодарность всем указанным товарищам за выполненную ими работу. При написании книги автор учел советы и замечания рецензентов члена-корреспондента АН СССР проф. Бусленко Н. П. и члена-корреспондента АН Казахской ССР проф. Акушского И. Я., за что выражает им признательность.

А. И. КИТОВ

Глава 1

ОБЩИЕ СВЕДЕНИЯ ОБ ЭКОНОМИЧЕСКИХ И УПРАВЛЕНЧЕСКИХ ЗАДАЧАХ И ОСОБЕННОСТЯХ ИХ ПРОГРАММИРОВАНИЯ

1. Кибернетика в экономике

В современных условиях в связи со значительным увеличением масштабов и темпов развития промышленности, усложнением производственных связей между различными звеньями народного хозяйства и увеличением потоков экономической информации резко возросла роль электронных вычислительных машин в решении задач обработки экономической информации, задач планирования и управления. Как показывает практика, внедрение ЭВМ в сферу экономики должно производиться не путем решения на них отдельных задач, а путем создания комплексных автоматизированных систем сбора и обработки экономической информации, обслуживающих целые предприятия или отрасли народного хозяйства, так как в этом случае обеспечивается высокая эффективность применения ЭВМ и других средств автоматизации в экономике. Автоматизированные системы управления, создаваемые в различных звеньях народного хозяйства, с кибернетической точки зрения относятся к классу больших управляющих систем и объединяются названием «экономические автоматизированные системы управления» (ЭАСУ).

Экономические автоматизированные системы управления обладают общими свойствами кибернетических систем, и в то же время имеют свои специфические свойства, которые могут быть сформулированы следующим образом:

— наличие большого количества составных элементов, связанных между собой каналами передачи информации. Свойства элементов, как правило, не определяют свойств системы и целом, и поэтому при проектировании и изучении этих управляющих систем необходим комплексный подход, обеспечивающий определение интегральных свойств системы на основе учета свойств ее элементов и характера их взаимодействия между собой и с внешней средой;

— иерархическая структура, допускающая включение в состав элементов управляющей системы первого уровня управляющих систем второго уровня, составными элементами которых могут быть управляющие системы третьего уровня и т. д.;

— наличие общей цели управления для всей большой управляющей системы и частных целей управления для каждой управляющей системы любого уровня; частные цели подчинены общей цели;

— расчленение управляющей системы любого уровня на отдельные функциональные подсистемы, цели и задачи которых подчинены целям и задачам системы управления данного уровня;

— функционирование управляющих систем всех уровней в условиях взаимодействия с внешней средой, являющейся источником входной информации и помех; к внешней среде для данной управляющей системы относятся как вышестоящие, так и нижестоящие управляющие системы, с которыми взаимодействует данная управляющая система;

— гибкость структуры и алгоритмов управления в управляющих системах всех уровней, входящих в состав большой управляющей системы, обеспечивающая устойчивость и надежность их функционирования;

— наличие людей в качестве элементов управления и контроля на всех уровнях иерархии большой управляющей системы и в связи с этим необходимость сочетания и постоянного взаимодействия людей и автоматических устройств в процессе выполнения функций управления.

В соответствии с перечисленными свойствами в каждой большой управляющей системе можно выделить собственно управляющую систему данного уровня и автономные управляющие системы более низких уровней. Собственно управляющая система данного уровня делится на функциональные подсистемы, которые являются частями системы, выполняющими определенные виды деятельности (определенные функции). Функциональная подсистема не является автономной; она не может работать вне данной управляющей системы, в то время как автономная управляющая система выполняет все виды деятельности системы, но в меньшем масштабе.

Так, например, большая управляющая система для отрасли промышленности включает в себя автономные управляющие системы, относящиеся к отдельным предприятиям; в то же время собственно управляющая система данного уровня (АСУ министерства) состоит из ряда функциональных подсистем (планирования, учета и отчетности, оперативного управления), которые могут функционировать только совместно. Таким образом, каждая большая управляющая система состоит из управляющей системы

данного уровня и определенного количества автономных управляющих систем более низких уровней иерархии, которые могут быть либо также большими управляющими системами, либо просто управляющими системами. Автономные управляющие системы самых низких уровней иерархии могут включать в себя только функциональные подсистемы и поэтому не являются по определению большими управляющими системами (хотя фактически они могут быть очень сложными и состоять из большого числа элементов).

Общая методика изучения любых управляющих систем основана на сочетании макро и микроподходов. При макроподходе управляющая система рассматривается как «черный ящик» и выясняются связи этой системы с внешней средой и выполняемые системой функции. При этом внутренняя структура системы остается вне поля зрения. При микроподходе рассматривается внутренняя структура данной управляющей системы, выделяются автономные управляющие системы, входящие в ее состав (если это большая управляющая система), и функциональные подсистемы, выявляются связи между составными частями данной управляющей системы. Затем к каждой из выявленных частей может быть снова применен макроподход, затем микроподход и т. д.

Указанная методика последовательной детализации больших управляющих систем отражает их иерархическую структуру.

Типовая блок-схема автоматизированной системы управления. С информационной точки зрения любая автоматизированная управляющая система состоит из нескольких независимых (несинхронизированных) центров переработки информации, связанных между собой каналами передачи информации — информационными связями. Центры переработки информации представляют собой физическую реализацию либо автономных систем, либо функциональных подсистем, либо их различные сочетания.

Информационной, в отличие от жесткой физической связи, называется связь, обеспечивающая только передачу информации, т. е. изменение состояния элементов памяти (ячеек, регистров и т. д.) центров, принимающих информацию. При этом обязательным условием реализации поступившей информации является обращение к этим элементам памяти, т. е. активные действия данного центра, определяемые его собственной программой работы и не зависящие от поступившей информации.

Каждый центр подобной системы должен иметь специальные входные запоминающие элементы (буферная или приемная память) определенной емкости и быстродействия, в которые производится прием информации от других центров.

Существенной особенностью центров переработки информации в ЭАСУ с точки зрения их проектирования и моделирования является дискретный характер их работы. Функционирование каждого центра во времени происходит по отдельным элементарным шагам, удовлетворяющим следующим условиям:

а) Порядок выполнения всех действий центра в течение шага полностью определяется состоянием центра в момент начала очередного шага и внешней информацией, поступившей в центр к этому моменту. Любая информация, поступившая в центр в процессе выполнения шага, никакого влияния на ход данного шага не оказывает и может быть учтена только при выполнении следующего шага;

б) в течение шага центр не выдает никакой информации другим центрам. Выдача информации может производиться только в конце шага, т. е. моменты окончания шагов совпадают с возможными моментами выдачи информации.

Примерами элементарных шагов являются некоторые технологические циклы работы заводов, которые могут прерываться только после их полного окончания. Подобные дискретные шаги, вообще говоря, могут быть выделены в любых системах, в том числе в системах непрерывного действия, если учесть наличие переходных процессов, инерционность исполнительных органов и ограничения в точности их работы.

Процесс функционирования большой управляющей системы сводится к одновременной работе ряда центров переработки информации и обмену информацией между ними.

В соответствии с изложенными определениями отдельные элементы управляющей системы, связанные между собой не информационной, а жесткой физической связью, должны при данном рассмотрении считаться одним центром.

Процесс функционирования центров переработки информации ЭАСУ имеет три особенности, которые принято определять общим термином «работа в реальном времени»:

1. Произвольность поступления отдельных порций информации во времени, обусловленная наличием внешних источников информации, которые не управляются данной управляющей системой. Информация может приходить в произвольные моменты времени и с различным темпом. Это не исключает и

поступления определенной информации в заранее предусмотренные интервалы времени.

2. Наличие оперативного взаимодействия в процессе работы между управляющей системой и людьми — операторами; при этом ход процесса переработки информации и управления может меняться в зависимости от команд операторов.

3. Наличие административных и организационных требований к процессу переработки информации (регламента) и динамических ограничений в работе каналов связи и исполнительных органов; регламент и динамические ограничения определяют производительность центров и их режим приема, переработки и выдачи информации.

Взаимодействие между отдельными центрами системы осуществляется по принципу обратной связи; при этом каждый из центров оказывает воздействие на работу других центров. Принцип обратной связи является характерным и для взаимодействия между собственно управляющей системой (центром переработки информации) и исполнительными органами, а также источниками внешней информации.

В составе каждой управляющей системы могут быть выделены следующие типовые части:

а) источники внешней информации;
б) устройство управления обменом информацией;
в) управляющая машина (комплекс);
г) устройство сопряжения с операторами;
д) исполнительные органы.

Каждый из этих объектов, вообще говоря, может сам состоять из нескольких независимых объектов. Общая блок-схема типовой управляющей системы показана на рис. 1.

В виде приведенной блок-схемы могут быть представлены весьма разнообразные управляющие системы: системы автоматического управления заводами, отраслями, транспортными и другими средствами.

Для обеспечения работы ЭАСУ в реальном масштабе времени необходимы гибкие и весьма сложные программы для ЭВМ. Совокупность этих программ называется математическим обеспечением ЭАСУ (МО ЭАСУ). Оно строится обычно в виде совокупности большого количества отдельных исполнительных

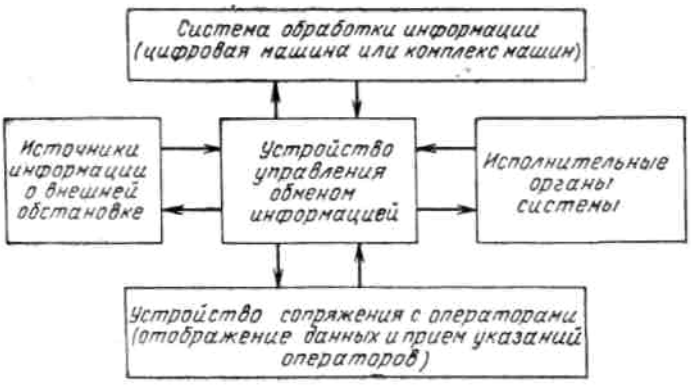


Рис. 1. Общая блок-схема типовой управляющей системы.

программ (программ конкретных задач), объединяемых главной программой-диспетчером. Главная программа-диспетчер определяет порядок работы исполнительных программ и включает их в работу в зависимости от поступающей внешней информации, указаний операторов, запросов исполнительных органов или в зависимости от текущего времени.

Основным требованием к исполнительным программам, входящим в состав МО ЭАСУ, является требование их максимальной автономности. Это значит, что каждая исполнительная программа должна быть связана со всей остальной программой при помощи минимального количества входных и выходных данных. Изменения в какой-либо исполнительной программе не должны оказывать влияния на всю программу, так же как и изменения в остальной программе не должны влиять на данную исполнительную программу. Таким образом, исполнительные программы по отношению ко всей программе могут рассматриваться как некоторые автономные блоки. Каждая исполнительная программа может, в свою очередь, состоять из ряда рабочих подпрограмм, объединенных частной (локальной) программой-диспетчером 2-го уровня.

Построение сложной программы в виде совокупности автономных исполнительных программ при наличии трех уровней управления (главная программа-диспетчер, частные программы-диспетчеры исполнительных программ и рабочие подпрограммы) обеспечивает:

а) гибкость и устойчивость работы всей сложной программы;

б) возможность одновременного и независимого составления различных исполнительных программ и рабочих подпрограмм и простоту их замены и корректировки;

в) упрощение и облегчение процесса отладки сложной программы.

Важным моментом в построении сложных программ МО ЭАСУ является выделение из всех исполнительных программ общих рабочих подпрограмм и объединение их в общую библиотеку так называемых стандартных подпрограмм. При этом одна и та же стандартная подпрограмма может использоваться во всех тех исполнительных программах, где она потребуется. Это позволяет существенно сократить общий объем программы.

Для повышения устойчивости работы сложной управляющей программы по отношению к случайным сбоям вычислительной машины, аппаратуры передачи данных, каналов связи и др. в ее состав иногда включается специальная программа анализа и исправления ошибок. Эта программа выполняет три функции:

1) контроль вычислений и обнаружение ошибок;

2) исправление ошибок (путем повторного счета, исключением искаженных данных и др.);

3) регистрацию сбоев, выдачу сигналов оператору и переключение работы на дублирующую аппаратуру при невозможности восстановить нормальный ход вычислений.

Методы выработки решений в ЭАСУ. Исходным моментом программирования задач для любой ЭАСУ является *выбор методов принятия решений* по проведению тех или иных операций, которые должны быть выполнены этой ЭАСУ. В зависимости от характера выбранных методов принятия решений управляющие системы могут быть разделены на 3 типа:

1) системы, действующие при наличии полной информации об обстановке и вырабатывающие детерминированные решения;

2) системы, действующие в стационарных условиях, обладающие «опытом» и имеющие вероятностную информацию о конкретной обстановке, в которой требуется принимать решения. Такие системы вырабатывают «решения с риском», используя вероятностные методы;

3) системы, действующие в нестационарных условиях (часто однократно) при отсутствии определенной информации (в том числе статистической) об обстановке, в которой приходится принимать решения.

Системы последнего типа должны обладать средствами для сбора данных об обстановке в ходе самого процесса управления и гибким (адаптирующимся) алгоритмом управления, обеспечивающим приспособление системы к непредвиденным условиям.

Любой процесс управления в экономических системах включает в себя три основные части: сбор информации об обстановке (учет и отчетность), планирование, т. е. определение порядка действий на некоторый отрезок времени, и оперативное управление, т. е. контроль за ходом выполнения плана и текущая корректировка действий системы. В планировании выделяют два вида:

1) планирование распределения материальных, денежных, людских и других ресурсов между участниками заданной работы;

2) определение причинно-временных зависимостей и логической последовательности действий различных участников работы, обеспечивающих достижение поставленной цели при заданных условиях.

При планировании первого вида применяют различные методы оптимального планирования, объединяемые общим названием «математическое программирование», а при планировании второго вида — методы сетевого планирования и управления, методы логического анализа (построение логических матриц следования событий).

Математическое программирование — это раздел прикладной математики, посвященный методам решения определенного класса экстремальных задач. Основной особенностью задач математического программирования, отличающей их от условных экстремальных задач классического анализа, является наличие неравенств среди ограничений, определяющих область изменения переменных. В этих случаях применение классических методов решения связано с большими трудностями, так как экстремум функций находится на границе области.

Другая особенность задач математического программирования заключается в обязательности условия неотрицательности переменных; это условие играет основную роль при построении методов решения подобных задач. Общая постановка задач математического программирования выглядит следующим образом.

Нужно вычислить вектор $X(x_1, x_2, \ldots, x_n)$, обеспечивающий экстремум функции $F(X) = F(x_1, x_2, \ldots, x_n)$, называемой целевой функцией и определяющей оптимальность плана при условиях:

$$g_i(x_1, x_2, \ldots, x_n) = 0, \qquad I = 1, 2, \ldots, m_1,$$
$$g_i(x_1, x_2, \ldots, x_n) \le 0, \qquad I = m_1 + 1, \ldots, m,$$
$$x_j \ge 0, \qquad j = 1, 2, \ldots, n.$$

В математическом программировании выделяется ряд частных разделов в зависимости от характера функций F и g_i. Если F и g_i линейные функции переменных x_j, то имеем *линейное программирование,* которое широко применяется при решении различных экономических задач. Для решения задач линейного программирования существует общий метод. Кроме того, для распространенных задач линейного программирования (например, широко известной транспортной задачи) существуют свои эффективные методы решения. Если F или некоторые из g_i нелинейны, то имеем *нелинейное программирование,* которое, в свою очередь, делится на выпуклое и невыпуклое.

Выпуклое программирование изучает методы определения максимума выпуклой вверх функции (или минимума выпуклой вниз функции). При выпуклом программировании имеет место совпадение локального и глобального экстремумов, что существенно упрощает методы решения. Наиболее разработан в выпуклом программировании раздел, называемый *квадратичным программированием,* при котором целевая функция является полиномом второй степени, а ограничения линейны.

При *невыпуклом программировании* необходимо нахождение и сравнение всех локальных экстремумов. Общих методов решения задач невыпуклого нелинейного программирования нет; решаются частные задачи, для которых подбираются специальные приемы. Если среди параметров, определяемых условием задачи, имеются случайные величины, то имеет место стохастическое программирование, решающее задачи планирования и управления в условиях неопределенности.

Если параметры по своему физическому смыслу могут быть только целыми числами, то имеет место *целочисленное программирование,* решающее в основном комбинаторные задачи. Иногда заранее известно, что параметры задачи (целевой функции и ограничений) могут меняться в заданных пределах, и важно выяснить влияние изменений этих параметров на решение задачи. Эти вопросы рассматриваются в разделе, называемом *параметрическим программированием.*

Большое значение при решении задач планирования и управления в экономике имеет *динамическое программирование,* предложенное в середине 50-х годов Р. Белманом. Это общий подход к решению многошаговых экстремальных задач, которые могут иметь различную конкретную математическую формулировку. Основной особенностью постановки таких задач является то, что оптимальный план для каждого последующего периода планирования зависит только от состояния системы в начале этого периода и не зависит от того пути, по которому система пришла в это состояние. Это утверждение, известное под названием принципа оптимальности Белмана, является исходным для построения метода решения. Кроме того, принимается допущение об аддитивности целевой функции

$$\Phi(v_0, v_1, v_2, \ldots, v_{n-1}) = \sum_{i=0}^{n-1} f(v_i)$$

Здесь Ф — целевая функция, определяющая, например, общий объем производства (прибыль, длина пути и т. д.) за n периодов (от 0 до n - 1); v_i— затраты на развитие производства в i-й период; $f(v_i)$ — объем производства в i-м периоде.

Обычно задаются начальное и конечное состояние системы и требуется определить оптимальный путь для перевода системы из начального состояния в конечное.

Управление движением системы осуществляется с помощью воздействий (v_i), выбираемых для каждого периода из множества воздействий, допустимых в этом периоде.

Ясно, что должны быть известны зависимости поведения системы в данном периоде от ее исходного состояния в начале этого периода и от воздействия в этом периоде.

Суть метода решения заключается в последовательном рассмотрении возможных переходов системы из одного состояния в другое, начиная с ее конечного состояния.

На первом шаге решения определяются все возможные состояния системы перед ее переходом в конечное состояние и из этих состояний выбирается такое, чтобы переход из него в конечное состояние являлся оптимальным.

Затем определяются все возможные состояния системы, из которых она может перейти за один шаг в данное выбранное состояние на предпоследнем шаге, а также выбирается оптимальный переход для

этого предпоследнего шага. Затем то же самое проделывается для предпоследнего шага и т. д. до тех пор, пока мы не доберемся до начального состояния и не определим первый шаг. Полученная при этом последовательность переходов (состояний системы) будет представлять собой оптимальный путь движения системы из начального состояния в конечное.

При решении многих экономических задач, и особенно при проектировании автоматизированных систем управления, используются также методы теории игр и статистических решений, обеспечивающие выбор оптимальных решений в конфликтных ситуациях при неполной информации, когда результаты мероприятий могут оцениваться только с вероятностной точки зрения, а также методы теории массового обслуживания, занимающейся задачами оптимальной организации работы производственных или снабженческих органов.

2. Общая характеристика ЭАСУ

Социалистическая экономика функционирует на основе сочетания принципов централизованного и децентрализованного планирования и управления. В соответствии с этими принципами можно выделить три основных уровня экономических автоматизированных систем управления (ЭАСУ):

1) общегосударственная автоматизированная система управления (ОГАС), использующая государственную сеть вычислительных центров (ГСВЦ) и предназначенная для обслуживания общегосударственных территориальных органов управления (Госплан, ЦСУ, Госснаб СССР, торговля, сельское хозяйство, государственные, советские и партийные органы);

2) отраслевые автоматизированные системы управления (ОАСУ), создаваемые для обслуживания отдельных отраслей промышленности (и обеспечивающие решение экономических задач в интересах функциональных и главных управлений министерства);

3) автоматизированные системы управления предприятиями (АСУП), предназначенные для автоматизации процессов обработки экономической информации и решения задач планирования и управления для одного или нескольких близкорасположенных предприятий.

Указанные системы, несмотря на различия в сферах действия и в выполняемых функциях, имеют ряд общих свойств и принципов построения. В связи с этим может быть дано общее определение экономической автоматизированной системы управления.

Экономическая автоматизированная система управления представляет собой синтез экономико-математических моделей, математических методов и информационных массивов и документов, отражающих производственно-экономические процессы в народном хозяйстве (в отрасли, на предприятии); вычислительной и организационной техники, предназначенной для сбора и обработки экономической информации и решения задач планирования и управления производством; коллектива людей, занятых в данном органе управления.

Совокупность таких систем в стране должна быть увязана между собой и представлять единую автоматизированную систему оптимального управления народным хозяйством страны. К числу первоочередных задач автоматизации управления народным хозяйством относятся:

1) оптимальное распределение имеющихся ресурсов между отраслями (главками и предприятиями) и автоматизация процессов материально-технического снабжения;

2) обеспечение сбалансированности производства в народном хозяйстве (отраслях, на предприятиях) при максимуме выпуска требуемой продукции и минимуме затрат;

3) определение оптимальных объемов капиталовложений (при перспективном планировании) и распределение их между отраслями (главками, предприятиями) по отдельным годам планируемого периода;

4) автоматизация сбора, обработки и передачи экономической информации, а также выработки команд управления и контроль за их выполнением.

В качестве более далекой задачи автоматизации управления народным хозяйством можно указать на задачу согласования текущего и перспективного планирования производства и строительства с прогнозированием развития науки, техники и экономики.

Создание указанных автоматизированных систем управления предполагает постепенную перестройку управления в народном хозяйстве, отраслях промышленности и на предприятиях на основе широкого применения электронной вычислительной техники и экономико-математических методов.

Внедрение этих систем должно обеспечить достижение следующих целей:

а) Значительное повышение уровня производства промышленности за счет более полного и

рационального использования производственных мощностей, кадров, материальных и денежных ресурсов, сокращения сроков подготовки производства к выпуску новых изделий и за счет более правильного определения объема и номенклатуры выпускаемых изделий.

б) Существенное сокращение сверхнормативных запасов (и вообще запасов на складах) за счет более точного определения потребностей в материалах, комплектующих изделиях и полуфабрикатах, более полного и точного учета изменений потребностей и движения материальных ценностей и оперативного маневрирования фондами и наличными запасами.

в) Высвобождение работников управленческого аппарата центральных органов управления, министерств и заводоуправлений от трудоемких расчетных работ.

Хотя общая численность персонала планирования и управления, по всей вероятности, и не уменьшится (если не увеличится за счет того, что потребуется дополнительный обслуживающий ЭВМ персонал), тем не менее сами задачи планирования и управления и их решение ставятся на качественно высшую ступень.

г) Коренное улучшение качества принятия решений и оперативности руководства на всех уровнях управления за счет соответствующего агрегирования и фильтрации избыточной информации техническими средствами.

Принципиальным отличием экономических автоматизированных систем управления от технических управляющих систем является *эволюционность* их разработки, внедрения и использования.

Эти системы создаются всегда в условиях функционирования какой-то существующей (сначала совершенно неавтоматизированной) системы управления. Скачкообразный переход от существующей системы к новой автоматизированной системе управления невозможен по следующим причинам:

а) Создание полностью законченной автоматизированной системы управления представляет собой сложный процесс; она создается частями и внедряется по этапам; при этом неизбежно сочетание автоматизированных и неавтоматизированных участков.

б) Невозможно создать комплексную систему управления без моделирования и экспериментальной проверки. Моделирование же, вернее подготовка для моделирования исходной информации, представляет собой весьма длительный процесс, и поэтому подготавливаемые модели (или их части) целесообразно использовать не только для исследовательских целей, но и для решения практических задач управления.

в) При внедрении новой автоматизированной системы неизбежен психологический барьер, который легче преодолевается при постепенной перестройке управления.

Эволюционность указанного процесса имеет свои преимущества, так как при этом представляется возможным постепенно уточнять состав подсистем и задач, корректировать структуру автоматизированной системы управления, т. е. создавать систему методом последовательных приближений.

Однако недостатком этого процесса является неизбежность перестройки и переделки уже сделанных участков и задач.

Структура ЭАСУ. Структура экономических автоматизированных систем управления наглядно может быть представлена в виде трехкоординатной схемы, предусматривающей расчленение этих систем на составные части по трем разрезам (рис. 2).

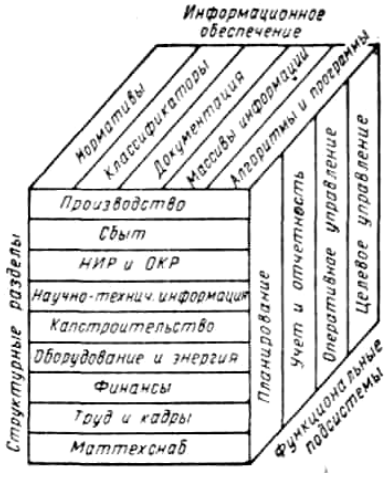


Рис.2. Структура ЭАСУ

Любая экономическая автоматизированная система управления может быть разделена на ряд

структурных разделов, каждый из которых решает задачи экономического управления в определенной области деятельности рассматриваемой экономической системы (в области производства продукции, в области обеспечения материалами, энергией и т. д., в области научно-исследовательской и опытно-конструкторской работы, в области капитального строительства и т. п.).

Структурные разделы в существующих системах характеризуются административным (организационным) единством и соответствуют в основном существующим органам управления (государственным комитетам, функциональным управлениям промышленных министерств, отделам отраслевых главных управлений и отделам заводоуправлений). Указанное соответствие не является случайным; оно обусловлено объективными свойствами современного промышленного производства, в котором необходимы такие виды деятельности, как производство продукции, снабжение его сырьем и материалами, обеспечение энергией и оборудованием, финансовыми ресурсами, кадрами, организация сбыта продукции, осуществление капитального строительства, обеспечение научно-технической информацией, осуществление научно-исследовательских и опытно-конструкторских работ, необходимых для технического прогресса производства.

Существующая система управления народным хозяйством имеет органы, управляющие указанными сторонами промышленности на различных уровнях. На верхнем уровне — это различные государственные комитеты, сфера деятельности которых охватывает все народное хозяйство; на среднем (отраслевом) уровне — функциональные управления в аппарате промышленных министерств (и отделы в отраслевых главных управлениях, образующих второй подуровень среднего уровня), на низшем уровне — соответствующие отделы заводоуправлений и предприятий.

По содержанию и характеру выполняемой работы в каждом структурном разделе (подразделений экономического органа) можно выделить три основные функции деятельности:

1) функции планирования (текущего и перспективного), 2) функции отчетности и учета, 3) функции оперативного управления (регулирования) деятельностью подведомственных объектов.

Кроме того, в некоторых министерствах выделяется четвертая функция, связанная с управлением процессом создания крупных технических объектов,— функция целевого управления. Указанные три (или четыре) функции можно рассматривать как три (четыре) функциональные подсистемы, на которые делится вся автоматизированная экономическая система управления (по второму разрезу).

Почему требуется, наряду с делением ЭАСУ на структурные разделы, делить всю систему на функциональные подсистемы? Дело в том, что деятельность промышленного министерства (предприятия) по различным его структурным разделам характеризуется тесной взаимосвязью этих структурных разделов. Нельзя выделить одно планирование производства, совершенно изолировав его от планирования материально-технического снабжения или от планирования сбыта, так как планирование охватывает все стороны деятельности предприятий или министерств и представляет собой единый процесс. Также нельзя полностью разделить по структурным разделам и отчетность о промышленной деятельности предприятий. Это обстоятельство особенно четко проявляется на примере отраслевой автоматизированной системы управления. В едином подходе к отчетности состоит одно из принципиальных отличий в построении автоматизированной системы управления по сравнению с неавтоматизированной существующей системой управления. Известно, что в неавтоматизированных системах управления учет и отчетность как бы привязаны к отдельным функциональным или отраслевым управлениям — отдельно собирают свою отчетность управление труда и заработной платы, планово-экономическое управление, центральная бухгалтерия и финансовое управление и т. д. Они являются как бы хозяевами получаемой ими отчетности предприятий.

При разработке автоматизированной системы имеется возможность «обобществления» отчетности. Различные формы отчетности, собираемой порознь, содержат во многих случаях общие показатели, одни и те же данные, и поэтому применение единых унифицированных форм отчетности позволяет сократить общий поток отчетности и в то же время более полно обеспечить потребности органов управлений в конкретных видах обработанной информации.

В ЭАСУ в отношении использования данных отчетности реализуется принцип «направленности» экономической информации. Он заключается в том, что работникам аппарата не выдается сразу вся собранная и обработанная отчетность, а готовятся и выдаются по запросам конкретные справки, необходимые для принятия того или иного решения.

Запросы могут быть постоянные, действующие в течение длительного периода, и разовые — апериодические. По своему содержанию запросы могут быть стандартными, предусматривающими выдачу справок по заранее установленной форме, и нестандартными, предусматривающими выдачу

произвольных сведений из тех данных, которые хранятся в памяти ЭВМ.

Особенностью обработки информации в подсистеме оперативного управления является непосредственный ввод оперативных данных из каналов связи в электронные вычислительные машины, находящиеся в главном вычислительном центре министерства.

Подсистема целевого управления является характерной для машиностроительных министерств, создающих сложные изделия. Эта подсистема основана на использовании методов сетевого планирования и управления (СПУ) и предназначена для контроля за разработками, ходом внедрения в производство и производством сложных и дорогостоящих объектов техники, например уникальных турбин, блюмингов, кораблей и т. д.

В создании подобных объектов участвуют обычно сотни и тысячи предприятий, входящих в различные министерства, и поэтому подсистема целевого управления носит межотраслевой характер.

В соответствии со структурой разработки сложной системы (например, системы запуска искусственных спутников Земли), т. е. в соответствии с делением этой системы на составные части (например, ракета, радиолокатор, связь и т. д.) организуются и службы СПУ различных уровней; они создаются при главных конструкторах отдельных подсистем, т. е. крупных частей разработки. Эти части, в свою очередь, делятся на отдельные изделия, у которых имеются свои конструкторы, при которых создаются органы СПУ.

Опыт применения систем СПУ как в межотраслевом масштабе, так и в масштабе отрасли или предприятия показывает существенную эффективность этого метода. Он позволяет осуществлять более точное и полное планирование разработки на ее начальной стадии; выявлять взаимосвязи между отдельными соисполнителями и своевременно обнаруживать «узкие» места в ходе выполнения работ.

Указанные свойства метода СПУ позволяют сосредоточить внимание руководства в основном на событиях критического пути, определяющих окончательный срок разработки и своевременно принимать меры по предотвращению срывов сроков выполнения проектов.

Перейдем к делению ЭАСУ по третьему разрезу, соответствующему верхней плоскости. На ней показано пять информационных служб, обеспечивающих все функциональные подсистемы ЭАСУ: 1) служба нормативного хозяйства, 2) классификации и кодирования, 3) документации, 4) управления массивами информации и 5) служба алгоритмов и программ. Рассмотрим по порядку назначение указанных служб информационного обеспечения ЭАСУ.

Служба нормативов ЭАСУ. Нормативы являются главной экономико-информационной основой любой ЭАСУ, они определяют допустимые затраты материалов, труда, энергии и других ресурсов на производство единицы продукции или на выполнение определенного объема работ и связывают ЭАСУ с реальными условиями функционирования производства. Нормативы различных видов (трудовые, финансовые, материальные и т. д.) разрабатываются соответствующими специалистами-технологами, конструкторами, экономистами.

Оформление машинных массивов нормативов (кодирование и размещение их в памяти ЭВМ) производится с учетом особенностей работы ЭВМ с накопителями информации большой емкости. При этом основным требованием является обеспечение надежности хранения, достоверной и быстрой выборки данных, простоты корректировки норм.

Основные нормативы ЭАСУ укрупненно можно разделить на три группы: материальные нормативы, нормативы трудоемкости, финансовые и общеэкономические нормативы. Каждая из первых двух групп нормативов, относящихся к выпускаемой продукции, делится на четыре вида: подетальные, сводные, комплексные и агрегированные нормативы. Подетальные нормативы используются в основном на предприятиях; они определяют материальные и трудовые затраты для изготовления отдельных деталей изделий. Сводные нормативы определяют расход ресурсов на конкретные изделия целиком, а не на его составные блоки и детали; они используются как на предприятиях, так и в отраслевых системах управления. Комплексные нормативы — это нормативы расхода ресурсов на сложный технический комплекс или систему, состоящую из большого количества составных частей, разрабатываемых различными предприятиями. Естественно, что для каждого из предприятий эта составная часть является отдельным изделием и предприятие составляет нормативы только на свои изделия, считая их сводными нормативами.

На основе сводных нормативов по изделиям, поступивших в главный вычислительный центр отрасли, составляются комплексные нормативы, относящиеся к целому комплексу или системе; при этом получаются весьма укрупненные нормативы, которые позволяют главному вычислительному центру вести расчеты первых вариантов проектов планов. Естественно, что комплексные нормативы

зависят от принятой комплектности сложного изделия или системы.

Четвертый вид нормативов — это агрегированные нормативы, которые формируются и используются в главном вычислительном центре отрасли на основе сводных и иногда комплексных нормативов. Агрегированные нормативы относятся не к отдельному конкретному изделию или техническому комплексу, а к абстрактному изделию (комплексу), полученному в результате обобщения нормативов по группе однотипных изделий (комплексов). Например, могут быть агрегированные нормативы на телевизор первого класса вообще, а не на конкретную модель телевизора этого класса, на вычислительную машину среднего класса и т. д. Естественно, что агрегированные нормативы зависят не только от сводных (комплексных) нормативов по конкретным изделиям, объединяемым в группу, но и от структуры плана, т. е. от соотношения между количествами конкретных изделий разных видов в плане. Агрегированные нормативы используются в основном при составлении и согласовании контрольных цифр планов.

Следует отметить, что финансовые и общеэкономические нормативы относятся не к отдельным видам продукции, а к предприятию целиком и регламентируют его производственную и финансовую деятельность.

Для любой ЭАСУ важное значение имеет вопрос определения нормативов на новые изделия в связи со сжатыми сроками запуска новых изделий в серийное производство. На рис. 3 показан алгоритм формирования таких нормативов с помощью ЭВМ (на примере трудовых нормативов).

Создание нормативного хозяйства представляет собой важнейшую предпосылку для эффективного внедрения любой ЭАСУ.

Опыт показывает, что новая экономическая реформа изменила отношение предприятий к ведению нормативного хозяйства. Предприятия, заинтересованные в получении напряженных планов, более тщательно относятся к отработке нормативов, и это позволяет добиться получения более полных и точных нормативов.

Следует отметить еще одну очень важную черту взаимосвязи между ЭАСУ и нормативным хозяйством. ЭАСУ не только нуждается в полном и точном нормативном хозяйстве, но и способствует его созданию. Только при наличии ЭАСУ нормативы начинают использоваться в полной мере и тщательно анализируются и проверяются.

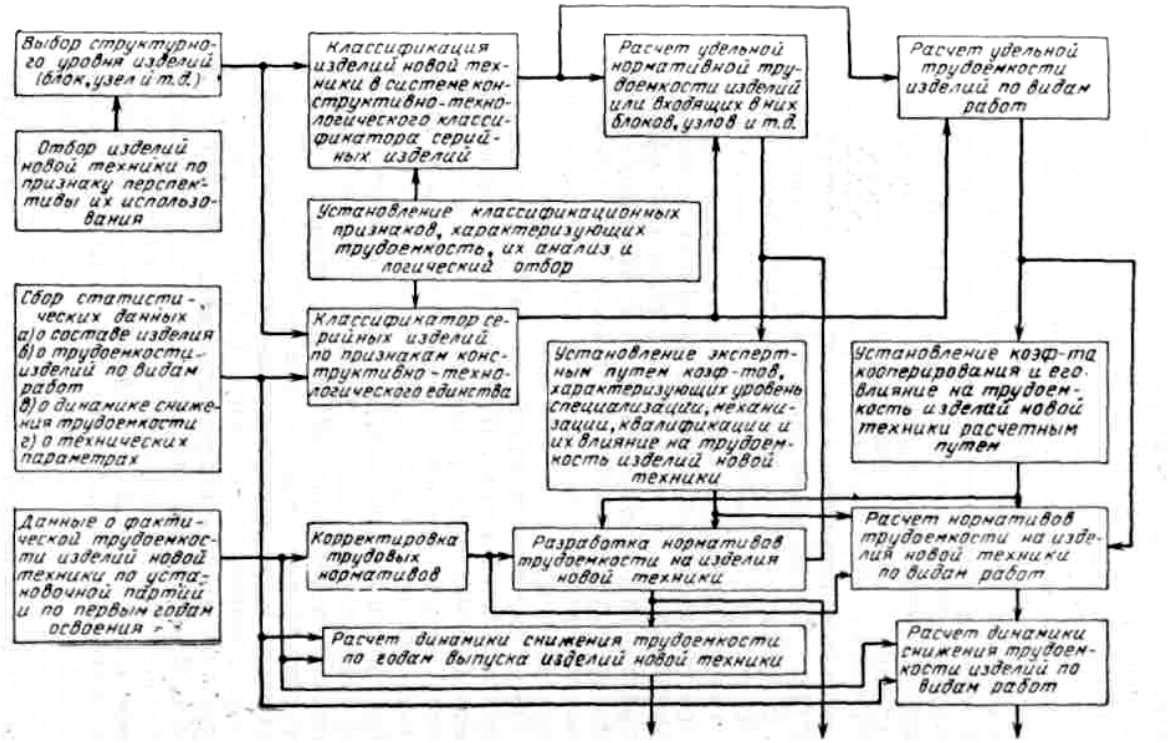


Рис. 3. Алгоритм формирования нормативов на новые изделия.

Такая взаимная связь между нормативным хозяйством и ЭАСУ является достаточно характерной для процессов перестройки управления в экономике под влиянием внедрения средств автоматизации обработки экономической информации.

Служба классификации и кодирования. Задачей этой службы является составление и ведение классификаторов (шифраторов) материалов, изделий, оборудования, кадров, документации,

поставщиков, потребителей и других объектов, информация о которых используется в ЭАСУ. Для решения внутренних задач предприятий и министерств обычно используются свои частные шифраторы и классификаторы. Создаваемый в настоящее время общесоюзный классификатор продукции является средством для получения единых по стране машинных шифров материалов и изделий, что должно существенно облегчить процессы автоматического обмена экономической информацией между различными предприятиями, ведомствами, министерствами, плановыми и статистическими органами страны.

Однако общесоюзный классификатор достаточно сложен с точки зрения принципов классификации изделий (особенно новой техники), что затрудняет его ведение. Частные классификаторы и шифраторы являются более простыми по структуре, их удобнее вести и использовать.

Поэтому для широкого применения внутри министерств и предприятий при массовой обработке экономической информации можно использовать частные классификаторы, которые строятся по весьма простому принципу — каждое изделие шифруется набором из нескольких групп цифр, указывающих номер предприятия или цеха и порядковый номер изделия.

Такой код значительно более удобен и с точки зрения ведения классификатора. Предметная классификация требует обязательно централизованного ведения классификатора, так как отдельное предприятие не может самостоятельно присваивать коды своим изделиям — это должно делаться в централизованном порядке специальной группой квалифицированных инженеров. Частные шифраторы допускают децентрализованное их ведение с обобщением всех кодов в вычислительном центре ЭАСУ.

Следует подчеркнуть значение общесоюзного классификатора продукции для общего упорядочения документооборота в стране. Общесоюзный классификатор продукции, особенно его высшие классификационные группировки, должен обеспечить систематизацию учета продукции в масштабе Советского Союза и способствовать упорядочению распределения номенклатуры и связей между различными плановыми, снабженческими и сбытовыми органами.

Существующее распределение номенклатуры продукции между различными плановыми, сбытовыми и снабженческими органами, относящимся к министерствам или общесоюзным ведомствам, в значительной мере определяется традиционно сложившимися связями и функциями этих органов и существенно расходится с делением продукции на группы, подгруппы и т. д. по общесоюзному классификатору. Указанные органы издают свои формы заявочных, плановых или отчетных документов, в которых группируют продукцию по своим признакам, в основном по принадлежности ее к тому или иному отделу, управлению, конторе и т. п. В этих документах часто проставляются даже коды, как правило, свои собственные, а иногда и коды из общесоюзного классификатора. Однако группировка изделий не соответствует общесоюзному классификатору, и эти коды не систематизированы в указанных документах в соответствии с высшими классификационными группировками общесоюзного классификатора.

Получая такие заявки, планы или отчеты, вычислительный центр не может использовать для упрощения процессов сортировки и обработки данных принцип универсальной десятичной классификации, на котором основан указанный классификатор и который является основным его достоинством. В силу указанного несоответствия между номенклатурой продукции, закрепленной за различными органами, и группировками продукции в общесоюзном классификаторе не представляется возможным даже внутри предприятий или главков пользоваться сокращенными кодами (например, только младшими разрядами для обозначения конкретных изделий или только старшими разрядами для обозначения групп или видов изделий), а необходимо пользоваться полными кодами, что загружает память ЭВМ и каналы связи.

Служба документации. Большое значение в ЭАСУ имеет работа по анализу различных экономических документов и их унификации. Документы в экономических системах играют тройную роль:

1) они являются непосредственно носителями входной (вводимой в ЭВМ) или выходной (выдаваемой ЭВМ) информации;

2) обеспечивают сохранение ответственности за информацию, так как они приспособлены для анализа и подписи их людьми;

3) являются основным средством взаимодействия между людьми и ЭВМ.

В связи с этим среди вопросов обработки экономической информации важнейшую роль играют процессы приема и обработки информации, вводимой с первичных документов (отчетов, накладных, заявок, планов и т. д.), и процессы формирования с помощью ЭВМ различных форм выходных

документов (сводных отчетов, планов, ведомостей и т. д.).

В ЭАСУ различаются два вида документации: внешняя документация, утверждаемая Госпланом, ЦСУ и другими вышестоящими органами, и внутренняя документация, разрабатываемая министерством или предприятиями и используемая внутри министерства и внутри предприятий.

Основным вопросом в деле разработки документации в ЭАСУ является ее приспособление к машинным методам обработки. При этом входная документация должна быть удобной для перфорации или для непосредственного чтения автоматическими считывающими устройствами, а выходная документация должна соответствовать требованиям печати документов на алфавитно-цифровых печатающих устройствах ЭВМ.

Служба управления массивами. Под массивами понимается вся постоянная и переменная информация, хранимая на машинных носителях в ЭАСУ. Данные для формирования массивов собираются и систематизируются однократно, в одном месте и по единой методике, определяемой службой управления массивами, чем достигается исключение дублирования информации и представление ее в виде, удобном для использования любой подсистемой в требуемых разрезах и формах. Служба управления массивами осуществляет подготовку и формирование массивов на машинных носителях и поддерживает их в рабочем состоянии. Этой службой ведется учет всех массивов, их обобщение (объединение для разных задач и подсистем); разрабатывается единая методика обращения к массивам и их обновления, ведется учет использования различных массивов в различных подсистемах и задачах. Таким образом, в ЭАСУ, в отличие от неавтоматизированных органов управления, где информация разделена по принципу принадлежности к разным органам управления и зачастую дублируется, обеспечивается создание единого, «общего» для всех подсистем информационного поля массивов.

Массивы информации ЭАСУ представляют собой огромный объем данных, который хранится в вычислительном центре на машинных носителях (перфокартах, лентах, магнитных лентах) и включает в себя справочные данные по предприятиям и изделиям, данные по составу оборудования, по планированию (текущему и прошлых лет), отчетные данные, нормативную информацию и целый ряд других данных. Здесь большое значение имеет создание и использование так называемых автоматизированных информационно-поисковых систем, которые обеспечивают быстрый поиск данных и выдачу их по запросам. В последующих главах будут подробно рассмотрены информационно-поисковые системы, применяемые в ЭАСУ.

Вторым важнейшим вопросом, относящимся к службе управления массивами, является построение специальной, достаточно сложной программы для ЭВМ, называемой операционной программой, которая обеспечивает управление процессом использования массивов, включение в работу различных рабочих программ и выдачу данных.

Служба математического обеспечения предназначена для создания и эксплуатации всех алгоритмов и программ, используемых в экономической автоматизированной системе управления, вместе с необходимыми инструкциями, определяющими порядок их разработки, применения и корректировки. Имеется две группы программ, образующих внутреннее и внешнее математическое обеспечение ЭАСУ. Внешнее МО представляет собой совокупность всех программ, которые решают непосредственно задачи учета, планирования, оперативного управления, т. е. конкретные экономические задачи, для которых создается ЭАСУ. Внутреннее математическое обеспечение включает программы, обеспечивающие автоматизацию процессов управления счетом и выработки программ, т. е. программы, автоматизирующие сам труд программистов. Известно, что ручное программирование задач представляет собой исключительно кропотливый и длительный процесс, и в связи с этим программистов постоянно не хватает.

Программист среднюю задачу составляет и отлаживает на машине в течение нескольких месяцев. Отладка программы на машине заключается в многократных попытках запустить эту программу при использовании заранее рассчитанных (вручную) контрольных данных. При этих пробных запусках обычно выявляются ошибки, причем процесс нахождения ошибок в программах весьма трудоемкий и в значительной мере случайный. На это тратится помимо большого ручного труда еще много машинного времени. Отсюда видно, что ручной труд программистов неэффективен по двум причинам, во-первых, потому, что он сам по себе медлителен и чреват ошибками, и, во-вторых, потому, что при этом еще тратится большое количество машинного времени на отладку программ. Для повышения производительности труда программистов применяются различные системы так называемой автоматизации программирования. Ниже будет подробно рассмотрена (в гл. 3) система, называемая АЛГЭМ, применяемая для автоматизации программирования экономических и математических задач во

многих экономических вычислительных центрах нашей страны. АЛГЭМ включает в себя специальный формальный язык, называемый алгоритмическим языком, на котором сравнительно просто можно записывать экономические задачи, и включает в себя так называемый транслятор.

Транслятор — это сложная программа, насчитывающая несколько тысяч отдельных команд, которая находится в вычислительной машине и позволяет машине осуществлять перевод той программы, которую записывает программист на упомянутом алгоритмическом языке в действительную машинную программу. Такая программа-транслятор для языка АЛГЭМ, приспособленного для экономических задач, полностью отработана и успешно облегчает труд многих программистов. При использовании средств автоматизации программирования резко повышается эффективность труда программистов, сокращаются сроки составления и отладки программ и экономится машинное время, затрачиваемое на отладку программ.

3. Основные алгоритмы ОАСУ

Для иллюстрации существа экономических задач рассмотрим более подробно основные задачи и алгоритмы в отраслевой автоматизированной системе управления, представляющей собой ЭАСУ среднего уровня. ОАСУ взаимодействует с АСУП — ЭАСУ предприятий и с ОГАС — государственной системой управления народным хозяйством.

Из того многообразия задач, которые можно себе представить на основе рассмотрения общей структурной схемы ЭАСУ, включающей в себя девять структурных разделов и четыре функциональные подсистемы (планирование, учет и отчетность, оперативное управление и целевое управление), в состав задач первой очереди ОАСУ входит комплекс задач, связанных с процессом годового планирования производства и материально-технического снабжения, а также отдельные задачи перспективного планирования, оперативного управления и целевого управления. Задачи оптимального планирования НИР и ОКР, планирования капитального строительства и ряд других задач оптимального планирования входит во вторую очередь ОАСУ. В первую очередь ОАСУ включена также задача механизированной обработки всех видов отчетности.

На рис. 4 показана общая модель ОАСУ первой очереди, представляющая собой функциональную укрупненную схему автоматизации процесса решения указанной комплексной задачи планирования и управления отраслью промышленности. Слева, в верхнем углу схемы, имеется группа блоков, объединенных надписью «Внешние связи»; она показывает процесс формирования заказов в промышленном министерстве. Сверху поступают задания от директивных органов, включающие в себя контрольные цифры плана, определяющие важнейшую номенклатуру, объем реализованной продукции, прибыль, фонд заработной платы и основные материально-технические ресурсы. Левый блок показывает поступление заказов от различных ведомств по номенклатуре продукции, планируемой самим министерством. Следующий блок, куда сходятся эти два потока заказов, предусматривает разукрупнение заказов и выявление потребности в изделиях межминистерской и внутриминистерской кооперации. Ниже находятся два блока, показывающие процесс составления межотраслевого баланса по группе взаимосвязанных отраслей промышленности. Многие отрасли тесно связаны между собой в процессе производства различных видов продукции.

Следующие три блока, обозначенные общим термином «Планирование», предусматривают выполнение трех основных этапов составления годового плана производства отрасли: распределение заданий между главками и заводами, расчет экономических показателей плана и расчеты данных по материально-техническому снабжению. Эти три этапа выполняются последовательно и многократно при постоянном сочетании машинных расчетов с анализом вариантов людьми. При разработке первой очереди ОАСУ принят принцип, согласно которому годовой план производства отрасли (и главков) должен составляться людьми (опытными плановиками, экономистами) с помощью машины. На упомянутом рисунке такой процесс совместной работы людей и машины проиллюстрирован. Для решения задачи планирования в память машины должны быть введены предварительно сведения о всех нормативах по изделиям, включая нормативы трудоемкости, расхода материалов и др.

Эти нормативы поступают из службы нормативного хозяйства ОАСУ, представленной на данном рисунке тремя верхними блоками, обозначенными общим названием «Нормирование». Нормативы по трудоемкости изделии задаются не вообще, а по типовым видам работ, нормативы расхода материалов - по группам материалов и комплектующих изделий с выделением дефицитных позиции. Кроме того в память машины должны быть введены сведения о мощностях и специализации заводов, состоянии

выполнения текущих планов, состоянии запасов, наличии фондов и т.д. Эти сведения получаются на основании обработки различных видов отчетности, что показано рядом нижних блоков рис. 4, объединенных названием «Отчетность». Следует отметить, что здесь должны учитываться данные и о новых мощностях, вводимых в текущем и планируемом году.

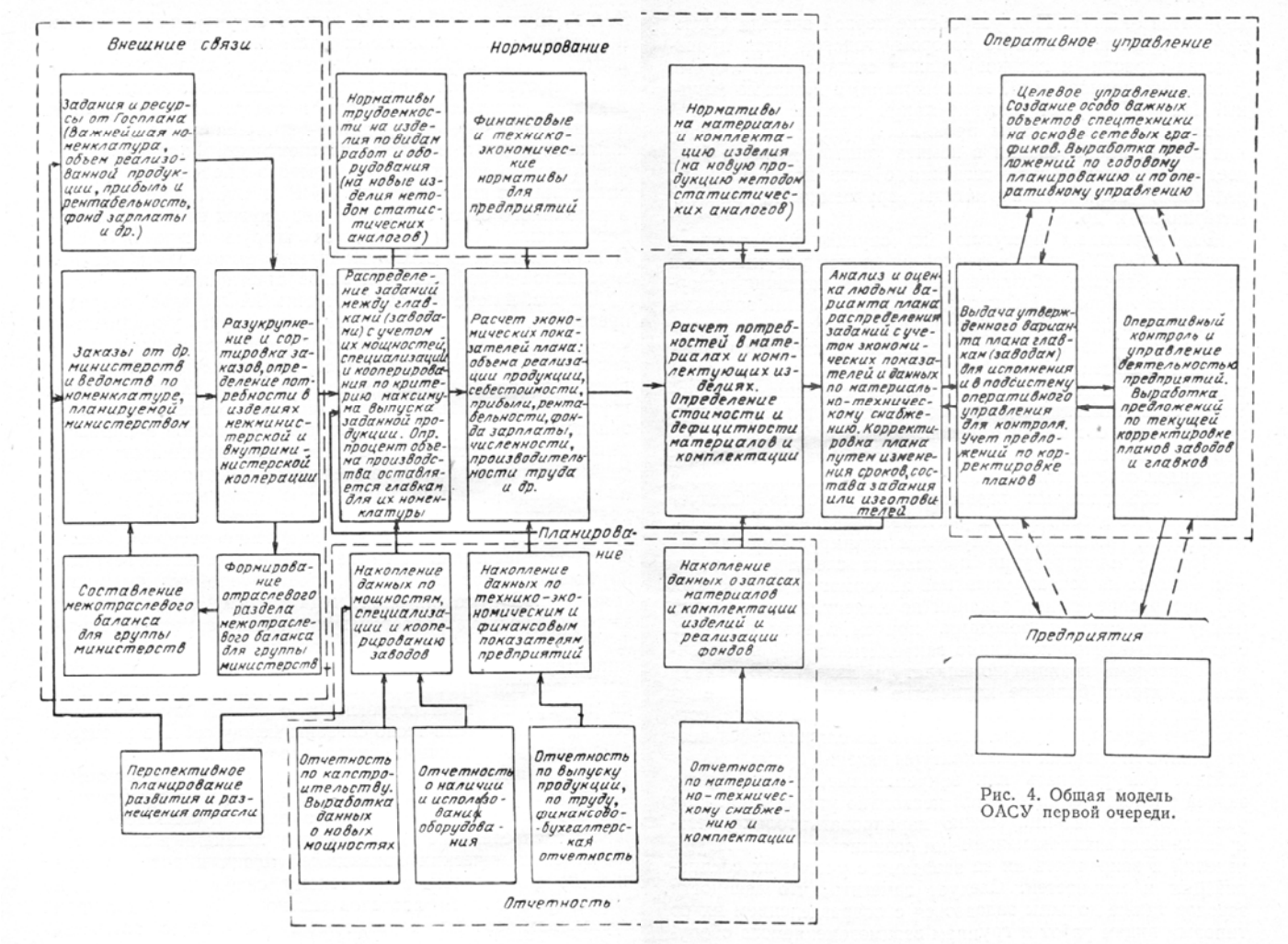


Рис. 4. Общая модель ОАСУ первой очереди.

Процесс планирования происходит следующим образом: машина на основе указанной информации дает первый вариант распределения заданий по номенклатуре изделий между главками и заводами, причем в этом варианте сразу учитываются постоянно закрепленные за определенными заводами позиции номенклатуры изделии. В планах всегда имеется большое количество изделии, которые из года в год изготавливаются на одном и том же заводе, есть сложные изделия, идущие постоянно по сложившейся кооперации. Эти позиции номенклатуры изделии будут в общей таблице фигурировать как закрепленные, и тем самым задача распределения заданий несколько упрощается. При распределении заданий можно варьировать только незакрепленными взаимозаменяемыми позициями номенклатуры изделий и закреплять их за заводами с учетом их специализации и мощностей. Следует заметить, что мощности заводов также должны задаваться с подразделением их по типовым видам работ и группам взаимозаменяемого оборудования. Как решение первого этапа указанной задачи машина вырабатывает определенный вариант распределения заданий и переходит ко второму этапу, заключающемуся в расчете экономических показателей данного варианта плана. Определяется объем реализуемой продукции, прибыль, рентабельность, фонд заработной платы; полученные данные сравниваются с теми данными, которые были выданы Госпланом в качестве контрольных цифр. После этого выполняется третий этап, на котором производится укрупненный расчет потребности в материалах и комплектации для этого варианта плана. В результате выполнения указанных трех этапов вырабатываются три таблицы: таблица варианта распределения заданий по номенклатуре, таблица экономических показателей, таблица потребностей в средствах материально-технического снабжения с указанием особо дефицитных позиций. Выданные машиной таблицы поступают в планово-экономическое управление, которое совместно с отраслевыми главками анализирует их, принимает решения по корректировке данного варианта плана либо путем изменения исполнителей, либо путем

изменения сроков, изменения номенклатуры изделий и т. д. Выработанные людьми корректировки плана вводятся снова в вычислительную машину, которая рассчитывает второй вариант и опять выдает такие же три таблицы (распределения заданий, экономических показателей, потребностей в средствах материально-технического снабжения). Эти таблицы снова анализируются планово-экономическим управлением, весь процесс повторяется и так до тех пор, пока вариант плана не будет признан приемлемым. После этого он передается на предприятия, которые на основе контрольных цифр, полученных от министерства, должны разработать свои планы. После того как планы министерства, главков и предприятий утверждены, они передаются в подсистему оперативного управления для контроля и в систему целевого управления для выработки по ним целевых планов, т. е. для корректировки сетевых графиков и для выборки из различных разделов общего плана тех сведений, которые относятся конкретно к определенному объекту техники, выпускаемому сложной кооперацией заводов.

Рассмотрим блок-схемы алгоритмов экономических задач, составленные для реализации на машинах типа «Минск-22». Это задачи управления сбытом, управления материально-техническим снабжением и комплектацией и управления главного механика и энергетика.

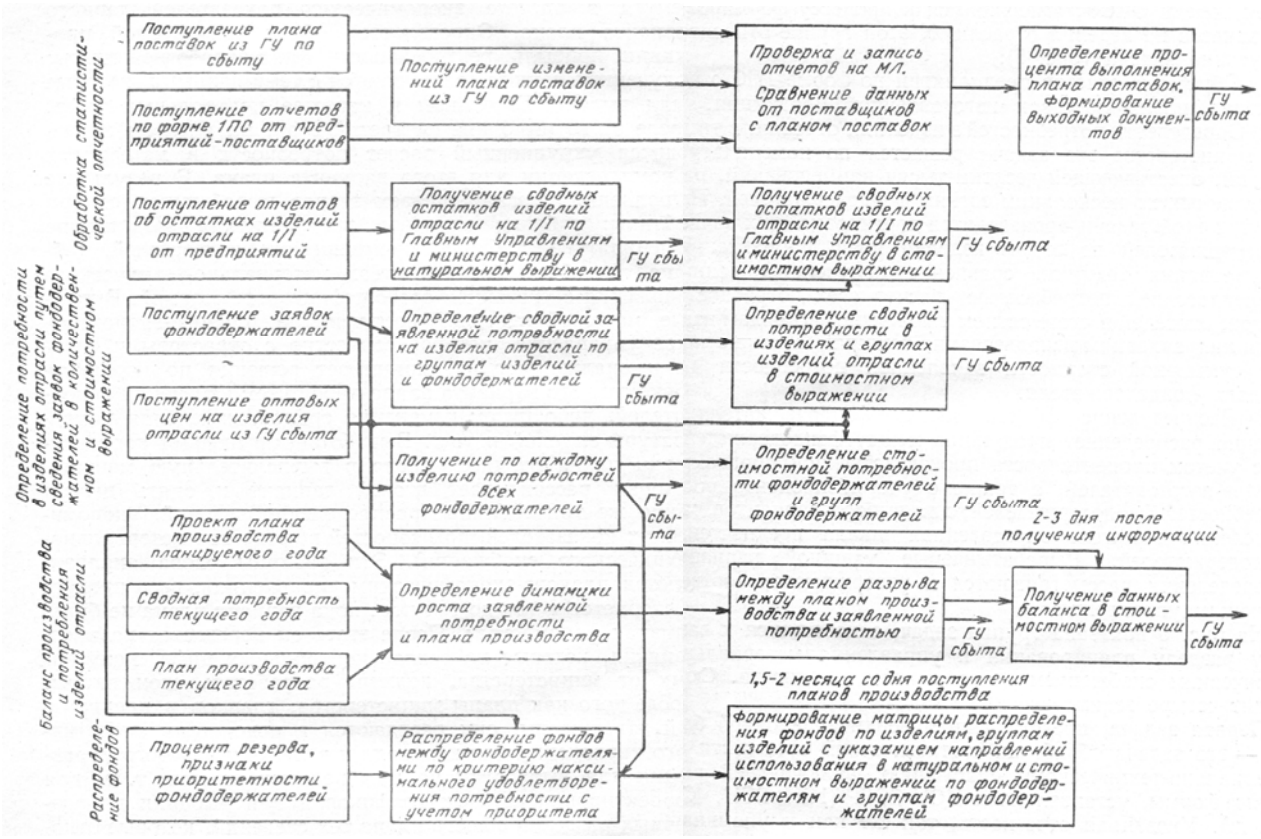


Рис.5. Задачи управления сбытом

Задачи управления сбытом соответствуют двум блока общей модели ОАСУ, относящимся к процессу формирования заказов на изделия отрасли. К этой группе относятся следующие задачи (рис. 5):

1) Обработка отчетов предприятий по форме 1ПС о выполнении плана поставок изготовленной продукции.

2) Определение потребностей в изделиях отрасли; в типовом министерстве эта задача решается по номенклатур изделий, охватывающей десятки тысяч наименований, распределяемых по нескольким сотням фондодержателей. При решении этой задачи производится сортировка всех заявок фондодержателей по видам изделий и по направления использования техники, сравнение полученных данных с прошлогодней потребностью, расчет всех потребностей фондодержателей в стоимостном выражении по видам изделий и направлениям использования техники, а также подсчет суммарной стоимости заявленной потребности для каждого фондодержателя.

3) Распределение фондов между фондодержателями Машина распределяет продукцию между фондодержателя ми с учетом процента роста производственной программы заводов-изготовителей, а также с учетом процента роста потребностей фондодержателей; для выбора вариантов распределения

вводится определенная шкала предпочтения фондодержателей. Вырабатываемые машиной варианты распределения рассматриваются и утверждаются соответствующим управлением.

На рис. 6 показана группа задач, относящихся к важному разделу планирования и управления материально-техническим снабжением отрасли промышленности. Сюда входят четыре взаимосвязанные задачи.

Первая задача, представленная верхней цепочкой блоков — это задача обработки статистической отчетности по материально-техническому снабжению, включающей различные формы, установленные ЦСУ СССР (1СН, 2СН, 4СН и т. д.). Указанная отчетность, так же как и упоминавшаяся раньше отчетность по сбыту, является частью общей для ОАСУ подсистемы учета и отчетности.

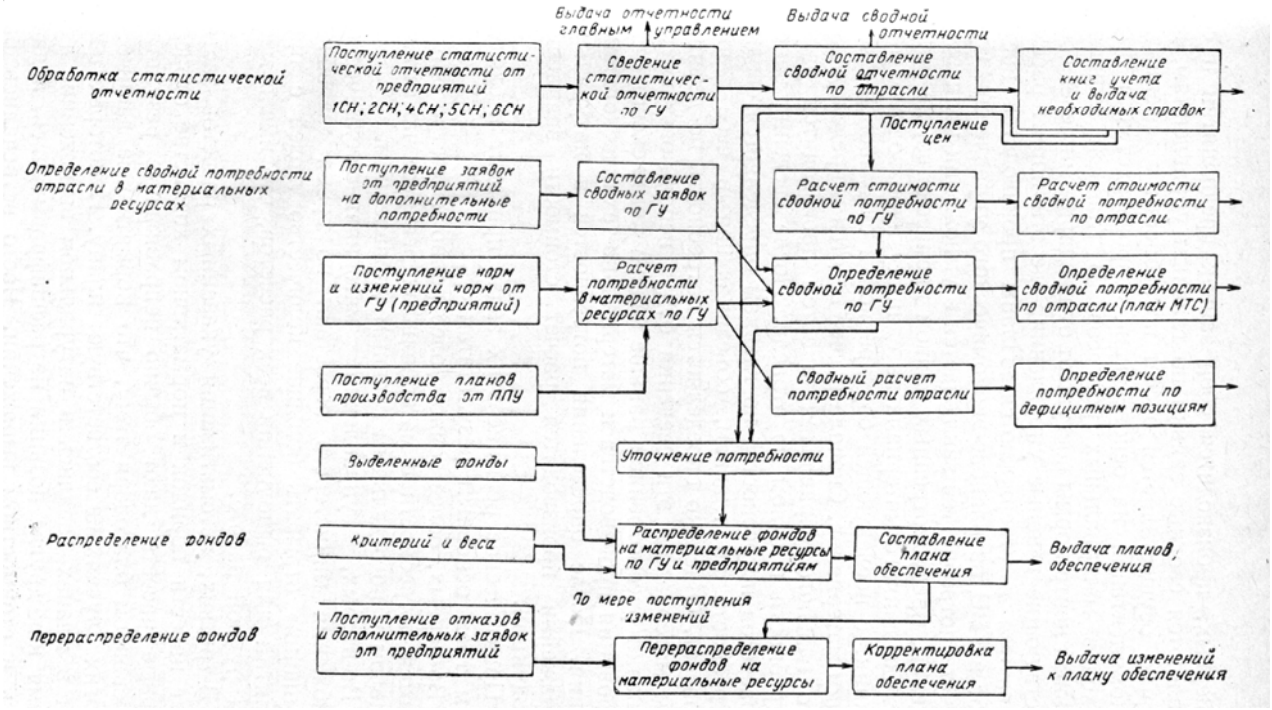


Рис. 6. Задачи управления материально-техническим снабжением.

Вторая задача рассматриваемой группы — это определение потребностей в материалах и комплектующих изделиях. Определение потребности ведется двумя способами: во-первых, производится сбор заявок предприятий и составляются сводные заявки по главным управлениям, во-вторых, осуществляется расчет потребности в материалах и комплектации на основе планов производства и норм расходов материальных ресурсов. В главный вычислительный центр нормы поступают от предприятий через соответствующие главные управления, где эти нормы рассматриваются и утверждаются. Планы производства поступают в ГВЦ из планово-экономического управления. Суть расчета потребностей заключается в умножении количества изделий, предусмотренных планом производства, на норму расхода данного материала на единицу производимой продукции, с последующей сортировкой и суммированием по видам материалов. Сначала расчет ведется в пределах каждого главного управления, а затем производится расчет потребности в целом по отрасли. После сведения заявок и расчетов потребностей по главкам определяется сводная потребность по главкам, объединяющая в себе заявки и расчет. Отдельно определяется потребность по дефицитным материалам; эти сведения будут использоваться при решении рассмотренной раньше задачи комплексного планирования производства и материально-технического снабжения. После определения потребностей в натуральном выражении производится расчет стоимости заявленной продукции.

Практическое решение этих задач приводит к следующим трем основным результатам.

Во-первых, более точно, более полно и быстро определяются потребности в материальных ресурсах по отрасли в целом и по отдельным главным управлениям.

Во-вторых, управления материально-технического снабжения министерств перестают во время «заявочной компании» отрывать большое количество людей с предприятий для выполнения всех расчетов, что имело место при ручной работе.

В-третьих, автоматизация указанных расчетов заставляет навести порядок в нормативном хозяйстве отрасли. Первые расчеты дают обычно неправильные результаты в основном благодаря тому, что

нормативные данные во многих случаях не соответствуют плану. Некоторые позиции в планах давались в натуральном выражении, в то время как другие позиции не расшифровывались, а давались в стоимостном выражении. Часто встречались расхождения между планами и нормами в единицах измерений, в названиях материалов и изделий и т. д. Требуется большая работа главного вычислительного центра министерства для того, чтобы выявить все эти ошибки и несоответствия и добиться их устранения. Для этих целей создаются специальные программы для ЭВМ.

Систематическая работа ГВЦ по проверке нормативов, их записи на машинные носители, корректировке и обновлению, а также практическое использование нормативов в расчетах потребностей оказывают влияние на отношение к нормативам работников предприятий. То обстоятельство, что в ГВЦ тщательно проверяются нормативы предприятий, заставляет работников предприятий более ответственно подходить к их составлению и оформлению.

Нижняя цепочка блоков рис. 6 показывает задачи распределения фондов на материальные ресурсы с помощью машины и перераспределения фондов. Распределение полученных фондов осуществляется в два этапа: сначала между главками, а затем между предприятиями внутри главка. При распределении фондов учитывается состояние запасов и важность выполняемых работ. Задача перераспределения фондов в процессе их реализации должна решаться в соответствии с изменяющимися условиями производства и материально-технического снабжения два раза в год. Для внедрения указанных задач в практику необходимы предварительные экспериментальные расчеты с целью проверки и уточнения алгоритмов. Суть экспериментальных расчетов заключается в том, что из материалов прошлых лет берутся данные, касающиеся выделенных фондов и их фактического распределения и перераспределения. Задачи распределения и перераспределения фондов решаются на машине с исходными данными о фондах, производственных программах и потребностях предприятий, соответствующими прошлому году, и полученные результаты сравниваются с тем фактическим распределением фондов (или перераспределением), которое осуществлялось в прошлом. Анализируются расхождения между машинным решением и фактическими данными и устанавливается, следует ли вносить изменения в машинный алгоритм или машинное решение является рациональным и менять алгоритм не следует.

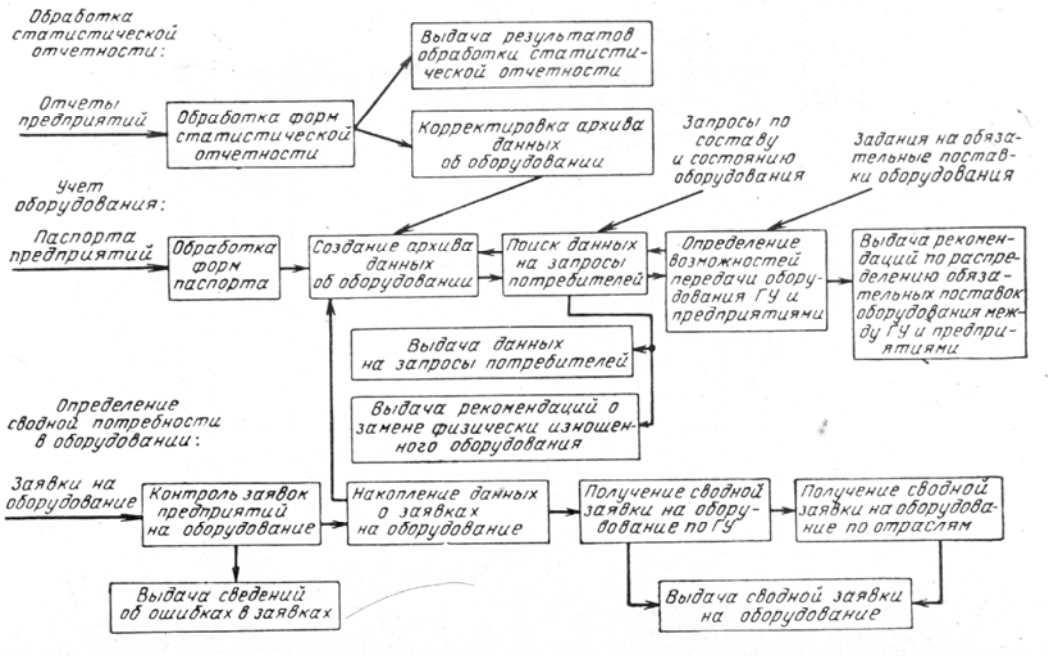


Рис. 7. Задачи управления главного механика

На рис. 7 показан комплекс задач, относящихся к учету и распределению оборудования. Верхние три блока представляют задачу обработки статистической отчетности предприятий по оборудованию и энергии. Средняя цепочка блоков представляет задачу учета оборудования; она решается на основе обработки паспортов предприятий с учетом данных статистической отчетности и единовременных отчетов. В результате формируется архив данных по оборудованию. В этом архиве используется упомянутая выше автоматизированная информационно-поисковая система (ИПС). Она позволяет получать любые справки по любому предприятию, главку или отрасли в целом о наличии и

использовании оборудования.

На основе накопленных в ИПС данных по специальной программе машиной решается задача определения возможности передачи излишнего оборудования от предприятий при выполнении обязательных поставок оборудования и выработки рекомендаций по распределению этих поставок.

Третьей задачей, которая решается с помощью указанной ИПС, является задача выработки рекомендации о замене физически изношенного и устаревшего оборудования. Несколько автономной является задача, также решаемая в интересах управления главного механика и энергетика, по определению потребностей в оборудовании и составлению сводных заявок на оборудование по главкам и отрасли в целом. Эта задача решается путем сбора и контроля сбора заявок предприятий и при ее решении используются сведения о наличии, состоянии и использовании оборудования на предприятиях, накопленные в ИПС.

Следующая большая и сложная задача ОАСУ, которая уже практически решается во многих промышленных министерствах, относится к проблеме оптимального перспективного планирования развития и размещения отрасли.

Рассмотрим постановку такой задачи на примере перспективного планирования развития и размещения производства радиоприемников и телевизоров.

При постановке этой задачи целесообразно использовать общепринятый в подобных случаях критерий оптимизации — получение минимума приведенных затрат. Учитываются три обычные группы факторов, характеризующих производство, а именно: потребность в данных видах продукции, возможные варианты расширения, реконструкции и строительства заводов и, наконец, возможные варианты перспективных моделей изделий с их характеристиками по трудоемкости изготовления и затратам.

Для того чтобы разработать все возможные варианты, необходимо подготовить огромный объем исходной информации; в такой работе обычно участвуют многие проектные, технологические институты и организации соответствующей отрасли промышленности.

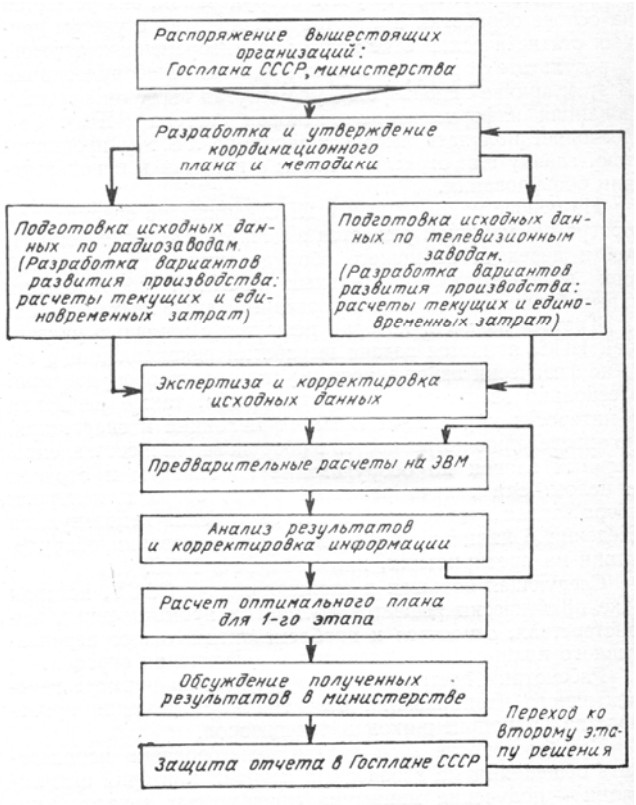


Рис. 8. Процесс решения задачи перспективного планирования производства.

На рис. 8 приведена математическая постановка задачи и общая схема процесса решения, состав исходной информации. Математически задача представляет собой один из видов задач линейного программирования. На первом этапе решения задачи может быть принят вариант, не учитывающий транспортных затрат ввиду их незначительной роли в данном виде производства.

Следует подчеркнуть трудность и большой объем предварительной работы по разработке различных вариантов развития, специализации и размещения заводов, по анализу этих вариантов, который должны

проделать высококвалифицированные специалисты, технологи, экономисты, проектировщики заводов и др. Особенно большая и сложная в техническом отношении работа должна быть проделана по анализу перспектив специализации предприятий, определению направлений технического прогресса в области разработок и производства соответствующих изделий, анализу реальных усовершенствований технологических процессов, которые могут быть достигнуты в планируемый период, определению ожидаемых нормативов расхода материалов, трудоемкости, определению зависимости себестоимости изделий от вариантов развития производства и объемов капитальных вложений.

Большое значение имеет и третий входной поток данных для решения этой задачи — это анализ ожидаемой потребности планируемых изделий со стороны населения и народного хозяйства. При определении потребности учитывается ожидаемый рост народонаселения.

4. Типовые информационные процессы ЭАСУ

В связи с огромным разнообразием существующих форм и методов управления в экономике (наличие разнотипных предприятий, трестов, главков, ведомств, министерств и. т. д.) большое значение приобретают вопросы унификации алгоритмов и программ для экономических автоматизированных систем управления. Решение этой проблемы осуществляется проведением работы в следующих 3-х направлениях:

1) построением математического обеспечения различных экономических автоматизированных систем управления из унифицированных блоков;

2) выделением более или менее однотипных экономических объектов (предприятий, отраслей, ведомств) и разработкой для них унифицированных систем математического обеспечения;

3) перестройкой существующих систем управления экономическими объектами в направлении типизации информационных процессов и их организационных структур с целью использования в них типовых алгоритмов и программ.

Работы по указанным трем направлениям унификации математического обеспечения должны проводиться комплексно; это значит, что при разработке конкретных автоматизированных систем управления должны учитываться как возможности приведения этих систем к какому-либо определенному типу, так и возможности построения их из унифицированных блоков.

Практически наибольшее значение имеет путь построения систем из унифицированных блоков (подпрограмм), так как унификация на уровне элементов систем более реальна, чем унификация на уровне систем в целом.

В решении этой проблемы важнейшую роль играет наличие отраслевых и межотраслевых фондов алгоритмов и программ, в которых должны накапливаться отработанные и проверенные алгоритмы и программы для различных типовых задач, учитывающих специфику определенных классов предприятий и отраслей промышленности.

Каждая функциональная подсистема в любом структурном разделе содержит комплекс взаимно связанных информационно-логических и вычислительных задач. В состав такого комплекса входит обычно 10—15 задач.

Например, в разделе материально-технического снабжения в комплекс задач планирования будут входить: определение потребности в материалах на планируемый период, распределение фондов, перераспределение фондов и запасов в соответствии с изменениями плана производства и фактической обеспеченностью предприятий и т. д.

Каждая информационно-логическая или вычислительная задача строится из набора стандартных и нестандартных блоков (подпрограмм). Примерами блоков могут служить: получение итоговых данных из массивов однотипных документов, формирование документов для выдачи на печать, сортировка записей данных, поиск объектов по признакам в массивах записей, расчет вариантов распределения ресурсов и т. д.

В конечном счете любая система, подсистема, раздел или задача должны строиться из стандартных блоков с добавлением минимального числа нестандартных блоков. Унификация общей структуры экономических автоматизированных систем управления путем разделения их на однотипные структурные разделы, функциональные подсистемы, информационные службы, а также унификация элементарных алгоритмов (блоков и подпрограмм), из которых строятся различные задачи ЭАСУ, позволяет разделить все математическое обеспечение экономической автоматизированной системы управления на более или менее автономные части, разработка которых может идти параллельно и в

значительной мере независимо.

Этот подход обеспечивает упрощение и сокращение сроков разработки и внедрения систем и, главное, возможность внедрения этих систем по частям (по подсистемам, разделам, задачам). С другой стороны, указанный подход облегчает решение задачи совместимости отдельных частей систем и различных взаимодействующих между собой экономических автоматизированных систем управления за счет применения общих алгоритмов планирования, учета, оперативного управления, а также методов программирования, организации массивов, за счет применения единых шифров и кодов, общих нормативов и т. д.

Особого внимания с точки зрения унификации алгоритмов и программ в экономических автоматизированных системах управления заслуживает использование так называемых универсальных алгоритмов обработки информации, рассчитанных на определенный класс задач и настраиваемых на конкретные задачи с помощью дополнительной информации.

Типовой состав основных задач, решаемых в экономических вычислительных центрах, с точки зрения их экономического содержания может быть разбит на следующие классы:

1) Обработка отчетности. Здесь главное внимание при программировании должно обращаться на запись и контроль исходных данных, формирование требуемых выходных документов, организацию рациональной сортировки. Для решения задач этого класса широкое применение находят универсальные алгоритмы обработки отчетов и вообще табличных документов, настраиваемые на конкретные формы документов с помощью таблиц параметров.

2) Расчеты потребностей в материалах по планам производства изделий и нормам расхода материалов. Суть задачи сводится к сортировке однотипных материалов и изделий и вычислению сумм парных произведений. Основное внимание при решении этих задач должно обращаться на рациональную организацию нормативного хозяйства, обеспечивающую полное и своевременное получение нормативов от предприятий. При машинной реализации этих задач важным является обеспечение быстрого поиска и простоты корректировки нужных норм. Для этих целей эффективно используются методы ассоциативного программирования.

3) Задачи оптимального перспективного планирования развития и размещения производства, сводящиеся в основном к построению различного вида балансов производства и распределения продукции (отраслевых, межотраслевых, региональных, межрайонных и др.). С точки зрения программирования особенностью этих задач является оперирование с матрицами большой размерности и применение стандартных методов и программ решения задач линейного программирования (симплекс-метод и др.).

4) Задачи оптимального текущего планирования производства и распределения заданий между производственными единицами (министерствами, главками, заводами, цехами, участками). Эти задачи решаются в основном методами линейного программирования.

5) Сбор и обработка оперативной информации. Эта информация поступает по каналам связи непосредственно в ЭВМ в реальном масштабе времени одновременно от многих удаленных абонентов. Особенностью данного класса задач является необходимость контроля передач сообщений и обеспечение оперативного взаимодействия (двустороннего обмена информацией) между ВЦ и удаленными абонентами.

6) Задачи управления запасами, включающие оперативный контроль уровней, оптимальное распределение между потребителями и оптимальное планирование заказов на пополнение запасов. При решении этих задач используются методы линейного программирования и теории массового обслуживания.

7) Информационно-логические документальные и фактографические (справочные) задачи. Характерным для этих задач является накопление в памяти машины больших массивов справочной информации об оборудовании, кадрах, площадях, выпускаемых изделиях, используемых материалах и полуфабрикатах и т. д. Программная реализация таких задач основана на применении так называемых информационно-характеристических таблиц, ассоциативных узловых структур и других приемов.

8) Задачи моделирования процессов производства и материально-технического снабжения (детерминистического и статистического). Характерным является использование специальных приемов и средств для получения случайных чисел, применение типовых процедур (подпрограмм) для синхронизации моделируемых событий, построение цепных списков для управления очередностью событий. Эти особенности требуют наличия специальных средств моделирования экономических систем, включаемых в состав универсальных алгоритмических языков, или применения

специализированных языков.

9) Задачи сетевого планирования и управления процессами создания сложных изделий. Здесь характерным моментом является применение методов оптимизации сетевых графиков с учетом времени и стоимости разработок и производства продукции. Создаются типовые комплексы алгоритмов и программ для расчета и обновления сетевых графиков и обеспечения взаимодействия ЭВМ и людей в процессе анализа сетевых графиков и принятия решений.

Теперь мы рассмотрим некоторые примеры типовых задач с точки зрения их машинной реализации. К числу таких задач относятся:

а) обработка массивов записей;

б) накопление и поиск данных в иерархических классификационных системах;

в) библиографический поиск;

г) фактографический поиск.

Обработка массивов записей. Этот тип процессов является основным при решении экономических задач. Записью называется точно установленный набор данных, характеризующих некоторый объект или процесс. Примерами записей могут служить товарные чеки, бланки для расчета заработной платы, наряды на выполнение работ, накладные для получения каких-либо материалов и т. д. В области учета кадров примерами записей могут служить анкеты или личные листки, в здравоохранении — истории болезней, статистические отчеты и т. п. Обычно различного рода записи используются большими группами — массивами, и обработка их носит массовый характер. Весь процесс обработки массивов записей складывается из ряда этапов:

1) Фиксация записей на первичном носителе (перфокарты, перфоленты, бланки со специальными магнитным и стилизованным шрифтом или бланки с графитовыми отметками, воспринимаемые непосредственно читающими устройствами машин, и др.).

2) Ввод записей в машину и размещение их на магнитных лентах. Как правило, для больших объемов данных (сотни миллионов знаков) используются магнитные ленты; магнитные диски и барабаны используются для средних (десятки миллионов знаков) и небольших массивов. Поэтому в качестве типового случая следует рассматривать использование магнитных лент (МЛ).

3) Обработка массивов записей, находящихся на магнитных лентах. Записи по одной или группами последовательно переписываются с МЛ в оперативную память машины, там обрабатываются и выводятся на магнитные ленты. Основной особенностью обработки является то, что один и тот же алгоритм обработки применяется ко всем однотипным записям массива, т. е. обработка носит *параллельно-циклический* характер.

4) В оперативной памяти машины обычно при подобных процессах обработки выделяются пять областей: для размещения программы (программа может тоже по частям вводиться в оперативную память), для размещения констант, для приема группы обрабатываемых записей, рабочая область для хранения промежуточных результатов и выходная для размещения обработанных записей перед выдачей их на МЛ.

5) Вывод обработанных данных с МЛ на печать в форме, удобной для чтения или дальнейшего использования на машинах.

Обработка записей включает в себя не только преобразование данных, находящихся в записях, и выдачу их в виде J записей другой формы, но и получение каких-либо общих показателей по всем записям массива или группам записей (например, подсчет общего количества и стоимости проданных товаров, если записями являются товарные чеки, или подсчет общей потребности в материалах различных видов, если записями являются заявки на материально-техническое снабжение, поступившие от многих предприятий и т. д.).

В отличие от процессов математических вычислений в подобных процессах обработки информации наибольшую трудность представляет не построение алгоритмов вычислений, а организация обращений к данным (их опознавание, выборка нужных величин из записей, перестроение записей, ввод и вывод и т. д.). Поэтому в алгоритмических языках, предназначенных для обработки информации, важное место занимает методика описания данных, как исходных, так и результирующих. Рассмотрим, например, следующую запись, представляющую собой отчет некоторого предприятия о наличии кадров:

Номер предприятия: 2462

Дата составления отчета: 25.05.66

Общее число работников: 698

Наличное число работников: 650

Больных: 24
В отпусках: 15
В командировках: 6
Отсутствующих по другим причинам: 3

Предположим, что подобные отчеты представляются многими предприятиями и они должны обрабатываться с помощью ЭЦМ. Ясно, что так как все эти отчеты имеют одинаковый формат, то на магнитную ленту достаточно записывать только сами числа без соответствующих названий данных, т. е. можно перечисление названий данных с указанием их разрядности и вида задать один раз в самой программе обработки в виде специального описания данных. Пользуясь этим описанием данных, машина по специальной программе сможет выделить из сплошной последовательности чисел на магнитной ленте запись, относящуюся к любому интересующему нас предприятию, а также выделить из выбранной записи любые требуемые данные (например, число больных, или число командированных).

Таким образом, для каждого массива, содержащего однотипные записи, должно быть сделано точное описание формата записей, указано общее их число и порядок расположения на магнитной ленте. Подобные описания, сделанные для всех типов записей, участвующих в задаче, являются неотъемлемой составной частью программы. Методика составления таких описаний будет подробно рассмотрена в дальнейшем.

Накопление и поиск данных в иерархических классификационных системах. Весьма распространенным способом организации информации о больших количествах различных объектов является использование иерархических (т. е. многоуровневых, древовидных) классификационных систем. Примерами таких систем могут служить универсальная десятичная классификация литературы (УДК), единая десятичная классификация материальных продуктов, различные частные десятичные классификаторы материальных продуктов (классификатор проката черных металлов, классификатор полупроводниковых приборов и т. д.). Все иерархические классификационные системы строятся по общему принципу: классифицируемая совокупность объектов делится сначала по основному признаку (например, по назначению) на несколько крупных классов, затем каждый класс делится по определенному признаку на ряд подклассов, которые, в свою очередь делятся на другие более мелкие подразделения (типы, виды и т. д.). В результате образуется классификационное дерево, отражающее выбранную схему деления совокупности предметов. Деревья не обязательно должны быть симметричными и иметь одинаковое число уровней во всех ветвях. Не обязательно также, чтобы было одинаковое количество разветвлений в каждой точке деления. В наиболее простых случаях основание классификационной системы выбирается постоянным для всех уровней дерева (например, число 10), практически же не все возможности деления используются сразу (т. е. при создании системы). На каждом уровне оставляют запасные подразделения классификации, обеспечивающие возможность включения новых разновидностей объектов.

В процессе работы классификационные системы могут дополняться и развиваться как в направлении заполнения свободных классификационных рубрик на различных уже имеющихся уровнях, так и в направлении наращивания новых уровней в тех или иных ветвях. Для того чтобы внести конкретный объект в заданную классификационную систему, ему необходимо приписать многопозиционный код, каждая из позиций которого определяет номер классификационного подразделения на соответствующем уровне, причем номер позиции в коде (слева направо) указывает номер уровня дерева сверху вниз. Так, например, трехзначное десятичное число соответствует трехуровневому дереву в десятичной системе классификации, причем старший разряд указывает номер классификационного подразделения на верхнем уровне, второй разряд — на среднем уровне и младший разряд — на нижнем уровне. Число уровней соответствует числу разрядов.

Примером иерархической десятичной классификационной системы может служить следующая система учета кадров. Допустим, что весь личный состав сначала делится на основные классы в зависимости от производственной категории (неквалифицированные рабочие, квалифицированные рабочие, средний технический персонал, инженерный состав, научные работники, административный состав, вспомогательный состав). Затем каждая из этих категорий делится на более узкие категории, например: рабочие — по разрядам, научные работники — по ученым степеням (без ученых степеней, кандидаты наук, доктора наук, члены-корреспонденты и академики). Далее можно представить себе деление на третьем уровне по специальности (например, для научных работников: математики, физики, механики, электрики, радисты), на четвертом уровне — по стажу работы (до 5, от 5 до 10, от 10 до 15, от 15 до 20, свыше 20 лет), на пятом— по возрасту (до 25, от 25 до 35, от 35 до 50, от 50 до 65, свыше 65

лет).

Для поиска человека, отвечающего определенному набору признаков, необходимо задать соответствующий многопозиционный код, каждая из позиций которого указывает номер рубрики на соответствующем уровне. Например, код 53133 будет обозначать: научный работник, доктор наук, математик, стаж работы от 10 до 15 лет, возраст между 35 и 50 годами. Поиск объектов по заданным признакам осуществляется путем прослеживания дерева сверху вниз. Сначала просматривается список подразделений верхнего уровня и там находится признаковый код, соответствующий цифре старшего разряда, заданного для поиска кода (53133). Там же указывается адрес, где находится в памяти машины список, представляющий собой второй уровень классификации, ответвляющийся от данного подразделения. Аналогичным образом просматривается этот список и происходит переход к списку третьего уровня. Этот процесс продолжается до тех пор, пока не будет достигнут список самого нижнего уровня, который указывает уже на список объектов, обладающих всеми заданными признаками.

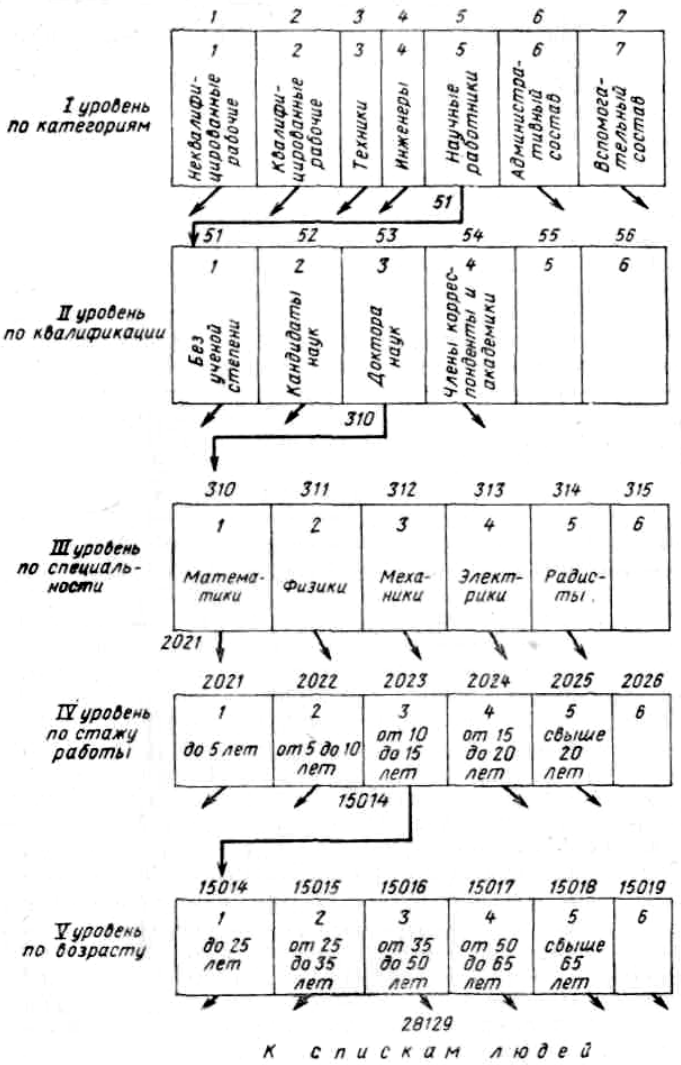


Рис. 9. Схема иерархической классификационной системы для учета кадров.

На рис. 9 приведен пример подобной классификационной системы. Над клетками указаны адреса соответствующих ячеек (взятые ориентировочно). Внутри клеток сверху указаны значения кодов признаков по соответствующим позициям, причем если ячейки расположены подряд, то эти значения могут в явном виде не храниться. Снизу под клетками, соответствующими коду запроса 53133, указаны адреса начальных ячеек подсписков, отходящих от данных клеток. Например, адрес 28129 взят условно как адрес ячейки, в которой хранятся данные о специалисте, обладающем признаками, отвечающими этому коду.

Основные трудности, которые возникают при реализации иерархических классификационных систем на электронных цифровых машинах, связаны с тем, что в процессе их эксплуатации происходит изменение классификационного дерева и состава объектов, информация о которых накапливается. Поэтому в таких системах, как правило, не представляется возможным заранее предусмотреть

полностью структуру дерева и особенно количество объектов, которые будут относиться к разным рубрикам классификации. Это обстоятельство не позволяет заранее произвести точное распределение всего объема памяти машины (в основном, магнитных лент) и в процессе работы часто приходится производить полную или частичную перезапись заполненных МЛ.

В настоящее время разрабатываются специальные способы гибкой адресации и записи на МЛ таких систем, исключающие необходимость перезаписи и обеспечивающие возможность эффективного использования объема МЛ. Одна из таких методик, называемая ассоциативным программированием, будет подробно рассмотрена нами в дальнейшем. Недостатком иерархических классификационных систем является то, что с их помощью трудно производить поиск объектов, относящихся одновременно ко многим классификационным подразделениям (многоаспектный поиск), а также чрезвычайная сложность и громоздкость этих систем, затрудняющая их эксплуатацию. Кроме того, иерархические системы при всей своей сложности все же не отражают полностью всего многообразия классификации объектов. Указанные недостатки присущи, в частности, и универсальной десятичной системе классификации литературы, в связи с чем сейчас применяются другие более гибкие системы, например системы, основанные на дескрипторном принципе.

Библиографический дескрипторный поиск. Дескрипторные информационно-логические системы накопления, хранения, обработки и поиска библиографической информации по способу организации информации в памяти машины можно условно разделить на два основных типа: простейшие дескрипторные системы без грамматики и дескрипторные системы с грамматикой.

Рассмотрим сущность простейшего дескрипторного метода поиска литературы. Содержание каждого литературного источника (книги, журнала, статьи, сборника и т. д.), называемого для краткости документом, может быть представлено в общих чертах при помощи набора характерных для данного текста слов. Эти слова называются ключевыми словами для данного текста. В различных текстах, посвященных одной и той же тематике, набор ключевых слов может сильно отличаться не только из-за различий в содержании текстов, но и благодаря наличию синонимов, а также возможности описывать одни и те же понятия различными формулировками. Для построения поисковой системы из всего многообразия ключевых слов, выбранных из ряда характерных для данной тематики текстов, составляется стандартный набор терминов со строго фиксированными значениями, в котором устранены синонимы. Эти термины, представляемые отдельными словами или группами слов, называются дескрипторами, а набор таких терминов — словарем дескрипторов. Например, статья об алгоритмических языках, предназначенных для программирования информационно-логических задач, может быть описана набором дескрипторов: *программирование, алгоритмический язык, информация, логическая обработка, вычислительная машина.* Для каждого из документов, образующих массив, в котором должен производиться поиск, составляется набор дескрипторов, называемый поисковым образом данного документа.

Для поиска литературы по определенному вопросу заказчик должен сформулировать свое требование в виде поискового образа запроса, который также должен представлять собой набор дескрипторов, характеризующих этот вопрос. Обычно заказчик не знает точно словарь дескрипторов и в запросах часто употребляет синонимы дескрипторов или близкие выражения. Поэтому первым этапом поиска является перевод запроса заказчика на язык дескрипторов, имеющихся в словаре. Затем производится поиск документов, целью которого является определение документов, наборы дескрипторов у которых соответствуют набору дескрипторов запроса.

Вопрос о критерии такого соответствия является достаточно сложным вопросом. В простейшем случае в качестве критерия соответствия принимается условие наличия всех дескрипторов запроса среди дескрипторов документа. Массив дескрипторных наборов документов может быть построен двумя различными способами: прямым и инверсным. При прямом способе в памяти машины последовательно записываются номера документов (или другие данные, показывающие их местонахождение) и за каждым номером документа указываются все коды дескрипторов, относящихся к нему. Процесс поиска заключается в последовательном сравнении для каждого документа всех дескрипторов запроса с дескрипторами этого документа и выделении тех документов, для которых выполняется критерий соответствия. Обычно эту простейшую схему поиска усовершенствуют путем разделения всего массива документов на некоторые разделы с тем, чтобы производить прямой дескрипторный поиск не по всем документам, а по их части, относящейся к требуемой рубрике.

При инверсном способе за основные позиции хранения информации принимают не документы, а дескрипторы. Для каждого дескриптора записываются все номера документов, которые имеют в своих

поисковых образах этот дескриптор. Поиск нужных документов, отвечающих заданному набору дескрипторов запроса, осуществляется путем обращения в словаре дескрипторов к тем дескрипторам, которые имеются в запросе, и выписывания всех номеров документов, которые относятся к каждому из этих дескрипторов. Затем производится отбор нужных документов путем последовательного сравнения номеров документов, относящихся к разным дескрипторам, указанным в запросе. Отобранными считаются те документы, номера которых оказались общими для всех дескрипторов, указанных в запросе.

На рис. 10 показан пример построения поискового массива. Если, например, требуется найти литературу по алгоритмическим языкам для программирования экономических задач, решаемых методом ПЕРТ, то запрос может быть сделан в таком виде:

0105, 0205, 0340.

Этому запросу отвечает документ с номером 0031. Как при прямом, так и при инверсном способе поиска возможны другие критерии соответствия документов запросу, а не только полное совпадение дескрипторов. Иногда дескрипторы запросов делятся на категории, указывающие их важность, что учитывается при поиске. Иногда алгоритм поиска предусматривает в случае отсутствия подходящих документов выдачу других, близких по смыслу дескрипторов, что позволяет заказчику уточнить или изменить запрос. В этом простейшем виде дескрипторного поиска дескрипторы, относящиеся к документам, и дескрипторы, входящие в состав запроса, представляют собой простые наборы, не связанные какой-либо грамматикой; в частности, порядок их расположения роли не играет.

Дескрипторы		Номера документов
Наименование	Коды	
Вычислительная машина	0101	0031, 0034, 0038, 0045, 0049, 0101
Логическая обработка	0102	0012, 0026, 0031, 0046, 0049, 0082
Алгоритмический язык	0105	0003, 0022, 0027, 0031, 0049
Экономика	0205	0031, 0168, 1342
ПЕРТ	0340	0010, 0031
Оптимизация	0520	0612, 0831, 1342

Рис. 10. Пример построения поискового массива.

Такой простейший способ, однако, приводит либо к выдаче излишних документов, не относящихся к интересующему вопросу, либо наоборот к пропуску (потере) нужных документов.

Например, по вопросу, содержащему 3 дескриптора: устройство, программа, выработка — могут быть выданы документы об устройствах для выработки программы, либо информация о программе выработки устройств, либо о выработке программы для устройств.

Более эффективными являются дескрипторные поисковые системы с грамматикой, в которых дескрипторы, относящиеся к документам, и дескрипторы, входящие в состав запросов, снабжаются дополнительными символами, указывающими на семантическую роль этих дескрипторов (субъект процесса, объект процесса, атрибут, процесс, причина и т.д.).

Иногда роль дескрипторов определяется их положением в наборе, а также некоторыми символами, указывающими связи между отдельными дескрипторами. В этом случае дескрипторные поисковые системы приближаются по своему принципу действия к поисковым системам, построенным на основе смыслового кодирования (например, система Перри — Кента (США), система информационного поиска Института кибернетики АН УССР и др.).

Принцип смыслового кодирования является основным для построения фактографических информационных систем, к рассмотрению которых мы и переходим.

Фактографические системы. Основными принципами построения фактографических систем является принцип смыслового кодирования, т. е. построение специального формализованного информационного языка, позволяющего записывать фактические сведения, относящиеся к тем или иным областям знаний.

Основой подобных информационных языков является некоторая обобщенная абстрагированная модель естественных языков, освобожденная от двусмысленностей и различных дополнительных эмоциональных средств, сильно усложняющих и обогащающих язык как средство общения людей, но не нужных для информационного накопления научных данных. Информационный язык как средство отображения закономерностей и связей внешнего мира должен отражать и его основную схему, а

именно наличие двух основных категорий: объектов и отношений между ними.

Информационные языки, построенные на основе смыслового кодирования, включают в себя обычно четыре основные части:

1) набор базисных терминов, обозначающих основные предметы данной области знаний. Например, для вычислительной техники такими терминами могут быть двоичный разряд, машинное слово, адрес, код операции и т. д.

Ясно, что выбор базисных терминов не является строго однозначным и может осуществляться в известной мере произвольно.

2) набор смысловых отношений между предметами. В отличие от базисных терминов, которые сильно связаны с конкретными областями знаний, наборы базисных отношений более универсальны и сходны для различных областей знаний (во всяком случае для многих областей науки).

Примеры отношений: один предмет является элементом класса, представляющего другой предмет; один предмет является частью предмета; предмет является субъектом процесса; предмет является объектом процесса и т. д.;

3) набор формальных правил (синтаксис), позволяющих из основных терминов и отношений языка строить более сложные термины и отношения, а также осуществлять переход между информационным и естественным языком;

4) систему кодирования понятий, отношений и синтаксических правил языка, позволяющую осуществлять преобразование и хранение информации, представленной на этом языке, с помощью цифровых вычислительных машин. Эта часть информационного языка включает в себя и алгоритмический язык для описания алгоритмов преобразования информации в машине.

По своим возможностям анализа и логической переработки информации фактографические информационные системы могут различаться в сильной степени. В качестве простейшего варианта таких систем можно представить себе систему, осуществляющую накопление информации и ее проверку на непротиворечивость. В такую систему должны вводиться некоторые утверждения и она должна выдавать один из трех ответов: «да», «нет», «неизвестно».

Ответ «да» свидетельствует о справедливости введенного для проверки утверждения, ответ «нет» показывает, что это утверждение противоречит той информации, которая имеется в памяти машины, и ответ «неизвестно» выдается в тех случаях, когда информации, имеющейся в машине, недостаточно для проверки поставленного утверждения.

Такие системы реализуют накопление и классификацию данных и проверку вводимых утверждений, используя основные аксиомы логики. В частности, подобные справочные фактографические системы могут быть использованы в химии для проверки возможности новых реакций, свойств новых веществ и т. д.

Подобную систему можно представить себе и в области законодательства, когда каждый новый закон или поправка проверяются на непротиворечивость всем ранее принятым законам.

В более сложном варианте фактографические системы должны не только проверять правильность утверждений, вводимых извне, но и выдавать фактический материал на различные вопросы (определенного круга и характера).

5. Программное моделирование ЭАСУ

Основной метод описания больших управляющих систем заключается в *алгоритмизации* процессов функционирования этих систем. В настоящее время отсутствуют математические методы, которые позволяли бы в общем виде аналитически описывать и исследовать большие управляющие системы подобно тому, например, как с помощью обыкновенных дифференциальных управлений описывается и исследуется механическое движение тел, с помощью уравнений в частных производных исследуются процессы движения газов и жидкостей, явления теплопроводности, упругости и т. д. В отдельных случаях большие управляющие системы или их части могут описываться с помощью дифференциальных уравнений и некоторых других методов (например, в теории автоматического регулирования), но более общим является метод алгоритмического описания этих систем. В связи с этим большое значение приобретает метод исследования больших управляющих систем на основе анализа их алгоритмов.

Исследование алгоритмов и программ может осуществляться либо путем их массовых реализаций на ЭВМ с различными исходными данными (программное моделирование), либо путем анализа их

структуры и выявления особенностей алгоритмов, определяющих свойства управляющей системы (разветвленность, связанность, устойчивость, степень цикличности и т. д.). Актуальной задачей является создание методов и критериев качественного анализа алгоритмов и программ, их типизация и классификация, выделение специфических по своим функциям блоков, описаний переменных и т. д. Однако в настоящее время такие методы анализа алгоритмов и программ отсутствуют; разработка этих вопросов, несомненно, будет способствовать созданию общей теории больших управляющих систем и методов их проектирования. В связи с невозможностью полного аналитического описания больших управляющих систем и отсутствием методов качественного анализа алгоритмов, основным методом проектирования и исследования больших управляющих систем является метод программного моделирования.

Метод программного моделирования предусматривает прежде всего построение математической модели исследуемой системы, т. е. совокупности уравнений, неравенств и алгоритмов, описывающих функционирование отдельных объектов и всей системы в целом. Для моделирования устанавливается критерий качества функционирования системы и определяются множества изменяемых и неизменяемых параметров системы, от которых зависит критерий качества функционирования системы.

В процессе моделирования варьируют значения изменяемых параметров и определяют их значения, обеспечивающие максимум критерия качества системы при заданных значениях изменяемых параметров.

Важной частью процесса моделирования является задание состояния внешней среды, в частности состава и моментов поступления заказов на выполнение работ, на поставку продукции и т. д. Эти условия могут задаваться как путем формирования последовательности конкретных ситуаций, так и статистическим путем (заданием вероятностных характеристик соответствующих потоков событий).

При построении программной модели большой управляющей системы выделяют три вида величин:

1) объекты системы (автономные управляющие системы, подсистемы или участки; исполнительные органы, средства связи и т. д.).

2) списки свойств объектов, имеющих значение с точки зрения целей моделирования (в списки входят упомянутые выше изменяемые и неизменяемые параметры системы);

3) списки классов объектов, определенных своими свойствами и функциональной ролью в моделируемой системе.

Построение модели начинается с выделения основных факторов, определяющих функционирование реальной системы, затем составляется алгоритм и программа для ЭВМ, проводится ее отладка и проверка и осуществляются рабочие прогоны модели на ЭВМ с варьированием изменяемых параметров модели. Различают три основных типа моделей:

1) Модели с непрерывным процессом изменения переменных. Эти модели представляются системами дифференциальных уравнений, и для их исследования широко применяются аналоговые машины. Упрощенные модели предприятий как информационных систем с обратной связью могут описываться системами дифференциальных уравнений. Такие модели в частных случаях могут представляться в виде сетей непрерывных потоков семи типов: изделий, энергии, материалов, денег, кадров, машин и информации. Потоки информации играют особую роль, так как с их помощью осуществляется управление. Выделяют два типа переменных в этих моделях: уровни и нормы (отношения), и соответственно два типа уравнений для их определения. Для каждого интервала времени определяются значения этих переменных с учетом случайных факторов, включенных в процесс моделирования.

2) Модели с фиксированным периодом изменения переменных. В моделях этого типа, применяемых в экономике для целей текущего планирования и управления, периодом изменения переменных является неделя или месяц. В моделях, связанных с перспективным планированием экономики, может использоваться в качестве интервала изменения переменных год или больший период времени.

3) Модели с дискретными событиями, наступающими в произвольные моменты времени. Подробнее о способах задания времени в моделях, реализуемых в виде программ для ЭВМ, будет сказано ниже.

Для реализации моделей второго и третьего типа необходимы электронные цифровые машины с большой памятью, так как большое число данных, описывающих модель, используется постоянно в процессе моделирования.

Возможны модели смешанного типа, в которых одни переменные изменяются непрерывно, а другие изменяются либо в фиксированные, либо в произвольные моменты времени. Для исследования таких

моделей применяются гибридные вычислительные машины, сочетающие в себе аналоговые и цифровые устройства. Важным вопросом методики моделирования является получение случайных и псевдослучайных чисел с заданными законами распределения. Эти числа могут получаться схемным путем (датчики случайных чисел), либо программным путем (вычисление псевдослучайных чисел). Типовыми примерами являются модели очередей и запасов, в частности, обслуживание заказчиков, резервирование транспортных средств, распределение заданий на механическую обработку деталей, планирование эксплуатации и ремонта оборудования, определение размеров складов. Подготовка исходных данных для моделей включает в себя: определение числовых значений параметров модели, оценку исходных данных и целей моделирования, оценку распределений случайных величин, оценку точности моделирования и ее влияния на объем расчетов.

Основным условием успешной разработки моделей является четкое определение целей моделирования и выделение факторов, определяющих поведение исследуемой системы в заданных условиях.

После выяснения целей моделирования необходимо тщательное изучение реальной системы, подлежащей моделированию. Это делается в основном путем опроса специалистов, работающих в этой системе; особое внимание обращается на выяснение «узких мест» системы и возможных путей ее улучшения. Создание модели начинается с построения ее предварительного варианта. Для этого варианта готовятся необходимые исходные данные и рассчитываются ожидаемые результаты, с помощью которых осуществляется проверка и отладка предварительного варианта модели.

Перед программированием модели на машину, целесообразно провести «проигрыш» модели вручную по ее блок-схеме с использованием упрощенных исходных данных. Это позволяет выявить многие логические и технические ошибки. В некоторых случаях переход от предварительной модели к рабочей может состоять в изменении значений параметров и границ изменения переменных. В более сложных случаях потребуется изменение логики отдельных блоков модели или всей структуры модели в целом.

Программирование модели может осуществляться либо с помощью машинных языков, либо с помощью универсальных алгоритмических языков, либо с помощью специальных языков моделирования (типа СИМУЛА, СИМСКРИПТ и др.) В последнем случае процесс программирования упрощается, но для этого необходимо хорошее знание программистом этих специализированных языков и наличие транслятора с них на имеющуюся ЭВМ.

При оценке времени, требуемого для программирования, нужно исходить не из общего ожидаемого числа команд в программе, а из ожидаемого числа команд переходов, так как это число лучше отражает логическую сложность программы, что в основном определяет время программирования. Количество переходов может быть ориентировочно определено по блок-схеме алгоритма модели.

При оценке времени работы программы необходимо исходить из общего количества событий, охватываемых моделированием. Зная содержание события модели, можно определить количество операций для его воспроизведения и на этой основе ориентировочно оценить общее число операций модели. При практическом использовании рабочей модели ее автор обычно хорошо представляет структуру модели и ее особенности, а также цели моделирования. Поэтому в процессе моделирования производится не случайное, а целенаправленное варьирование параметров модели системы и внешних условий с тем, чтобы достигнуть целей моделирования при минимальных затратах машинного времени. При моделировании целесообразно использовать некоторые дополнительные средства вычислительной техники: схемные способы автоматического наращивания модельного времени, средства для определения правил выбора решений, основанные на списковых структурах, средства для обеспечения вероятностного отбора данных (схемные и программные), средства для фиксации результатов отдельных прогонов и выдачи сообщений.

Методика временной синхронизации и общая структура моделирующих программ. В отличие от аналоговых машин, которые по своему принципу действия приспособлены для моделирования сложных управляющих систем в реальном времени, процесс работы цифровых программно-управляемых машин состоит в последовательном выполнении во времени отдельных элементарных операций и не может быть непосредственно использован для моделирования в реальном масштабе времени одновременной работы нескольких независимых объектов.

В связи с этим при программном моделировании процесс функционирования нескольких независимых объектов должен быть развернут во времени в виде последовательности элементарных действий, относящихся к различным моделируемым объектам.

Это требует специального построения программы моделирования и специального учета временных зависимостей моделируемой системы.

Принципиально возможны два способа задания временного режима или темпа работы моделируемой системы.

1) *Способ постоянного приращения времени.* При этом способе выбирается общая для всех объектов моделируемой системы единица приращения времени и общий счетчик времени. Процесс моделирования состоит в последовательном увеличении на единицу показаний счетчика времени и расчете для каждого фиксированного таким образом момента времени состояний всех объектов моделируемой системы. Очевидно, что при этом способе единица приращения времени должна быть выбрана достаточно малой, чтобы дискретность времени не вносила искажений в работу моделируемой системы. При наличии в составе системы нескольких несинхронизированных объектов с различной степенью дискретности времени работы такой способ приводит к значительному увеличению объема вычислений и в то же время не позволяет полностью избежать искажений в динамике моделируемого процесса. Этот способ удобен для моделирования синхронизированных систем или систем с одинаковой дискретностью работы. Кроме того, этот способ применяется при приближенных кинематических исследованиях управляющих систем без учета их реальных динамических характеристик. Например, такой способ применяется при моделировании структуры проектируемой цифровой вычислительной машины. Отдельные части моделирующей программы (объектные подпрограммы) моделируют работу отдельных устройств машины. При включении объектных подпрограмм для каждого устройства в специальный счетчик времени записывается заранее рассчитанное время, в течение которого это устройство будет выполнять заданную операцию. В ходе моделирования из этого времени последовательно вычитается значение постоянного временного интервала, равного шагу моделирования; равенство нулю значения, записанного в счетчике данного устройства, будет свидетельствовать о том, что данное устройство закончило свою операцию и является свободным.

2) *Способ приращения времени по событиям.* При этом способе для каждого объекта системы устанавливается свой счетчик реального времени, который, в отличие от предыдущего случая, не служит для фиксации занятости объекта, а показывает текущее время его работы.

Процесс моделирования заключается в расчетах для каждого объекта системы его состояния путем последовательного приращения времени на характерные для каждого объекта (и для каждого шага его работы) интервалы. Указанный способ обеспечивает возможность независимого изменения масштабов времени для любого из объектов и в связи с этим возможность моделирования этих объектов с высокой степенью точности без чрезмерного увеличения объема вычислений.

Таким образом, моделирование сложных управляющих систем на электронных цифровых программно-управляемых машинах заключается в представлении последовательностью команд программы процесса одновременной работы ряда независимых (несинхронизированных) объектов, образующих систему, при следующих условиях:

а) Работа каждого независимого объекта системы, представляемая дискретной последовательностью элементарных шагов, должна моделироваться отдельной программой. Эта программа помимо выполнения основных функций, характерных для данного объекта, ведет учет реального времени его работы путем последовательного прибавления к счетчику реального времени данного объекта соответствующих временных интервалов. Для каждого объекта устанавливается специальный счетчик реального времени его работы, т. е. отводится отдельная ячейка в памяти ЭВМ. В эту ячейку последовательно прибавляются приращения времени, соответствующие реальным временам выполнения объектом различных элементарных шагов.

Значения приращений времени устанавливаются заранее на основе физических характеристик работы моделируемого объекта и задаются либо в виде таблицы, либо с помощью функциональной зависимости.

Например, моделируя команды одной программно-управляемой цифровой машины на другой машине, нужно после выполнения одной команды первой машины с помощью ряда команд второй машины прибавить к счетчику реального времени первой машины фактическое время выполнения этой команды первой машиной.

б) Синхронизация работы всех частных моделирующих программ осуществляется синхронизирующей программой, которая периодически проверяет значения счетчиков реального времени всех объектов и включает в работу ту частную программу, которая имеет минимальное

значение счетчика реального времени.

Включенная частная моделирующая программа воспроизводит очередной элементарный шаг данного объекта, прибавляет соответствующее приращение реального времени к своему счетчику и передает управление обратно синхронизирующей программе; последняя снова проверяет значения всех счетчиков реального времени, выбирает из них минимальный и т. д.

Таким образом, процесс функционирования всей моделируемой системы воспроизводится в виде поочередного выполнения элементарных шагов отдельными объектами этой системы. При этом каждый раз очередной шаг делает тот объект, который имеет наименьшее значение счетчика реального времени, т. е. наиболее «отставший» во времени объект. При таком порядке работы значение счетчика реального времени объекта, получающего разрешение на работу, всегда будет меньше значений счетчиков реального времени всех остальных объектов, но оно будет больше значений этих счетчиков для предыдущих моментов, соответствующих началам последних, уже выполненных этими объектами элементарных шагов.

в) Выполнение каждого очередного шага любого объекта должно производиться для момента времени, определенного значением счетчика времени при включении в работу данной частной моделирующей программы. Таким образом, все выходные сигналы данного объекта и выдаваемая им вовне информация должны относиться к моменту окончания шага. Это значит, что каждый раз, для каждого объекта моделируется предыдущий элементарный шаг, т. е. шаг, начало которого определяется значением счетчика реального времени в момент включения в работу данной частной программы.

Действительно, так как включаемый в работу объект имеет минимальное значение счетчика реального времени, то передаваемая от него другим объектам информация не может прийти слишком рано; таким образом исключаются положения, когда какой-либо объект получит информацию раньше, чем она могла бы быть в реальных условиях выработана другими объектами.

Но, с другой стороны, включаемый в работу объект имеет значение счетчика реального времени большее, чем, значения счетчиков других объектов в моменты начала ими последних шагов. Поэтому передаваемая этим объектом информация не может прийти к другим объектам и слишком поздно; таким образом исключаются положения, когда какой-либо объект, начиная выполнять очередной шаг, не будет иметь той информации от других объектов, которая к этому моменту должна была бы прийти в реальных условиях.

Так как в процессе выполнения элементарных шагов объекты не воспринимают извне и не выдают вовне информацию, то очевидно, что при указанном принципе обмена информацией между объектами обеспечивается соблюдение правильных причинно-временных соотношений, имеющих место в реальной системе.

На рис. 11 приведена схема синхронизации процесса моделирования четырех объектов (№ 1, 2, 3, 4). Ломаная линия показывает последовательность приращений времени для указанных объектов.

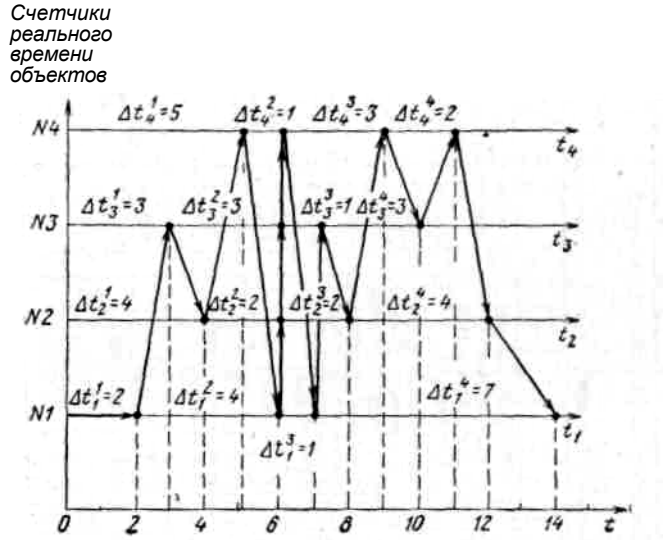


Рис. 11. Схема синхронизации процесса моделирования.

На рис. 12 приведена укрупненная блок-схема моделирующей программы для автоматизированной системы управления, состоящей из пяти типовых частей: источника внешней информации, устройства управления обменом данными между АСУ и внешними абонентами, управляющего вычислительного комплекса, операторов и ряда исполнительных органов.

Помимо синхронизирующей программы и совокупности частных моделирующих программ в состав общей моделирующей программы входит ряд вспомогательных блоков: блок регистрации результатов и выдачи их на печать, блок варьирования изменяемых параметров, выработки случайных чисел и др.

С точки зрения общей методики моделирования сложных управляющих систем в реальном масштабе времени представляет интерес вопрос об учете обратных связей от управляющей системы к источникам информации о внешней обстановке.

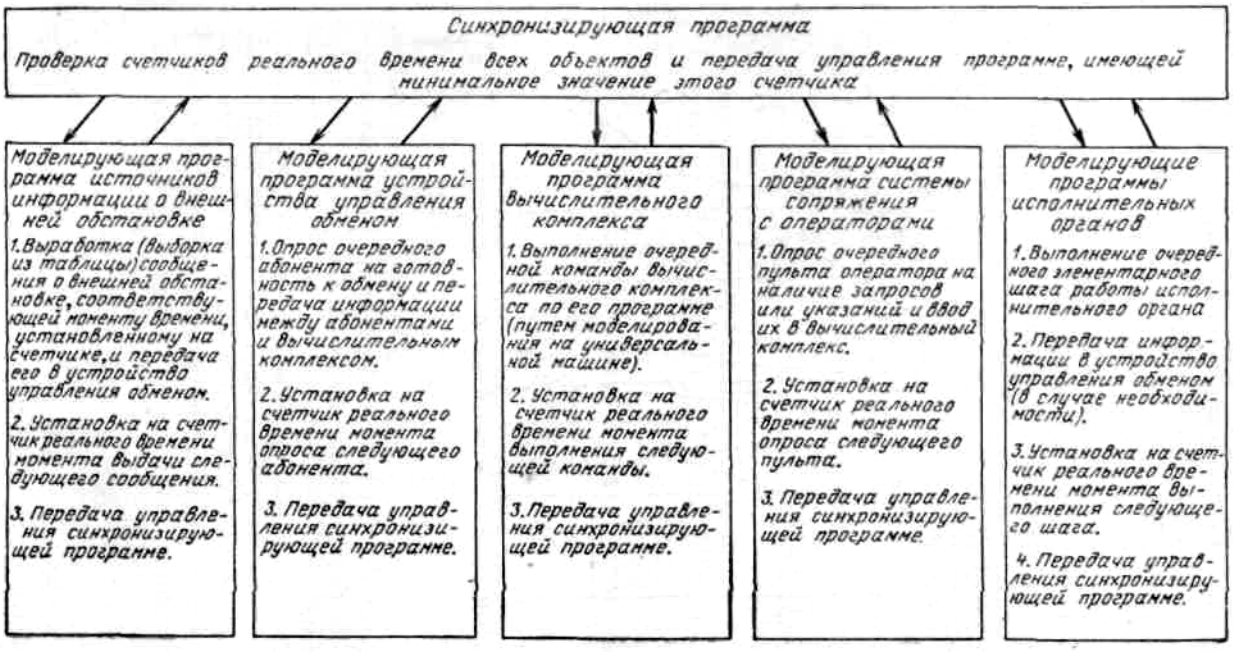


Рис. 12. Блок-схема моделирующей программы.

Обычно характер поступления информации о внешней обстановке не зависит от работы управляющей системы. Содержание, объем, темп и моменты поступления внешней информации определяются фактическим состоянием внешней среды и объективными свойствами источников информации (характеристиками устройств съема и передачи данных).

При отсутствии обратных связей моделирование процесса поступления внешней информации в управляющую систему осуществляется достаточно просто при помощи заранее составленной таблицы сообщений, содержание и моменты поступления которых рассчитываются исходя из заданного варианта внешней обстановки и заданных характеристик источников информации. Кроме указанного случая независимого поступления информации в управляющую систему возможны случаи, когда эта система управляет некоторыми источниками внешней информации и регулирует темп и характер поступления внешней информации. В этом случае процесс поступления внешней информации не может быть представлен заранее рассчитанной таблицей сообщений. Для моделирования таких процессов может быть применен способ так называемой подвижной временной шкалы, суть которого состоит в следующем.

В отличие от простой таблицы сообщений, в которой сообщения располагаются подряд вместе с временами их поступления, в данном случае заранее составляется только таблица сообщений, поступающих от неуправляемых источников информации. Времена поступления записываются в порядке возрастания отдельно, вместе с адресами соответствующих сообщений, образуя временную шкалу. В эту шкалу в процессе моделирования вносятся времена поступления новых сообщений от управляемых источников информации. Вместе со временем указывается и адрес соответствующего сообщения; само сообщение составляется уже в ходе моделирования с учетом работы моделируемой системы. Программа моделирования внешней обстановки, просматривая временную шкалу, определяет очередные сообщения и вводит их в работу в общем порядке очередности по мере возрастания шкалы времени. Таким образом, выдерживается единый порядок поступления сообщений как от неуправляемых, так и от управляемых источников информации. Временная шкала строится в виде цепного списка (см. гл. 4), обеспечивающего простоту включения

новых членов в любое место шкалы, без перемещения всех ранее включенных членов, и простоту исключения ненужных членов шкалы. Последнее может потребоваться при отмене некоторых ранее предусмотренных сообщений в ходе самого процесса моделирования.

Общий подход к моделированию больших управляющих систем. Н. П. Бусленко разработана

методика и теория моделирования сложных систем, одним из видов которых являются большие управляющие системы экономического назначения.

В основу этой методики положены два основных принципа:

1) типизация функций элементарного объекта сложной системы;

2) типизация структур сложных систем, формируемых путем объединения указанных элементарных объектов.

В качестве элементарного объекта вводится понятие агрегата, для которого дается четкое математическое описание процесса функционирования.

Агрегат — это некоторый «черный ящик», внутренняя структура которого нам неизвестна, но свойства которого, необходимые для моделирования, известны. Агрегат функционирует дискретно, воспринимая и выдавая некоторую информацию в определенные моменты времени. Состояние агрегата описывается заданным числом координат — случайных функций времени и входной информации. Для координат агрегата задаются вероятностные характеристики (параметры), которые в ходе моделирования могут изменяться под воздействием некоторого алгоритма G, реализуемого данным агрегатом. Этот алгоритм G преобразует входную информацию в выходную и может изменять вероятностные характеристики переменных, являющихся параметрами агрегата.

Сам алгоритм G может изменяться под влиянием входной информации, поступающей по особому каналу и называемой управляющей информацией агрегата.

Переход агрегата из одного состояния в другое (изменение координат агрегата) зависит от данного состояния и внешней информации, поступившей в данный момент на вход агрегата. Таким образом, агрегат характеризуется двумя алгоритмами: *внутренним*, позволяющим определять очередное состояние агрегата по его вероятностным характеристикам, внешней информации и текущему состоянию, и *внешним*, позволяющим определять выходную информацию агрегата по входной информации. Этот же внешний алгоритм изменяет и вероятностные характеристики состояний агрегата, а сам изменяется под влиянием управляющей информации, поступающей в агрегат извне.

Если теперь задать последовательность входных сообщений и управляющих воздействий для заданных моментов времени, а также исходное состояние агрегата и его вероятностные характеристики, то можно последовательно для каждого момента времени рассчитать выходные сообщения, изменения вероятностных характеристик агрегата, значения его текущих координат. Модель типового агрегата учитывает также и возможность нарушений нормального функционирования агрегата, а также возможность восстановлений нормальной работы, что очень важно для исследования вопросов надежности сложных систем.

Для этого в математическое описание агрегата добавляются функции, показывающие зависимости внутреннего и внешнего алгоритмов агрегата от специальных управляющих сообщений, содержащих информацию о нарушениях или восстановлениях в работе агрегата. Для моделирования процессов функционирования агрегатов с целью анализа их надежности необходимо задать законы распределения вероятностей нарушений (сбоев) в работе агрегата и времен восстановления (ремонта) агрегата. Моделирующая программа воспроизводит процесс функционирования агрегата по элементарным шагам, воспринимая внешние сообщения и управляющие воздействия, и выдает выходную информацию в рассчитываемые внешним алгоритмом моменты времени. В единой временной последовательности с внешним алгоритмом работает и внутренний алгоритм, обеспечивающий необходимые изменения состояний агрегата, так что эти изменения отражаются на работе внешнего алгоритма в последующие моменты времени.

Агрегатированная сложная система представляет собой совокупность взаимосвязанных типовых агрегатов, в которой выходные сообщения одних агрегатов являются входными или управляющими сообщениями для других агрегатов. В системе выделяются входные полюса, т. е. агрегаты, у которых входная или управляющая информация частично или полностью поступает из внешней среды, и выходные полюса — агрегаты, у которых выходная информация поступает во внешнюю среду. В сложных агрегатированных системах различают внутреннюю информацию, циркулирующую внутри системы, и внешнюю информацию, приходящую извне или уходящую во внешнюю среду. При анализе подобных систем принимается допущение о том, что информация между агрегатами передается мгновенно; момент выдачи информации одним агрегатом совпадает с моментом приема ее другим агрегатом. Агрегат считается управляющим по отношению к другому агрегату, если его выходная информация является управляющей информацией для другого агрегата.

В зависимости от конфигураций, в которых соединяются между собой агрегаты, входящие в

систему, различают несколько типов агрегатированных систем. В первую очередь выделяются агрегатированные системы, называемые *комплексами*; в комплексах любые два входящих в него агрегата связаны между собой либо непосредственно, либо через цепочку других агрегатов. Комплексами являются в общем случае подсистемы сложных агрегатированных систем. Затем выделяется класс *m*-фазных систем. Эти системы представляют собой последовательные цепочки из *m*-звеньев, каждое звено которых является комплексом. Класс *n-канальных* агрегатированных систем образуют системы, состоящие из *n* параллельно действующих, не связанных между собой комплексов.

Иерархическая система с одним уровнем управления образуется путем подчинения нескольких комплексов одному управляющему комплексу. Если несколько таких иерархических систем подчиняются одному управляющему комплексу, образуется иерархическая система с двумя уровнями управления. Таким путем можно получать многоуровневые иерархические системы. Среди агрегатированных сложных систем выделяется узкий класс систем, называемых *базисными* и имеющих наибольшее практическое значение. Этот класс систем характерен тем, что входящие в его состав агрегаты и перерабатываемая ими информация описываются методами теории массового обслуживания. Целью функционирования таких систем является обслуживание некоторых внешних объектов, называемых клиентами. В состав базисных систем должны входить агрегаты определенного функционального назначения: датчики информации о клиентах и самой системе, пункты сбора информации, устройства передачи и переработки информации, исполнительные органы. Методом статистического моделирования изучаются свойства подобных систем и выбираются оптимальные значения их параметров в зависимости от характеристик потоков заявок клиентов на обслуживание. Выбирается режим обслуживания, быстродействие, количество агрегатов и т, п.

6. Методика разработки математического обеспечения ЭАСУ

Разработка ЭАСУ включает в себя три основные части:

1) разработку экономико-организационных основ ЭАСУ;
2) разработку математического обеспечения ЭАСУ;
3) разработку или выбор технических средств ЭАСУ.

Начнем с комплекса технических средств ЭАСУ. Он состоит из одной или нескольких ЭВМ, средств связи и сопряжения ЭВМ с каналами связи, аппаратуры отображения информации и взаимодействия людей-операторов с ЭВМ; кроме того, важной составной частью комплекса технических средств ЭАСУ является набор датчиков первичной информации, устанавливаемых на различных участках производственного процесса.

Основными принципами выбора и разработки технических средств ЭАСУ являются: а) использование унифицированного оборудования, серийно выпускаемого промышленностью, и б) агрегатирование средств в зависимости от объема перерабатываемой информации и срочности решения задач.

Экономико-организационные основы ЭАСУ определяют состав экономических и управленческих задач и работ, подлежащих автоматизации; общую модель ЭАСУ, систему нормативов (материальных, трудовых, финансовых и др.); систему входной и выходной документации; порядок функционирования и взаимодействия ЭАСУ и людей с ней.

С возрастанием мощности и сложности вычислительных машин и систем возрастает сложность и стоимость их математического обеспечения и повышается его роль в эффективном использовании этих машин. Как уже говорилось, математическое обеспечение ЭАСУ делится на две основные части: общее (внутреннее) математическое обеспечение (ОМО) и специальное (внешнее) математическое обеспечение (СМО). Общее МО должно позволять эффективно использовать ЭВМ как некоторую универсальную систему обработки информации, поэтому оно должно представлять совокупность алгоритмических языков и программ, рассчитанных на применение ЭВМ в различных областях, для различных классов задач и типов ЭАСУ. Кроме того, эта совокупность средств служит для создания специального математического обеспечения.

Общее МО делится на три составные части:

1) Испытательные программы (тесты) для наладки и технической эксплуатации ЭВМ.
2) Операционная система (программа-диспетчер), устанавливающая порядок работы ЭВМ и ее вспомогательных (внешних) устройств, включение в работу исполнительных программ в соответствии с приоритетом задач, обмен информацией между ЭВМ и внешней средой (каналы связи, машинный архив

и т. д.), а также связь с операторами (пользователями).

3) Система автоматизации программирования задач, включающая в себя входные алгоритмические языки различных уровней и типов, трансляторы с входных языков, а также библиотеку стандартных подпрограмм.

Совокупность алгоритмических языков, входящих в состав математического обеспечения ЭАСУ, должна представлять собой семейство языков, построенных на общей синтаксической основе, единой символике и общей структуре. Эта структура должна предусматривать наличие минимального ядра, общего для всех языков семейства, и ряда языков различной сложности и назначения, в том числе специализированных языков для описания определенных классов задач (языки моделирования, обработки табличных документов, поиска информации и т. д.).

Использование *семейства языков* обеспечит совместимость различных программ, т. е. возможность автоматической компоновки программ из блоков, выдаваемых трансляторами при трансляции отдельных частей программ, записанных для разных типов задач, а также облегчит процесс подготовки кадров и процесс обмена алгоритмами и программами.

Общий порядок разработки математического обеспечения ЭАСУ может быть представлен следующим образом.

I этап. Теоретическое исследование и разработка аванпроекта ЭАСУ. Этот этап обычно называют этапом исследования операций в существующей системе. Он включает в себя следующие вопросы:

а) описание и анализ существующей системы управления и выявление ее недостатков, которые должны быть устранены с помощью автоматизации;

б) определение целей, критериев и границ ЭАСУ, намечаемой к разработке;

в) определение общей структуры, составных частей и системы внешних и внутренних связей, состава задач и объемов информации в ЭАСУ;

г) определение требований к техническим средствам ЭАСУ и предварительный выбор этих средств.

Описание и анализ действующей системы управления включает в себя рассмотрение следующих вопросов:

— определение основных функций действующей системы управления;

— разделение основных функций на конкретные вопросы и задачи;

— описание структурной схемы системы управления и схемы действующих каналов связи;

— определение существующих методов принятия решений и контроля за выполнением решений;

— описание информационных потоков, т. е. состава, количества и периодичности поступления документов;

— составление таблиц показателей, имеющихся в документах, единиц измерения и пределов изменения величин;

— составление таблиц использования различных показателей отчетной информации при решении задач управления;

— оценка затрат труда и времени на решение задач управления и обработки информации.

Особое внимание обращается на выявление случаев дублирования информации, получения ненужной информации и случаев дублирования в решении аналогичных задач разными подразделениями.

После общего изучения действующей системы управления проводится тщательный анализ тех функций и задач, решение которых оказывается неудовлетворительным. Причинами неудовлетворительных решений могут быть следующие обстоятельства:

— применение несовершенных («волевых») методов распределения ресурсов в сложных условиях;

— некачественная или неполная информация;

— большой объем информации, который невозможно обработать в допустимые сроки с помощью старых методов и средств;

— время старения информации больше допустимого;

— низкая пропускная способность каналов связи, приводящая к потере информации;

— слабая помехозащищенность каналов и технических средств, приводящая к искажению информации;

— большой объем затрат труда и времени на решение задач управления.

В результате анализа отбирается определенное количество задач, которые решаются в существующей системе управления неудовлетворительно и в первую очередь нуждаются в автоматизации. Из них выбираются задачи, которые могут быть автоматизированы с помощью

современных средств вычислительной техники. Очевидно, автоматизировать можно лишь те задачи, которые отвечают следующим требованиям:

— методы решения задачи либо уже известны, либо имеются все предпосылки для их разработки в ближайшее время;

— для выбранных методов решения может быть построен алгоритм машинного решения;

— для выбранных методов решения имеется либо может быть получена необходимая информация;

— объем этой информации не превышает пределов, установленных возможностями имеющейся или намечаемой к приобретению вычислительной техники;

— время решения задачи с помощью средств вычислительной техники укладывается в требуемые сроки.

При этом надо иметь в виду, что применение ЭВМ может быть целесообразным в следующих случаях:

— при выполнении небольших расчетов над большими последовательностями данных;

— при выполнении длинных расчетов над сравнительно небольшими объемами информации;

— при выполнении расчетов, для которых исходная информация уже имеется в памяти ЭВМ или же на машинных носителях;

— при выполнении сложных расчетов с большим числом итераций.

Одновременно с отбором первоочередных задач, которые должны быть автоматизированы, производится разработка общей схемы будущей ЭАСУ. При этом важно, чтобы отобранные для первоочередной разработки задачи вписывались в общую схему ЭАСУ и составляли первый этап ее внедрения. Для составления общей схемы ЭАСУ всю систему управления нужно разделить на структурные разделы и подсистемы. Основным критерием выделения разделов и подсистем служит целенаправленность, независимость и законченность группы решаемых задач по получению и выдаче конечных результатов и использованию входных данных. На основе анализа составляется предварительная общая модель всей автоматизированной системы первой очереди управления и укрупненная структурная схема с делением на разделы и подсистемы и указанием связей между ними. После этого ориентировочно определяется состав требуемых технических средств.

II этап. Разработка и экспериментальное внедрение отдельных задач и подсистем:

а) алгоритмизация и программирование выбранных первоочередных задач; выявление типовых информационных процессов в. разных задачах и подсистемах (например, унификация процессов обработки отчетности, процессов выдачи таблиц, процессов сортировки и др.);

б) отладка и экспериментально-производственный счет и отработка полной технологической схемы прохождения
отдельных задач в подсистемах;

в) оформление документации на отдельные задачи, передача ее в межотраслевой фонд алгоритмов и программ и производственное внедрение задач с подготовкой фактических массивов справочной, нормативной и другой информации;

г) стыковка задач в пределах функциональных подсистем, унификация входных и выходных документов и исключение промежуточных выдач информации из ЭВМ (например, использование данных обработки отчетности по форме СН1 для решения задач перераспределения фондов и уточнения потребностей).

На этом этапе происходит дальнейшая детализация постановки задач:

— уточняется вопрос применения определенного типа ЭВМ или другой вычислительной техники;

— определяются способы дистанционного получения исходных данных и выдачи результатов, в том числе решаются вопросы использования электрических каналов связи, пересылки документов или машинных носителей (перфокарт, перфолент, магнитных лент). Выбирается оптимальный вариант передачи данных;

— определяются способы кодирования информации для автономного ввода исходных данных в ЭВМ (с помощью перфокарт, перфолент, автоматического считывания);

— определяются способы и формы выдачи информации из ЭВМ (на цифровую или алфавитно-цифровую широкую печать, на перфокарты или перфоленты, на магнитные ленты);

— решаются вопросы объединения алгоритмов задач в общий алгоритм подсистемы и определяется порядок решения задач и обмена данными (т. е. вопросы взаимосвязи алгоритмов, создания алгоритмов организации счета, программ взаимодействия с каналами связи и вывода и др.).

Важнейшим вопросом, который решается на этом этапе, является вопрос контроля правильности

получения и ввода в ЭВМ исходных данных. Для этого применяется ряд способов: повторная перфорация с контролем, двойной ввод в ЭВМ, ввод в ЭВМ и выдача на печать для визуального контроля (сплошного или выборочного), арифметический или логический машинный контроль на основе избыточной информации, введенной вместе с исходными данными. Следует подчеркнуть, что контроль исходных данных требует больших затрат машинного времени, труда перфораторщиков и программистов.

Программирование задач ЭАСУ должно проводиться с использованием, как правило, средств автоматизации программирования, что особенно важно на начальном этапе, когда идет обработка алгоритмов и возможны частые переделки программ.

При этом не столь существенны характерные для получаемых автоматическим способом программ недостатки (их большая длина и большой расход машинного времени на выполнение), сколько важна возможность быстро вносить изменения и проводить экспериментальные расчеты.

Одним из основных вопросов программирования задач ЭАСУ является организация памяти ЭВМ, включающая в себя разделение всех входных, промежуточных и выходных величин на массивы и составление их описаний, а также описаний отдельных величин, определение общего количества и объема программ, распределение всей информации и программ в памяти ЭВМ.

В случае ручного программирования решаются также следующие вопросы:

— определение адресов и размеров участков памяти ЭВМ для исполнительных программ и стандартных подпрограмм;

— определение адресов рабочих ячеек, констант и отдельных переменных:

— определение адресов и размеров участков памяти (на МЛ) для размещения массивов информации.

Процесс отладки программ и испытания системы математического обеспечения можно разделить на следующие стадии:

— частная отладка автономных участков программ и стандартных подпрограмм;

— отладка и стыковка программ по отдельным задачам;

— комплексная стыковка программ задач по подсистемам;

— накопление в ЭВМ исходной информации;

— стыковка программ решения задач с программами ввода и вывода данных;

— комплексная стыковка полной программы данной подсистемы с программами обмена информацией и служебными программами.

Начиная с момента стыковки программ задач, проводится проверка алгоритмов и практически доводятся до работоспособного состояния алгоритмы и методы решения задач.

III этап. Уточнение состава задач, разделов и подсистем и их объединение в единую систему первой очереди (разработка эскизного проекта ЭАСУ):

а) построение общей функциональной модели ЭАСУ, объединяющей в единую систему основные подсистемы ЭАСУ;

б) построение информационной стыковочной матрицы для всех задач ЭАСУ, определяющей информационные связи между конкретными задачами.

Эта матрица показывает: состав задач каждой подсистемы и раздела; какую информацию вырабатывает данная подсистема (раздел) и в какую подсистему (для какой задачи) она передает эту информацию; какую исходную информацию и откуда получает данная подсистема (раздел) для решения задачи. Описываемая матрица является квадратной; в ней каждой подсистеме (разделу) соответствует одна колонка и одна строка. В клетке, находящейся на пересечении данной колонки и соответствующей строки, перечисляются задачи данной подсистемы (раздела). В остальных клетках данной строки указывается информация, получаемая данной подсистемой от тех подсистем, которым соответствуют колонки. При этом после каждого вида данных указываются в скобках номера задач этой подсистемы, в которых используются эти данные. При таком заполнении строк стыковочной матрицы ее колонки, естественно, будут указывать для каждой подсистемы в каждом разделе все данные, которые она выдает другим подсистемам (расположенным по строкам), а также номера задач этих подсистем, в которых используются выдаваемые данные. Эта матрица позволяет установить необходимую и возможную последовательность решения задач в различных разделах и подсистемах с тем, чтобы для каждой задачи в момент ее решения были готовы все исходные данные;

в) построение временных диаграмм поступления вход ной информации и загрузки устройств подготовки информации, процессов решения задач и выдачи выходных данных и определение состава технического комплекса ЭАСУ. Разработка общей технологической схемы работы ВЦ ЭАСУ и

аппарата управления;

г) уточнение алгоритмов и программ, организующих совместную работу программ различных задач и подсистем и обеспечивающих формирование и использование общих массивов информации. Построение специальных программ – диспетчеров и информационно-поисковых систем, обеспечивающих взаимодействие различных частей ЭАСУ.

В процессе эскизного проектирования (и последующего рабочего проектирования) неизбежны переделки алгоритмов и программ отдельных задач, разделов и подсистем, разрабатывавшихся ранее автономно. Это объективно необходимое свойство метода последовательных приближений, который является характерным для разработки и внедрения больших экономических автоматизированных систем управления.

IV этап. Моделирование и техническое проектирование ЭАСУ. Моделирование ЭАСУ производится с целью уточнения и оптимизации структуры системы и ее основных параметров, обеспечивающих заданные эксплуатационные и технические характеристики с учетом стоимости, надежности, сроков создания и внедрения, экономической эффективности и окупаемости.

Моделирование производится сначала по частям (по разделам и подсистемам), а затем в комплексе (ЭАСУ в целом). При этом уточняются необходимые характеристики технических средств ЭАСУ и состава математического обеспечения ЭАСУ.

В связи со сложностью подготовки экономических моделей, и особенно исходной информации, для их функционирования необходимо сочетать моделирование на ЭВМ для целей проектирования систем с экспериментальным и практическим решением задач и последовательным поэтапным внедрением частей системы. Таким образом, создание автоматизированных экономических систем представляет непрерывный процесс разработки, внедрения и практического использования частей системы с постоянным их расширением, объединением и совершенствованием. При этом важно четко выделять автономные этапы внедрения с тем, чтобы обеспечить практическую эксплуатацию внедренных подсистем и задач без изменений достаточно длительное время.

Обязательным условием успешной разработки ЭАСУ является систематическое изучение вопросов совместимости частей (подсистем) большой управляющей системы с целью возможно раннего выявления случаев несовместимости.

Для оптимизации хода разработки и внедрения больших управляющих систем целесообразно применение методов СПУ.

V этап. Рабочее проектирование ЭАСУ.

Основными задачами рабочего проектирования ЭАСУ являются:

а) уточнение состава и содержания задач в подсистемах и алгоритмов их решения. Оптимизация программ с точки зрения времени счета, их объемов и удобства эксплуатации (упрощение подготовки данных, введение служебных подпрограмм, согласование с другими программами и т. п.). Некоторые задачи (особенно типовые и часто повторяемые) могут на этом этапе перепрограммироваться с целью их оптимизации;

б) подготовка полных рабочих массивов используемой информации на машинных носителях (нормативы, характеристики изделий и заводов, классификаторы и т. д.);

в) отработка официальных документов (инструкций, приказов и т. д.), определяющих порядок взаимодействия ЭАСУ с аппаратом административного органа (министерства, заводоуправления и т. д.), в том числе ответственность за исходную информацию и результаты решения задач, а также порядок принятия решений в процессе совместного функционирования ЭАСУ и административного органа.

Отличие технического проектирования от рабочего заключается в том, что при техническом проектировании осуществляется в основном отработка и оптимизация алгоритмов на базе моделирования всей системы и ее подсистем а при рабочем проектировании в основном идет оптимизация программ при некоторой доработке алгоритмов, а также производится проектирование рабочей технологии процессов решения задач, подготовка рабочих машинных массивов и оформление технической документации.

Важным условием успеха в создании больших управляющих систем является систематическое и тщательное ведение документации при разработке, отладке, испытаниях и внедрении этих систем.

К числу основных требований системной документации относятся:

— унификация форм входной и выходной документации;

— блочный иерархический принцип построения и классификации алгоритмов и программ и

стандартизация их описаний;

— стандартизация форм и иерархический принцип представления и учета структурных и функциональных схем подсистем и всей системы;

— наличие строгой формализованной методики согласования подсистем по алгоритмам и информации;

— унификация информационных массивов — нормативов, классификаторов, планов, отчетов, справочных данных по продукции, материалам, оборудованию, кадрам — и наличие строго формализованной методики по их ведению и проверке. Особенно важно систематическое ведение классификаторов и нормативов;

— наличие полной и тщательно разработанной системы тестов, позволяющих контролировать неисправности или ошибки как в системе в целом, так и в отдельных ее подсистемах;

— наличие строго формализованной методики и утвержденного порядка внесения исправлений и дополнений во все виды документации большой управляющей системы. При этом следует иметь в виду, что разработка вопросов документации для большой управляющей системы и практическое ведение и оформление этой документации представляет собой весьма сложную и трудоемкую работу.

Глава 2

АЛГОРИТМИЧЕСКИЙ ЯЗЫК АЛГОЛ

Для решения любой задачи на электронной машине должна быть составлена программа, представляющая собой (в форме, воспринимаемой машиной) алгоритм решения задачи.

В первые годы развития электронных вычислительных цифровых машин применялось ручное непосредственное программирование задач, т. е. перевод алгоритмов на машинный язык осуществлялся непосредственно людьми-программистами. Программисты должны были хорошо знать систему команд машины и алгоритм решения задачи и полностью выписывать подряд все команды программы. Такая работа в случае сложных и громоздких программ чрезвычайно трудоемка, сопровождается значительным количеством ошибок, и в связи с этим составляемые вручную программы требуют, как правило, длительной доработки и проверки (отладки) на машине.

Первым этапом автоматизации процесса программирования явился переход к так называемому символическому программированию, когда адреса команд и коды операций пишутся программистом не в виде конкретных числовых значений, а в символической или буквенной форме. При этом адреса обозначаются, как правило, теми же буквами, что и величины, которые должны храниться в соответствующих ячейках, а коды операций — обычными математическими символами, например, +, —, ×, : и т. п.

Переход от символических адресов и кодов операций

конкретным машинным кодам осуществляется самой машиной, которая снабжается для этого специальной переводящей программой. Хотя при символическом программировании полностью сохраняется соответствие между символическими и машинными командами, процесс программирования существенно облегчается и сокращается время на отладку программ.

Следующим этапом автоматизации программирования явилось введение *метода автокодов*, при котором для некоторой конкретной машины составлялся набор так называемых макрокоманд, каждая из которых соответствует определенной совокупности машинных команд (машинным подпрограмм). В макрокомандах использовались символические обозначения для кодов этих команд и для адресов операндов. Например, для некоторых одноадресных машин автокодами служили системы команд трехадресных машин. Переход к укрупненным командам (макрокомандам) в сочетании с символическим заданием адресов обеспечил дальнейшее сокращение объема программ, составляемых вручную. Как символическое программирование, так и метод автокодов привязан к конкретным машинам и в этом отношении не обеспечивает необходимой гибкости и универсальности составляемых программ.

Дальнейшим этапом явилось применение *алгоритмических языков,* ориентированных не на определенные типы машин, а на определенные классы задач. Так, широкое применение в международном масштабе для программирования научно-технических вычислений имеет язык АЛГОЛ-60; в США и Западной Европе широко распространен язык ФОРТРАН (II и IV), применяемый для научно-технических задач; для программирования экономических задач в США и Западной Европе применяется КОБОЛ; для описания алгоритмов машинного перевода — язык КОМИТ; для задач поиска информации — язык РЕКОЛ.

Широкое применение в настоящее время в США, Западной Европе и Японии получил универсальный алгоритмический язык PL-1, в котором собраны различные средства, необходимые для описания задач различных классов, в том числе для моделирования процессов в реальном масштабе времени.

Язык PL-1 построен на основе языков ФОРТРАН, КОБОЛ, и ряда других и представляет собой по существу объединение и дальнейшее развитие этих языков на базе ФОРТРАНА. Практически используются, как правило, различные подмножества языка PL-1, ориентированные на определенные области применения.

Особую группу составляют языки для неарифметической обработки информации: ИПЛ, ЛИСП. К числу алгоритмических языков следует отнести также адресный язык, в котором учитываются машинные особенности решения задач (принцип адресности).

Мы будем рассматривать алгоритмические языки для описания экономических и управленческих задач. Следует заметить, что, как правило, указанные задачи включают в себя и обычные

вычислительные процессы (расчеты по формулам), поэтому язык для описания таких процессов должен позволять описывать обычные вычислительные задачи.

В настоящее время определилась тенденция к созданию общего алгоритмического языка, который бы представлял собой единое семейство языков, приспособленных к различным классам задач. Это семейство языков должно иметь общую основную часть (ядро), служащую для описания вычислительных процессов, встречающихся во всех классах задач, и общий синтаксис, обеспечивающий единообразие построения семейства языков.

Единый алгоритмический язык (точнее, семейство языков) позволяет упростить обучение практических работников различных областей пользованию этими языками в необходимом для них объеме, облегчает переходы от одного языка к другому, позволяет стандартизировать построение трансляторов на машинные языки и между алгоритмическими языками, стандартизировать символику, применяемую в устройствах ввода и вывода данных в машинах. Особенно удобно применение такого языка при программировании сложных процессов переработки информации, включающих в себя задачи различных классов. Примером такого семейства языков может служить упоминавшийся выше язык PL-1.

Мы в качестве основы для построения алгоритмического языка для программирования экономических задач примем алгоритмический язык АЛГОЛ. Этот язык дополняется средствами, необходимыми для описания процессов обработки экономической информации в двух вариантах. Первый вариант создан в период 1964—1965 гг., он называется АЛГЭМ-1 и является входным языком транслятора для машины «Минск-22», получившего уже широкое практическое применение. Второй вариант АЛГЭМ-2 создан в 1968 г.; он представляет собой более широкий и гибкий язык, в котором использованы многие средства других известных языков (PL-1, АЛГЭК).

Для удобства изучения и использования описываемого здесь алгоритмического языка принят метод последовательного изложения отдельных его составных частей. Сначала описывается АЛГОЛ, затем излагаются дополнения и изменения к АЛГОЛу, образующие вместе с ним язык АЛГЭМ-1, после того описываются дополнения и изменения, образующие вместе с АЛГОЛом язык АЛГЭМ-2.

Язык АЛГЭМ-2 можно рассматривать как некоторое достаточно полное подмножество языка PL-1, построенное в отличие от него не на базе ФОРТРАНА, а на базе АЛГОЛа.

Для условий Советского Союза, в котором АЛГОЛ получил весьма широкое применение, а ФОРТРАН почти не используется, более перспективным с точки зрения освоения и внедрения будет язык, построенный на базе АЛГОЛа.

7. Общие сведения об АЛГОЛе

Название АЛГОЛ (ALGOL) происходит от сокращения двух английских слов Algoritmic Language, обозначающих алгоритмический язык. Число 60 указывает год принятия этого варианта языка. Основным назначением АЛГОЛа является описание алгоритмов выполнения математических и научно-технических вычислений с помощью автоматических программно-управляемых машин. Он используется в качестве международного алгоритмического языка для обмена алгоритмами в международном масштабе. АЛГОЛ служит также эталонным языком для построения конкретных алгоритмических языков (входных языков).

Конкретные входные языки типа АЛГОЛ сохраняют все основные свойства эталонного АЛГОЛа и отличаются от эталонного составом используемых символов, что связано с различиями в конструкций клавиатур клавишных устройств, служащих для ввода данных в разные машины. В основу АЛГОЛа положен общепринятый язык математических формул, дополненный средствами, необходимыми для формального описания процессов автоматического выполнения алгоритмов. Программа, записанная на АЛГОЛе, перед выполнением должна быть переведена с помощью специальной (достаточно сложной) программы, называемой транслятором, в рабочую машинную программу для данной конкретной машины. Отличие АЛГОЛа от обычного языка математических формул заключается в линейности записи (в одну строку) всех символов (в том числе индексов, показателей степеней), а также в более полной формализации записи действия (явное указание операций умножения, четкое определение типов величин при вычислениям и т. п.), в строгом разграничении способов использования всех символов (скобок, запятых, точек с запятой и т.д.).

Кроме эталонного язык АЛГОЛ используется еще как язык публикаций. В этом варианте языка допустимы отклонения от эталонного языка, связанные с удобством печатания и чтения людьми

алгоритмов, записанных на АЛГОЛе (в частности, индексы и показатели степеней пишутся так, как обычно принято их писать в формулах).

В основу построения АЛГОЛа положен блочный принцип. Любая программа на АЛГОЛе представляет собой блок и сама может состоять из блоков, внутри которых могут быть другие блоки и т. д.

Блок в АЛГОЛе — это автономный участок программы, включающий в себя информацию двух видов: описания данных и других объектов, участвующих в вычислениях, и указания о непосредственно выполняемых *0* программе действиях. Эти указания принято называть операторами. Оператор охватывает законченную последовательность действий, например вычисление по формуле, переход к другому варианту вычислений и др. Описания и операторы можно рассматривать как отдельные предложения языка. После каждого такого предложения ставится точка с запятой.

Блоки АЛГОЛ-программы строятся по общему плану. В начале блока идут все описания, затем пишутся подряд все операторы в порядке их выполнения. Описания имеют силу только внутри данного блока. Блок начинается словесной скобкой **начало** и заканчивается словесной скобкой **конец.** Эти же скобки используются и для образования так называемых составных операторов, представляющих собой группы операторов, разделяемых точками с запятой. В составном операторе в отличие от блока отсутствуют описания. Заметим, что внутри любого блока или составного оператора в качестве операторов могут фигурировать снова составные операторы и блоки, так как они являются разновидностями операторов. Блочный принцип построения программ удобен при программировании сложных задач, так как при этом всю задачу можно разделить на ряд автономных участков (блоков), которые будут разрабатываться одновременно несколькими программистами. Каждый блок получает некоторые входные данные (от других блоков или в процессе ввода информации) и выдает выходные данные другим блокам или на выводные устройства машин (печать, перфорацию). Помимо входных и выходных данных в блоке используются и свои местные (локальные)

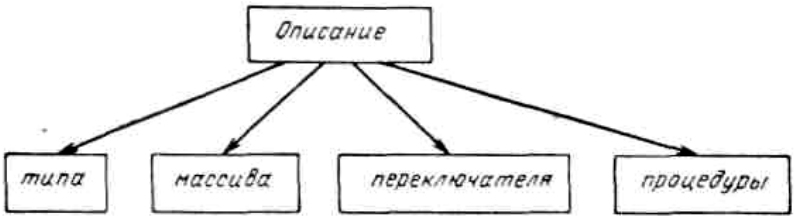


Рис. 13. Виды описаний в АЛГОЛе.

величины, которые не используются вне данного блока. Эти величины должны быть описаны в данном блоке с помощью описаний.

За символом **конец** ставится, как правило, точка с запятой, либо другой символ **конец,** либо символ **иначе** (это будет пояснено ниже). Кроме того, за символом **конец** можно писать добавочный пояснительный текст, который не влияет на работу программ. Конец пояснительного текста указывается либо точкой с запятой, либо новым символом **конец,** либо символом **иначе,** а сам этот текст не должен содержать в себе указанных трех символов.

Блок обладает свойством локализации переходов внутри блока. Обращение к блоку возможно только в его начало; все операторы, используемые внутри блока, доступны только для операторов перехода, находящихся внутри блока. Переходы извне могут совершаться только в начало блока.

Перечислим все виды описаний и операторов, используемых в АЛГОЛе (приводимые названия будут пояснены ниже). Описания делятся на четыре вида в соответствии со схемой, показанной на рис. 13.

Классификация операторов в АЛГОЛе представляется схемой, показанной на рис. 14. Существует всего восемь видов основных операторов, причем некоторые из этих видов объединяются в более крупные группировки (основной оператор, безусловный оператор).

Для осуществления переходов между операторами некоторые из них снабжаются метками, которые ставятся перед операторами и отделяются от них двоеточиями. Перед одним оператором может стоять несколько меток, разделенных двоеточиями.

В АЛГОЛ-программе может стоять просто метка с пустым оператором. Понятие пустого оператора введено специально для возможности помещения одной метки, что бывает необходимо

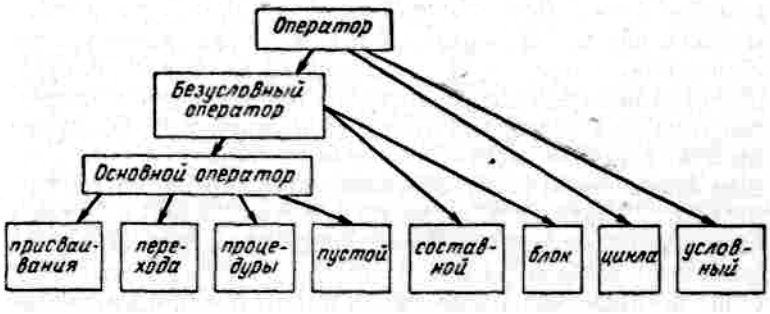


Рис. 14. Виды операторов в АЛГОЛе.

для отсылок к определенному месту программы, например, для выхода в конец цикла (см. ниже).

Метаязык для описания алгоритмических языков. В алгоритмических языках, как и вообще в языках, в основу построения их синтаксиса положен принцип иерархического определения более сложных понятий через более простые. Однако некоторые из сложных понятий языка не могут быть определены путем перечисления их компонентов. Например, точное формальное определение числа невозможно дать перечислением всех возможных значений чисел. В подобных случаях используются так называемые рекурсивные определения, в которых определяемое понятие само участвует в своем определении.

В качестве примера можно привести определение понятия блока. Можно считать, что блок состоит из двух половин, разделенных точкой с запятой. Первая половина называется «начало блока», а вторая — «конец составного» (имеется в виду конец составного оператора).

Начало блока может содержать за символом **начало** одно или несколько описаний, разделенных точками с запятой, а конец составного может содержать перед символом **конец** несколько операторов, разделенных точками с запятой, в числе этих операторов могут быть блоки, являющиеся одним из видов операторов.

Таким образом, в определение понятия входит само понятие блока. Для представления подобных определений и вообще для описания различных конструкций языка удобна символика, известная под названием метаязыка Бэкуса.

В метаязыке используются металингвистические формулы, имеющие подобно обычным математическим формулам правую и левую части. Эти части соединяются специальным символом «: : = », обозначающим «равно по определению». Левая часть является определяемой, а правая часть — определяющей. В металингвистических формулах участвуют основные символы языка, играющие роль констант и представляющие самих себя, и так называемые металингвистические переменные, представляющие более сложные понятия языка (операторы, описания, блоки и т. д.). Металингвистические переменные при подстановке в металингвистические формулы заключаются в специальные угловые скобки (⟨ — открывающая и ⟩ — закрывающая). В левых частях металингвистических формул ставятся только определяемые металингвистические переменные, заключенные в угловые скобки.

В правых частях формул пишутся определяющие металингвистические выражения, которые строятся с использованием двух операций метаязыка:

1) операции перечисления;

2) операции построения определяющего выражения по составлению.

Операция определения путем перечисления имеет вид

$$\langle x\rangle : : = A \mid B \mid C \mid D$$

где x — определяемое понятие, являющееся металингвистической переменной; угловые скобки ⟨ ⟩ обозначают некоторое конкретное значение величины, заключенной в скобки;

A, B, C, D — некоторые элементы, являющиеся конкретными значениями определяемого понятия;

вертикальная черта — символ операции перечисления (операция ИЛИ).

В данном случае конкретными значениями некоторого понятия x будут: либо величина A, либо B, либо C, либо D.

Операция определения путем составления имеет вид

$$\langle x\rangle : : = ABC$$

где ABC есть конкретное выражение, образованное путем составления (последовательного написания

рядом) трех выражений A, B и C и являющееся частным значением определяемого понятия x.

Возможно комбинированное использование указанных двух операций определения. Например,

⟨x⟩: : =AB | BC | CD

При рекурсивном способе определения понятий определяемое понятие, указанное в левой части, снова выступает в правой части в качестве одного из элементов определяющего выражения. Например, целое число без знака может определяться так:

⟨целое число без знака ⟩ : : = ⟨цифра ⟩ | ⟨целое число без знака⟩ ⟨цифра⟩

Основные символы АЛГОЛа. Основные символы — это наименьшие неделимые элементы языка, используемые для построения всех остальных конструкций языка. Пользуясь метаязыком, приведем следующее определение понятия основного символа АЛГОЛа:

⟨основной символ⟩ : : = ⟨буква⟩ | ⟨цифра⟩ | ⟨логическое значение⟩ | ⟨ограничитель⟩

⟨буква⟩ : : = ⟨латинская малая буква⟩ ⟨ только латинская большая буква ⟩ |Ж | Ф | Щ | Э | Ы | Ю | Я | Ш | Н | Ч | И

В эталонном АЛГОЛе применяются только латинские малые и большие буквы. В конкретных вариантах АЛГОЛа допускается изменение алфавита. Для удобства обозначений мы вводим в состав алфавита АЛГОЛа еще буквы русского алфавита, несовпадающие с латинскими буквами. Состав цифр АЛГОЛа строго фиксирован; другие виды цифр не применяются:

⟨цифра⟩ ::=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Цифры используются для представления чисел; буквы и цифры используются для представления различного рода наименований.

Логические значения будем обозначать с помощью словесных основных символов: **истинно** и **ложно.** Иногда для обозначения истинного значения применяют единицу (1), а ложного значения — нуль (0).

В эталонном АЛГОЛе для этого используются английские слова **(true** — истинно, **false** — ложно):

⟨логическое значение⟩ : : = **истинно | ложно**

Логические значения являются логическими константами; они применяются для проверки логических условий, а также для образования логических выражений, т. е. выражений, которые могут иметь только два значения (истинно или ложно).

Ограничители — это служебные символы, входящие в состав языка и предназначенные для построения более сложных конструкций языка:

⟨ограничитель⟩ : : = ⟨знак операции⟩ | ⟨разделитель⟩ | ⟨скобка⟩ | ⟨описатель⟩ | ⟨спецификатор⟩

⟨знак операции⟩ : : = ⟨знак арифметической операции⟩ | ⟨знак операции отношения⟩ | ⟨знак логической операции⟩ | ⟨знак операции следования⟩

⟨знак арифметической операции⟩::= + | — | × | / | ÷ | ↑

Символ «÷» обозначает операцию целочисленного деления; результатом ее является наибольшее целое число, не превышающее фактического значения результата. Символ ↑ обозначает операцию возведения в степень.

Заметим, что в АЛГОЛе операция умножения должна указываться явно. Например, a × b нельзя записать как ab; последняя запись будет воспринята как одно обозначение некоторой величины. Знаки операций отношения и логических операций имеют общепринятый смысл:

⟨знак операции отношения ⟩::= < | ≤ | = | ≥ | > | ≠ |

⟨знак логической операции ⟩::= ∨ | ∧ | ¬ | ⊃ |≡

Перечислим названия логических операций:

∨ — «или» (логическое сложение, дизъюнкция);

∧ — «и» (логическое умножение, конъюнкция);

¬ — «не» (отрицание), заметим, что в АЛГОЛе операция отрицания пишется не в виде черты над величиной, а в виде уголка, помещаемого перед величиной;

⊃ — «влечет» (импликация);

≡ — «эквивалентно» (равнозначность).

Логические операции служат для образования логических выражений, используемых для выбора направления дальнейших вычислений.

⟨знак операции следования⟩ : : = **на** | **если | то | иначе | для | цикл**

К символам операций следования относятся такие символы, которые изменяют порядок выполнения операторов. Символ **на** используется для построения безусловного перехода; символы **если, то, иначе** используются при построении условных переходов и символы **для** и **цикл** — при построении циклов. Следует отметить, что словесные основные символы не имеют никакого отношения к буквам, из которых они составлены; они являются такими же основными, как и символы. Вместо них можно было бы ввести какие-нибудь стрелки или другие значки, но выбраны слова, чтобы их было легче запоминать. Словесные основные символы при написании принято подчеркивать; при печатании они выделяются жирным шрифтом:

⟨разделитель ⟩:: =, | · |$_{10}$ | : | ; | : = | ␣ | **шаг** | **до** | **пока** | **примечание**

Перечисленные разделители имеют следующие назначения[1] : запятая служит для разделения элементов списков, например списка индексов [i, j, k] или списка параметров процедуры (a, b, c); точка служит для отделения целой части числа от дробной, точнее для обозначения дробной части числа (десятичной); число 10, написанное с понижением, используется как символ десятичного порядка числа; следующее за этим символом число является порядком; двоеточие служит для разделения граничных пар в описаниях массивов, а также для отделения меток от операторов; точка с запятой служит для разделения отдельных участков программы (операторов и описаний); двоеточие и равенство является символом присваивания и используется в операторах присваивания, в операторах циклов и в описаниях переключателей; пробел ␣ применяется при построении строк; слова **шаг, до, пока** используются при построении операторов цикла; словесный символ **примечание** ставится после символа **начало** или точки с запятой и позволяет включать после него любой пояснительный текст, не содержащий точек с запятой, который транслятором не воспринимается, а служит только для удобства понимания программы людьми. Конец пояснительного текста указывается точкой с запятой. Пояснения, не содержащие символа **конец** и точек с запятой можно включать и после символа **конец.** Они также не воспринимаются транслятором.

В АЛГОЛе используется четыре вида скобок:

⟨скобка⟩ : : = (|) | [|] | **начало** | **конец** |‘ |’

Круглые скобки используются при построении выражений для определения в них порядка выполнения действий и при построении так называемых процедур. Квадратные скобки используются для заключения индексов величин, граничных пар в описании массивов величин, индексов в так называемых указателях переключателей (см. ниже). Словесные скобки **начало** и **конец** служат для объединения операторов в составные операторы и блоки. Кавычки используются для образования строк, обозначающих нечисловые величины (например, собственные имена, качественные характеристики и др.).

Описатели служат для построения описаний идентификаторов, используемых в АЛГОЛ-программах. Описания, как уже упоминалось, ставятся в начале каждого блока; состав описаний в разных блоках может быть различным.

⟨описатель⟩ : : = **целый** | **вещественный | логический | массив | собственный** | **переключатель** | **процедура**

Описатели **целый, вещественный** и **логический** служат для описания типов переменных и массивов.

Описатель **массив** служит для описания массивов, т. е. групп величин (например, матриц), а описатели **переключатель** и **процедура** служат для описания типовых алгоритмических процессов, играющих роль подпрограмм при обычном машинном программировании:

⟨спецификатор⟩ : : = **метка | значение | строка**

Спецификаторы — это такие описатели, которые используются только в одном разделе языка, а именно в разделе описаний процедур. Они служат для характеристики величин, используемых в процедурах. В качестве спецификаторов в этом разделе могут использоваться и другие перечисленные выше описатели, но роль их несколько меняется. Описатели, используемые в качестве спецификаторов, менее полно определяют соответствующие величины по сравнению с тем случаем, когда они используются в описаниях в блоках. Например, в спецификации **массив** M не указывается размерность массива M. Подробно эти вопросы будут рассмотрены в дальнейшем при описании процедур.

Необходимые дополнительные пояснения всех основных символов языка будут даваться по мере их

[1] В дальнейшем назначение каждого символа будет рассмотрено более подробно.

появления в процессе изложения.

Числа. В АЛГОЛе используются только десятичные числа. Простейшим числом является целое число без знака, представляющее собой последовательность цифр:

⟨целое без знака ⟩ : : = ⟨цифра ⟩ | ⟨целое без знака ⟩ ⟨(цифра ⟩

⟨целое ⟩: : = ⟨целое без знака ⟩ | + ⟨целое без знака⟩ | — ⟨целое без знака⟩

⟨правильная дробь⟩ : : =. ⟨целое без знака⟩

Заметим, что за рубежом точкой принято отделять целую часть от дробной, поэтому в АЛГОЛе для указания правильной дроби принята точка, а не запятая.

⟨десятичное число⟩ : : = ⟨целое без знака⟩ | ⟨мантисса ⟩ | ⟨целое без знака⟩ ⟨правильная дробь⟩

⟨порядок⟩ : : = $_{10}$ ⟨целое⟩

⟨вещественное без знака⟩ : : = ⟨десятичное число⟩ | ⟨порядок⟩ | ⟨десятичное число⟩ ⟨порядок⟩

⟨число без знака⟩ : : = ⟨целое без знака⟩ | ⟨вещественное без знака⟩

⟨вещественное⟩ : : = ⟨вещественное без знака⟩ | + ⟨вещественное без знака⟩ | — ⟨вещественное без знака⟩

⟨число⟩ : : = ⟨целое⟩ | ⟨вещественное⟩

В указанных определениях правильная дробь, десятичное число и порядок определяются как числа без знака, при этом, например, число вида —0,135 не будет подходить под определение «правильная дробь», а будет подходить под определение «число».

Комплексные числа в алгоритмическом языке специально не выделяются. При программировании они представляются парами чисел.

Порядок — это масштабный множитель, представляющий собой степень десяти. Показатель степени может быть только целым десятичным числом (положительным и отрицательным). Отрицательный показатель степени в скобки не заключается.

Идентификаторы. Идентификаторы — это наименования различных величин. В отличие от принятого в математике правила обозначать величины отдельными буквами (возможно с индексами) в алгоритмических языках величины могут обозначаться группами букв или букв и цифр, в том числе словами.

Идентификатором называется любая последовательность букв или цифр начинающаяся с буквы:

⟨идентификатор ⟩ : : = ⟨буква ⟩ | ⟨идентификатор ⟩ ⟨(буква⟩ | ⟨идентификатор⟩ ⟨цифра⟩

Идентификаторы не имеют закрепленного за ними смысла, а используются для обозначения различных величин: простых переменных, массивов, меток, переключателей и процедур. Все идентификаторы, кроме постоянно закрепленных для стандартных функций (sin, cos и т. д., см. ниже), могут выбираться произвольно.

Чтобы облегчить человеку чтение алгоритмов, иногда удобно в качестве идентификаторов величин использовать названия этих величин (например, *высота, скорость* и т. п.). Каждый идентификатор должен быть описан в том блоке, внутри которого он используется. Исключение составляют идентификаторы ряда стандартных функций, за которыми в АЛГОЛе закреплены постоянные идентификаторы. Описание определяет вид величины, которая обозначается этим идентификатором (например, массив, процедуру и т. д.). Различные величины, используемые в одном и том же блоке программы, должны обозначаться обязательно различными идентификаторами. Нельзя один идентификатор описывать в одном и том же блоке несколько раз.

Строки. Строками называются группы основных символов, представляющие собой нечисловые константы. В отличие от идентификаторов, используемых для обозначения переменных величин, принимающих в процессе вычислений каждый раз некоторое конкретное числовое значение, строки представляют самих себя, т. е. являются конкретными нечисловыми значениями. Например, ‘ИВАНОВ’, ‘ПЕТРОВ’, ‘СИДОРОВ’ и т. д. Для отличия строк от идентификаторов используются кавычки, представляющие собой один из видов скобок.

Определение строки, данное с помощью метаязыка, имеет следующий вид:

⟨пусто⟩ : : =

⟨чистая строка⟩ : : = ⟨любая последовательность основных символов не содержащая символ ‘или символ ’⟩ | ⟨пусто⟩

«Пусто» обозначает строку, в которой нет ни одного символа.

⟨открытая строка⟩ : : = ⟨чистая строка⟩ |‘⟨открытая строка⟩’| ⟨открытая строка⟩ ⟨открытая строка⟩

Примеры открытых строк:

ПЛАТЕЖНАЯ ⊔ ВЕДОМОСТЬ
АНКЕТА ⊔ ‘ИВАНОВА’
‘ГОРОД ⊔ ТАШКЕНТ’

Открытая строка — это группа основных символов, которая может включать в себя кавычки, но не обязательно охватывается ими. Строка отличается от открытой строки тем, что она обязательно охватывается кавычками: ⟨строка⟩ : : — ‘⟨открытая строка⟩’

В эталонном языке АЛГОЛ предусматривается возможность использования строк только в качестве фактических параметров процедур. В рассматриваемом нами ниже варианте языка (АЛГЭМ) предусматривается более широкое использование строк для построения выражений и операторов; при этом вводятся строчные величины, которые могут иметь своим значением как отдельные строки, так и группы строк (в случае строчных массивов).

Метки. Как уже говорилось, некоторые операторы могут иметь перед собой метки, отделяемые от операторов двоеточиями. Метки — это по существу те же строки, но они используются не для обозначения нечисловой информации, подлежащей обработке, а для различения операторов программы с целью осуществления переходов. Кроме этого функционального отличия меток от строк в эталонном языке АЛГОЛ дается ограничение на структуру меток: метками могут быть только числа (целые без знака) или идентификаторы, а не любые наборы символов основного языка, как это определено для строк.

Однако это отличие не является принципиальным: метки подобно строкам представляют самих себя и им не могут присваиваться какие-либо значения, как это делается для идентификаторов. Метки могут являться значениями именующих выражений, в частности значениями указателей переключателей.

Переменные. Переменная — это наименование некоторой величины, которая может принимать определенные значения. Переменные в зависимости от типа значений, которые они могут принимать, делятся на числовые — целые и вещественные — и нечисловые — логические и строчные (последние не в АЛГОЛе).

Значение для числовых переменных (целых и вещественных) — это число или множество чисел (вектор, матрица и т. д.); для логических переменных — одно логическое значение **(истина —** 1 или **ложь** — 0) или множество логических значений; для строчных переменных — одна открытая строка или множество открытых строк. Значения переменных могут изменяться в процессе вычислений с помощью так называемого оператора присваивания (см. ниже). Тип значения переменной определяется описанием этой переменной или массива.

Приведем формальное синтаксическое определение понятия переменной. Для приближения данного изложения к эталонному АЛГОЛу мы сейчас не вводим в рассмотрение составные переменные, а даем определение переменной, подразумевая под ней элементарную (не составную) переменную.

⟨идентификатор переменной⟩: : = ⟨идентификатор ⟩
⟨простая переменная⟩ : : = ⟨идентификатор переменной⟩
⟨индексное выражение⟩ : : = ⟨арифметическое выражение⟩
⟨список индексов⟩ : : = ⟨индексное выражение⟩ | ⟨список индексов⟩, ⟨индексное выражение⟩
⟨идентификатор массива⟩ : : = ⟨идентификатор⟩
⟨переменная с индексами⟩ : : = ⟨идентификатор массива⟩ [⟨список индексов⟩]
⟨переменная⟩: : = ⟨простая переменная⟩ | ⟨переменная с индексами⟩

Для каждой переменной с индексом дается описание массива, элементом которого она является. Описание массива содержит все необходимые сведения об этом массиве (его идентификатор, размерность, размеры, тип значений). Переменная с индексами — это наименование одного элемента массива, заданного индексами.

Примеры переменных с индексами:

Обычные обозначения	Обозначения по АЛГОЛу
a_1	a [1]
$y_{i,j}$	y[i, j]
A_{x_i, y_j}	A [x [i], y [j]]

Список индексов состоит из одного или нескольких арифметических выражений, разделенных запятыми.

Арифметическое выражение, являющееся индексным выражением, по своему смыслу может быть

только целым числом. Если в результате вычислений получится нецелое число, в качестве значения индекса берется ближайшее целое. Это делается по формуле Е (А + 0,5), где Е означает целое, не превосходящее значения выражения в скобках, а А — фактическое значение арифметического выражения.

Значение переменной с индексами определенно только в том случае, если значение индекса не выходит за пределы границ индекса в соответствующем описании массива.

Указатели функций. Переменные величины в АЛГОЛе могут быть представлены также в виде так называемых указателей функций. В АЛГОЛе имеется способ для однократного представления часто используемых в разных местах программы автономных участков. Это делается с помощью так называемых процедур. Описание процедуры содержит полное описание (также на АЛГОЛе) данного участка программы и помещается в начале блока вместе с другими описаниями. Обращения к процедурам могут совершаться из любого места блока при помощи оператора процедуры или *указателя функции.* Обращения имеют стандартный вид: пишется идентификатор соответствующей процедуры, за которым в круглых скобках указываются ее аргументы, разделенные запятыми. Аргументы называются фактическими параметрами процедуры.

Процедуры в общем случае могут быть построены для выполнения разных действий и дают в результате одно или несколько значений переменных или не дают вообще ни одного значения, а осуществляют, например, перегруппировку величин. Если процедура используется для вычисления функции, то она обязательно в результате дает одно значение.

В тех местах программы, в которых нужно производить вычисления с помощью данной процедуры, ставятся обращения к этой процедуре, которые могут иметь, например, такой вид: SIN(x) или ПЛОШАДЬ (a, h) и т. д.

В последнем примере идентификатором процедуры является слово «площадь», а величины a и h представляют собой параметры этой процедуры и обозначают, например, основание и высоту треугольника.

Подобные процедуры, служащие для вычисления какой-либо одной величины (числовой, логической), называются процедурами-функциями.

Приведенные выше примеры обращения к процедурам-функциям называются указателями функций. В отличие от указателей функций обращения к процедурам, не являющимся функциями, называются операторами процедур. Операторы процедур по внешнему виду похожи на указатели функций, но между ними имеются следующие отличия:

а) обращение к описанию процедуры-функции осуществляется при помощи указателя функции, который в результате выполнения всех действий, заданных соответствующим описанием процедуры, принимает одно определенное значение, в то время как оператор процедуры может вычислять несколько значений или вообще не вычислять никаких значений, а выполнять некоторые другие действия в процессе вычислений (контроль вычислений, перепись данных и т. д.);

б) указатель функции стоит всегда внутри какого-нибудь выражения, в то время как оператор процедуры всегда фигурирует в виде отдельного оператора, т. е. после него ставится точка с запятой.

Примеры:

a + sin(x) и ВВОД (a, b**,** c);

Здесь sin (x) является указателем функции; ВВОД (a, b, c) — оператором процедуры;

в) процедура–функция в своем описании должна обязательно содержать оператор присваивания, осуществляющий присваивание вычисленного значения идентификатору процедуры–функции (который совпадает всегда с идентификатором указателя функции).

Синтаксическое определение указателя функции в АЛГОЛе имеет следующий вид:

⟨идентификатор процедуры ⟩ : : = ⟨идентификатор ⟩

⟨фактический параметр ⟩ : : = ⟨строка ⟩ | ⟨выражение⟩ | ⟨идентификатор массива⟩ | ⟨идентификатор переключателя ⟩ | ⟨идентификатор процедуры⟩

⟨строка букв ⟩ : : = ⟨буква⟩ | ⟨строка букв⟩ ⟨буква⟩

⟨ограничитель параметра⟩ : : = , |) ⟨строка букв⟩ : (

⟨список фактических параметров⟩ : : = ⟨фактический параметр⟩ | ⟨список фактических параметров⟩ ⟨ограничитель параметра⟩ ⟨фактический параметр⟩

⟨совокупность фактических параметров⟩ : : = ⟨пусто⟩ | (⟨список фактических параметров⟩)

⟨указатель функции⟩ :: = ⟨идентификатор процедуры⟩ ⟨совокупность фактических параметров⟩

Ограничитель параметра в виде) ⟨строка букв⟩ : (позволяет вставлять пояснительный текст между параметрами. Например, функция v, зависящая от величин t, D, H, т. е. v (t, D, H), может быть записана так:

v (t) дальность: (D) высота: (H)

Здесь величина t не имеет пояснений, а величины D и H поясняются.

В качестве фактических параметров наиболее часто фигурируют числа и переменные, которые являются частными случаями выражений.

Из приведенного синтаксического описания указателя функции следует, что аргументы функции всегда должны заключаться в скобки. Например, нельзя писать, как принято в математике, sin x, а нужно писать sin(x), так как запись sin x будет воспринята согласно правилам АЛГОЛа просто как идентификатор некоторой переменной.

Далее, так как в описании процедуры-функции обязательно должен быть оператор присваивания, осуществляющий присваивание вычисленного значения идентификатору процедуры-функции, то здесь могут встретиться необычные для математических формул записи. Например, присваивание некоторого значения может производиться просто идентификатору sin без указания аргумента x; этот идентификатор sin будет иметь определенное числовое значение, равное значению sin(x).

В АЛГОЛе для ряда стандартных функций постоянно закреплены идентификаторы, которые запрещается использовать для других целей.

Стандартные функции не описываются в программах, а используются сразу в тех выражениях, где они необходимы. К числу стандартных функций относятся следующие функции от выражения E, которое является аргументом (фактическим параметром):

abs (E) — для вычисления модуля (абсолютной величины) значения E,
sign (E) — для знака значения E (+1 для E > 0, 0 для E = 0 и -1 для E < 0),
sqrt (E) — для квадратного корня из значения E,
sin (E) — для синуса значения E,
cos (E) — для косинуса значения E,
arctan (E) — для главного значения арктангенса значения E,
ln (E) — для натурального логарифма значения E,
exp (E) — для показательной функции значения E (e^E).

В АЛГОЛе принято, что все эти функции могут быть использованы как с аргументами, имеющими тип **вещественный,** так и с аргументами, имеющими тип **целый.** Все эти функции, кроме функции sign (E), abs (E), дают значение, имеющее тип **вещественный.** Функция sign (E) дает значение целого типа. Тип значения указателя функции abs (E) совпадает с типом значения выражения E.

Выражения — это конструкции языка, имеющие определенное функциональное назначение; они показывают, какие действия и в каком порядке должны выполняться над величинами или как эти величины должны использоваться (например, в качестве операндов арифметических или логических операций, в качестве индексов или параметров и т. п.).

Составными частями выражений могут быть переменные, указатели функций, числа, логические значения, ограничители и некоторые символы операций.

Так как в синтаксические определения переменных и указателей функций входят выражения (например, индексное выражение, являющееся арифметическим выражением), то определение самих выражений, а также переменных и указателей функций может быть произведено только рекурсивным способом.

В АЛГОЛе используются три типа выражений: арифметические, логические и именующие:

⟨выражение ⟩: : = ⟨арифметическое выражение ⟩ | ⟨логическое выражение⟩ | ⟨именующее выражение⟩

Выражения делятся на два вида: простые и условные. Мы рассмотрим сначала простые арифметические и логические выражения, затем условные арифметические и логические выражения. Именующие выражения будут рассмотрены вместе с операторами перехода.

Простое арифметическое выражение, Это выражение представляет собой формулу, определяющую порядок вычислений некоторого числового значения.

Элементарными компонентами формул являются первичные арифметические выражения, которые будем называть просто первичными выражениями:

⟨знак операции типа сложения⟩ : : = + | -

⟨знак операции типа умножения⟩ : : = × |/ | ÷

⟨первичное выражение⟩: : = ⟨переменная⟩ | ⟨ указатель функции ⟩ | (⟨арифметическое выражение⟩)

Переменные и функции, являющиеся первичными арифметическими выражениями, могут быть только числовыми, т. е. иметь тип **целый** или **вещественный**.

Из первичных выражений образуются множители, представляющие собой степени первичных выражений:

⟨множитель⟩ : : = ⟨первичное выражение⟩ | ⟨множитель⟩ ↑ ⟨первичное выражение⟩

Показатель степени должен заключаться в скобки (круглые), кроме случаев, когда он является числом без знака, переменной или указателем функции. Отрицательный показатель заключается в скобки.

Примеры записи множителей:

$$a \uparrow b \uparrow c = (a^b)^c; \qquad a \uparrow (b \times c) = a^{bc}.$$

Несколько операций возведения в степень выполняются последовательно слева направо. Суть правил, определяющих тип результата, сводится к следующему: при нулевом основании результат равен нулю или не определен (при показателе, равном или меньшем нуля). При целом положительном или равном нулю показателе тип результата совпадает с типом основания. При целом отрицательном показателе и основании, не равном нулю, тип результата вещественный.

При основании a, большем нуля, и вещественном показателе r (любом) результат для a ↑ r вычисляется по формуле exp (r × ln (a)) и имеет тип вещественный. При основании, меньшем нуля, и вещественном показателе результат не определен.

⟨терм⟩ : : = ⟨множитель⟩ | ⟨терм⟩ ⟨знак операции типа умножения⟩ ⟨множитель⟩

В данном случае используется промежуточное понятие «терм», представляющее собой одночлен без знака.

Пример терма:

$$a \times b/c$$

Тип результата определяется следующими правилами: если множители целые, то и результат целый, в противном случае — вещественный.

Если делитель равен нулю, то операция «/» не определена. Операция «÷»— целочисленное деление — определена только для целых при условии, что делитель не равен нулю. Результат операции всегда целый и определяется по формуле

$$m \div n = \text{sign}\,(m/n) \times \text{entier}(\text{abs}\,(m/n))$$

Здесь слово entier обозначает взятие целой части, т. е. наибольшего целого, не превышающего фактического значения аргумента.

⟨простое арифметическое выражение) : : = ⟨терм⟩ | ⟨знак операции типа сложения⟩ ⟨терм⟩ | ⟨простое арифметическое выражение⟩ ⟨знак операции типа сложения⟩ ⟨терм⟩

При вычислении арифметических выражений с вещественными величинами точность вычислений может контролироваться методами численного анализа с помощью операций, задаваемых на алгоритмическом языке.

Следует остановиться еще на порядке выполнения действий при вычислении арифметических выражений. Сначала вычисляются выражения в скобках, затем выполняются операции в порядке следующего приоритета:

↑ первый приоритет;
× / ÷ второй приоритет;
+ — третий приоритет.

Операции одинакового старшинства выполняются в порядке записи слева направо.

Для выяснения возможности выполнения очередной операции каждый раз производится анализ следующей по порядку операции, и если она оказывается одинаковой или младшей по приоритету, то выполняется данная операция. В противном случае производится анализ следующей операции и т. д. до тех пор, пока не будет встречена младшая или равная по приоритету операция.

Пример:

$$a + b \times c \uparrow (d + e)$$

Это выражение будет вычисляться в следующем порядке:

(d + e)
c ↑ (d + e)
(c ↑ (d + e)) × b
((c ↑ (d + e)) × b) + a

Простые логические выражения. Логические выражения и логические переменные и функции могут иметь одно из двух значений : **истинно** (1) или **ложно** (0). Синтаксическое определение простого логического выражения имеет следующий вид:

⟨знак операции отношения⟩:: = < | ⩽ | = | > |⩾ | ≠

⟨отношение ⟩ : : = ⟨простое арифметическое выражение ⟩ ⟨знак операции отношения ⟩ ⟨простое арифметическое выражение ⟩

⟨первичное логическое выражение⟩ : : = ⟨логическое значение⟩ | ⟨переменная⟩ | ⟨указатель функции⟩ | ⟨отношение⟩ | (⟨логическое выражение⟩)

⟨вторичное логическое выражение⟩:: = ⟨первичное логическое выражение⟩ | ⌝⟨первичное логическое выражение⟩

⟨логический одночлен⟩ : : = ⟨вторичное логическое выражение⟩ | ⟨логический одночлен⟩ /\ ⟨вторичное логическое выражение⟩

⟨логический терм⟩ : : = ⟨логический одночлен⟩ | ⟨логический терм⟩ \/ ⟨логический одночлен⟩

⟨импликация⟩: : = ⟨логический терм⟩ | ⟨импликация⟩⊃ ⟨логический терм⟩

⟨простое логическое выражение⟩ : : = ⟨импликация⟩ | ⟨простое логическое выражение⟩ ≡ ⟨импликация⟩

Переменные и функции, являющиеся первичными логическими выражениями, должны иметь тип **логический.**

Примеры простых логических выражений:

A\/B/\C⊃(D\/F)
X > У /\ a ⩽b /\ ⌝d

Таким образом, первичное логическое выражение — это величина, принимающая одно из возможных логических значений, а простое логическое выражение — это комбинация таких величин, соединенных знаками логических операций.

Определения пяти логических операций, включенных в состав АЛГОЛа, даны в табл. 1

Таблица 1

A	0	0	1	1
B	0	1	0	1
⌝A	1	1	0	0
A ∧ B	0	0	0	1
A ∨ B	0	1	1	1
A ⊃B	1	1	0	1
A ≡ B	1	0	0	1

В логических выражениях, как мы видели, могут участвовать арифметические операции, отношения и логические операции. Для всех этих операций имеет место следующий общий порядок приоритета (старшинства):

арифметические операции

↑
× | ÷
+ —

операции отношения

< ⩽ = ⩾ > ≠

При рассмотрении операций отношения вопрос о их старшинстве по отношению друг к другу не возникает;

логические операции

⌝

/\
\/
⊃
≡

Во всех случаях, когда нужно изменить порядок выполнения операций (всех трех видов), используются круглые скобки.

Условные арифметические выражения. Условные арифметические выражения включают в себя наряду с арифметическими операциями и проверку некоторых условий, определяющих выбор того или иного варианта вычисления данного арифметического выражения. Проверяемые условия формулируются всегда в виде логических выражений:

⟨условие ⟩ : : = **если** ⟨логическое выражение ⟩ то

⟨условное арифметическое выражение⟩ : : = ⟨условие⟩ ⟨простое арифметическое выражение⟩ **иначе** ⟨арифметическое выражение⟩

⟨арифметическое выражение⟩ : : = ⟨простое арифметическое выражение⟩ | ⟨условное арифметическое выражение⟩

Следует обратить внимание на то, что после условия всегда следует простое арифметическое выражение, а после символа **иначе** — арифметическое выражение, в том числе условное арифметическое выражение и т. д.

Конструкция условных арифметических выражений схематически может быть представлена в виде

если Л1 **то** А1 **иначе** А

если Л1 **то** А1 **иначе если** Л2 **то** А2 **иначе если** Л3 **то** А3 **иначе** А

где Л1, Л2, Л3 — логические выражения;

Al, A2, A3 — простые арифметические выражения;

А — арифметическое выражение.

Вычисление подобных условных арифметических выражений производится следующим образом. Последовательно проверяются слева направо логические выражения (Л1, Л2, Л3), стоящие после символов если.

Первое истинное логическое выражение указывает, что следует взять в качестве значения данного условного арифметического выражения то простое арифметическое выражение, которое стоит за символом то, следующим за истинным логическим выражением. Если все логические выражения окажутся ложными, то в качестве значения условного выражения берется последнее арифметическое выражение, которое, конечно, будет простым.

Пример.

Пусть величины x, y, a, b имеют тип вещественный. Тогда можно записать такое условное арифметическое выражение:

если x = a /\ y < b то КОЭФФИЦИЕНТ 2 + b **иначе** КОЭФФИЦИЕНТ 2 — b.

Заметим, что в случае истинности проверяемого условия значением этого условного арифметического выражения будет сумма двух величин: КОЭФФИЦИЕНТ 2 и b, а в случае ложности — разность этих же величин.

Условные логические выражения строятся подобно условным арифметическим выражениям, только вместо простых арифметических выражений используются простые логические выражения, а вместо арифметического выражения в конце условного выражения ставится логическое выражение:

⟨условное логическое выражение⟩ : : = ⟨условие⟩ ⟨простое логическое выражение⟩ **иначе** ⟨логическое выражение⟩

⟨логическое выражение⟩ : : = ⟨простое логическое выражение⟩ | ⟨условное логическое выражение⟩

Условие определяется точно так же, как и в случае условных арифметических выражений. Общий вид условного логического выражения может быть представлен следующей схемой:

если Л1 **то** L1 **иначе если** Л2 **то** L2 **иначе** Л3

Здесь Л1, Л2, Л3 — логические выражения;

L1, L2 — простые логические выражения.

Особенностью вычисления подобных условных логических выражений является то, что стоящие после символов **если** логические выражения (Л1, Л2, и т. д.) сами могут быть условными логическими выражениями, и, таким образом, могут встретиться несколько подряд стоящих символов **если** и за

последним из них будет стоять простое логическое выражение.

Способ вычисления сводится к последовательному выделению условных логических выражений, начиная с самого внутреннего простого логического выражения, и замене этих условных логических выражений их значениями.

При этом нужно придерживаться основного правила, что каждому символу **если** соответствует ближайший справа символ **иначе,** не считая уже выделенных выражений. Пример условного логического выражения показан на рис. 15, *а.*

Просматриваем выражение и после третьего **если** находим простое логическое выражение L1. Вычисляем это выражение, и если оно истинно, то берем L2, если ложно, то находим ближайший справа символ **иначе** и берем стоящее за ним логическое выражение (11). В данном примере оно оказалось также условным логическим выражением; мы его вычисляем аналогичным образом, причем подобная цепочка действий может продолжаться, так как Л1 может также оказаться условным логическим выражением. Допустим, что мы нашли значение выражения 11 и при L1 ложном оно будет значением выражения 12. Теперь наше условное логическое выражение примет вид, показанный на рис. 15, *б.*

Подобным же образом вычисляем условное логическое выражение 13 и находим его значение (L5 или Л2). Самое внешнее условное логическое выражение будет иметь вид

если 13 **то** L6 **иначе** Л3 Оно вычисляется описанным выше способом.

В условных выражениях весьма важно указание, что после символа то обязательно должно стоять простое (а не условное) выражение. Это обеспечит соответствие между символами **если, то** и **иначе,** относящимися к одному и тому же выражению. При отсутствии такого ограничения могла бы возникнуть неопределенность в их понимании.

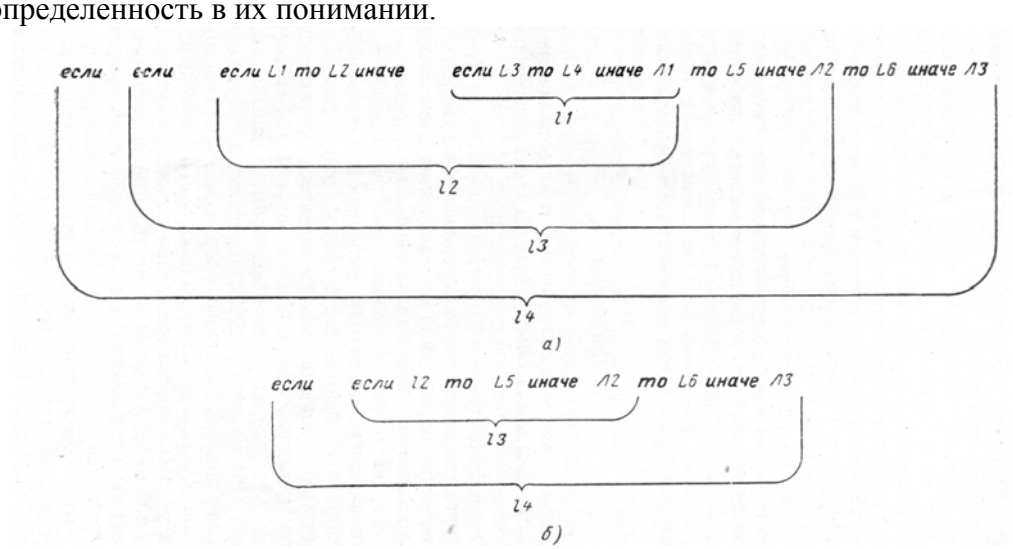


Рис. 15. Пример условного логического выражения.

Далее, в условных выражениях, в отличие от условных операторов (см. ниже), обязательно каждому символу **если** соответствует символ **иначе.**

Любое условное логическое выражение может быть представлено эквивалентным ему простым логическим выражением. Использование условных логических выражений может в ряде случаев существенно сократить процесс вычислений по сравнению с представлением той же логической зависимости простым логическим выражением, так как при этом вычисляется одно из двух выражений: либо стоящее после **то,** либо стоящее после **иначе.**

Однако различия в порядке вычислений этих двух видов логических выражений могут привести иногда к побочным эффектам, если в этих логических выражениях имеются функции, которые при одном порядке вычислений должны вычисляться, а при другом — не должны вычисляться.

8. Операторы

Операторы — это единицы действий в алгоритмическом языке. Операторы могут объединяться в составные операторы и блоки, которые являются частными видами операторов, и поэтому определение понятия «оператор» по необходимости рекурсивное; при этом предполагается также, что и синтаксические определения описаний уже даны и они могут использоваться при определении

операторов.

Естественным порядком выполнения операторов называется тот порядок, когда они выполняются в порядке их написания в программе сверху вниз и слева направо. Этот естественный порядок может измениться при помощи операторов перехода, которые явно определяют своего преемника, т. е. оператор, который должен выполняться следующим. При помощи условных операторов могут осуществляться пропуски некоторых операторов. При помощи операторов цикла осуществляются циклические повторения некоторых участков программы, а при помощи операторов процедур (указателей функций) осуществляется вклинивание в выполняемую последовательность операторов некоторых заданных участков программы, называемых процедурами.

Для того чтобы можно было указывать операторы, к которым должны совершаться переходы, эти операторы снабжаются метками. Приведем общую классификацию операторов и синтаксические определения понятий, относящихся к операторам:

⟨непомеченный основной оператор⟩ : : = ⟨оператор присваивания⟩ | ⟨оператор перехода⟩ | ⟨пустой оператор⟩ | ⟨оператор процедуры⟩

⟨основной оператор⟩ : : = ⟨непомеченный основной оператор ⟩ | ⟨метка⟩ : ⟨основной оператор⟩

⟨безусловный оператор⟩ : : = ⟨основной оператор⟩ | ⟨составной оператор⟩ | ⟨блок⟩

⟨оператор⟩ : : = ⟨безусловный оператор⟩ | ⟨условный оператор⟩ | ⟨оператор цикла⟩

Синтаксические определения понятий «составной оператор», «блок» и «программа» выглядят следующим образом:

⟨конец составного⟩ : : = ⟨оператор⟩ **конец** | ⟨оператор⟩; ⟨конец составного⟩

⟨начало блока⟩: : = **начало** ⟨описание⟩ | ⟨начало блока⟩; ⟨описание⟩

⟨непомеченный составной⟩ : : = **начало** ⟨конец составного⟩

⟨непомеченный блок⟩ : : = ⟨начало блока⟩; ⟨конец составного⟩

⟨составной оператор⟩ : : = ⟨непомеченный составной⟩ | ⟨метка)⟩ : ⟨составной оператор⟩

⟨блок⟩ : : = ⟨непомеченный блок⟩ | ⟨метка⟩ : ⟨блок⟩

⟨программа⟩ : : = ⟨блок⟩ | ⟨составной оператор⟩

Оператор присваивания. Значения арифметических и логических выражений (а также строчных выражений в АЛГЭМе) могут присваиваться одной или нескольким переменным или функциям. (Последнее применяется только в описаниях процедур-функций.) Значения присваиваются с помощью так называемого оператора присваивания, играющего основную роль в любом алгоритмическом языке. В машинной интерпретации оператор присваивания соответствует посылке в ячейку (или группу ячеек).

Синтаксическое определение оператора присваивания в АЛГОЛе:

⟨левая часть⟩ : : = ⟨переменная⟩ : = | ⟨идентификатор процедуры⟩ : =

⟨список левой части⟩ : : = ⟨левая часть⟩ | ⟨список левой части⟩ ⟨левая часть⟩

⟨оператор присваивания⟩ : : = ⟨список левой части⟩ ⟨арифметическое выражение⟩ | ⟨список левой части⟩ ⟨логическое выражение⟩

Символ : = означает «принимает значение» или «становится равным».

Примеры операторов присваивания:

ПРИЗНАК СОВПАДЕНИЯ : = **если** a = b **то** 5 **иначе** 10;

z : = y: = x: = a + b;

В первом примере в правой части оператора присваивания стоит условное арифметическое выражение. Как было сказано раньше, в конце оператора ставится точка с запятой.

В отличие от обычных математических формул в списке левой части оператора присваивания может стоять переменная, используемая и в правой части, например:

n: =n + 1;

Идентификатору n в правой части соответствует значение этой переменной довыполнения оператора присваивания, а идентификатор n в левой части представляет значение этой переменной уже после выполнения оператора присваивания.

Типы переменных в левой и правой частях оператора присваивания не обязательно должны быть одинаковыми. Если переменная в левой части имеет тип целый, а в правой части результат вычисления арифметического выражения имеет тип вещественный, то при выполнении оператора присваивания производится округление результата до ближайшего целого по формуле entier (A + 0,5), где A — фактическое значение результата.

Для обеспечения однозначности результатов в тех случаях, когда в операторах присваивания встречаются переменные с индексами, изменяемыми этим оператором присваивания, установлено следующее правило, определяющее порядок действий при выполнении таких операторов присваивания.

Сначала вычисляются значения всех индексных выражений, встречающихся в списке левой части оператора присваивания, затем вычисляется значение правой части оператора присваивания. Полученное значение правой части присваивается всем переменным списка левой части, взятым со значениями ранее вычисленных индексных выражений. В АЛГОЛе индексы у переменных указываются в квадратных скобках.

Например:

```
i : = j : = k : = m + 1;
x [i, j, k] : = i: = j: = k : = m + n;
```

Величина m + n будет присвоена величине x, взятой со значениями индексов i, j, k, равными m + 1; а сами индексы после этого получат значение m + n.

Именующие выражения и оператор перехода. В алгоритмических языках, как вообще в математике, действует правило, согласно которому порядок выполнения вычислений по нескольким формулам определяется порядком их записи: слева направо и сверху вниз. В таком порядке выполняются обычно и операторы в АЛГОЛ - программах.

При необходимости изменить этот естественный порядок выполнения операторов используется оператор перехода, соответствующий по своему смыслу команде безусловного перехода в машинах.

Оператор перехода определяется синтаксически следующим образом:

⟨оператор перехода⟩ : : = **на** ⟨именующее выражение⟩

где **на** — это основной символ, обозначающий одну из операций следования, а именно операцию перехода. Именующее выражение является правилом, определяющим метку того оператора, к которому должен быть совершен переход.

Как говорилось раньше, те операторы программы, к которым должны или могут совершаться переходы, снабжаются метками; метки пишутся перед этими операторами и отделяются от них двоеточиями.

Именующее выражение по своему назначению близко к строчному выражению, используемому в АЛГЭМе ⟨см. ниже⟩. Отличие заключается в том, что строчные выражения служат для представления обрабатываемо ой информации, а именующее выражение — для описания самого процесса обработки. Но и те и другие выражения служат для представления нечисловых значений.

Приведем синтаксическое определение именующего выражения:

⟨метка⟩ : : = ⟨идентификатор⟩ | ⟨целое без знака⟩

⟨идентификатор переключателя⟩:: = ⟨идентификатор⟩

⟨указатель переключателя⟩ : : = ⟨идентификатор переключателя⟩ [⟨индексное выражение⟩]

⟨простое именующее выражение⟩:: = ⟨метка⟩ | ⟨указатель переключателя⟩ | ⟨именующее выражение⟩

⟨именующее выражение⟩ : : = ⟨простое именующее выражение⟩ | ⟨условие⟩ ⟨простое именующее выражение⟩ **иначе** ⟨именующее выражение⟩

Как видно из этой записи, структура условного именующего выражения полностью совпадает со структурой условных выражений арифметического и логического типов.

Понятие метки было рассмотрено раньше; меткой может быть идентификатор или целое без знака, причем это целое, используемое в качестве метки, трактуется просто как строка символов. Часто ограничиваются использованием в качестве меток только идентификаторов, считая, что применение целого без знака в качестве метки может привести к некоторым неудобствам.

Используя арифметические и логические выражения, операторы присваивания, метки и операторы перехода, можно уже строить разнообразные программы, в том числе программы, включающие в себя циклы.

Рассмотрим, например, задачу о нахождении суммы парных произведений двух групп чисел p**[**i**]** и q**[**i], где i меняется от 1 до n.

Пусть S обозначает вычисляемую сумму.

Программа на АЛГОЛе (без описания информации) будет иметь вид:

```
S: =0;          i : = 1;
M:S: =p[i] × q[i] + S;          i : = i + 1;
```

на если i≤n то M **иначе** ДАЛЬШЕ;

Здесь идентификатор ДАЛЬШЕ обозначает метку некоторого оператора, к которому нужно перейти после окончания данного расчета. В программе использован оператор перехода с условным именующим выражением **если** i ≤ n **то** M **иначе** ДАЛЬШЕ;

Рассмотрим назначение и строение *переключателя.* Часто при программировании встречаются случаи, когда необходимо из одного и того же места программы перейти в разные места программы в зависимости от выполнения некоторых условий. Такие переходы можно запрограммировать, используя оператор перехода с условным именующим выражением, но более экономно переходы представляются с помощью специального средства, предусмотренного в АЛГОЛе и называемого переключателем.

Переключатель состоит из двух частей: указателя переключателя и описания переключателя. Указатель переключателя представляет собой переменную с индексом; эта переменная в зависимости от значения индекса может принимать одно из значений, заданных описанием переключателя.

Значением указателя переключателя в конечном счете должна быть метка. Указатель переключателя ставится на место именующего выражения в операторе перехода, например:

на П [i];

здесь П — идентификатор переключателя, а i — индексное выражение.

Описание переключателя имеет вид

переключатель П : = И1, И2, И3, . . ., Иn;

Здесь переключатель — основной символ, П — идентификатор переключателя (в данном примере именно П), а идентификаторы И1, И2, И3, . . ., Иn образуют так называемый переключательный список, содержащий те именующие выражения, которые могут представлять значения данного переключателя. Они могут быть просто метками, а могут быть и снова указателями переключателей или условными именующими выражениями.

Описание переключателя помещается вместе с остальными описаниями, относящимися к данному блоку, т. е. в его начало. Действие переключателя происходит следующим образом.

Перед выполнением оператора перехода, содержащего указатель переключателя, должен быть выполнен оператор, определяющий значение индексного выражения в указателе переключателя. После этого выполнение оператора перехода сводится к тому, что по идентификатору переключателя находится описание соответствующего переключателя в начале блока, а по значению индекса указателя переключателя в переключательном списке выбирается именующее выражение, стоящее на том месте (считая слева направо), номер которого равен значению индекса. В общем случае найденное именующее выражение может оказаться снова указателем переключателя или даже условным именующим выражением, которое укажет другой указатель переключателя; в этом случае процесс повторяется аналогичным образом, т. е. производится обращение к другому описанию переключателя и т. д.

В эталонном АЛГОЛе допускается произвольная глубина подобных переходов. Практически же эти возможности, как правило, не используются.

Значение указателя переключателя определено только в том случае, если его индексное выражение имеет значение одного из целых чисел от i до n, где n — число членов в переключательном списке. Если значение указателя переключателя в операторе перехода не определено, то оператор перехода также не определен.

Как будет видно из дальнейшего, условные переходы можно осуществлять не только с помощью условных именующих выражений, но и с помощью условных операторов, что, вообще говоря, практически применяется чаще.

Приведем синтаксическое определение описания переключателя:

⟨идентификатор переключателя ⟩ : : = ⟨идентификатор ⟩

⟨переключательный список⟩ : : = ⟨именующее выражение⟩ | ⟨переключательный список⟩, ⟨именующее выражение ⟩

⟨описание переключателя⟩ : : = **переключатель** ⟨идентификатор переключателя⟩ : = ⟨переключательный список⟩

Для пояснения назначения переключателя рассмотрим следующий пример. Пусть из некоторого места программы требуется осуществить переход к четырем различным операторам, имеющим метки Ml, M2, M3, M4; причем переход должен происходить в зависимости от значения некоторой величины

К, которая может принимать целые значения 1, 2, 3 или 4.

Используя условное именующее выражение, этот переход можно записать так:

на если К = 1 **то** Ml **иначе если** К = 2 **то** М2 **иначе**
если К = 3 **то** М3 **иначе** М4;

Этот же переход при помощи переключателя будет иметь вид:

переключатель П : =М1, М2, М3, М4;
. . . .
на П[К];
. . . .

Ясно, что в обоих случаях предварительно должна быть вычислена величина **К**. Одно и то же описание переключателя может использоваться многими указателями переключателя, находящимися в разных местах программы.

В связи с рассматриваемым вопросом о реализации переходов в программе следует сделать несколько замечаний относительно пустого оператора. Это понятие, как мы уже говорили, введено для того, чтобы можно было помещать в программах метки, за которыми не стоят операторы. Такие метки нужны, в частности, для указания перехода на конец блока или составного оператора.

Например: **начало . . .;** КОНЕЦ: **конец**

Здесь идентификатор КОНЕЦ перед символом **конец** означает метку пустого оператора.

Для иллюстрации применения переключателя приведем пример программы вычисления суммы парных произведений двух групп величин p [i] и q [i], где i изменяется от 1 до n (без описаний данных):

начало переключатель M: =M1, ВЫХОД;

i : = 1; S : = 0;

Ml :S: =S + p[i] × q [i]; i : = i + 1;
на M [если i ≤ n, **то** 1 **иначе** 2];
ВЫХОД: ПЕЧАТЬ (‘S’=, S)
конец

Этот же пример можно записать в следующем виде:

начало переключатель M : **= если** i ≤ n **то** Ml **иначе** ВЫХОД;
i: = 1; S: =0;

Ml :S: =S + p[i] ×q[i]; i : = i + 1;
на M [1];
ВЫХОД : ПЕЧАТЬ (‘S-’ , S)
конец

Заметим, что после метки ВЫХОД стоит оператор специальной процедуры выдачи данных на печать. В этом операторе в круглых скобках указаны в качестве фактических параметров (т. е. аргументов) этой процедуры те величины, которые должны быть выданы на печать. Первая величина является строкой, состоящей из двух символов S — , заключенных в кавычки. Вторая величина — просто величина S, обозначающая результат вычислений. Так как любая строка представляет самое себя и не принимает никаких значений, то на печать в качестве первой величины будут выданы два символа S—, за которыми будет напечатано то числовое значение, которое получит S в результате вычислений.

Условный оператор. Как мы уже видели, изменение порядка выполнения операторов в зависимости от некоторого условия может быть осуществлено с помощью оператора перехода, в котором после символа **на** стоит условное именующее выражение. Однако более употребительным является другой способ, основанный на использовании условных операторов. Структура условного оператора напоминает структуру условных выражений, только вместо выбираемых выражений в нем стоят операторы. Синтаксическое определение условного оператора имеет вид:

⟨условие⟩ : : = **если** ⟨логическое выражение⟩ **то**

⟨безусловный оператор ⟩ : : = ⟨основной оператор⟩ | ⟨составной оператор ⟩ | ⟨блок ⟩

⟨оператор «если»⟩ : : = ⟨условие⟩ ⟨безусловный оператор⟩

⟨условный оператор⟩ : : = ⟨оператор «если»⟩ | ⟨оператор «если»⟩ **иначе** ⟨(оператор⟩ | ⟨условие⟩ ⟨оператор цикла⟩ | ⟨метка⟩ : ⟨условный оператор⟩

Заметим, что в операторе «если» разрешается использовать только безусловный оператор, а

условный оператор с участием оператора цикла разрешается строить только без продолжения **иначе** ⟨оператор⟩. Оба эти ограничения имеют целью предотвратить неопределенность в согласовании символов **если** и **иначе.** Эта неопределенность могла бы возникнуть в связи с тем, что условный оператор может быть двух видов: **с** продолжением **иначе** ⟨оператор⟩ и без такого продолжения. Поэтому если бы разрешить в условном операторе ставить после символа то снова условный оператор, то могла бы возникнуть, например, такая картина:

если Л1 **то если** Л2 **то** S1 **иначе** S2;

Здесь неясно, какому из двух **если** соответствует **иначе.**

Как будет видно из дальнейшего, в самом операторе цикла разрешается использовать условный оператор, что тоже могло бы привести к подобной неопределенности, если бы не было указанных ограничений. Если все же нужно образовать условный оператор с оператором цикла, после которого должен идти символ **иначе** с последующим оператором, то это может быть достигнуто путем заключения оператора цикла в скобки **начало** и **конец.** При этом оператор цикла превращается в составной оператор, т. е. относится уже к безусловным операторам. Заметим, что благодаря наличию понятия пустого оператора перед символом **конец** можно ставить или не ставить точку с запятой. Перед символом **иначе** точку с запятой ставить нельзя ни в каких случаях.

Из приведенного выше синтаксического определения видно, что условный оператор может быть двух видов:

а) односторонний условный оператор вида

если Л **то** S1 ; S ;

где Л — логическое выражение, S1 — безусловный оператор или оператор цикла и S — какой-то оператор, который следует за данным условным оператором. Суть работы одностороннего условного оператора состоит в том, что в зависимости от того, истинно или ложно логическое выражение Л, оператор S1 выполняется или пропускается. В обоих случаях следующим выполняется оператор S;

б) двусторонний условный оператор вида

если Л **то** S1 **иначе** S2 ; S ;

Здесь S1 — безусловный оператор, а S2 и S — операторы. Если логическое выражение Л истинно, то выполняется оператор S1 и пропускается оператор S2, в противном случае пропускается оператор S1 и выполняется оператор S2. В обоих случаях следующим за условным оператором выполняется оператор S (если операторы S1, S2, а также S1 в предыдущем варианте не содержат внутри себя операторов перехода). Оператор, стоящий после символа **иначе,** сам может быть условным оператором, и, таким образом, могут образовываться цепочки произвольной длины, включающие в себя несколько символов **если** с соответствующими логическими выражениями (Л1, Л2, Л3, и т. д.). Например,

если Л1 **то** S1 **иначе если** Л2 **то** S2 **иначе** S ;

Здесь S1 и S2 — безусловные операторы, а S — оператор.

Выполнение подобных условных операторов сводится к последовательной проверке слева направо логических выражений. Первое истинное логическое выражение указывает, что должен быть выполнен безусловный оператор, следующий за этим выражением (за символом то). Если все логические выражения окажутся ложными, то выполняется последний оператор, стоящий за символом **иначе,** а если его нет, то выполняется оператор, стоящий за данным условным оператором. Следует заметить, что возможны переходы извне к какому-нибудь оператору, входящему в состав условного оператора (и имеющему, конечно, свою метку). В этом случае, если оператор, к которому совершен переход извне, сам не определяет своего преемника, т. е. того оператора, который должен выполняться за ним, следующим будет выполняться преемник всего условного оператора таким же образом, как если бы выполнение условного оператора происходило с самого начала и выбор для выполнения данного оператора (т. е. оператора, к которому совершен переход) произошел в результате проверки предшествующих условий.

В качестве примера применения условных операторов приведем программу вычисления двух сумм положительных и отрицательных чисел, выбранных из общей последовательности чисел $a_0, a_1, \ldots, a_n$ (без описания данных):

начало S: = t : = i : = 0;

ПРОВЕРКА : **если** $i \leq n$ **то**

начало если a [i] > 0 **то** S : = S + a [i]

иначе t : = t + a [i] i : = i + 1;
на ПРОВЕРКА;
конец
конец

Вторым примером будет служить запись **с** помощью условного оператора программы вычисления скалярного произведения двух n-мерных векторов p и q, которая раньше была записана с помощью переключателя и условного именующего выражения табл. 2.

Таблица 2

Условный оператор	Условное именующее выражение

начало
i: = l; s:=0;
M1:s: = s + p[i] × q[i];
i:=i+l;
если i ≤n **то на** Ml**;**
ПЕЧАТЬ (‘s = ’ , s);
конец
	начало
	переключатель M: =M1, ВЫХОД;
	i: = l; s: = 0;
	Ml : s: = s + p[i]×q[i];
	i: = i+l;
	на M **[если** i≤n **то** 1 **иначе** 2];
	ВЫХОД : ПЕЧАТЬ (‘s = ’ , s)
	конец

Пустой оператор.

⟨пустой оператор⟩ : : = ⟨пусто⟩

Пустой оператор не выполняет никакого действия; он может служить для помещения метки.

Оператор цикла. В связи с важностью циклических вычислительных процессов для решения самых разнообразных задач с помощью цифровых программно-управляемых машин в АЛГОЛе предусматривается специальный оператор цикла. Этот оператор обеспечивает более наглядную и компактную запись таких процессов по сравнению с представлением их с помощью операторов перехода и условных именующих выражений или условных операторов. Синтаксическое определение оператора цикла:

⟨элемент списка цикла ⟩ : : = ⟨арифметическое выражение⟩ | ⟨арифметическое выражение⟩ **шаг** ⟨арифметическое выражение⟩ **до** ⟨арифметическое выражение⟩ | ⟨арифметическое выражение⟩) **пока** ⟨логическое выражение⟩

⟨список цикла⟩ : : = ⟨элемент списка цикла⟩ | ⟨список цикла⟩, ⟨элемент списка цикла⟩

⟨заголовок цикла⟩ : : = **для** ⟨переменная⟩ : = ⟨список цикла⟩ **цикл**

⟨оператор цикла⟩ : : = ⟨заголовок цикла⟩ ⟨оператор⟩ | ⟨метка⟩ : ⟨оператор цикла⟩

Как видно из приведенного определения элемента списка цикла, эти элементы могут быть трех типов, и соответственно им принято различать три типа циклов: цикл с перечислением, цикл с арифметической прогрессией и цикл с проверкой (последний тип цикла называют иногда циклом с пересчетом).

Цикл с перечислением имеет вид

для v : = Al, A2, A3, . . ., An **цикл** S;

Здесь **для** является символом начала, а **цикл** — символом конца заголовка цикла. Идентификатор переменной v представляет собой так называемый параметр цикла, т. е. переменную величину, которая изменяется с каждым повторением цикла. В различных конкретных циклах этот параметр может быть обозначен любыми другими идентификаторами; здесь в общем случае может стоять и переменная с индексами, хотя часто делают ограничение, что параметром цикла может быть только простая переменная. Символ присваивания „: = “ в данном случае условно показывает, что параметр цикла v должен последовательно принимать значения арифметических выражений Al, A2, A3, . . ., An, вычисляемых непосредственно перед каждым выполнением оператора S. Этот символ S условно обозначает некоторый оператор, который должен быть многократно повторен (выполнен в цикле) при различных значениях параметра цикла v.

Цикл с арифметической прогрессией имеет вид

для v : = A1 **шаг** A2 **до** A3 **цикл** S;

Здесь A1 — арифметическое выражение, показывающее начальное значение параметра цикла V. Этот параметр при данном типе элемента списка цикла должен изменяться по закону арифметической

прогрессии с шагом А2. Арифметическое выражение А3, представляющее собой ограничение значения параметра цикла, определяет момент окончания повторений цикла для данного элемента списка цикла. Все три выражения Аl, А2, А3 могут быть как положительными, так и отрицательными; между ними должны соблюдаться естественные для арифметической прогрессии соотношения.

Порядок выполнения написанного выше оператора цикла может быть представлен следующим составным оператором:

начало v : =A1;
M : **если** (v — A3) × sign (A2) < 0 **то**
начало S**;** v : = v + A2 ; **на** M **конец**
конец

Цикл с элементом списка типа пересчета имеет вид

для v : = A1 **пока** Л **цикл** S;

Здесь А1 — значение параметра цикла v, которое он принимает при каждом повторении цикла, а Л — логическое выражение, которое проверяется перед каждым новым повторением цикла (т. е. перед выполнением оператора S). Пока это выражение является истинным, цикл повторяется; как только Л становится ложным, повторение цикла прекращается.

Как видно из общего синтаксического определения списка цикла, в этом списке допускается участие элементов различных типов, расположенных в различном порядке. Например:

для v : = A1 **шаг** A2 **до** A3, A4, A5, A6 **пока Л цикл** S;

Здесь сначала будет выполняться оператор цикла с элементом списка цикла типа арифметической прогрессии, затем два раза повторится цикл со значениями параметра цикла v, равными А4 и А5, а затем будет выполняться цикл с элементом списка цикла типа пересчета.

В отношении оператора цикла необходимо сделать следующие замечания.

Переходы извне к оператору S, который должен выполняться в цикле, или к операторам, входящим в его состав, правилами АЛГОЛа не допускаются, и результат действия оператора перехода, находящегося вне оператора S и ведущего внутрь оператора, будет неопределенным.

Выходы из этого оператора могут быть двух видов:

1) естественный выход, происходящий при выполнении числа повторений цикла, заданного всем списком цикла;

при таком выходе значение параметра цикла v считается неопределенным, т. е. эту величину нельзя использовать в операторах, которые будут выполняться вслед за оператором цикла;

2) «досрочный» выход, происходящий при наличии в составе оператора S, выполняемого в цикле, оператора перехода, ведущего к какому-нибудь оператору, находящемуся вне данного цикла. При таком («досрочном») выходе из цикла считается, что параметр цикла v сохраняет то значение, которое он имел непосредственно перед выходом из цикла, и эту величину можно, таким образом, использовать в последующих вычислениях (например, для того чтобы определить, сколько раз был повторен данный цикл).

В качестве примера программы с оператором цикла рассмотрим задачу о вычислении пяти значений полинома:

$$y_j = \sum_{i=0}^{n} a_i x_j^i \qquad \text{для} \quad j = 1, 2, 3, 4, 5.$$

Вычисления ведутся по схеме Горнера:

начало
для j : = 1, 2, 3, 4, 5 **цикл**
начало y [j] : = 0;
для i = n **шаг —** 1 до 0 **цикл**
y [j] : =y [j] × x [j] + a [i];
конец
конец

Пять значений x_j (j = 1, 2, 3, 4, 5) считаются заданными. Они могут быть расположены в последовательных ячейках памяти машины, так же как и величины коэффициентов a_n, $a_{n-1},...,a_0$ и результаты вычислений y_j(j =1,2,3,4,5).

Другой вариант этой программы:

начало
для j : = 1, 2, 3, 4, 5 **цикл**
начало
y [j] : = 0;
для i : = n, i — 1 **пока** i ≥ 0 **цикл**
y [j] : = y [j] × x[j] + a[i];
конец
конец

В АЛГОЛе допускается возможность изменения в процессе выполнения оператора S, входящего в цикл, величин, входящих в заголовок цикла (шага цикла, конечного значения параметра цикла, переменных, образующих логическое выражение). Это свойство оператора цикла АЛГОЛа называется динамической интерпретацией оператора цикла.

Например:

начало
h : =10; i : = 1;
для v : = 0 **шаг** h **до** 100 **цикл**
начало
h : = h + 10; v : = v × 2 + h;
y [i] : = h × v
i: =i + 1;
конец
конец

В этом примере параметр цикла v меняется не только по правилу арифметической прогрессии, определяемому заголовком цикла, но и самим оператором, выполняемым в цикле так же, как и шаг h. В результате получим всего два значения величины y:

y [1] = 400, y [2] = 3300.

При третьем заходе (i = 3) параметр v уже примет значение 110 + 30 = 140, т. е. будет больше 100, и третий раз цикл выполняться не будет.

Рассмотрим примеры выхода из цикла.

Пусть требуется для заданной последовательности положительных чисел $a_1, a_2, \ldots, a_n$ определить номер первого числа a_i, находящегося в заданных пределах $p \le a_i \le q$. Это может быть сделано с помощью следующего оператора цикла:

для i : = 1, i + 1 **пока** i ≤ n **цикл**
если ⌉ (a [i] ≤ q /\ a [i]≥) **то иначе на** M;

В этом примере возможны два случая выхода из цикла:

1) при исчерпании всех элементов ряда чисел, т. е. при i > n. После выхода из цикла следующим будет выполняться оператор, стоящий за этим оператором цикла, значение параметра цикла i будет неопределенным, а искомый член ряда не будет найден (так как ни один член ряда не удовлетворяет заданному условию $p \le a_1 \le q$).

При этом в цикле производится проверка условия ⌉ (a [i] ≤ q /\ a [i] ≥p) для каждого члена ряда a **[i];** так как это условие выполняется, то каждый раз выполняется пустой оператор, стоящий за символом то;

2) второй случай выхода будет иметь место при обнаружении члена ряда a [i], удовлетворяющего заданному требованию. При этом будет выполнен оператор перехода, стоящий за символом **иначе.** Метка M указывает некоторый оператор, который должен выполняться в случае обнаружения члена ряда, удовлетворяющего заданному требованию. Такой выход из цикла будет являться «досрочным» выходом, даже если он произошел после проверки последнего члена ряда (i = n), так как условие окончания цикла, требуемое заголовком цикла (i > n), еще не будет соблюдено. При этом параметр цикла i сохранит свое последнее значение, т. е. он укажет номер члена ряда, удовлетворяющего заданному условию.

Оператор процедуры. В параграфе, посвященном величинам языка АЛГОЛ-60, мы рассматривали процедуры-функции: там было дано их синтаксическое определение и указаны их отличия от операторов процедур.

Основное отличие заключается в характере использования: оператор процедуры всегда фигурирует в программе в виде отдельного оператора (т. е. он может иметь метки, после него ставится точка с

запятой), а процедура-функция — в виде так называемого указателя функции, который используется при построении выражений. Описания процедур-функций и операторов процедур и обращения к ним строятся по одним и тем же правилам, которые будут рассмотрены ниже.

Отличие сводится к тому, что в описании процедуры функции указывается тип величины, который должен иметь результат вычисления функции.

Синтаксическое определение оператора процедуры имеет следующий вид:

⟨фактический параметр ⟩ : : = ⟨строка ⟩ | ⟨выражение ⟩ | ⟨ идентификатор массива ⟩ | ⟨идентификатор переключателя ⟩ | ⟨идентификатор процедуры⟩

⟨строка букв ⟩ : : = ⟨буква ⟩ | ⟨строка букв ⟩ ⟨буква ⟩ ⟨ограничитель параметра⟩ : : = , |) ⟨строка букв⟩ : (

⟨список фактических параметров⟩ : : = ⟨фактический параметр⟩ | ⟨список фактических параметров⟩ ⟨ограничитель параметра⟩ ⟨фактический параметр⟩

⟨совокупность фактических параметров⟩ : : = ⟨пусто ⟩ | (⟨ список фактических параметров⟩)

⟨оператор процедуры⟩ : : = ⟨идентификатор процедуры ⟩ ⟨ совокупность фактических параметров⟩

Обращение к процедуре либо с помощью указателя функции, либо с помощью оператора процедуры задает конкретные исходные данные в виде так называемых фактических параметров для выполнения стандартного участка алгоритма, определенного описанием процедуры. Эти фактические параметры представляются перечнем величин в круглых скобках после идентификатора процедуры. Один фактический параметр от другого отделяется ограничителем параметра либо в виде запятой, либо в виде последовательности символов, состоящей из круглой закрывающей скобки, буквенного пояснения с двоеточием и открывающей круглой скобки.

Фактическими параметрами могут быть числа и переменные, которые являются частными случаями выражений. Следует заметить, что идентификаторы массивов, процедур и переключателей, являющиеся фактическими параметрами, указываются в списке фактических параметров без сопровождающих их дополнительных выражений (индексов, именующих выражений или параметров). Фактические параметры должны быть описаны в том блоке, в котором находится оператор процедуры или указатель функции, или во внешнем блоке (см. ниже). При их включении в состав фактических параметров оператора некоторой процедуры использование соответствующих величин внутри тела процедуры производится в соответствии с этими описаниями. При этом, например, использование в качестве фактического параметра идентификатора некоторой процедуры приводит к тому, что внутри первой процедуры будет использоваться вторая процедура и, естественно, что для этого в первой процедуре должна быть предусмотрена подготовка всех фактических параметров для второй процедуры. Введение в состав языка такого средства, как процедуры, сильно усложняет язык, но в то же время придает ему большие возможности для представления различных процессов обработки информации и, главным образом, обеспечивает возможность использования предшествующего опыта при программировании новых задач. Если накоплен достаточно большой запас различных процедур, обеспечивающих выполнение типовых вычислительных процессов, то программирование новых задач будет в значительной степени сводиться к компоновке подходящих процедур в общую программу с добавлением недостающих участков. При этом сокращается время программирования и отладки программ.

Заметим, что более полное понимание процедур и особенностей их использования можно получить после ознакомления с правилами построения описаний процедур, которые изложены в следующем параграфе.

9. Описания

Описания являются важной неотъемлемой частью каждой программы; они сообщают транслятору (переводящей программе) необходимые сведения для построения машинной программы: для распределения памяти, выполнения округлений и др. Описания служат для того, чтобы определить некоторые свойства величин, испрльзуемых в программе, и связать эти величины с идентификаторами. Описания следует отличать от примечаний, которые могут появляться в АЛГОЛ-программах. Примечания служат только для пояснения отдельных мест программы человеку, читающему эту программу. Описания же воспринимаются транслятором, и поэтому они должны быть написаны в строгом соответствии с синтаксисом АЛГОЛа. Описания ставятся в начале блока, к которому они относятся, и оформляются в виде предложений, заканчивающихся точкой с запятой. Описания имеют силу (т. е.

область действия) только внутри этого блока. Расположение различных описаний в начале блока может быть произвольным.

Описывать следует все идентификаторы, кроме меток и идентификаторов стандартных функций. Один и тот же идентификатор не может быть описан в одном блоке больше одного раза. В АЛГОЛе существует четыре вида описаний — описания типа, массивов, переключателей и процедур. Описания переключателей были приведены нами раньше при рассмотрении именующих выражений. Рассмотрим описания типа и массивов, а затем описания процедур.

Описания типа. Описания типа служат для указания типа простых переменных и переменных, являющихся элементами массивов. В описания простых переменных могут входить только описания типа, в описания массивов входят, кроме того, сведения о размерности массива и пределах изменения индексов переменных. В АЛГОЛе предусматривается три типа переменных: **целый, вещественный, логический.** Тип переменной определяет характер операций, которые могут выполняться над этой переменной (например, с округлением или без).

Каждый из этих типов переменных может быть локальным или собственным. Часто используют еще термин глобальные переменные как противоположность локальным, но этот термин определяет не тип переменных, а только степень их локальности. Переменные, являющиеся локальными в одном блоке, могут быть глобальными для другого блока. Подробнее этот вопрос будет рассмотрен несколько позже. Приведем синтаксическое определение описания типа:

⟨список типа ⟩ : : = ⟨простая переменная⟩ | ⟨простая переменная⟩ , ⟨список типа⟩

⟨тип ⟩ : : = **целый | вещественный | логический**

⟨локальный или собственный тип⟩ : : = ⟨тип⟩ **собственный** ⟨тип⟩

⟨описание типа⟩ : : = ⟨локальный или собственный тип⟩ ⟨список типа⟩

Если переменная, имеющая по описанию тип **целый,** получает в процессе вычислений нецелое значение, то при выполнении присваивания будет произведено ее округление до целого значения по правилу

a : = entier (A + 0,5);

где A — фактическое значение переменной, а функция entier — наибольшее целое, не превышающее значения аргумента, стоящего в круглых скобках.

Примеры описаний типа:

целый i, j, k; вещественный x, y;

логический ОМЕГА, ПРИЗНАК СРОЧНОСТИ;

Переменные, имеющие вещественный тип, могут принимать любые вещественные значения (в пределах разрядной сетки конкретной машины), а переменные логического типа могут быть представлены в машине одноразрядными двоичными кодами.

Описания массивов. Для описания массивов вводится вспомогательное понятие — сегмент массива. Сегмент массива — это группа однотипных массивов, отличающихся только своими идентификаторами; эти массивы должны иметь одинаковый локальный или собственный тип переменных, одинаковую размерность и одинаковые пределы изменения индексов.

Приведем синтаксическое определение описания массивов:

⟨нижняя граница⟩ : : = ⟨арифметическое выражение⟩

⟨верхняя граница⟩ : : = ⟨арифметическое выражение⟩

⟨граничная пара⟩ : : = ⟨нижняя граница⟩ : ⟨верхняя граница⟩

⟨список граничных пар⟩: : = ⟨граничная пара⟩ | ⟨список граничных пар⟩, ⟨граничная пара⟩

⟨сегмент массива⟩ : : = ⟨идентификатор массива⟩ [⟨список граничных пар⟩] | ⟨идентификатор массива⟩, ⟨сегмент массива⟩

⟨список массивов⟩ : : = ⟨сегмент массива⟩ | ⟨список массивов⟩, ⟨сегмент массива⟩

⟨описание массива⟩ : : = **массив** ⟨список массивов⟩ | ⟨локальный или собственный тип⟩ **массив** ⟨список массивов⟩

Размерность массива определяется числом граничных пар в списке.

Пример сегмента массива:

массив A, B, C, D [al: b1, a2 : b2, a3 : b3]

A, B, C, D — идентификаторы трехмерных массивов (в списке граничных пар имеется три пары);

al, a2, a3 — нижние границы;

b1, b2, b3 — верхние границы.

Таким образом, в одно описание массивов может входить несколько сегментов, т. е. несколько групп массивов, каждая из которых имеет одинаковые списки граничных пар. Если в блоке используется несколько массивов различных типов, то должны записываться соответственно несколько описаний массивов. Описание массива может не содержать описателя типа; в этом случае считается, что массив имеет вещественный тип. Например:

массив A, B, C **[**1 : a], X [b : c];

Здесь указаны два сегмента массивов, в первом имеется три массива A, B, C, во втором — один массив X.

Все массивы одномерные и имеют тип вещественный.

Примеры описаний массивов с явным указанием типов:

собственный целый массив Z, X [0 : 100, a : b], У [1 : 50];

вещественный массив A [0 **: если** i ≤ n **то** p **иначе** 2 × q];

Арифметические выражения, представляющие собой границы изменения индексов, по своей природе должны быть целыми. Индексы переменных могут принимать все целые значения, начиная с нижней границы и кончая верхней. Если при вычислении границ получаются нецелые числа, то они округляться до ближайшего целого значения. Все переменные или функции, входящие в арифметические выражения, представляющие границы индексов, должны быть описаны в одном из внешних блоков, в который входит блок, содержащий данное описание массивов. При каждом входе в этот внутренний блок заново происходит определение границ массивов, и эти границы остаются неизменными в процессе выполнения данного внутреннего блока. При этом может быть случай, когда границы зависят от переменных, описанных во внешнем блоке, но являющихся глобальными для внутреннего блока (т. е. они используются и во внутреннем блоке). При работе внутреннего блока эти переменные, от которых зависят границы массивов, могут менять свое значение, но при этом границы массивов будут сохранять то значение, которое они имели при входе в этот внутренний блок.

Отсюда следует, что в самом внешнем блоке, представляющем собой всю программу, могут содержаться описания массивов, имеющих только постоянные (числовые) границы.

Нижняя граница всегда должна быть не более соответствующей верхней границы, а значения индексов переменных не должны выходить за пределы границ. При невыполнении этих условий значение переменной с индексами считается неопределенным.

Области действий описаний в программах. Основой АЛГОЛа является блочный принцип построения программ, при котором отдельные ее участки, так называемые блоки, могут строиться одновременно и независимо разными лицами, что очень удобно при работе над сложными задачами. Это достигается строгим определением областей действий различных описаний или, как говорят, локализацией описаний. Все переменные, не имеющие в своем описании символа **собственный,** считаются локальными по отношению к тому блоку, в котором они описаны. Следует подчеркнуть, что все переменные, используемые в данном блоке, должны быть обязательно описаны в нем самом или в каком-нибудь внешнем блоке. Требуется также обязательно описывать и другие объекты, используемые в блоке (процедуры, переключатели), кроме стандартных функций (sin, cos, arctan, ln, exp, abs, sign, sgrt, entier). Смысл локализации переменных заключается в том, что такие переменные могут использоваться только в пределах того блока, в котором они описаны (за исключением, может быть, некоторых внутренних блоков, входящих в данный блок, о чем подробнее будет сказано ниже). При выходе из данного блока эти переменные теряют свое значение, и те же идентификаторы переменных (простых или с индексами) могут быть использованы для других целей (обозначения других переменных, меток, процедур, переключателей). Если же в своем описании некоторые переменные имеют символ **собственный,** то они также могут быть использованы только в пределах своего блока, но при выходе из данного блока сохраняют свое значение. При повторном обращении к этому блоку собственные переменные принимают то значение, которое они имели при выходе из него. В отличие от этого локальные переменные при первоначальном или повторном входе в блок не имеют никаких значений, и перед тем, как их использовать в данном блоке, им нужно присваивать определенные значения.

Отдельно нужно остановиться на том случае, когда в качестве собственных описаны массивы с переменными границами. Эти границы, как уже говорилось, могут зависеть только от переменных, описанных в некотором внешнем блоке по отношению к тому блоку, в котором описаны данные

массивы. При каждом новом входе в блок границы собственных массивов могут изменяться операторами внешнего блока, поэтому значения элементов таких массивов сохраняются только в пределах совпадения границ массивов, которые они имели при последнем выходе из блока, с границами массивов, получающимися при данном входе в блок. Если массив расширит свои границы, то новые его области будут неопределенными; если массив уменьшит свои границы, то элементы отпавших областей будут утрачены, т. е. их нельзя использовать в вычислениях.

Рассмотрим теперь понятие глобальности переменных. Если необходимо, чтобы некоторые переменные сохраняли свое значение в пределах некоторого блока, включая и все внутренние блоки, входящие в его **состав,** то необходимо такие переменные описать в самом внешнем блоке и не описывать их во внутренних блоках. Такие переменные будут называться глобальными по отношению к внутренним блокам, ими можно пользоваться одинаково в пределах всего внешнего блока, включая и те внутренние блоки, в которых эти переменные не описаны. Как уже говорилось, переменным, описанным в данном блоке и не являющимся собственными перед их использованием в данном блоке для вычислений необходимо произвести присваивание определенных значений, так как при входе в этот блок их значения являются неопределенными.

Особенностью использования собственных переменных является то, что в том блоке, где они описаны, должны быть операторы, присваивающие им первоначальное значение при первом входе в данный блок. Эти операторы не должны выполняться при повторных входах в данный блок. При повторных входах собственные переменные будут сохранять то значение, которое они имели в момент предыдущего выхода из этого блока.

Как мы уже говорили, сохранить значения некоторых переменных, используемых во внутреннем блоке, при выходе из этого блока можно, описав эти переменные во внешнем блоке и не описывая во внутреннем. Однако между этим способом и способом, основанным на использовании описателя **собственный,** имеется следующее различие. При первом способе указанные переменные будут доступными для всех, входящих в состав внешнего блока внутренних блоков, в которых эти переменные не описаны; при втором способе эти переменные будут недоступны никому, кроме операторов своего блока, даже если те же идентификаторы будут использованы в других блоках и описаны во внешнем блоке. Таким образом, можно обойтись без собственных переменных, но это потребует некоторого согласования использования общих переменных. Если же какой-то соседний блок (т. е. блок, входящий в состав того же внешнего блока) будет иметь описанными те же переменные, которые описаны во внешнем блоке, то внешние значения этих переменных будут недоступны операторам внутреннего блока, а внутренние переменные будут недоступны операторам внешнего блока.

Таким образом, повторное описание одних и тех же переменных во внутреннем и внешнем блоках устанавливает полное различие между внешним и внутренним использованием этих переменных, так как если бы это были совершенно разные переменные.

Локализация меток. Метки в АЛГОЛ-программах специально не описываются, а вводятся в действие при появлении. Областью действия метки является та совокупность операторов перехода, которые могут вести к оператору с данной меткой. Считается, что областью действия метки является тот внешний блок, в котором такая метка является единственной. Если одна и та же метка используется и во внутреннем и во внешнем блоках, то операторы перехода, находящиеся во внутреннем блоке, будут вести к внутренней метке, а во внешнем блоке — к внешней; при этом невозможно будет осуществить переход к оператору во внешнем блоке, имеющем эту метку, из внутренних блоков, имеющих внутри себя оператор с такой же меткой. Как уже говорилось, в АЛГОЛе не допускаются переходы извне к отдельным операторам, входящим в состав блока. Переходы могут быть только в начало блока. Выходы же из внутренних блоков к другим операторам, входящим в состав одного и того же внешнего оператора, допустимы, если при этом выполняются указанные выше правила локализации меток.

Описания процедур. ***Структура описания процедур.*** Каждая процедура, используемая в блоке, должна иметь в начале этого блока или какого-нибудь внешнего блока, включающего данный блок, описание, которое определяет порядок входа в процедуру и действия, выполняемые процедурой. Описания процедур являются единственным видом описаний, в которых используются операторы, образующие так называемое тело процедуры.

Синтаксис описания процедуры имеет вид:

⟨формальный параметр⟩: : = ⟨идентификатор⟩

⟨список формальных параметров⟩ : : = ⟨формальный параметр⟩ | ⟨список формальных параметров⟩ ⟨ограничитель параметра⟩ ⟨формальный параметр⟩

⟨совокупность формальных параметров⟩ : : = ⟨пусто⟩ | (⟨список формальных параметров⟩)

⟨список идентификаторов⟩ : : = ⟨идентификатор⟩ | ⟨список идентификаторов⟩, ⟨идентификатор⟩

⟨список значений⟩: : = **значение** ⟨список идентификаторов⟩; | ⟨пусто⟩

⟨спецификация⟩ : : = **строка |** ⟨тип⟩ **| массив |** ⟨тип⟩ **массив | метка | переключатель | процедура |** ⟨тип⟩ **процедура**

⟨совокупность спецификаций⟩ : : = ⟨пусто⟩ | ⟨спецификация ⟩ ⟨список идентификаторов⟩; | ⟨совокупность спецификаций⟩ ⟨спецификация⟩ ⟨список идентификаторов⟩;

⟨заголовок процедуры⟩ : : = ⟨идентификатор процедуры⟩ ⟨совокупность формальных параметров⟩; ⟨список значений⟩ ⟨совокупность спецификаций⟩

⟨тело процедуры⟩ : : = ⟨оператор⟩ | ⟨код⟩

⟨описание процедуры⟩: : = **процедура** ⟨заголовок процедуры ⟩ ⟨тело процедуры⟩ | ⟨тип⟩ **процедура** ⟨заголовок процедуры⟩ ⟨тело процедуры⟩

Операторы, входящие в состав тела процедуры, указывают те действия, которые должны выполняться процедурой.

В АЛГОЛе предусматриваются две возможности представления тела процедуры: запись необходимых операторов на языке АЛГОЛ или представление тела процедуры на каком-то другом языке, например в виде машинной программы для конкретной машины. Последнее представление называется представлением процедуры в виде кода. Таким образом, понятие «код» является не определяемой в АЛГОЛе синтаксической единицей.

Типичными примерами процедур, задаваемых сразу в машинном коде, являются процедуры ввода и вывода данных, обмена между запоминающими устройствами. Определения таких процедур-кодов будут даны ниже при изложении АЛГЭМа.

Список значений и совокупность спецификаций не являются обязательными частями всех заголовков процедур. Для многих процедур они могут отсутствовать.

В качестве формальных параметров могут фигурировать только идентификаторы, а в качестве фактических параметров, указываемых в обращениях к процедурам, могут, использоваться также выражения и строки.

Количество фактических параметров в обращении к процедуре должно быть равно числу формальных параметров в заголовке процедуры. Соответствие формальных и фактических параметров устанавливается путем их перечисления слева направо.

Формальные параметры, перечисленные в заголовке, являются обозначениями переменных или других объектов (переключателей или процедур), которые используются в операторах тела процедуры. При обращении к процедуре конкретный смысл каждого формального параметра задается соответствующим ему фактическим параметром. При этом возможны два способа конкретизации формальных параметров:

1) каждому формальному параметру присваивается численное значение соответствующего ему фактического параметра, которое вычислено к моменту обращения к процедуре. Этот способ называется конкретизацией формальных параметров значением (вызов по значению);

2) каждый формальный параметр заменяется в теле процедуры соответствующим ему фактическим параметром. При этом (так как фактические параметры могут быть сложными выражениями) там, где возможно и необходимо, фактические параметры заключаются в скобки. Такой способ называется конкретизацией формальных параметров наименованием (вызов по наименованию).

Для одной и той же процедуры возможно применение сразу обоих способов, при этом часть формальных параметров конкретизируется значением, а часть — наименованием. Для указания того, какие формальные параметры должны конкретизироваться значением, в заголовке процедуры пишется спецификатор значение, после которого перечисляются соответствующие формальные параметры. Этот перечень называется списком значений:

⟨список значений⟩: := **значение** ⟨ список идентификаторов⟩; | ⟨пусто⟩

Для всех формальных параметров, перечисленных в списке значений, обязательно должны быть заданы спецификации, т. е. указаны виды и типы объектов, которые представляются идентификаторами. Эти спецификации образуют четвертую составную часть заголовка процедуры,

называемую совокупностью спецификаций.

Так как фактическими параметрами наряду со строками, выражениями, идентификаторами массивов и идентификаторами переключателей могут быть и идентификаторы процедур, то в составе спецификаций формальных параметров предусматривается и такой вид спецификации ⟨тип⟩ **процедура.**

Последний вид спецификаций используется для процедур-функций. Кроме обязательного указания для формальных параметров, перечисленных в списке значений, спецификации могут указываться и для других формальных параметров, не перечисленных в списке значений. Это бывает необходимо для упрощения автоматического программирования процедуры транслятором. Ясно, что программирование процедур должно быть сделано независимо от вида конкретного обращения к ней. Вообще говоря, обращение к процедурам содержит дополнительную информацию о характере объектов, с которыми должна работать процедура. Такой информацией являются описания, относящиеся к фактическим параметрам, указанным в обращениях. Эти описания находятся либо в том блоке, из которого производится обращение к процедуре, либо в одном из внешних блоков. Так как процедуры рассчитываются на использование их в разных местах программы (в разных блоках), то в своем описании они должны содержать сведения, накладывающие необходимые ограничения на характер обращений к ним и обеспечивающие возможность программирования процедуры независимо от остальных частей программы. Для этой цели и служат спецификации, которые накладывают необходимые ограничения на фактические параметры и тем самым обеспечивают общность в построении и использовании данной процедуры.

Локализация величин в процедурах. Важной функцией списка значений является локализация перечисленных в нем величин в теле процедуры. При этом процедуру следует рассматривать как блок, хотя внешне она не оформляется в виде блока. Все формальные параметры, перечисленные в списке значений, являются локальными по отношению к телу данной процедуры. Это значит, что при входе в процедуру они получают определенные значения. В процессе выполнения процедуры эти величины могут изменяться операторами, входящими в тело процедуры, но соответствующие им величины, находящиеся вне процедуры, будут оставаться неизменными, т. е. будут сохранять те значения, которые они имели в момент входа в процедуру. При выходе из процедуры значения локальных объектов будут теряться, поэтому результаты работы процедуры нельзя перечислять в списке значений.

Формальные параметры, конкретизируемые наименованием, не являются локальными по отношению к данной процедуре. Если при выполнении процедуры соответствующие им фактические параметры изменяют свое значение, то изменения распространяются на весь блок, в котором описаны эти фактические параметры. При выходе из процедуры такие объекты сохраняют то значение, которое они получили в процедуре. Кроме объектов, перечисленных в списке формальных параметров, в теле процедуры могут использоваться различные объекты без указания их в списке формальных параметров и, естественно, без спецификации. Такими объектами могут быть простые переменные, массивы, переключатели, процедуры. Все эти объекты можно разделить на две группы: 1) объекты, описанные в блоке, являющемся телом процедуры (здесь мы уже имеем в виду не воображаемый, а явный блок, оформленный по всем правилам АЛГОЛа; такие объекты, естественно, будут локальными для этого блока); 2) объекты, не описанные в блоке, являющемся телом процедуры, и не являющиеся формальными параметрами. Такие глобальные по отношению к процедуре объекты должны быть обязательно описаны в каком-нибудь внешнем блоке, включающем в себя блок, содержащий описание данной процедуры, или в самом блоке, содержащем описание процедуры, т. е. область действия описаний таких объектов должна обязательно распространяться на блок, содержащий описание данной процедуры.

Отсюда вытекает, что все описания типа, массивов, переключателей и процедур, содержащиеся в начале данного блока, могут использоваться любой процедурой, описанной в этом блоке.

Заметим, что область действия меток, стоящих перед операторами, входящими в тело процедуры, или перед всем телом процедуры, ограничена телом процедуры, т. е. к этим операторам невозможны переходы извне тела процедуры.

Рассмотрим пример процедуры с локализацией переменных, перечисленных в списке значений:

начало вещественные m, x, y, z, p, q;
вещественная процедура П (a, z, Q);
значение a, z; **вещественное** a, z, Q;

П : = (a + y ↑ m) × Q/(z + y);

. . . . . .
m : = 3; p : = 100; q : = 10;
x : = 5; y : =15; z : = 20;
. . . . . .
R : = П (x, p, q) + z ↑ m;
. . . . . .

конец

Заметим, что величина R должна быть описана в каком-нибудь внешнем блоке. Выполнение последнего оператора присваивания, использующего в правой части процедуру-функцию П, будет эквивалентно выполнению следующего составного оператора:

начало начало вещественное a, z;
a : = 5; z : = 100; П : = (a × y ↑m) × q/(z + y)
конец; R : =П +z ↑ m **конец**

В теле процедуры П используются переменные a, z, Q, являющиеся формальными параметрами, и переменные m и y, являющиеся глобальными переменными по отношению к этой процедуре. Кроме того, в блоке, содержащем эту процедуру, описана переменная z, которая совпадает по обозначению с одним из формальных параметров. Два формальных параметра a и z конкретизируются значением, а один Q конкретизируется наименованием.

При выходе из процедуры переменная z получит прежнее значение, а переменная a будет иметь неопределенное значение. Переменная z, используемая в теле процедуры, будет иметь значение фактического параметра p =100, а переменная z, используемая в операторе присваивания R : = . . ., будет иметь то значение, которое ей было присвоено перед выполнением данного оператора присваивания, т. e. z = 20.

В соответствии с правилами локализации действий описаний в блоках все описания, помещенные во внутренних блоках, не оказывают влияния на внешние блоки, и поэтому нельзя обратиться из внешнего блока к процедуре, описанной во внутреннем блоке. Наоборот, из внутренних блоков в общем случае возможны обращения к процедурам, описанным во внешних блоках. При этом если в процедуре используются глобальные величины, не являющиеся формальными параметрами, но совпадающие по виду с величинами, описанными в том внутреннем блоке, из которого совершается обращение к процедуре, то при выполнении процедуры эти величины принимают те значения и смысл, которые они имели во внешнем блоке, содержащем описание процедуры.

Следует подчеркнуть, что это правило относится только к нелокальным для процедуры величинам, не являющимся формальными параметрами.

Что же касается формальных параметров, то во всех случаях обращения к процедурам, в том числе из внутренних блоков, конкретизация формальных параметров фактическими происходит с использованием описаний того блока, из которого произведено обращение, или тех описаний внешних блоков, действие которых распространяется на блок, содержащий обращение к процедуре. Таким образом, обращение к процедуре как бы несет с собой всю необходимую информацию о параметрах для работы процедуры, взятую из того блока, где помещается это обращение.

Рассмотрим пример некоторой процедуры, описанной во внешнем блоке и используемой во внутреннем блоке:

A : **начало вещественные** b, x, y, z;
вещественная процедура p (a);
значение a; **вещественное** a;
начало x:=a×x; p : = x ↑ 2 + y **конец;**
. . . . . .
x : = 15; b : = 50; y : = 10;
. . . . . .
B : начало вещественные x, b;
. . . . . .
x : = 100; b : =10;
. . . . . .
z : = p (b) + y;
. . . . . .

конец. . . . . **конец**

Выполнение процедуры будет эквивалентно выполнению блока:

начало вещественное a;

a : = 10; x : = a × 15; p : =x ↑ 2 + 10

конец

При выполнении операторов тела процедуры использовано значение величины b = 10 (являющейся фактическим параметром), полученное во внутреннем блоке, имеющем метку B. Такая же величина b, описанная во внешнем блоке, имеет значение b = 50. В то же время глобальная переменная x в теле процедуры используется со значением x = 15, полученным во внешнем блоке, хотя такая же переменная x, описанная во внутреннем блоке, имеет значение 100.

Порядок выполнения процедур. При реализации в программе какого-нибудь обращения к процедуре, описанной на АЛГОЛе, с помощью оператора процедуры или указателя функции выполняются следующие действия:

а) формальные параметры, перечисленные в списке значений, конкретизируются значением, т. е. им присваиваются значения соответствующих фактических параметров;

б) остальные формальные параметры, не перечисленные в списке значений, конкретизируются наименованием, т.е. везде в теле процедуры вместо них подставляются выражения или идентификаторы соответствующих фактических параметров;

в) выполняется оператор, составляющий тело процедуры, так как если бы он находился в блоке, в котором находится описание данной процедуры, и к нему был бы совершен переход при помощи оператора перехода из того места, где находится оператор процедуры или указатель функции;

г) если в теле процедуры не содержится оператор перехода, выводящий из этой процедуры в какое-то другое место программы, то после окончания процедуры выполняются очередные действия программы, т. е. операторы, следующие заданным оператором процедуры или указателем функции.

Роль спецификаций и списка значений. Спецификации в описании процедуры накладывают определенные ограничения на фактические параметры. Кроме того, сам характер использования различных переменных и других объектов языка в теле процедуры также предъявляет определенные требования к фактическим параметрам, которые могут задаваться для данной процедуры. Обе группы ограничений играют в общем одну и ту же роль. Можно сказать, что ограничения, связанные с характером использования объектов в теле процедуры и характером обращений, являются более полными, чем ограничения, задаваемые в явном виде спецификациями, так как спецификации не охватывают всех возможных случаев использования объектов. Поэтому всегда при составлении или использовании процедур нужно иметь в виду содержательную сторону дела и, используя общие принципы локализации объектов и конкретизации параметров, не допускать непредусмотренных вариантов работы процедур.

Однако применение спецификаций бывает полезно и необходимо с точки зрения упрощения работы транслятора, так как спецификации в явном виде указывают типы и виды объектов, допустимых в качестве фактических параметров в процедуре. К числу основных ограничений, задаваемых спецификациями, относятся:

1) Указание типа (вещественный, целый, логический) определяет, что для этого формального параметра фактический параметр должен быть выражением указанного типа. При конкретизации значением должно производиться (в случае различия в типах) преобразование типа к тому, который задан спецификациями.

2) Формальному параметру, конкретизируемому наименованием и используемому в теле процедуры в качестве левой части оператора присваивания, может соответствовать в качестве фактического параметра только переменная (а не более сложное выражение, так как операторы вида ⟨выражение⟩: = ⟨выражение⟩ в АЛГОЛе недопустимы).

3) Формальным параметрам, специфицированным как массивы, должны соответствовать фактические параметры-массивы того же типа, что и формальные параметры. При конкретизации значением также может происходить преобразование типов (целый в вещественный или наоборот). Если этот массив описан в блоке тела процедуры и имеет фиксированную размерность, то соответствующий фактический параметр-массив должен иметь ту же размерность. При конкретизации значением формальный параметр-массив принимает значения элементов и границ индексов соответствующего фактического параметра-массива.

4) Формальному параметру, имеющему спецификацию **метка** или используемому в качестве метки в теле процедуры, должно соответствовать в качестве фактического параметра именующее выражение.

5) Спецификация **строка** в АЛГОЛе указывает, что данный формальный параметр может иметь фактическим параметром строку, по правилам АЛГОЛа такие формальные параметры могут использоваться только в процедурах-кодах, т. е. написанных не на языке АЛГОЛ.

Рассмотрим роль списка значений. Указание списка значений позволяет использовать в процедуре в качестве исходных данных значения любых переменных или выражений, имеющихся в основной программе, сохраняя их в то же время в основной программе неизменными. Кроме того, используя конкретизацию значением, можно в качестве фактических параметров применять любые выражения, в том числе и для формальных параметров, фигурирующих в теле процедуры в качестве левых частей операторов (хотя последнее вряд ли имеет какой-либо смысл). Ясно, что конкретизация значением неприменима к формальным параметрам, играющим роль идентификаторов процедур и переключателей. Формальные параметры, играющие роль именующих выражений, могут конкретизироваться как наименованием, так и значением, причем эти параметры могут использоваться для указания переходов из тела процедуры в основную программу, и в этом случае формальный параметр, конкретизированный значением, будет иметь смысл вне тела процедуры, т. е. в отличие от общего правила не будет локализован в теле процедуры.

Пример процедуры. Определение суммы элементов, находящихся на главной диагонали квадратной матрицы (вычисление следа матрицы).

Описание процедуры:

процедура СЛЕД (a, n, S); **значение** n;
массив a; **вещественное** S; **целое** n;
начало целый к; S : = 0;
для k : = 1 **шаг** 1 **до** n **цикл** S : = S + a [k, k[
конец

Оператор процедуры для обращения к этой процедуре может иметь, например, вид, показанный в следующем фрагменте программы:

начало вещественное z; **массив** A [1 : 20,1 : 20]; **целый** b;
b : =20, СЛЕД (A, b, z); . . . **конец**

Выполнение оператора процедуры СЛЕД (A, b, z); может быть представлено в виде блока:

начало целое n;
n : = b;
начало целое к;
z : =0;
для k : = 1 **шаг** 1 **до** n **цикл**

z : = z + A [k, k]

конец
конец

Переменная k локализована описанием в теле процедуры. Переменная n локализована указанием в списке значений. Формальные параметры a и S конкретизируются наименованием, и результат процедуры сохраняется в виде значения переменной z.

В заключение настоящего параграфа, посвященного описаниям процедур, следует сделать следующие замечания.

Во-первых, при выполнении процедур, использующих нелокальные переменные, может происходить иногда присваивание новых значений таким переменным, даже если они не являются результатами процедуры. Это явление носит название побочного эффекта процедуры и его нужно учитывать при пользовании процедурами.

Во-вторых, правилами АЛГОЛа допускается такое использование процедур, когда в процессе выполнения одной процедуры (до ее окончания) происходит вновь обращение к этой же процедуре либо непосредственно, либо через какую-нибудь другую процедуру. Такие многократно вызываемые процедуры называются рекурсивными. Их применение позволяет в ряде случаев весьма компактно записывать алгоритмы, носящие рекурсивный характер.

Глава 3

СИСТЕМА АВТОМАТИЗАЦИИ ПРОГРАММИРОВАНИЯ АЛГЭМ

10. Алгоритмический язык АЛГЭМ-1

В настоящей главе приводятся добавления к АЛГОЛу и некоторые изменения имеющихся в АЛГОЛе средств, составляющие вместе с АЛГОЛом содержание алгоритмического языка АЛГЭМ-1, предназначенного для программирования экономических и математических задач. Слово АЛГЭМ образовано от сокращения и объединения трех слов АЛГоритмы Экономические и Математические. Индекс 1 указывает на первый практически реализованный вариант этого языка в качестве входного языка в системе автоматизации программирования АЛГЭМ-СТ-3 для машины «Минск-22». СТ-3 в названии упомянутой системы показывает, что в нее, кроме алгоритмического языка АЛГЭМ, входит еще синтаксический транслятор в его третьей (усовершенствованной) модификации.

Заметим, что до появления в 1969 г. нового расширенного варианта языка АЛГЭМ, так называемого языка АЛГЭМ-2 (который также будет рассмотрен в дальнейшем), описываемый в настоящей главе язык АЛГЭМ-1 был известен просто как АЛГЭМ.

Язык АЛГЭМ-1 может быть использован не только при решении экономических и математических задач, но и при решении различных других информационно-логических задач, связанных с обработкой больших массивов информации, имеющей сложную, но полностью детерминированную структуру (обработка историй болезней в клиниках, анкет переписей, результатов экспериментов и т. д.).

Описываемые ниже добавления и изменения языка АЛГОЛ связаны со спецификой решения экономических задач, которые характеризуются использованием в качестве исходной и выходной информации сложных структур данных (отчетов, заявок, нормативов, планов и т. п.), большим удельным весом операций по вводу исходных данных и выдаче на печать готовых форм документов, выработанных с помощью машины, большим удельным весом операций с накопителями информации на магнитных лентах, а также необходимостью воспринимать, обрабатывать и выдавать из из машины не только числовую, но и текстовую информацию и осуществлять проверки значений различных признаков и соотношений между числовыми и текстовыми (строчными) величинами.

В АЛГЭМ-1 по сравнению с АЛГОЛом добавляются следующие средства:

1) Строчные переменные и выражения. Они позволяют обрабатывать текстовую информацию и описывать процессы выдачи различных документов на АЦПУ-128 (алфавитно-цифровое печатающее устройство со 128 символами в строке; используется в составе машины «Минск-22»).

Строчные переменные в языке АЛГЭМ-1 применяются в качестве фактических параметров в операторе процедуры-кода и в операторе присваивания, как в левой, так и в правой его частях. Предусматривается возможность преобразования значений типа **строчный** в значения типа **целый** или **вещественный** и обратного преобразования.

2) Указание формата для переменных типа **целый, вещественный,** и **строчный.** Форматы определяют состав символов в значениях этих величин и тем самым позволяют редактировать их при выдаче на печать, а также осуществлять более плотное их размещение (упаковку) в ячейках памяти. Указание формата в предшествовавших описаниях языка АЛГЭМ [8, 18] называлось указанием вида.

3) Составные величины, включающие в себя составные переменные и составные массивы. Описания составных величин служат для задания структуры различных документов, обеспечения возможности обращения к любым элементам данных, имеющихся в этих документах, и осуществления обмена данными между оперативной памятью и внешней памятью и устройствами ввода — вывода.

4) Описание внешних величин, располагаемых на магнитных лентах. Такие описания позволяют задавать в явном
виде размещение данных на любых ЛПМ (лентопротяжных механизмах) машины «Минск-22» и осуществлять обмен данными между ЛПМ и оперативной памятью.

5) Конкретное представление оператора процедуры-кода с широкими возможностями модификации назначения этого оператора (его спецификации) и состава фактических параметров. В неразрывной связи с оператором процедуры-кода находится разработанная в языке АЛГЭМ-1 методика построения БСП (библиотеки стандартных программ), обеспечивающая достаточно широкие возможности включения в БСП новых процедур-кодов (стандартных подпрограмм), необходимых различным пользователям.

6) Два служебных оператора — специальный условный оператор, обеспечивающий условный переход в программе в зависимости от включения ключа на пульте управления машины, и оператор останова, обеспечивающий прекращение процесса выполнения программы машиной. Оба служебных оператора предназначены для облегчения процесса отладки программ, выработанных с помощью системы АЛГЭМ-СТ-3.

Наряду с указанными выше добавлениями в языке АЛГЭМ-1 сделаны следующие сокращения некоторых средств АЛГОЛа по сравнению с эталонным языком:

1) Исключены условные выражения. Выбор направлений вычислительного процесса может осуществляться при этом с помощью условных операторов (односторонних).

2) Исключены величины типа **логический** и логические выражения. Их функции выполняются с помощью отношений, которые представляют собой простой частный вид логических выражений.

3) Введено ограничение на состав операндов (величин, участвующих в операциях) внутри одного арифметического выражения. Эти операнды (переменные, числа и указатели стандартных функций) должны быть одного типа — либо целые, либо вещественные.

4) В операторе цикла в качестве списка цикла можно использовать либо один элемент типа арифметической прогрессии, либо один элемент типа списка выражений (перечисление выражений).

5) Исключены двусторонние условные операторы. Условные переходы можно осуществлять только с помощью односторонних условных операторов вида

если ⟨отношение⟩ **то** ⟨оператор⟩

6) Исключен раздел процедур АЛГОЛа. Допускается использование только упомянутых выше процедур-кодов и указателей стандартных функций (представленных также процедурами-кодами).

7) В качестве границ в описаниях массивов допускается использование целых чисел без знака, со знаком «минус» и простых переменных целого типа.

8) В переключательном списке допускается использование только меток (а не любых именующих выражений, как это предусмотрено в АЛГОЛе).

9) Запрещается использование целых чисел без знака в качестве меток (можно использовать в качестве меток только идентификаторы).

10)Исключены собственные величины.

Все перечисленные ограничения подмножества языка АЛГОЛ, входящего в состав языка АЛГЭМ-1, по сравнению с эталонным языком связаны в основном с упрощением построения транслятора и повышением скорости его работы, а также с облегчением возможности выработки транслятором более качественных рабочих программ, т. е. программ, имеющих приемлемые размеры и работающих не слишком долго. Как подтверждает практика использования системы АЛГЭМ-СТ-3, указанные ограничения входного языка не являются существенными при программировании экономических задач и не создают каких-либо заметных трудностей при пользовании этим языком.

Рассмотрим более подробно упомянутые выше добавления и изменения АЛГОЛа, составляющие специфику языка АЛГЭМ-1. Начнем с определения состава основных символов:

⟨основной символ АЛГЭМ-1⟩ : : = ⟨буква⟩ | ⟨цифра⟩ | ⟨ограничитель строки⟩ | ⟨ограничитель⟩

Здесь исключено понятие ⟨логическое значение⟩, но введено понятие ⟨ограничитель строки⟩, необходимое для оперирования со строками и заменяющее строчные кавычки АЛГОЛа.

⟨ограничитель строки⟩ : : = ‘ | ’

В состав букв АЛГЭМ-1 входят только большие буквы русского алфавита и недостающие буквы латинского.

Из состава ограничителей в АЛГЭМ-1 исключены спецификаторы, в связи с отсутствием в этом языке раздела процедур. Из состава операций исключены логические операции, а из состава арифметических операций исключена операция целочисленного деления:

⟨знак операции следования⟩: : = **на** | **если** | **то** | **для** | **цикл | ко | стоп**

Здесь исключен основной символ **иначе** и включены знаки **ко** и **стоп,** смысл которых будет пояснен ниже.

Из состава разделителей исключен символ **пока** в связи с отсутствием в АЛГЭМ-1 элементов списка цикла типа пересчет.

В состав скобок включены две словесные скобки **составной** и **уровень** и из состава скобок исключены кавычки, которые образуют отдельную группу символов — ограничителей строки.

Из состава описателей исключены символы **логический, собственный, процедура** и включены символы **строчный, вид,** необходимые для описания строчных величин и форматов переменных.

Определения чисел в АЛГЭМ-1 отличаются от АЛГОЛа только тем, что значение порядка в АЛГЭМ-1 может иметь только одну цифру (десятичную), со знаком или без знака. В отношении идентификаторов в АЛГЭМ-1 вводится дополнительное ограничение: идентификаторы не могут начинаться с буквы Я, которая зарезервирована для обозначения внешних величин.

Строки в АЛГЭМ-1 имеют некоторые ограничения на состав символов:

⟨строчный символ⟩ : : = ⟨буква⟩ | ⟨цифра⟩ | ⟨знак арифметической операции⟩ | ⟨знак операции отношения⟩| · |10 ⊔|I

Последний символ используется в качестве вертикального разделителя при печати строк на АЦПУ.

⟨элемент строки⟩ : : = ⟨строчный символ⟩ | ⟨строка⟩

⟨открытая строка⟩ : : = ⟨элемент строки⟩ | ⟨открытая строка⟩ ⟨элемент строки⟩

⟨строка⟩ : : = ‘⟨открытая строка⟩’

Открытые строки представляют собой значения строчных переменных.

Существенным дополнением в АЛГЭМ-1 по сравнению с АЛГОЛом является понятие составной переменной. Понятие АЛГОЛа «переменная» в АЛГЭМ-1 получает название «элементарная переменная», а новое понятие «переменная» определяется следующим образом:

⟨идентификатор составной переменной⟩ : : = ⟨идентификатор ⟩

⟨идентификатор составного массива⟩ : : = ⟨идентификатор⟩

⟨уточнение⟩ : : = ⟨идентификатор составной переменной ⟩ | ⟨идентификатор составного массива⟩ [⟨список индексов⟩]

⟨переменная⟩ : : = ⟨элементарная переменная⟩ | ⟨переменная⟩ ⟨уточнение⟩

Составная переменная — это наименование определенной совокупности элементарных величин, расположенных в заданном порядке. Значением идентификатора составной переменной является упорядоченное множество значений элементарных величин, входящих в ее состав.

Составной массив — это массив составных переменных одинаковой структуры с индексами. Значение идентификатора составного массива есть упорядоченное множество значений массива составных переменных с индексами.

Составная переменная или составная переменная с индексами могут включать в себя элементарные величины различных типов, и поэтому составные величины не имеют описаний типа; такие описания могут быть только у несоставных величин. Составные величины могут иметь сложную многоуровневую структуру, т. е. в составную величину первого уровня могут входить не только элементарные величины, но и составные величины второго уровня, а в них могут входить составные величины третьего уровня и т. д. В конечном счете любая составная величина состоит из элементарных величин, но эти элементарные величины могут группироваться внутри составной величины различными способами.

Для того чтобы можно было обратиться к любой элементарной переменной, входящей в составную величину любого уровня, используются уточнения.

Уточнение представляет собой ссылку на более крупные единицы информации (составные величины), в состав которых входит данная элементарная переменная согласно описанию. Эти идентификаторы составных величин указываются последовательно слева направо в соответствии с глубиной уровней этих величин и разделяются точками. Наличие уточнений позволяет использовать для обозначения различных величин, входящих в разные составные величины, одни и те же идентификаторы. При этом неопределенность исключается за счет различий в идентификаторах составных величин, являющихся уточнениями.

В АЛГЭМ-1 (в отличие от некоторых других языков) выписывать следует все уточнения в порядке их иерархии.

Выражения и операторы языка АЛГЭМ-1. В состав выражений языка АЛГЭМ-1 входят простые арифметические, именующие и строчные выражения, а также отношения. Особенностью языка АЛГЭМ-1 является разделение арифметических выражений на два отдельных типа: целые и вещественные.

Рассмотрим определение целого арифметического выражения:

⟨первичное целое выражение⟩ : : = ⟨целое без знака ⟩ | ⟨переменная⟩ | ⟨указатель стандартной функции⟩ | (⟨целое выражение⟩)

⟨целый множитель⟩: : = ⟨первичное целое выражение⟩ | ⟨первичное целое выражение⟩ ↑ ⟨целое без знака⟩

⟨целый одночлен⟩ : : = ⟨целый множитель⟩ | ⟨целый одночлен⟩ × ⟨целый множитель⟩

⟨операция типа сложения⟩ : : = + | —

⟨целое выражение⟩: : = ⟨целый одночлен⟩ | ⟨операция типа сложения⟩ ⟨целый одночлен⟩ | ⟨целое выражение⟩ ⟨операция типа сложения⟩ ⟨целый одночлен⟩

Компоненты целых выражений (за исключением фактических параметров указателей стандартных функций и индексных выражений) должны быть типа **целый.** Операция деления в целом выражении не допускается.

При операции возведения в степень (символ операции ↑) ⟨первичное целое выражение⟩ является основанием, а ⟨целое без знака⟩ — показателем степени. В АЛГЭМ-1 повторное возведение в степень в целых выражениях не допускается, а показателем степени может быть только целое число без знака.

Синтаксис вещественного арифметического выражения получается из синтаксиса целого выражения заменой во всех определениях слов «целое» на «вещественное» и введением дополнительного понятия

⟨операция типа умножения⟩ : : = × |/,

которое заменяет знак операции умножения во второй альтернативе определения вещественного одночлена. Кроме того, во второй альтернативе определения вещественного множителя в качестве основания степени выступает ⟨вещественный множитель⟩ (это обеспечивает возможность повторного возведения в степень в вещественных выражениях), а показателем степени является ⟨первичное вещественное выражение⟩

Переменная и указатель функции, входящие в состав первичных вещественных выражений, по описанию должны быть вещественными.

В качестве одного из способов контроля точности вычислений в вещественных арифметических выражениях на языке АЛГЭМ-1 (и АЛГЭМ-2) может использоваться задание форматов чисел в описаниях переменных и массивов (см. ниже).

Строчные выражения в АЛГЭМ-1 представляют собой либо строку, либо переменную типа **строчный.** Значением строчного выражения является открытая строка.

Именующие выражения в АЛГЭМ-1 могут представляться либо меткой (которая в АЛГЭМ-1 может быть только идентификатором), либо указателем переключателя.

Отношения в АЛГЭМ-1 —это либо два целых выражения, соединенные знаком операции отношения, либо два вещественных выражения, соединенных знаком операции отношения. При использовании отношений с вещественными выражениями необходимо иметь в виду возможные отклонения в точности вычислений вещественных выражений на конкретной вычислительной машине. Результатом вычисления отношения является значение признака: **истинно** или **ложно,** которое используется в условных операторах для выбора направления дальнейших вычислений.

Примеры.

1) Числа:

349 -013. 17_{10}+3 0.272_{10}—7 +0.131

2) Строки:

‘29 июня ⊔1953 г.’ ‘ИВАНОВ ⊔ П. ⊔ М.’

3) Переменные с индексами и уточнениями:

М. К [x, y + sin (a)].A [i, 100]

B643 [i + j, P.Q [R]]

4) Целые арифметические выражения:

A5 + A8 ↑ P × sign (x) 164

5) Вещественные арифметические выражения:

$+8.149_{10}$ — 2 sin (x+0.5 ×y)

(A + B [i])/(C + D [j]) - Q ↑ x ↑ y ↑ (0.5- M)

6) Отношения: A + B ≤ C + D

x[i, j] ≥ 0.148_{10} —3

Особенности операторов АЛГЭМ-1. Синтаксическое определение операторов в АЛГЭМ-1 отличается от определения в АЛГОЛе только тем, что в состав основных операторов вместо оператора процедуры входит оператор процедуры-кода, а программа должна обязательно быть блоком (т. е. не может быть составным оператором). В АЛГЭМ-1 полностью сохраняют свою силу принципы локализации величин в блоках (в том числе и принципы локализации меток), принятые в АЛГОЛе.

Оператор присваивания в АЛГЭМ-1 может иметь в списке левой части только переменные, а в правой части — либо арифметическое, либо строчное выражение. Все переменные в списке левой части должны быть по описаниям одного типа (целого, вещественного или строчного). Если тип выражения в правой части отличается от типа переменных в левой части, то перед присваиванием вычисленного значения правой части производится автоматически его преобразование к типу переменных в левой части.

Если переменные типа **целый** или **вещественный** имеют в своих описаниях форматы, то присваивание им значения выражения правой части производится в соответствии с их форматами (см. ниже). При этом если выражение в правой части является строчным, то это должны быть либо просто строчная переменная, либо строка.

Если в левой части находятся строчные переменные, а в правой части — арифметическое выражение, то таким арифметическим выражением может быть только переменная, имеющая в своем описании формат. Во всех случаях, когда переменная списка левой части имеет в своем описании формат, присваивание ей значения идет в соответствии с этим форматом посимвольно слева направо.

Оператор перехода в АЛГЭМе-1 может иметь в качестве именующего выражения либо метку, либо указатель переключателя. Переход не может осуществляться внутрь блока, но может происходить внутрь составного оператора.

Условный оператор в АЛГЭМ-1, помимо одностороннего условного оператора АЛГОЛа с отношением в качестве логического выражения в условии, включает в себя и такую разновидность, как переход по ключу, устанавливаемому на пульте управления машины «Минск-22»:

⟨переход по ключу⟩: : = **ко** ⟨целое без знака⟩ **на** ⟨метка⟩ Символ **ко** означает **к**люч **о**тключен. Целое без знака, стоящее за этим символом, должно представлять собой номер ключа на пульте ЭВМ «Минск-22». Если этот ключ отключен, то выполняется переход по метке, указанной в операторе перехода, если включен, то переход не осуществляется и весь оператор перехода по ключу равносилен пустому оператору.

Оператор цикла в АЛГЭМ-1 может иметь в качестве параметра цикла только простую переменную типа целый или вещественный, не имеющую формата в описании, а список цикла может представлять собой либо список арифметических выражений, либо арифметическую прогрессию. Если в списке цикла вида арифметической прогрессии выражения имеют тип, отличный от типа переменной, являющейся параметром цикла, то перед каждым использованием значений этих выражений производится их преобразование к типу переменной, являющейся параметром цикла.

Операторы процедур-кодов в АЛГЭМ-1 представлены достаточно широко. Оператор процедуры-кода вызывает обращение к подпрограмме, записанной в коде ЭВМ «Минск-22» и входящей в библиотеку стандартных подпрограмм. Операторы процедур-кодов вместе с библиотекой стандартных подпрограмм составляют специфическое средство системы автоматизации программирования АЛГЭМ-СТ-3, неразрывно связанное с машиной «Минск-22» (§ 11).

В АЛГЭМ-1 имеется оператор останова:

⟨оператор останова⟩ : : = **стоп** ⟨целое без знака⟩

При выполнении данного оператора машина останавливается и на пульте управления высвечивается целое число без знака, использованное в данном операторе, а также адрес команды, соответствующий этому оператору. Если нажать кнопку «ПУСК», то программа будет выполняться со следующего оператора.

Описания в АЛГЭМ-1. По сравнению с АЛГОЛом в АЛГЭМ-1 добавляются описания составной переменной, составного массива, внешнего массива, внешней составной переменной, внешнего составного массива и исключается описание процедур. Кроме того, в описании типа вводятся форматы, исключается логический тип и вводится строчный тип.

С помощью форматов осуществляется распределение памяти под величины и редактирование значений величин

при выдаче их на печать. Описание типа в АЛГЭМ-1 имеет следующий синтаксис.

⟨постоянный повторитель⟩ : : = (⟨целое без знака⟩)

⟨переменный повторитель ⟩:: = (⟨идентификатор⟩)

⟨9-позиция⟩ : : = 9 | 9 ⟨постоянный повторитель⟩

⟨1-позиция⟩ : : = 1 | 1 ⟨постоянный повторитель⟩

⟨повторитель⟩ : : = ⟨постоянный повторитель⟩ | ⟨переменный повторитель⟩

⟨С-позиция⟩ : : = С | С ⟨повторитель⟩

⟨формат целого без знака⟩ : : = ⟨9-позиция⟩ | ⟨формат целого без знака⟩ ⟨9-позиция⟩

⟨формат целого⟩: : = ⟨формат целого без знака⟩ | + ⟨формат целого без знака⟩ | — ⟨формат целого без знака⟩

⟨формат порядка⟩ : : = $_{10}$+9

⟨формат правильной дроби⟩ : : =. ⟨формат целого без знака⟩

⟨формат дроби⟩ : : = ⟨формат правильной дроби⟩ | ⟨формат целого без знака⟩ ⟨формат мантиссы⟩

⟨формат вещественного без знака⟩ : : = ⟨формат дроби)⟩ | ⟨формат дроби⟩ ⟨формат порядка⟩

⟨формат вещественного⟩ : : = ⟨формат вещественного без знака⟩ | + ⟨формат вещественного без знака⟩ | — ⟨формат вещественного без знака⟩

⟨формат числа⟩ : : = ⟨формат целого⟩ | ⟨формат вещественного⟩ | ⟨1-позиция⟩

⟨формат строки⟩ : : = ⟨С-позиция⟩

⟨формат⟩: : = ⟨формат числа⟩ | ⟨формат строки⟩

⟨элемент списка типа⟩ : : = ⟨идентификатор⟩ | ⟨идентификатор⟩ **вид** ⟨формат⟩

⟨список типа⟩ : : = ⟨элемент списка типа⟩ | ⟨список типа⟩, ⟨элемент списка типа⟩

⟨тип⟩ : : = **целый** | **вещественный** | **строчный**

⟨описание типа⟩ : : = ⟨тип⟩ ⟨список типа⟩

Примеры **целый** A, B, C **вид** 9 (6)
строчный ИМЯ **вид** С (12)
вещественный SA **вид** 9.9999

Переменные, которым предписан тип строчный, принимают значения, являющиеся открытыми строками. Эти переменные должны иметь в описании формат. Переменные типа целый и **вещественный** могут иметь в описании формат, а могут и не иметь. Форматы переменных определяют для их значений длину и фиксированный состав символов в заданных позициях. Формат характеризует только ту переменную, за идентификатором которой он непосредственно следует. Для задания вида и положения символов, которые могут представлять значения переменных, используются указатели символов.

Указатель «9» означает, что в данной позиции значения числовой величины может стоять одна из десятичных цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Кроме того, на таких позициях в начале числа могут находиться пробелы, а в конце числа (справа от десятичной точки) могут помещаться нули. Указатель «1» показывает, что в данной позиции числа может стоять двоичная цифра: либо 0, либо 1.

Указатель « .» предписывает помещение в данном разряде десятичной точки (в явном виде), отделяющей целую часть числа от дроби.

Указатель $_{10}$ предписывает помещение в данном разряде символа $_{10}$, обозначающего десятичный порядок. Цифра справа от этого символа представляет значение порядка; в АЛГЭМ-1 в формате порядка допускается только один указатель символа 9.

Указатель «+», использованный в начале формата числа, предписывает помещение в данном разряде знака «плюс» для положительных чисел и знака «минус» для отрицательных чисел. Аналогичное действие вызывается в том случае, когда он стоит после указателя $_{10}$. Указатель «—», использованный в начале формата числа, предписывает помещение в данном разряде знака «пробел» (и) для положительных чисел и знака «минус» для отрицательных чисел. После указателя $_{10}$ в формате может стоять указатель «+» и не может стоять указатель «—».

Указатель «С» допускает любой символ в данной позиции значения строчной переменной.

Для сокращения записей форматов служат повторители. Повторитель показывает, сколько раз должен быть повторен символ, стоящий непосредственно перед ним. Так, форматы +9999 и +9 (4) эквивалентны. Число символов в самом формате с учетом повторителей (повторитель содержит три символа) не должно превышать девятки. В переменном повторителе допускается использование простой переменной типа **целый,** не имеющей в своем описании формата. Указанная переменная не должна быть локальной в блоке, для которого имеет силу соответствующее описание. Повторение символа «С» возможно только с помощью повторителя.

Общее число указателей символов, показанных в формате переменной, определяет длину слова ее значения. Эта длина для переменных типа **целый** и **вещественный** не должна превышать девяти символов (если формат задается с помощью 9-позиций) или тридцати шести символов (если формат

задается с помощью 1-позиции). Для символа «С» ограничения на количество повторений практически нет. Указатели символов «+», «—», «.», «$_{10}$» в формате могут встречаться только один раз.

Примеры действия форматов. Пусть фактическое значение переменной +182, 643. Тогда при записи числа в ячейку памяти машины и при выдаче его на печать оно будет иметь разный вид в зависимости от формата, указанного в описании переменной (табл. 3).

Таблица 3

Формат	Запись числа
+ 999.999	+ 182.643
99.999	82.643
9999.9999	␣ 182.6430
-(3)9.9(5)	␣␣␣ 2.64300

Редактирование значений переменных по форматам производится при выполнении оператора присваивания. При этом переменная, находящаяся в левой части оператора присваивания, должна иметь в своем описании формат; значение выражения, находящегося в правой части этого оператора, будет отредактировано по формату переменной левой части перед выполнением присваивания.

Редактирование целых значений осуществляется путем совмещения младшей цифры числа с правой (младшей) позицией формата. Если количество цифр в числе меньше количества позиций в формате, то избыточные позиции заполняются нулями. Если, наоборот, количество цифр в числе превышает количество позиций в формате, то старшие разряды числа, не поместившиеся в формат, отбрасываются. Знак числа всегда ставится на то место, которое отведено ему форматом.

Знак числа проставляется в получающемся числовом значении в соответствии со следующими правилами. Если в формате числа (вне формата порядка) стоит знак плюс, то в соответствующей позиции получающегося числового значения помещается при неотрицательном числе символ плюс, а при отрицательном — символ минус.

Если в формате числа (вне формата порядка) стоит знак минус, то в соответствующей позиции получающегося числового значения помещается при неотрицательном числе символ пробела, а при отрицательном — символ минус.

Если в формате числа (вне формата порядка) знак отсутствует, то числовое значение должно быть неотрицательным.

Если числовое значение по абсолютной величине слишком велико, чтобы его можно было представить в указанной форме, то результат редактирования не определен.

Если значение порядка отрицательно и слишком велико по абсолютной величине, чтобы его можно было представить в форме, заданной форматом, то редактирование выполняется так, как если бы числовое значение было равно нулю. Так же редактируются и двоичные коды по формату, состоящему из 1-позиции. При необходимости недостающие разряды слева заменяются нулями.

Редактирование вещественных чисел может производиться двумя путями, в зависимости от того, имеется ли в формате числа формат порядка или нет. Если формат порядка отсутствует, то совмещаются позиции десятичной точки числа и формата. При этом, если число имеет порядок, то оно предварительно приводится к виду десятичного числа без порядка (путем соответствующего перемещения десятичной точки). Затем разряды числа, находящиеся справа и слева от десятичной точки, совмещаются с соответствующими позициями формата. Целая часть редактируется по правилам редактирования целого числа. Дробная часть, имеющая больше цифр, чем это предусмотрено форматом дроби, округляется. Наоборот, если дробная часть имеет меньше цифр, чем это предусмотрено форматом дроби, то она дополняется справа нулями.

При наличии в формате числа формата порядка редактирование производится путем совмещения старшей значащей цифры числа со старшей (левой) цифровой позицией (9) формата числа. При этом, если необходимо, чтобы совпали положения десятичных точек, то корректируется порядок числа. Получившаяся дробная часть числа округляется (если она выходит за формат) или дополняется справа нулями. Значение порядка числа ставится в ту позицию (одну), которая для него предусмотрена форматом порядка, с обязательным указанием знака порядка. Редактирование строчных значений производится путем посимвольного совмещения символов строки, представляющей строчное значение, с указателями символов «С» слева направо. Если строка имеет больше символов, чем это предусмотрено форматом, то лишние символы отбрасываются. Если строка имеет меньше символов, то

вместо недостающих символов ставятся пробелы.

Следует заметить, что в АЛГЭМ-СТ-3 символы строк кодируются 7-разрядными кодами АЦПУ (в том числе, если в состав строк входят и десятичные цифры). Десятичные числа кодируются 4-разрядными двоичными кодами (по отдельным цифрам). Наконец, двоичные коды представляются двоичными разрядами в пределах разрядной сетки ячейки машины «Минск-22».

Описание массивов. Описание массивов в АЛГЭМ-1 имеет следующий синтаксис:

⟨нижняя граница⟩ : : = ⟨целое без знака⟩ | — ⟨целое без знака⟩ | ⟨идентификатор⟩

⟨верхняя граница⟩ : : = ⟨целое без знака⟩ | — ⟨целое без знака⟩ | ⟨идентификатор⟩

⟨восьмеричная цифра⟩ : : =0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

⟨адрес⟩ : : =

$$\underbrace{\langle\text{восьмеричная цифра}\rangle \ldots \langle\text{восьмеричная цифра}\rangle}_{\text{восемь раз}}$$

⟨идентификатор внешнего массива⟩ : : = Я ⟨адрес⟩

⟨элемент описания массивов⟩ : : = ⟨идентификатор массива⟩ **вид** ⟨формат⟩ | ⟨идентификатор массива⟩ | ⟨идентификатор внешнего массива⟩ **вид** ⟨формат⟩ | ⟨идентификатор внешнего массива⟩

⟨описание массивов⟩ : : = **массив** ⟨список массивов⟩ | **целый массив** ⟨список массивов⟩ | **строчный массив** ⟨список массивов⟩

Понятия ⟨граничная пара⟩, ⟨список граничных пар⟩, ⟨сегмент массивов⟩, ⟨список массивов⟩ такие же, как в АЛГОЛе.

Примеры описаний массивов.

массив C, D, Я00042000 [M : N, 12 : H], Q [—12 : 20]

целый массив S **вид** 9 (3) [10 : 50]

строчный массив R1 **вид** C (12) **[**—8 : 15], N **вид** CC, AB **вид** C (2) 99

Если в описании массива есть формат, то задаются длина и состав символов значений переменных с индексами. Формат распространяется на идентификатор массива, стоящий непосредственно перед ним. Так же как и в АЛГОЛе, все массивы, данные в одном описании, имеют один и тот же тип. Если описание типа отсутствует, то подразумевается тип **вещественный.**

Простые переменные, использованные в списке граничных пар, должны иметь тип **целый** и не содержать в описании формата. Новым видом массивов в АЛГЭМ-1 по сравнению с АЛГОЛом являются внешние массивы. Идентификатор внешнего массива может быть использован только в качестве фактического параметра процедуры-кода со спецификациями ‘ЗАПИСЬ’ и ‘ЧТЕНИЕ’ (см. ниже), (адрес) представляет собой истинный начальный адрес внешнего массива на магнитной ленте.

Как уже упоминалось раньше, в АЛГЭМ-1 для представления сложных единиц информации (отчетов, заявок анкет и т. п.) введены составные переменные и составные массивы. Их описания строятся в соответствии со следующим синтаксисом:

⟨элемент описания⟩ : : = ⟨описания типа⟩ | ⟨описание массивов⟩ | ⟨описание составной величины⟩

⟨составная величина⟩ : : = ⟨идентификатор составной переменной⟩ | **массив** ⟨идентификатор составного массива⟩ [⟨список граничных пар⟩]

⟨структура составной величины⟩:: = ⟨элемент описания⟩ **уровень** | ⟨элемент описания⟩; ⟨структура составной величины⟩

⟨описание составной величины⟩ : : = **составной** ⟨составная величина⟩; ⟨структура составной величины⟩

Примеры: составной Q**; строчный** A **вид** C (10);
целый M;
составной B; **целый** A **вид** 99.99,
D **уровень;**
массив S **вид** 9.99 [P : R[**уровень составной массив** ОТЧЕТ [1 : 100];
целый НОМЕР ЗАВОДА **вид** 9 (3);
составной ДАТА;
целый ДН, МЦ, ГД **уровень;**
целый ПОЛУЧЕНО **вид** 999,
ОТГРУЖЕНО **вид** 999,
ОСТАТОК **вид** 999 уровень

Составные величины могут быть двух видов: составные переменные в виде:

составной ⟨идентификатор⟩; ⟨структура составной величины⟩

и составные массивы в виде:

составной массив ⟨идентификатор⟩ [⟨список граничных пар⟩]; ⟨структура составной величины⟩)

В описании составной переменной перечисляются идентификаторы, первый из которых именует саму составную переменную, а остальные — переменные или массивы (элементарные или составные), входящие в ее состав. Последовательность расположения величин в составной переменной и использование скобок **составной** . . . **уровень** позволяет задавать иерархию величин (т. е. вхождение одних величин в другие).

Составной массив — это массив, состоящий из одинаковых по структуре составных переменных.

Составная переменная и составной массив не имеют типа и формата, так как в общем случае могут состоять из величин различных типов. Только элементарная величина может иметь тип и формат, что и указывается в ее описании на соответствующем уровне.

Поскольку элементами составных переменных и составных массивов, в свою очередь, могут служить также составные величины и массивы, то определение составных величин является рекурсивным.

Как уже указывалось при рассмотрении синтаксиса переменной, обращение к элементу составной переменной или составного массива содержит ряд идентификаторов, разделенных точками. Оно содержит в качестве уточнений обязательно все промежуточные уровни иерархии, включая самый внешний.

Заметим, что одни и те же идентификаторы можно использовать в одном блоке программы для обозначения различных величин, если эти величины являются частями составных переменных и массивов, и различие между ними может быть установлено с помощью уточнений.

Внешние составные величины представляют собой средство для обеспечения обмена данными между МЛ и МОЗУ. Внешние составные величины имеют следующий синтаксис:

⟨идентификатор внешней составной⟩ : : = Я ⟨адрес⟩

⟨идентификатор внешнего составного массива⟩ : : = Я ⟨адрес⟩

⟨внешняя составная величина⟩ : : = ⟨идентификатор внешней составной⟩ | **массив** ⟨идентификатор внешнего составного массива⟩ [⟨список граничных пар⟩]

⟨указатель начала⟩ : : = ⟨целое выражение⟩

⟨указатель длины⟩ : : = ⟨целое выражение⟩

⟨описание внешней составной⟩: : = составной ⟨внешняя составная величина⟩; ⟨указатель начала⟩, ⟨указатель длины⟩ **уровень**

Примеры.

составной массив Я00000120 [i : j, k : 1]; A + 10, B + 5 уровень

составной Я00007020; К, М **уровень**

Идентификаторы внешних составных величин могут выступать только в качестве фактических параметров оператора процедуры-кода со спецификациями ‘ЗАПИСЬ’ и ‘ЧТЕНИЕ’ (см. ниже). По существу, в описаниях внешних составных переменных и массивов не задается их фактическая структура, а фиксируется только их размер (число ячеек) и начальное положение на МЛ.

В указателе начала и указателе длины внешней составной допускается использование целых выражений. Переменные, входящие в состав этих выражений, не должны быть локальными в блоке, для которого имеет силу описание внешней составной.

Указатель начала представляет собой сдвиг (число ячеек) относительно начального адреса, указанного после буквы Я. Указатель длины представляет собой длину (в ячейках) одного элемента внешнего составного массива или длину внешней составной переменной.

11. Транслятор и библиотека стандартных подпрограмм системы «АЛГЭМ-СТ-3»

Транслятор и библиотека стандартных подпрограмм наряду с входным алгоритмическим языком АЛГЭМ-1, рассмотренным в предыдущем параграфе, образуют три основные составные части системы автоматизации программирования АЛГЭМ-СТ-3.

Буквы СТ означают синтаксический транслятор, а цифра 3 — третий (усовершенствованный) вариант этого транслятора, реализованного на машине «Минск-22».

ЭВМ «Минск-22» относится к классу машин средней производительности; она имеет быстродействие 5—6 тысяч операций в секунду, оперативное запоминающее устройство на ферритовых сердечниках емкостью 8196 ячеек по 37 двоичных разрядов, внешнее запоминающее устройство на магнитных лентах емкостью 1,6 млн. ячеек. Информация на каждой МЛ записывается зонами по 2048 ячеек; на одной МЛ может быть 50 зон. Ввод данных осуществляется с помощью перфолент и перфокарт, а вывод — с помощью перфолент, перфокарт и с помощью цифрового и алфавитно-цифрового печатающих устройств.

Основные принципы и алгоритмы работы синтаксических трансляторов разработаны и подробно описаны Айронсом и Ингерманом [7].

В трансляторе АЛГЭМ-СТ-3 осуществлена реализация этих принципов применительно к языку АЛГЭМ и усовершенствованы основные алгоритмы по сравнению с описанными в литературе, а также произведена разработка комплекса обслуживающих программ транслятора.

Синтаксический транслятор отличается от трансляторов функционально-блочного типа прежде всего тем, что он строится не для определенной конкретной пары языков (входного, с которого переводит, и выходного, на который переводит), а для классов входных и выходных языков. Обычные трансляторы строятся в виде набора различных функциональных блоков, каждый из которых осуществляет обработку и перевод конструкций определенных видов в заданной паре конкретных языков (входном и выходном). Синтаксический транслятор состоит из двух основных частей: собственно транслятора, представляющего собой комплекс машинных программ, осуществляющих предварительную подготовку, анализ и перевод текстов с входного языка на выходной, и набора синтаксических правил, определяющих структуру и взаимосвязь этих языков и используемых транслятором в процессе его работы.

Собственно транслятор строится для класса языков, а информация о конкретных языках, между которыми нужно производить перевод, задается транслятору с помощью указанного набора правил. Таким образом, только синтаксические правила (представленные в виде таблиц) привязаны к конкретным входному и выходному языкам, и, изменяя эти таблицы, можно использовать один и тот же транслятор для перевода между разными парами языков, относящихся к определенному классу.

Основным признаком, определяющим принадлежность некоторого языка к заданному классу, является возможность описания синтаксиса этого языка в форме, «понятной» транслятору, т. е. на метаязыке, принятом для построения данного синтаксического транслятора. Метаязыком по отношению к данному языку является язык, с помощью которого описывается данный язык. Таким метаязыком, принятым в системе АЛГЭМ-СТ-3, является метаязык Бэкуса или, как принято называть, бэкусовская нормальная форма (БНФ). Таблицы синтаксических правил языка, используемые транслятором, представляют собой закодированные определенным способом определения понятий языка АЛГЭМ-1, приведенные в предыдущем параграфе.

Каждая такая формула представляет собой определение некоторой единицы языка (понятия) через более простые единицы языка (понятия и основные символы). Кроме того, для каждого определяемого понятия приводится соответствующий семантический перевод, показывающий, какие элементы выходного языка должны быть записаны в выходную строку при обнаружении во входной строке данного понятия.

В рассматриваемом трансляторе применен двухступенчатый способ перевода: исходная программа, представленная на входном языке, сначала переводится синтаксическим транслятором на некоторый промежуточный язык (ПЯ), а затем вспомогательным транслятором (построенным по функционально-блочному принципу) — с ПЯ на машинный язык ЭВМ «Минск-22». Такой способ упрощает построение транслятора и облегчает возможность перехода на другие машины, так как при этом должен меняться в основном только второй транслятор. Часть транслятора, осуществляющая синтаксический анализ входной строки (т. е. программы, записанной на входном алгоритмическом языке с помощью упомянутых металингвистических формул), называется анализатором. Часть транслятора, осуществляющая «сборку» промежуточной программы на ПЯ с помощью семантических определений,— компилятором.

Процесс анализа программы, представленной на входном языке, осуществляется последовательно символ за символом по принципу «снизу вверх». Упрощенно этот принцип может быть описан так. В начале работы анализатор имеет перед собой конечную цель — получить «программу» (в таблице синтаксических правил понятие «программа» определено как наиболее крупная синтаксическая единица языка). После выбора первого символа текста из входной строки анализатор определяет с

помощью специальной таблицы, можно ли из этого символа (в принципе) получить эту конечную цель. Например, если первым символом будет знак «+», то ответ будет отрицательный, так как никакая программа не начинается с этого символа. Если ответ будет положительный (например, первым символом была буква), то анализатор проверяет по синтаксическим правилам, в какую более крупную синтаксическую единицу может входить этот символ, и ставит себе в качестве ближайшей цели получение этой единицы. Он выбирает следующий символ и определяет, какую конструкцию образуют два уже выбранных символа и какую более крупную конструкцию можно из них получить и т. д. В процессе построения каждой синтаксической единицы анализатор ставит перед собой новые цели. При этом временно запоминается более далекая предшествующая цель, в том числе конечная цель — «программа». Из текста на входном языке последовательно выбираются очередные основные символы. Когда будет достигнута какая-то ближайшая цель, т. е. сформирована какая-то синтаксическая единица (например, «отношение»), происходит возврат (восстановление) предшествующей цели (например, «конец составного»). После этого анализатор выбирает и анализирует следующие символы входной строки и формирует более крупную синтаксическую единицу, соответствующую предшествующей цели.

При получении новой единицы (при завершении работы с данным синтаксическим правилом) анализатор выписывает номер семантического определения, связанного с этим правилом. Этот номер будет использован компилятором при построении строки на промежуточном языке. В состав синтаксического транслятора входит еще часть, называемая составителем таблиц. Эта часть служит для дополнения исходной совокупности металингвистических правил («глобальных» правил) новыми правилами («локальными»), образуемыми в процессе просмотра входной программы (на основе описаний, имеющихся в блоках).

В трансляторе реализован разработанный Ф. Ф. Шиллер способ контроля программы на выходном языке. Для этой цели множество синтаксических правил языка расширено таким образом, чтобы можно было анализировать не только правильные, но и ошибочные конструкции, а семантическая часть этих правил содержит указания о печати в выходной строке сведений о встретившихся синтаксических или семантических ошибках.

Описываемый в данной главе транслятор АЛГЭМ-СТ-3 обладает более высокой эффективностью работы в части требуемого времени трансляции и увеличения объема допускаемых к трансляции программ по сравнению с ранее выпущенным транслятором АЛГЭМ-СТ-2, получившим достаточно широкое применение. Это повышение эффективности достигнуто в основном за счет тщательной отработки Ю. П. Кирюхиным основных алгоритмов транслятора (блока ввода и кодировки, составителя таблиц, анализатора), в которых широко использованы оригинальные приемы размещения и обработки информации, в частности ассоциативные списковые структуры и магазины.

Система АЛГЭМ создавалась в период 1965—1969 гг. и в настоящее время получила практическое применение. В процессе опытной эксплуатации системы, а затем практического ее использования были проверены и тщательно отработаны все алгоритмы и программы транслятора, выявлены и исправлены ошибки в синтаксических правилах и программах и произведена технологическая доработка системы, обеспечивающая расширение ее практических возможностей и удобство пользования. К числу технологических особенностей системы АЛГЭМ-СТ-3 относятся: наличие автоматического синтаксического контроля и печати ошибок, удобство внесения исправлений в программу и исходные данные; широко развитая система контрольных остановок, гибкая система процедур-кодов, обеспечивающая включение в библиотеку стандартных программ любых новых специализированных подпрограмм, отвечающих нуждам пользователя, а также наличие комплекса обслуживающих программ транслятора (тест сохранности транслятора на МЛ, программа дублирования транслятора с МЛ на МЛ, программа проверки перфорации рабочих программ и др.).

Операторы процедур-кодов и библиотека стандартных подпрограмм системы АЛГЭМ-СТ-3. Операторы процедур-кодов служат для обращения к стандартным подпрограммам, написанным на машинном языке и включенным в библиотеку стандартных подпрограмм системы АЛГЭМ-СТ-3. Оператор процедуры-кода имеет следующий синтаксис:

⟨групповой параметр⟩ : : = ⟨идентификатор массива⟩ | ⟨идентификатор составной переменной⟩ | ⟨идентификатор составного массива⟩ [⟨список индексов⟩] | ⟨идентификатор составного массива⟩ | ⟨групповой параметр⟩ ⟨уточнение⟩

⟨внешний групповой параметр⟩ : : = ⟨идентификатор внешнего составного массива⟩ | ⟨идентификатор внешнего составного массива⟩ [⟨список индексов⟩] | ⟨идентификатор внешней составной⟩ |

⟨идентификатор внешнего массива⟩

⟨фактический параметр⟩ : : = ⟨число без знака⟩ | ⟨строка ⟩ | ⟨переменная⟩ | ⟨групповой параметр⟩ | ⟨внешний групповой параметр⟩

⟨восьмеричная цифра⟩ : : =0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

⟨номер подпрограммы⟩ : : = ⟨адрес⟩

⟨идентификатор внешнего массива⟩ : : = Я ⟨адрес⟩

⟨идентификатор внешней составной⟩ : : = Я ⟨адрес⟩

⟨идентификатор внешнего составного массива⟩ : : = Я ⟨адрес⟩

⟨спецификация процедуры-кода⟩ : : =‘ 2—10’ | ‘10—2’ |
| ‘П4’ | ‘П4 ⊔ 10’ | ‘П4 ⊔ 2—10’ |‘П4 ⊔ АЦПУ’ | ‘ПРЛ’ |
‘ПРЛ ⊔ 10’ | ‘ПРЛ ⊔ 2—10’ | ‘ПРЛ ⊔ С’ | ‘ПРК’ |
‘ПРК ⊔ 10’ | ‘ПРК ⊔ 2—10’ | ‘ПРК⊔ С’ | ‘ВВОДЛ’ |
‘ВВОДЛ⊔ 10—2’ | ‘ВВОДЛ ⊔ С’ | ‘ВВОДК’ |
‘ВВОДК ⊔ 10’ | ‘ВВОДК ⊔ 10—2’ | ‘ВВОДК ⊔ С’ |
‘Д’ | ‘ЗАПИСЬ’ | ‘4ТЕНИЕ’ | ‘ВЫХОД’ | ‘⟨номер подпрограммы⟩’

⟨список фактических параметров⟩ : : = ⟨спецификация процедуры-кода⟩ | ⟨список фактических параметров⟩, ⟨фактический параметр⟩

⟨оператор процедуры-кода⟩ : : = КОД (⟨список фактических параметров⟩)

Примеры операторов процедур-кодов.

КОД (‘4ТЕНИЕ’ Я27310010, М) — чтение с МЛ внешнего массива Я27310010 и присваивание его значения массиву М.

КОД (‘ЗАПИСЬ’, М, Я27310010) — на место внешнего массива Я27310010 записывается массив М.

КОД (‘ПРК ⊔ 10’, A, B, C, D) — ввод с перфокарт в десятичной системе с присваиванием значений переменным A, B, C, D.

КОД (‘П4 ⊔ 2—10’, М [i, j, k]) — печать с переводом из двоичной в десятичную систему счисления значений элементарных переменных, входящих в состав элемента составного массива М со значениями индексов i, j, k.

Спецификации процедур-кодов имеют следующий смысл:

‘2—10’ — перевод чисел из двоичной системы в десятичную систему счисления.

‘10—2’ — перевод чисел из десятичной в двоичную систему счисления.

‘П4’ — печать на ТБПМ (цифровое печатающее устройство машины «Минск-22») чисел в восьмеричной системе. Буква «Ч» везде в спецификациях представляется цифрой 4.

‘П4⊔10’ — печать на ТБПМ чисел в десятичной системе счисления

‘П4⊔2—10’ — печать чисел на ТБПМ с предварительным переводом из двоичной в десятичную систему счисления

‘П4⊔АЦПУ’ — печать на АЦПУ (алфавитно-цифровом печатающем устройстве ЭВМ «Минск-22») значений строчных величин и строк.

‘ПРЛ’ — перфорация чисел на перфоленту в восьмеричной системе счисления.

‘ПРЛ⊔10’ — перфорация чисел на перфоленту в десятичной системе счисления.

‘ПРЛ⊔2—10’ — перфорация чисел на перфоленту с предварительным переводом из двоичной в десятичную систему счисления.

‘ПРЛ⊔С’ — перфорация на перфоленту значений строчных величин (посимвольно).

‘ПРК’ — перфорация на перфокарты восьмеричных чисел.

‘ПРК⊔10’ — перфорация на перфокарты десятичных чисел.

‘ПРК⊔2—10’ — перфорация на перфокарты чисел с предварительным переводом из двоичной в десятичную систему счисления.

‘ПРК⊔С’ — перфорация на перфокарты значений строчных величин (посимвольно).

‘ВВОДЛ’ — ввод чисел с перфоленты.

‘ВВОДЛ⊔10—2’ — ввод чисел с перфоленты с переводом из десятичной системы в двоичную систему счисления.

‘ВВОДЛ⊔С’ — ввод с перфоленты значений строчных величин.

‘ВВОДК’ — ввод с перфокарт восьмеричных чисел.

‘ВВОДК␣10’ — ввод с перфокарт десятичных чисел.

‘ВВОДК␣10—2’ — ввод с перфокарт чисел с переводом их из десятичной в восьмеричную систему счисления.

‘ВВОДК␣С’ — ввод с перфокарт значений строчных величин.

В операторах процедур-кодов со всеми вышеперечисленными спецификациями число фактических параметров может быть любым.

‘Д’ — подпрограмма деления целых чисел с выдачей частного и остатка; в списке фактических параметров задаются кроме спецификации процедуры-кода четыре параметра: первый — делимое, второй — делитель, третий — частное, четвертый — остаток.

Делимое и делитель могут быть целыми без знака или простыми переменными типа целый, не имеющими форматов. Операция определена только для неотрицательных значений делимого и делителя (для делителя, исключая нуль). Частное и остаток — простые переменные типа целый, без форматов.

‘ЗАПИСЬ’ — запись величины, находящейся в оперативной памяти (так называемой внутренней величины), заданной своим идентификатором, на магнитную ленту по адресу, заданному идентификатором внешней величины. Внешней величиной может быть внешний массив, внешняя составная величина, внешний составной массив или идентификатор внешнего составного массива с индексами.

‘4ТЕНИЕ’ — чтение с магнитной ленты значения внешней величины и присваивание ее значения внутренней величине.

В операторах процедур-кодов со спецификациями ‘ЗАПИСЬ’ И ‘4ТЕНИЕ’ в качестве фактических параметров могут стоять только групповые параметры, и число их должно быть четным. В каждой паре параметров первым должен стоять отправитель, а вторым — получатель.

‘ВЫХОД’ — сегментация программы; появление оператора процедуры-кода с данной спецификацией означает конец сегмента программы; после этого сегмента управление должно быть передано следующему сегменту программы, номер которого указан фактическим параметром этого оператора процедуры-кода.

В этом операторе используется один фактический параметр, который может быть либо целым числом без знака, либо переменной целого типа, не имеющей формата в своем описании. Каждое включение в программу оператора процедуры-кода со спецификацией ‘ВЫХОД’ означает выделение в программе еще одного сегмента, и, таким образом, n включений делят программу на n + 1 сегмент, которые нумеруются в порядке следования указанных операторов процедур-кодов.

Все эти сегменты программы будут записаны в процессе трансляции на магнитную ленту, начиная с адреса, заданного на пульте управления «Минск-22». При выполнении оттранслированной программы оператор процедуры-кода со спецификацией ‘ВЫХОД’ обеспечивает вызов с магнитной ленты сегмента программы с номером, заданным значением фактического параметра, и передачу управления первому оператору вызванного сегмента.

Вызов следующего сегмента всегда производится после окончания выполнения данного сегмента, последним оператором которого должен быть оператор процедуры кода со спецификацией ‘ВЫХОД’.

Передача управления частям программы, находящимся на МЛ (т. е. другим сегментам программы), может осуществляться только с помощью оператора процедуры-кода со спецификацией ‘ВЫХОД’, а не операторами перехода, которые могут обеспечивать переходы только внутри сегментов.

В операторах процедур-кодов с указанными спецификациями не допускается использование в качестве фактических параметров переменных типов **целый** и **вещественный,** являющихся элементами массивов и составных величин, а также величин, имеющих форматы в своих описаниях.

Библиотека стандартных программ (БСП) системы автоматизации программирования АЛГЭМ-СТ-3. БСП является неотъемлемой составной частью системы автоматизации программирования АЛГЭМ-СТ-3 для машины «Минск-22» и включает в себя подпрограммы с перечисленными выше спецификациями, а также другие подпрограммы, оформленные в соответствии с правилами составления стандартных подпрограмм для БСП машины «Минск-22». Обращение к стандартным подпрограммам при автоматическом программировании осуществляется с помощью операторов процедур-кодов. Выполнение стандартных подпрограмм производится в режиме интерпретации с помощью интерпретирующей системы, входящей в состав типового математического обеспечения машины «Минск-22». Имеется возможность включения в БСП новых подпрограмм, необходимых для решения задач того или иного класса.

Стандартные подпрограммы должны составляться с учетом следующих требований, указанных в описании библиотеки стандартных программ для машины «Минск-22», изданном ЦСУ СССР в 1963 г.:

1) Стандартные программы (СП) могут иметь не более трех уровней вхождения одних СП в другие. СП, которая не содержит в себе обращений к другим СП или внешним по отношению к ней блокам программы, называется простой. СП, содержащая в себе обращения к другим СП или к внешним блокам, называется сложной. СП (простая или сложная), обращение к которой производится непосредственно из основной рабочей программы, называется СП первого уровня. Сама основная программа образует нулевой уровень. Если СП первого уровня содержит обращение к некоторой СП, то последняя называется СП второго уровня. Если она сама является сложной и содержит обращение к другой СП, то последняя может быть только простой СП; она является СП третьего уровня. Как основная программа, так и СП любого уровня (кроме, естественно, третьего) могут содержать произвольное количество обращений к СП, однако общая длина основной программы, вырабатываемой транслятором СТ-3, вместе с СП, находящимися одновременно в МОЗУ в процессе работы, ограничена размером зоны МОЗУ, выделенной под размещение программы.

Рабочая программа, составляемая транслятором СТ-3, размещается вместе со своими константами в МОЗУ-1, а информация для рабочей программы — в МОЗУ-2, там же размещаются фактические параметры процедур-кодов.

2) СП составляются в действительных адресах и размещаются начиная с адреса 7000_8. Объем любой СП не должен превышать 1000_8 или 512_{10} команд. В качестве рабочих ячеек СП можно использовать область памяти от 00020_8 до 02077_8, а в качестве индексных ячеек — ячейки от 00001_8 до 00015_8. Следует заметить, что, когда в процессе работы сложной СП происходит обращение к другой СП, содержимое индексных ячеек запоминается и восстанавливается при возвращении к прежней СП. Содержимое же рабочих ячеек при переходах от одной СП к другой и обратно автоматически не запоминается и не восстанавливается, и поэтому сохранение необходимых промежуточных величин должно предусматриваться при составлении самих СП.

Вход в любую СП осуществляется через ячейку 7000, где размещается первая команда СП; выход из СП осуществляется через ячейку 00017, куда должна засылаться заранее команда, необходимая для продолжения работы программы. Выход из интерпретирующей программы в рабочую основную программу происходит по ячейке 00016. Аргументы, необходимые для работы СП, засылаются в рабочие ячейки, начиная с адреса 00030.

3) Каждая СП состоит из двух частей: перерабатываемой (адреса в командах этой части меняются при настройке СП на определенное место памяти) и неперерабатываемой (адреса в командах этой части не меняются). Изменяемые адреса в командах должны начинаться с 7, что является признаком для отнесения этих команд к перерабатываемой части СП. Исключение составляют условные числа в команде печати, адреса МЛ в командах обращения к МЛ и др., которые, хотя и начинаются с цифры 7, но не подвергаются изменениям при настройке СП.

Последняя ячейка каждой СП является информационной: она содержит в 12 левых разрядах (не считая знакового) длину перерабатываемой части СП, и для сложных СП — в 12 правых разрядах общую длину данной СП с учетом тех блоков (СП), к которым будет обращаться данная СП в процессе работы.

Предпоследняя ячейка каждой СП оставляется свободной; она предназначается для контрольной суммы СП.

Задание информации о фактических параметрах процедур-кодов. Перед обращением к СП в рабочие ячейки, начиная с адреса 00030, должна быть заслана информация о фактических параметрах данной СП. Такой информацией являются адреса и некоторые другие характеристики фактических параметров, но не сами значения фактических параметров. Все фактические параметры (за исключением фактических параметров, являющихся константами рабочей программы) находятся в МОЗУ-2. Признаком того, что данный фактический параметр находится в МОЗУ-1 (константы программы, размещаемые вместе с ней в МОЗУ-1), является единица в знаковом разряде ячейки, содержащей адрес указанного фактического параметра.

Информация о фактических параметрах задается числами типа **целый,** которые размещаются в правой части ячейки с фиксированной запятой после младшего разряда.

Информацией о простой переменной или переменной с индексами, не имеющих в описаниях форматов, является просто адрес ячейки, в которой находится данная переменная.

В качестве информации о групповых параметрах засылается начальный адрес группового параметра

(в одну ячейку) и количество ячеек, занимаемых этим групповым параметром (во вторую ячейку).

Значения простых переменных типа **целый** и **вещественный,** имеющих в описаниях форматы, представляются в двоично-десятичной системе (если в формате стоят 9-по-зиции) либо в двоичном коде (если в формате стоят 1-позиции) и располагаются, начиная с левого конца ячейки (не считая знакового разряда). Информацией о таких переменных является адрес ячейки, содержащей эту переменную (заносится в одну ячейку), и формат этой переменной (заносится во вторую ячейку). Кодирование значений переменной и формата производится следующим образом:

Символ	0	1	2	3	4	5	6	7	8	9	+	—	(	)	.	10
Двоичный код	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Информация о переменной с индексами, имеющей в описании формат, и о переменной типа **строчный** состоит из четырех чисел (четырех аргументов). Первый аргумент представляет собой номер без единицы разряда МОЗУ-2, начиная с которого располагается данный фактический параметр. Второй аргумент представляет собой количество разрядов, занимаемых данным фактическим параметром. Третий аргумент—двухразрядный двоичный код, определяющий тип переменной (**целый —** 01, **вещественный —** 10, **строчный** — 11). Первый, второй и третий аргументы располагаются в одной ячейке, причем третий аргумент занимает в ней 11-й и 12-й разряды.

Четвертым аргументом является формат переменной, который заносится в отдельную ячейку. Для переменных типа **строчный** формат записывается равным 0.

Значение переменной типа **целый** или **вещественный** и их форматы кодируются так, как показано было выше.

Значение переменных типа **строчный** кодируется посимвольно с помощью семиразрядного двоичного кода, принятого в АЦПУ; при этом в каждой ячейке помещается 5 символов. Для того чтобы по номеру разряда найти адрес ячейки, в которой он находится, необходимо в случае переменных типа **целый** и **вещественный** номер разряда разделить на 36ю, а в случае переменных типа **строчный** — на 35_{10}. Информация о строчной константе включает в себя адрес ячейки, начиная с которой располагается строка (адрес будет иметь единицу в знаковом разряде, так как константы находятся в МОЗУ-1, адрес записывается в одну ячейку), количество ячеек без единицы, занимаемых данной константой (записывается во вторую ячейку), и количество незанятых разрядов справа в последней ячейке (записывается в третью ячейку). Значения строчных переменных (строки) всегда размещаются начиная с левого конца одной ячейки и могут занимать подряд несколько ячеек.

Информацией о числовых константах (целых или вещественных) являются сами эти константы, причем в этом случае не имеет значения, где была эта константа в МОЗУ-1 или в МОЗУ-2.

Некоторые сведения о работе транслятора СТ-3. Приведем некоторые сведения о работе этого транслятора, необходимые для более эффективного использования входного языка при составлении исходных программ для указанной системы автоматизации программирования.

Как уже упоминалось, транслятор СТ-3 строит рабочие программы таким образом, что сами программы располагаются в МОЗУ-1, а информация для них в МОЗУ-2. При этом в МОЗУ-2 образуются два магазина данных, направленных навстречу друг другу. Массивы, составные переменные, составные массивы (идентификаторы которых не начинаются с буквы Я) и простые переменные типа **строчный** располагаются начиная с ячейки 00400 МОЗУ-2 в сторону увеличения адресов.

Каждый массив, составная переменная, составной массив и каждая группа переменных, объединенных в одно описание типа, располагаются с начала ячейки (а затем их элементы идут сплошным потоком). Внутри составных величин каждая величина, представленная в описании отдельным элементом (т. е. заканчивающаяся точкой с запятой), располагается с начала ячейки. В операциях обмена с магнитными лентами и операциях ввода и вывода всегда участвует целое число ячеек. Простые переменные типа **строчный** располагаются сплошным потоком символ за символом, без учета границ ячеек (но начиная с начала ячейки). Они кодируются семиразрядными кодами, принятыми в АЦПУ.

Простые переменные типа **целый** и **вещественный** (с форматами или без них) располагаются в отдельных ячейках, начиная с адреса 7777 МОЗУ-2 в сторону уменьшения адресов. При этом если в

описаниях этих переменных имеются форматы, то переменные располагаются начиная с левого конца ячейки (без учета знакового разряда). Они кодируются двоично-десятичными или двоичными кодами в зависимости от форматов.

После трансляции программы транслятор СТ-3 по заданию программиста может выдать на печать распределение памяти. При работе с большим количеством малоразрядных числовых величин, являющихся элементами массивов или составных, можно с помощью указанного распределения памяти упаковывать их по несколько величин в одну ячейку, снабжая их описания соответствующими форматами.

В трансляторе СТ-3 реализован блочный принцип построения программ, характерный для всех языков типа АЛГОЛ. Поэтому описанное выше распределение памяти производится транслятором для каждого блока программы. Зная эти принципы распределения памяти, можно иногда пользоваться переопределением данных одного блока для другого блока, т. е. использовать фактические значения данных, остающиеся в памяти ЭВМ (МОЗУ-2) после окончания работы данного блока, для работы следующего блока, присвоив им новые описания. При этом следует иметь в виду, что для двух соседних блоков, входящих внутрь одного внешнего блока, будут использоваться одни и те же начала отсчета ячеек в МОЗУ-2 при образовании верхнего и нижнего магазинов данных для этих блоков.

С целью сокращения длины рабочей программы, вырабатываемой транслятором, и упрощения процесса трансляции необходимо все массивы, имеющие одинаковые типы и граничные пары, в описаниях объединять в единые сегменты массивов, т. е. писать **массив** А, В, С, [1 : К]; а не **массив** А [1 : К], В [1 : К], С [1 : К].

С целью ускорения работы рабочих программ, вырабатываемых транслятором, необходимо сокращать в программах количество обращений к переменным с индексами, элементам составных переменных и составных массивов. В частности, если требуется обратиться к подобной переменной несколько раз (к одному и тому же ее значению), то целесообразно присвоить ее значение при первом обращении какой-нибудь промежуточной простой переменной, а затем обращаться к этой переменной. Также необходимо по возможности выносить обращения к подобным переменным из циклов. Это связано с тем, что при каждом обращении к переменной с индексами или элементу составной переменной или составного массива производится вычисление ее адреса, а если переменная имеет еще в своем описании формат, то вычисление адреса производится с точностью до номера разряда МОЗУ-2, начиная с которого размещается эта переменная; в последнем случае вычисляется также количество разрядов, занимаемых переменной. Эти вычисления увеличивают время работы программы.

Следующее замечание касается использования оператора процедуры-кода со спецификацией «П4␣АЦПУ», обеспечивающего печать на АЦПУ-128. С помощью этого оператора можно печатать только строки или значения переменных и массивов типа **строчный,** а также составных переменных и составных массивов, состоящих из элементов типа **строчный.** При этом каждый фактический параметр печатается с новой строки.

Следующее замечание касается возможности объединения нескольких участков программы, составленных автоматическим способом, в единую программу с использованием объединяющей управляющей программы (диспетчера), составленной вручную.

Составляемые транслятором рабочие программы обладают таким свойством, что при включении ключа 0100 на пульте управления машины «Минск-22» после окончания работы данной программы управление передается в ячейку 10400 (МОЗУ-2). (При выключенном ключе 0100 происходит останов программы после ее окончания.) Если, начиная с ячейки 10400, будет располагаться программа-диспетчер, то она может вызвать с МЛ другой участок программы и передать ей управление. Для вызова следующего участка необходимо, чтобы вызванная в данный момент программа также обеспечила запись программы-диспетчера (той же или другой) в МОЗУ-2, начиная с ячейки 10400 и т. д.

Большое значение при программировании экономических задач имеют процессы обработки больших массивов информации, включающих в себя сотни тысяч и миллионы чисел. Такие массивы записываются при вводе в ЭВМ на магнитные ленты, а в процессе обработки переписываются по частям (по отдельным блокам или зонам фиксированного размера) в оперативную память. Обычно в машине «Минск-22» обмен данными между накопителями на магнитной ленте и оперативной памятью осуществляется позонно. Каждая зона на МЛ содержит 2048 ячеек.

Для того чтобы запрограммировать на алгоритмическом языке АЛГЭМ-1 такие процессы обмена данными между МЛ и МОЗУ, можно воспользоваться следующим приемом. Большой массив информации, который должен храниться на МЛ, описывается в виде составного внешнего массива,

элементы которого должны иметь размер, равный тому блоку данных, который будет переписываться за один раз. В качестве внутреннего массива должен быть описан массив (или составная величина) с таким же размером. В описании составного внешнего массива, как упоминалось раньше, должны быть указаны кроме его идентификатора, начинающегося с буквы Я и содержащего начальный машинный адрес этого массива на магнитной ленте, и граничной пары, определяющей число элементов этого массива, также величина сдвига фактического адреса начала элемента относительно начального машинного адреса и количество ячеек в одном элементе. Для переписи с МЛ в МОЗУ используется оператор процедуры-кода со спецификацией ‘4ТЕНИЕ’, за которым ставится идентификатор внешнего составного массива с индексом в квадратных скобках и идентификатор внутреннего массива. При этом значение одного элемента внешнего составного массива будет присвоено всему внутреннему массиву (ячейка на ячейку). После этого может проводиться обработка внутреннего массива в соответствии с его описанием, которое, естественно, может определять любое расположение величин в ячейках или их частях.

Аналогично программируются и операции обратной переписи из МОЗУ на магнитную ленту с использованием оператора процедуры-кода со спецификацией ‘ЗАПИСЬ’.

Глава 4

АЛГОРИТМИЧЕСКИЙ ЯЗЫК АЛГЭМ-2

12. Отличия АЛГЭМ-2 от АЛГЭМ-1

АЛГЭМ-2 представляет собой попытку развития и расширения алгоритмического языка АЛГЭМ, построенного на базе АЛГОЛа. В языке АЛГЭМ-2 по сравнению с языком АЛГЭМ-1 сделаны следующие изменения:

1) Введен раздел процедур, которые строятся по типу процедур-блоков языка PL-1.

2) Введен раздел операторов обмена данными, которые также строятся по типу языка PL-1.

3) Сняты ограничения на оператор цикла, структуру и состав выражений и описаний типа и массивов, которые были введены в АЛГЭМ-1 по сравнению с АЛГОЛом.

4) Введено развернутое описание форматов переменных, соответствующее описаниям форматов языка АЛГЭК (с несущественными отличиями).

5) Введены логические многоразрядные переменные и массивы.

Заметим, что язык АЛГЭМ-2 не представляет собой реализованного варианта языка, а служит лишь прототипом для ознакомления с наиболее существенными сторонами подобных языков.

Изложение языка АЛГЭМ-2 мы будем вести следующим образом. Новые разделы и разделы языка, имеющие существенные отличия по сравнению с АЛГОЛом или АЛГЭМ-1, будут излагаться полностью; разделы, совпадающие полностью с АЛГОЛом или АЛГЭМ-1 (например, оператор цикла, переключатель и именующее выражение, арифметические выражения, операторы перехода и присваивания), вообще не будут описываться. Для разделов, имеющих некоторые несущественные отличия, эти отличия будут перечислены. Для описания языка АЛГЭМ-2 используется тот же метаязык БЭКУСА, что и при описании АЛГОЛа и АЛГЭМ-1.

Основные символы, идентификаторы, числа, логические константы и строки.

⟨основной символ⟩: : = ⟨буква⟩ | ⟨бит⟩ | ⟨цифра⟩ | ⟨кавычка⟩ | ⟨строчная кавычка⟩ | ⟨ограничитель⟩

⟨бит⟩ : : = 1 | 0

⟨кавычка⟩ : : = ”

⟨строчная кавычка⟩ : : = ’

Бит представляет собой логическое значение. Кавычка используется внутри строк для выделения наименований, а строчная кавычка — для ограничения строк. В составе ограничителей по сравнению с АЛГОЛом отсутствует спецификатор. К знакам операций добавлен знак текстовой операции:

⟨знак текстовой операции⟩ : : = ← | **текст** | **смысл**

В состав разделителей добавлены символы: **или** | **позиция** |*

Существенно расширено количество описателей в основном за счет описателей, используемых в операторах обмена:

⟨описатель⟩ : : = **логический** | **целый** | **вещественный** | **строчный** | **вид** | **массив** | **переключатель** | **метка** | **вход** | **ярлык** | **имя** | **ввод** | **вывод** | **ввод — вывод** | **поток** | **запись** | **последовательный** | **прямой** | **ключ** | **обратное** | **печать** | **строка** | **столбец** | **данные** | **из** | **ключ из** | **в** | **ключ в** | **пропуск** | **СП** | **СВ** |

Символ **процедура** исключен из числа описателей и введен в число скобок; он используется вместе с символом **конец** для описания процедур. Ограничители имеют фиксированный смысл, который в большинстве случаев приближается к общепринятому смыслу соответствующих символов или слов; точный смысл новых ограничителей будет указываться по мере необходимости. **СП** — стандартный массив для вывода, **СВ** — стандартный массив для ввода.

В АЛГЭМ-2 вводится понятие логической константы, которая определяется следующим образом:

⟨ряд битов⟩ : : = ⟨бит⟩ | ⟨ряд битов⟩ ⟨бит⟩

⟨логическая константа⟩:: = ’⟨ряд битов⟩’ В

Символ «В» используется в данном случае для того, чтобы отличать логические константы от строк. Для сокращения записи логических констант допускается конструкция следующего вида:

⟨повторитель ряда⟩ : : = ⟨пусто⟩ | (⟨целое без знака⟩)

Конструкция вида

⟨повторитель ряда⟩ ’ S’ В

где S — ряд битов, эквивалентна логической константе

$$\underbrace{\text{'S ... S' B}}_{n \text{ раз}}$$

где *n* — целое без знака в повторителе ряда.

Синтаксис строки выглядит следующим образом:

⟨строчный символ⟩ : : = ⟨буква⟩ | ⟨цифра⟩ | ⟨кавычка⟩ | ⟨ограничитель⟩

⟨открытая строка⟩ : : = ⟨строчный символ⟩ | ⟨открытая строка⟩ ⟨строчный символ⟩

⟨строка⟩ : : = ⟨повторитель ряда⟩ ' ⟨открытая строка⟩ '

Примеры.

(3)'Минск-23'

' 5K-''(((''''↑⊔ ×'''

' 28 ⊔ СЕНТЯБРЯ ⊔ 1948 г.'

Строки введены для того, чтобы дать возможность оперировать в программе произвольными последовательностями всех основных символов, кроме строчных кавычек. Открытые строки могут использоваться как значения переменных, имеющих по описанию тип **строчный,** строчных выражений и функций.

Конструкция вида ⟨(повторитель ряда)⟩'S', где S — некоторая открытая строка, эквивалентна строке $\underbrace{\text{'S . . . S',}}_{n \text{ раз.}}$ где *n* — целое без знака в повторителе ряда.

В АЛГЭМ-2 в состав величин вместо процедур включаются так называемые имена входов, заменяющие собой, по существу, идентификаторы процедур.

Область действия величины — это совокупность операторов и выражений, внутри которой имеет силу описание идентификатора, связанного с этой величиной.

Значение есть некоторое упорядоченное множество, элементами которого могут быть числа, логические константы и открытые строки либо метка. В частном случае это может быть одно число, одна логическая константа, одна открытая строка или совокупность чисел, совокупность Логических констант, совокупность открытых строк. В АЛГЭМ-2 в состав выражений по сравнению с АЛГОЛом добавляются строчные выражения. Существенным дополнением в АЛГЭМ-2 по сравнению с АЛГОЛом и АЛГЭМ-1 является возможность выделения отдельных позиций в составе переменных (отдельных ее разрядов). В связи с этим синтаксис переменной имеет следующий вид:

⟨начальная позиция⟩ : : = ⟨арифметическое выражение⟩

⟨конечная позиция⟩ : : = ⟨арифметическое выражение⟩

⟨граничная пара для позиций⟩ : : = ⟨начальная позиция⟩ : ⟨конечная позиция⟩

⟨список позиций⟩ : : = ⟨граничная пара для позиций⟩ | ⟨арифметическое выражение⟩ | ⟨список позиций⟩, ⟨граничная пара для позиций⟩ | ⟨список позиций⟩, ⟨арифметическое выражение⟩

⟨идентификатор переменной⟩ : : = ⟨идентификатор⟩

⟨простая переменная⟩ : : = ⟨идентификатор переменной⟩ | ⟨идентификатор переменной⟩ **[позиция** ⟨список позиций⟩]

⟨индексное выражение⟩ : : = ⟨арифметическое выражение⟩

⟨список индексов⟩ : : = ⟨индексное выражение⟩ | список индексов⟩, ⟨индексное выражение⟩

⟨идентификатор массива⟩ : : = ⟨идентификатор⟩

⟨переменная с индексами⟩ : : = ⟨идентификатор массива⟩ [⟨список индексов⟩] | ⟨идентификатор массива⟩ [⟨список индексов⟩ **позиция** ⟨список позиций⟩]

⟨элементарная переменная⟩ : : = ⟨простая переменная⟩ | ⟨переменная с индексами⟩

⟨идентификатор составной переменной⟩ : : = ⟨идентификатор⟩

⟨идентификатор составного массива⟩ : : = ⟨идентификатор⟩

⟨уточнение⟩ : : = · ⟨идентификатор составной переменной⟩ | · ⟨идентификатор составного массива⟩ [⟨список индексов⟩]

⟨переменная⟩ : : = ⟨элементарная переменная⟩ | ⟨переменная⟩ ⟨уточнение⟩

Примеры. A 17

Б2 **[позиция** 1,3 : 10]

Q [7,2]

X[SIN(N × Pl/2), Q[3, N, 4]]
AB1 [I, J]·P
M[K, L **позиция** 1 : 1 + 2,15] · N · R[J]

Переменная — это наименование, данное некоторому отдельному значению. Это значение можно использовать в выражениях для образования других значений, а также изменять посредством оператора присваивания. Тип значения данной переменной определяется описанием самой переменной или описанием соответствующего идентификатора массива.

Список позиций может относиться только к элементарной переменной. Последовательность позиций составляет значение данной переменной. Ссылка может производиться на десятичные или двоичные позиции числа или на позиции строки (в зависимости от формата в описании переменной). Если формат для числовой переменной в описании отсутствует, то значит определена ссылка только на двоичные разряды (с учетом длины конкретного машинного слова). Нумерация позиций идет слева направо.

Уточнения используются для исключения двусмысленностей, связанных с использованием одного и того же идентификатора для обозначения двух переменных, имеющих по описанию пересекающиеся области действия. Уточнения представляют собой ссылку на более крупные единицы информации, в состав которых входит данная переменная, и отделяются друг от друга и от элементарной переменной, которую они поясняют, точками. В АЛГЭМ-2, в отличие от АЛГЭМ-1, выписывать следует только необходимый минимум уточнений в порядке их иерархии.

Указатели функций имеют такой же смысл и синтаксис, как и в АЛГОЛе, за исключением того, что вместо идентификатора процедуры используется *имя входа* в процедуру и в качестве фактических параметров могут использоваться групповые параметры, имеющие такой же смысл, как и в АЛГЭМ-1.

⟨имя входа⟩: : = ⟨идентификатор⟩

⟨фактический параметр⟩: : = ⟨выражение⟩ | ⟨групповой параметр⟩ | ⟨имя входа⟩ | ⟨идентификатор переключателя⟩

⟨список фактических параметров⟩ : : = ⟨фактический параметр⟩ | ⟨список фактических параметров⟩, ⟨фактический параметр⟩

⟨совокупность фактических параметров⟩ : : = ⟨пусто⟩ | (⟨список фактических параметров⟩)

⟨указатель функции⟩ : : = ⟨имя входа⟩ ⟨совокупность фактических параметров⟩

Примеры.

SIN (X + Y)
РАСЧЕТ (ЗАВОД, МЕСЯЦ)

Указатели функций могут определять значения величин разных типов (целых, вещественных, логических, строчных).

В АЛГЭМ-2 к стандартным функциям, принятым в АЛГОЛе, добавляются следующие функции:

ДРОБЬ (В) —для дробной части значения В;

ЗНАК (В) — для определения знака значения В;

ПЕРВЫЙ (В) — для номера позиции первой слева единицы значения В;

СДВИГ (В1, В2) — сдвиг значения В1 на количество позиций, определяемое значением В2;

ДЛИНА (В) — значение функции ДЛИНА равно числу основных символов открытой строки, являющейся значением В.

Функции ДРОБЬ, ЗНАК, ПЕРВЫЙ оперируют аргументами типа **вещественный** и **целый.** Значение функции ДРОБЬ имеет тип **вещественный;** значения функций ЗНАК, ПЕРВЫЙ, ДЛИНА имеют тип **целый.**

Функция СДВИГ (Bl, B2) оперирует аргументами типа **вещественный, целый** и **строчный.** Тип значения функции СДВИГ (В1, В2) определяется типом аргумента В1; аргумент В2 должен быть типа **целый** или **вещественный** (в последнем случае величина сдвига определяется выражением entier (B2 + 0.5); значением функции entier (B) является наибольшее целое, не превосходящее значения В.

Арифметические и логические выражения имеют в АЛГЭМ-2 такой же смысл и синтаксис, как и в АЛГОЛе (с учетом сделанных выше дополнений, связанных с понятием переменной).

Некоторые отличия имеют место в логических выражениях. Логической константой, как было определено раньше, называется конечная последовательность битов, т. е. нулей и единиц. Переменная или указатель функции типа **логический** могут иметь формат в соответствующем описании. В этом случае количество битов в логической константе (т. е. ее длина) определяется форматом. Если формат в описании отсутствует, предполагается, что логическая константа имеет один двоичный разряд (один

бит).

Смысл операций отношения над значениями типа **вещественный** или **целый** очевиден.

Для операций отношения над значениями типа **строчный** принимается следующий порядок старшинства основных символов (некоторые из этих символов в языке АЛГЭМ-2 не используются):

0 1 2 3 4 5 6 7 8 9 + - / , . ⊔ $_{10}$ ↑ ( ) × = ; [ ] * ‘ ’ ≠
< > : А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х
Ц Ч Ш Щ Ы Ь Э Ю Я D F G I J L N Q R S U V W Z-
≤ ≥ ∨ ∧ ⊃ ⏋ ÷ ≡ ■ % ◊ | ¯ _ ! « № ← → ? ↓ ∅ ± ∇

При выполнении операции отношения над значениями типа **строчный** разной длины более короткое из них считается дополненным справа символами пробелов. Сравнение строчных символов выполняется в порядке слева направо.

При выполнении операции отношения над значениями типа **логический** разной длины более короткое из них считается дополненным справа нулями. Сравнение выполняется поразрядно слева направо.

Если операнды отношения имеют различные типы, то перед выполнением операции отношения должны выполняться соответствующие преобразования типов значений. Определяется следующий порядок старшинства типов:

первый — **вещественный;**
второй — **целый;**
третий — **строчный;**
четвертый — **логический.**

Из двух типов значений высшим считается тот, который имеет меньший порядок старшинства. Указанная функция преобразования преобразует значение операнда низшего типа к высшему.

Результат логической операции имеет тип **логический,** а длина результата равна наибольшей из длин операндов. Логическое выражение может являться правой частью оператора присваивания, в левой части которого должна быть логическая переменная.

Строчные величины имеют такой же смысл и назначение, что и в АЛГЭМ-1. Синтаксис строчного выражения в АЛГЭМ-2 предусматривает использование операции ← (присоединения) и условные строчные выражения:

⟨первичное строчное выражение⟩ : : = ⟨строка⟩ | ⟨указатель функции⟩ | ⟨переменная⟩ | (⟨строчное выражение⟩)

⟨простое строчное выражение⟩ : : = ⟨первичное строчное выражение⟩ | ⟨простое строчное выражение⟩ ← ⟨первичное строчное выражение⟩

⟨строчное выражение⟩ : : = ⟨простое строчное выражение⟩ | ⟨условие⟩ ⟨простое строчное выражение⟩ **иначе** ⟨строчное выражение⟩

Примеры

A1 [**позиция** K]
’ИВАНОВ’
СТАВКА [1]
если МАСШТАБ [N] < 5 **то** ’ТАШКЕНТ’ **иначе** ’МОСКВА’

Результатом операции присоединения (←) является последовательность символов значения левого операнда, за которой следует последовательность символов значения правого операнда. Например:

P : =’A’ ← ’B’

Значением P служит последовательность символов AB.

Строчное выражение может быть в правой части оператора присваивания, в левой части которого должна быть строчная переменная.

Именующие выражения в АЛГЭМ-2 совпадают с именующими выражениями в АЛГЭМ-1.

Особенности операторов и процедур в АЛГЭМ-2. Как известно, операторами алгоритмического языка называются автономные единицы действий определенного функционального назначения. В АЛГЭМ-2 вводятся некоторые операторы, которые выполняют роль описаний подобно тому, как это имеет место в языке PL-1. Операторы могут в явном виде осуществлять преобразование информации или изменять ход выполнения программы (переход к другому месту программы, присваивание нового

значения переменной и т. п.), а могут служить для введения в действие или запрещения обращений к определенным массивам, для изменения их наименований и характеристик, т.е. выполнять функции, сходные с функциями описаний.

Основное отличие операторов АЛГЭМ-2 от АЛГОЛа заключается в процедурах и операторах обмена данными. Процедуры в АЛГЭМ-2 строятся по типу процедур-блоков языка PL-1. Они включают в себя собственно процедуры и операторы процедур (указатели функций). Смысл операторов процедур (указателей функций) совпадает с АЛГОЛом. Процедуры в данном языке по существу аналогичны описаниям процедур в языке АЛГОЛ, но в отличие от АЛГОЛа они могут размещаться в любом месте программы (не обязательно в начале блоков, а также вне какого-либо блока). При последовательном выполнении операторов встречающиеся процедуры пропускаются; обращения к ним могут осуществляться только с помощью операторов процедур или указателей функций.

В состав основных операторов в АЛГЭМ-2 добавляется оператор обмена. Понятия основного оператора, безусловного оператора, оператора цикла, составного оператора, блока, а также понятие совокупности формальных параметров в АЛГЭМ-2 совпадают с аналогичными понятиями в АЛГОЛе. Программа в АЛГЭМ-2 может состоять только из процедур, которые называются внешними. В состав процедур могут входить блоки, составные операторы, процедуры (не внешние) и другие операторы, а также описания. Синтаксис процедуры в АЛГЭМ-2 имеет вид:

⟨элемент процедуры⟩ : : = ⟨оператор⟩ | ⟨вторичная точка входа⟩

⟨имя входа⟩: : = ⟨идентификатор⟩

⟨вторичная точка входа⟩ : : = ⟨имя входа⟩ : **вход** ⟨совокупность формальных параметров⟩ | ⟨имя входа⟩ : ⟨вторичная точка входа⟩

⟨конец процедуры⟩ : : = ⟨элемент процедуры⟩ **конец** | ⟨элемент процедуры⟩; ⟨конец процедуры⟩

⟨спецификация⟩ : : = **метка** ⟨список меток⟩ | **вход** ⟨список имен входов⟩ | тип **вход** ⟨список имен входов⟩

⟨список меток⟩: : = ⟨метка⟩ | ⟨список меток⟩, ⟨метка⟩

⟨список имен входов⟩ : : = ⟨имя входа⟩ | ⟨список имен входов⟩, ⟨имя входа⟩

⟨описание в процедуре⟩ : : = ⟨описание⟩ | ⟨спецификация ⟩

⟨начало процедуры⟩ : : = **процедура** ⟨совокупность формальных параметров⟩ ; ⟨описание в процедуре⟩ | ⟨начало процедуры⟩; ⟨описание в процедуре⟩

⟨непомеченная процедура⟩ : : = ⟨начало процедуры⟩; ⟨конец процедуры⟩

⟨процедура⟩ : : = ⟨имя входа⟩ : ⟨непомеченная процедура ⟩ | ⟨имя входа⟩ : ⟨процедура⟩

⟨программа⟩ : : = ⟨процедура⟩ | ⟨программа⟩ ; ⟨процедура ⟩

Этот синтаксис можно проиллюстрировать следующим образом: обозначим произвольные операторы, описания, метки и формальные параметры буквами S, D, L и F соответственно. Процедура в общем случае имеет вид:

L : L : . . . : L : **процедура** (F, F, . . ., F); D; . . . ; L;
S; S; . . . ; S;
L : L : **вход** (F, F, . . ., F);
S; S; . . .; S;
L : L : **вход** (F, F, . . ., F); S; S; . . .; S
конец

При этом каждый из операторов может, в свою очередь, быть составным оператором, блоком и процедурой.

Пример записи процедуры:

```
R : процедура  (Z);   . . .
M : процедура (X, Y); . . . L: вход (x); . . .
K (A + B, D + E);
. . . P(Z0);   . . .
конец;
. . .
M (5, 6);
. . .
K : процедура (Z,  T);   ...  L (XO);
M (1, 2);  ... P:  вход (Z);
```

. . .
конец
. . .
K (7, 8);
. . .
конец процедуры R

Процедуры могут быть внешними и внутренними. Внешней процедурой называется процедура, не являющаяся оператором никакого другого блока или процедуры. Программа представляет собой последовательность внешних процедур, разделенных точками с запятой. Программа начинает работать с заданной процедуры. Внешние процедуры могут транслироваться независимо друг от друга. Каждый блок и процедура автоматически вводят новый уровень обозначений. Это реализуется следующим образом. Любой идентификатор, встречающийся внутри блока (процедуры), можно определить посредством соответствующего описания либо как локальный в данном блоке (процедуре), либо как внешний.

Идентификатор, описанный в блоке или процедуре с использованием символа **внешний,** считается описанным во внешней процедуре, одним из операторов которой служит данный блок или процедура. Областью действия такого идентификатора считается совокупность всех внешних процедур, содержащих описания этого идентификатора как внешнего, за исключением тех процедур и блоков, в которых этот идентификатор описан не как внешний. Метка считается описанной в заголовке наименьшего блока (процедуры), включающего в себя оператор, перед которым стоит эта метка.

Если обращение к процедуре производится с помощью указателя функции, то все метки, помещенные перед основным символом **процедура,** должны быть описаны в начале процедуры, так как эти метки будут представлять результаты действия процедуры. Необходимо, чтобы в процедуре находился один или несколько явных операторов присваивания с меткой (являющейся именем входа этой процедуры) в левой части; по крайней мере один из них должен быть выполнен.

Именами входов называются метки, т. е. идентификаторы, помещенные перед основными символами **процедура** или **вход.** Имя входа, помещенное перед основным символом **процедура,** называется первичной точкой входа; имя входа, помещенное перед основным символом **вход,** называется вторичной точкой входа.

Имена входов процедуры (ее метки) считаются локальными в непосредственно охватывающем эту процедуру блоке (процедуре). Имена входов внешних процедур имеют областью действия всю программу (за исключением тех блоков и процедур, в которых эти идентификаторы описаны заново).

Совокупность формальных параметров вторичных точек входа должна являться подмножеством совокупности формальных параметров соответствующей первичной точки входа. Все формальные параметры, перечисленные в списке формальных параметров первичной точки входа, должны быть описаны в начале этой процедуры. Таким образом, в процедуре могут использоваться три группы величин: формальные параметры, локальные величины, описанные в этой процедуре, и глобальные величины, описанные в этой процедуре как внешние. Все формальные параметры должны быть описаны в начале процедуры, причем некоторые из них могут иметь еще и описатель **значение,** показывающий, что их вызов должен осуществляться по значению.

Глобальные величины могут быть также описаны в начале процедуры с помощью описателя **внешний.** В этом случае глобальные величины становятся внешними, т. е. как бы описанными в самом внешнем (фиктивном) блоке, представляющем всю программу. Эти внешние описания служат для обеспечения независимости трансляции блоков и процедур. В блоках тоже могут в виде внешних величин описываться глобальные величины. Различие между понятиями внешний и глобальный состоит в том, что глобальные величины являются глобальными относительно данного блока и могут быть локальными в другом блоке, охватывающем данный блок.

Внешние величины являются абсолютно глобальными, т. е. один и тот же идентификатор, описанный с помощью описателя внешний, представляет собой один и тот же объект независимо от того, на каком уровне иерархии (вхождения) находится тот блок или процедура, в которой имеется это описание. Если некоторая процедура, имеющая описание внешних величин, используется во внутреннем блоке, имеющем описания тех же идентификаторов не как внешних величин, то при выполнении этой процедуры используются внешние величины, а не те, которые описаны в этом блоке, включающем в себя процедуру. Все внешние величины, взятые из всех блоков и процедур, образуют единую систему внешних величин, общую для всей программы. Различные описания (т. е. описания, находящиеся в

разных процедурах и блоках) одних и тех же внешних величин не должны противоречить друг другу.

Имена входов процедур, определяющих значения указателей функций, могут и должны встречаться в качестве переменных в левых частях операторов присваивания. В отношении оператора присваивания в АЛГЭМ-2 заметим, что все переменные и имена входов процедур, перечисленные в списке левой части, должны иметь по описанию один и тот же тип. Тип, соответствующий имени входа процедуры, задается его описанием, которое находится в составе описаний данной процедуры.

Выражение в правой части оператора присваивания может быть любого типа. Если тип выражения отличается от типа переменных списка левой части, то считается, что после вычисления значения выражения автоматически включается соответствующая функция преобразования. Для тех переменных и имен входов в списке левой части, которые имеют в соответствующем описании формат, присваивание производится с редактированием (т. е. с учетом формата). Рассмотренные выше процедуры, построенные по типу процедур-блоков языка PL-1, обладают следующими преимуществами по сравнению с процедурами АЛГОЛа.

1. Возможность расположения процедуры в любом месте программы (не обязательно в начале блока) обеспечивает удобство в написании программы и использовании процедур. Каждая процедура пишется в программе тогда, когда она впервые понадобилась.

2. Введение в процедуры наряду с первичными также вторичных точек входа обеспечивает гибкость в использовании процедур и их отдельных частей.

3. Требование обязательного описания в начале процедуры всех ее формальных параметров облегчает процесс построения трансляторов и процесс трансляции, так как при этом исключается необходимость извлекать сведения о неописанных формальных параметрах из тела процедуры.

4. Введение внешних процедур и внешних величин обеспечивает возможность независимой трансляции внешних процедур. Последнее свойство весьма важно для построения сложных и больших программ, так как оно позволяет осуществлять их сегментацию на автономные части — внешние процедуры, которые будут транслироваться независимо друг от друга.

Оператор цикла АЛГЭМ-2 отличается от этого оператора в АЛГОЛе только тем, что в нем допускается такая конструкция, когда вместо оператора стоит именующее выражение, показывающее оператор, который должен выполняться в цикле:

⟨оператор цикла⟩ : : = ⟨заголовок цикла⟩ ⟨оператор⟩ | ⟨заголовок цикла⟩ ⟨именующее выражение⟩ | ⟨метка⟩ : ⟨оператор цикла⟩

Пример.

для Q : = 1 **шаг** 1 **до** N, N + 15, 2, N + 2 **пока** A < B **цикл** M5

Заголовок цикла заставляет выполняться нуль или более раз либо следующий за ним оператор, либо оператор, метка которого служит значением именующего выражения, помещенного после заголовка цикла.

Оператор процедуры и указатель функции в АЛГЭМ-2, как уже говорилось, совпадают с соответствующими конструкциями АЛГОЛа, только вместо идентификаторов процедур ставятся имена входов этих процедур:

⟨оператор процедуры⟩ : : = ⟨имя входа⟩ ⟨совокупность фактических параметров⟩

Список фактических параметров оператора процедуры должен иметь то же число членов, что и список формальных параметров соответствующей точки входа. Соответствие получается путем сопоставления членов этих двух списков в одном и том же порядке. Класс, структура и тип каждого фактического параметра должны быть совместимы с классом, структурой и типом соответствующего формального параметра. Формальному параметру, входящему в процедуру в качестве идентификатора массива, составной переменной или составного массива, может соответствовать в качестве фактического параметра только идентификатор массива, составной переменной или составного массива соответственно (возможно, с уточнениями). При этом структуры формального и соответствующего ему фактического параметра должны совпадать с точностью до числа и состава уровней, до размерностей массивов и значений границ. Если в описании формального параметра массива в качестве границ использованы символы *, это значит, что значениями этих границ становятся значения соответствующих границ фактического параметра массива.

13. Операторы обмена

Операторы обмена служат для обмена данными между внутренней и внешней памятью (на магнитных лентах, дисках, на перфокартах, лентах и др.). Доступ к данным, расположенным во внешней памяти, возможен только посредством операторов обмена:

⟨оператор обмена⟩ : : = ⟨оператор ОТКРЫТЬ⟩ |
⟨оператор ЗАКРЫТЬ⟩ | ⟨оператор ВЫБРАТЬ⟩|
⟨оператор ПОСЛАТЬ⟩ | ⟨оператор ПЕЧАТЬ⟩ |
⟨оператор ЧИТАТЬ⟩ | ⟨оператор ПИСАТЬ⟩ |
⟨оператор УДАЛИТЬ⟩ | ⟨оператор ПОДВЕСТИ⟩

Данные, находящиеся во внешней памяти, будем называть внешними массивами.

При выборке и чтении передача данных идет из внешней памяти во внутреннюю, при посылке, записи и печати — из внутренней во внешнюю. Прочие операторы являются служебными и выполняют некоторые подготовительные или вспомогательные операции. Рассмотрим синтаксис и семантику перечисленных выше операторов обмена.

Оператор ОТКРЫТЬ **и оператор** ЗАКРЫТЬ:

Синтаксис.

⟨внешний массив⟩ : : = ⟨идентификатор⟩ | **СП** | **СВ**
⟨ярлык массива⟩ : : = ⟨пусто⟩ | **ярлык** (⟨выражение⟩)
⟨переименование⟩ : : = ⟨пусто⟩ | **имя** (⟨выражение⟩)
⟨вид обмена⟩ : : = **ввод | вывод | ввод — вывод**
⟨вид массива⟩ : : = **поток | запись**
⟨вид доступа⟩ : : = ⟨пусто⟩ | **последовательный** | **прямой**
⟨указание ключа⟩ : : = ⟨пусто⟩ | **ключ** (⟨индексное выражение⟩)
⟨направление⟩ : : = ⟨пусто⟩ | **обратное**
⟨признак печати⟩ : : = ⟨пусто⟩ | **печать**
⟨характеристика массива⟩ : : = ⟨ярлык массива⟩ | ⟨переименование⟩ | ⟨вид обмена⟩ | ⟨вид массива⟩ | ⟨вид доступа ⟩ | ⟨указание ключа⟩ | ⟨направление⟩ | ⟨признак печати⟩
⟨список характеристик⟩ : : = ⟨характеристика массива⟩ | ⟨список характеристик⟩ ⟨характеристика массива⟩
⟨оператор ОТКРЫТЬ⟩ : : = ОТКРЫТЬ (⟨внешний массив⟩ ⟨список характеристик⟩
⟨список для закрытия⟩ : : = ⟨внешний массив⟩ ⟨ярлык массива⟩ | ⟨список для закрытия⟩, ⟨внешний массив⟩ ⟨ярлык массива⟩
⟨оператор ЗАКРЫТЬ⟩: : = ЗАКРЫТЬ (⟨список для закрытия⟩)

Примеры.

ОТКРЫТЬ (Р **ярлык** АНТЕЙ **имя** ’ТУРБОВИНТОВОЙ’ **ввод — вывод запись прямой ключ** (10))
ЗАКРЫТЬ (ВЕДОМОСТЬ, ТАБЛИЦА А, Р **ярлык** ’1956’)

Семантика. Оператор ОТКРЫТЬ представляет собой описание внешнего массива в том смысле, что появление его **в** программе определяет характеристики последующих операций обмена, которые будут выполняться над данным массивом. Ни одна из характеристик не может появляться в списке характеристик более одного раза.

Внешний массив может иметь ярлык, в котором приводятся сведения, необходимые для контроля доступа и отождествления указанного массива, в частности, в начале массива записывается головная метка, а в конце массива — хвостовая метка. Отождествление внешнего массива осуществляется следующим образом:

1) для массивов с характеристикой **ввод** производится сверка значения переменной (в данном случае в качестве выражения может выступать лишь переменная типа строчный), помещенной после символа **ярлык** в операторе ОТКРЫТЬ, с меткой, помещенной в начале внешнего массива (головной меткой). В случае несовпадения выдается соответствующий сигнал, а также выполняется присваивание переменной, помещенной после символа ярлык, фактического значения головной метки внешнего массива;

2) для массивов с характеристикой **вывод** значение выражения, стоящего после символа **ярлык** в операторе ОТКРЫТЬ, преобразуется к типу **строчный.** Полученное значение становится головной

меткой записываемого внешнего массива;

3) для массивов с характеристикой **ввод — вывод** выполняются те же действия, что и для массивов с характеристикой **ввод.** В том случае, если внешний массив имеет характеристику **обратное** (совместимую только с характеристикой **ввод),** выполняются действия, перечисленные в п. 1, с той разницей, что сравниваемая метка является хвостовой, т. е. находится в конце внешнего массива.

Наличие характеристики (переименование) приводит к следующим действиям: вычисляется и преобразуется к типу **строчный** (если оно нестрочное) значение выражения, стоящего после символа **имя** в операторе ОТКРЫТЬ. Полученная открытая строка будет новым идентификатором для данного внешнего массива; эта информация получает характеристики, перечисленные в данном операторе ОТКРЫТЬ. Если внешний массив, указанный слева от символа **имя,** к этому моменту еще не был открыт, он открывается автоматически, т. е. к нему допускаются обращения с помощью последующих операторов программы.

Вид обмена используется для задания режима работы внешнего массива. **Ввод** означает, что данный внешний массив может быть использован только в операциях ввода и чтения; вывод и запись блокируются. **Вывод** означает, что соответствующий внешний массив может использоваться лишь в операциях вывода, записи и печати. Характеристика **ввод — вывод** задает универсальный режим работы.

Характеристика (вид массива) определяет структуру внешнего массива. **Поток** задает внешний массив как непрерывную последовательность элементов данных. Любое обращение к такому внешнему массиву с помощью операторов обмена вызывает передачу данных в массив (из массива), начиная с первого элемента данных. **Запись** определяет внешний массив как совокупность порций (записей) данных. В динамике работы программы в зависимости от состояния вычислительной системы и от вида доступа один и тот же оператор обмена может вызывать обращения к различным порциям (записям) внешнего массива.

Вид доступа определяет способ обращения к отдельным записям внешнего массива с характеристикой **запись. Прямой** означает, что обращение к записям внешнего массива производится непосредственно по ключу (т. е. по значению признака записи). **Последовательный** означает, что производится лишь последовательное обращение к записям внешнего массива в соответствии с порядком их (физического) расположения на носителе.

Указание ключа в массиве записей означает, что каждая запись внешнего массива должна быть снабжена ключом (признаком), представляющим собой текст заданной длины. Длина текста задается значением индексного выражения, помещенного после символа **ключ. Прямой** массив всегда должен иметь ключ. Последовательный массив может иметь ключ.

Характеристика **обратное** может быть задана только для последовательных массивов, имеющих характеристику **ввод.** Она указывает, что считывание записей происходит в обратном порядке, т. е. в направлении от последней записи к первой.

Характеристика **печать** указывается только для массивов вида **поток,** имеющих характеристику **вывод.**

Оператор ЗАКРЫТЬ лишает внешний массив характеристик, приписанных ему в момент появления в программе соответствующего оператора ОТКРЫТЬ. Областью действия характеристик внешнего массива, перечисленных в операторе ОТКРЫТЬ, является совокупность операторов обмена, находящихся между оператором ОТКРЫТЬ для данного массива и соответствующим ему оператором ЗАКРЫТЬ. Закрытие внешнего массива не означает его уничтожения, а означает только запрещение доступа к этому массиву до его открытия. Повторное открытие открытого массива, закрытие закрытого массива, а также закрытие неоткрытого массива эквивалентны пустому оператору. Ярлык закрываемого внешнего массива используется для контроля и отождествления этого массива следующим образом:

1) для массивов с характеристикой **ввод** производится сверка значения переменной (в данном случае в качестве выражения может выступать лишь переменная типа **строчный),** помещенной после символа **ярлык** в операторе ЗАКРЫТЬ, с меткой, помещенной в конце внешнего массива (хвостовой меткой). В случае несовпадения выдается соответствующий сигнал, а также производится присваивание переменной, помещенной после символа **ярлык,** фактического значения хвостовой метки внешнего массива;

2) для массивов с характеристикой **вывод** значение выражения, стоящего после символа **ярлык** в операторе ЗАКРЫТЬ, преобразуется к типу **строчный.** Полученное значение становится хвостовой меткой закрываемого внешнего массива;

3) для массивов с характеристикой **ввод — вывод** выполняются те же действия, что и для массивов с характеристикой **ввод.**

В том случае, если внешний массив с характеристикой **ввод** имеет еще характеристику **обратное,** то при закрытии массива выполняются действия, перечисленные в п. 1, за исключением того, что сравниваемая метка является головной и находится в начале внешнего массива. Оператор ОТКРЫТЬ служит для открытия только одного внешнего массива, в то время как оператор ЗАКРЫТЬ может использоваться для закрытия сразу нескольких внешних массивов.

Операторы обмена элементами данных. Операторы ВЫБРАТЬ, ПОСЛАТЬ, ПЕЧАТЬ служат для обмена элементами данных; их разрешается употреблять только для внешних массивов, имеющих характеристику **поток.**

Синтаксис.

⟨внутренний массив⟩ : : = ⟨выражение⟩ | ⟨групповой параметр⟩

⟨список внутренних массивов⟩ : : = ⟨внутренний массив ⟩ | ⟨список внутренних массивов⟩, ⟨внутренний массив⟩

⟨совокупность внутренних массивов⟩ : : = ⟨внутренний массив⟩ | (⟨список внутренних массивов⟩) | (⟨список внутренних массивов⟩) **для** ⟨параметр цикла⟩ : = ⟨список цикла⟩

⟨спецификация обмена⟩ : : = ⟨строчное выражение⟩

⟨знак размещения⟩ : : = ␣ | / | ×

⟨указатель размещения⟩ : : = ⟨повторитель⟩ ⟨знак размещения⟩ |

⟨элемент списка форматов⟩ : : = ⟨повторитель⟩ ⟨формат⟩ | ⟨указатель размещения⟩ | X (⟨индексное выражение⟩) | **строка** (⟨индексное выражение⟩)| **столбец** (⟨индексное выражение⟩)

⟨список форматов⟩ : : = ⟨элемент списка форматов⟩ |

⟨список форматов⟩, ⟨элемент списка форматов⟩

⟨совокупность форматов⟩ : : = ⟨пусто⟩ | (⟨список форматов⟩)

⟨форма данных обмена⟩ : : = ⟨совокупность форматов⟩ | ⟨совокупность форматов⟩ **данные**

⟨элемент обмена⟩ : : = ⟨совокупность внутренних массивов⟩ ⟨форма данных обмена⟩

⟨параметр обмена⟩ : : = ⟨спецификации обмена⟩, ⟨внешний массив⟩, ⟨элемент обмена⟩

⟨список параметров обмена⟩ : : = ⟨параметр обмена⟩ | ⟨список параметров обмена⟩; ⟨параметр обмена⟩

⟨размер страницы⟩): : = ⟨пусто⟩ | ⟨индексное выражение⟩ : ⟨индексное выражение⟩,

⟨оператор ВЫБРАТЬ⟩ ::= ВЫБРАТЬ (⟨список параметров обмена⟩)

⟨оператор ПОСЛАТЬ⟩: : = ПОСЛАТЬ (⟨список параметров обмена⟩)

⟨оператор ПЕЧАТЬ⟩: : = ПЕЧАТЬ (⟨размер страницы⟩ ⟨список параметров обмена⟩)

Примеры.

ВЫБРАТЬ (’10 — 2’, P, A [I, J] **для** I : = 1 **шаг** 1 **до** 9 **для** I J : = 1 **шаг** 1 **до** 5)

ПОСЛАТЬ (’2—10’, P, (A [I, J], B [J, I]) **для** I : = 1 **шаг** 1 **до** 9 **для** J : = 1 **шаг** 1 **до** 3)

ВЫБРАТЬ (’ПЛ’, К, ((A, B, C,) **для** I : = 1 **шаг** 1 **до** 7) (9.99, (3)C(5)))

ПОСЛАТЬ (’МЛ’, К, A, (B, C) **для** I := 1 **шаг** 1 **до** 10 **данные)**

ВЫБРАТЬ (’ПЕРФОКАРТЫ’, H, A (9.99))

ПЕЧАТЬ (25 : 35, 128, П, (М, К) (X, (2)C(2)))

Семантика. Внутренний массив определяет упорядоченное множество, элементами которого являются числа, логические значения или открытые строки, образующие значение массива, составной переменной или составного массива.

Порядок расположения элементарных переменных в составных переменных и составных массивах определяется порядком их расположения в описаниях путем перечисления слева направо. Порядок, в котором располагаются элементы массива, соответствует лексикографическому порядку значений индексов, т. е.

$A[K_1\ K_2,...,K_m]$ предшествует $A\ [J_1\ J_2, ..., J_m]$ если

$K_1 = 1, \quad (i = 1, 2, ..., p-1)$

$K_p < J_p \quad (1 \le p \le m)$

Количество и порядок передаваемых элементов динамически определяется описанием внутреннего массива, за исключением того случая, когда при вводе данных совокупность внутренних массивов сопровождается символом данные (см. ниже).

Заголовок цикла, помещаемый после совокупности внутренних массивов, определяет итеративный

процесс, применяемый последовательно, слева направо, ко всем внутренним массивам из совокупности внутренних массивов согласно правилам, установленным для заголовка цикла. Например, оператор

ВЫБРАТЬ (’10—2’, P, (A, B) для i : = 1 **шаг** 1 до 3) эквивалентен оператору

ВЫБРАТЬ ’10—2’, P, (A, A, A, B, B, B)) или оператор

ПЕЧАТЬ (’АЦПУ’, M, A [i] **для** i : = 1 **шаг** 2 **до** 6) эквивалентен оператору

ПЕЧАТЬ (’АЦПУ, M, (A [1], A [3], A [5]))

В общем случае одному внешнему массиву соответствует совокупность внутренних массивов. Порядок заполнения (исчерпания) внешних массивов при выполнении оператора ВЫБРАТЬ, ПОСЛАТЬ и ПЕЧАТЬ (т. е. операторов, оперирующих с внешними массивами типа **поток)** можно пояснить схемой на рис. 16.

Форматы обмена служат для преобразования потока данных, участвующих в обмене.

Элемент списка форматов вызывает соответствующее редактирование величины, являющейся элементом внешнего массива. В случае присваивания этого значения величины соответствующему элементу внутреннего массива производится ее редактирование в соответствии с форматом, заданным в описании внутреннего массива.

При выполнении операторов обмена для элементов потока данных процесс передачи данных управляется описаниями соответствующих внутренних массивов и списком форматов, имеющимся в операторе обмена.

Количество передаваемых элементов определяется их числом в соответствующем внутреннем массиве с учетом выборки, осуществляемой с помощью заголовка цикла, если он есть. Например, если в операторе ВЫБРАТЬ указан просто идентификатор внутреннего массива, то производится последовательно присваивание значений очередных элементов внешнего массива очередным элементам

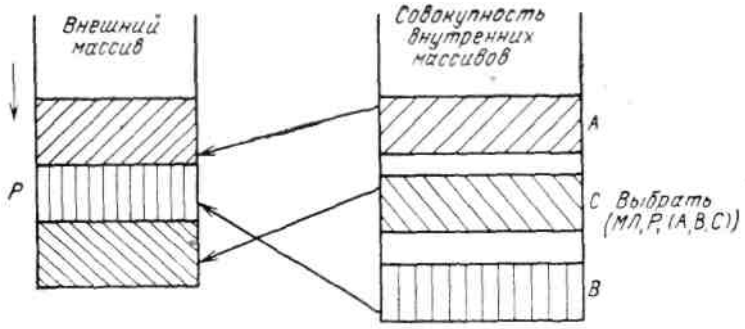


Рис. 16. Пример заполнения внешнего массива.

данного внутреннего массива (всем по очереди с учетом редактирования; при этом используется только один очередной формат без переходов по списку форматов). Если же указан идентификатор составной переменной, то производится последовательно присваивание значений очередных элементов внешнего массива элементарным компонентам этой составной переменной (при этом происходит переход по списку форматов).

Процесс обмена заканчивается по исчерпании всех элементов внутренних массивов. Если внешний массив исчерпается раньше, то процесс продолжается циклически, т. е. с начала внешнего массива. Если внутренние массивы исчерпаются раньше, то процесс обмена закончится; этот случай представляет собой естественное окончание процесса обмена. В операторах обмена ВЫБРАТЬ, ПОСЛАТЬ, ПЕЧАТЬ могут указываться форматы данных, а могут и не указываться. Если формат указан, то данные редактируются при передаче в соответствии с этим форматом. Если в операторе формат не указан, то используется формат переменной, указанный в описании этой переменной. Если указан формат в операторе и имеется формат в описании этой же переменной, то при обращении к внешнему массиву используется формат, заданный в операторе, а при обращении к внутреннему массиву — формат, заданный в его описании. Если, например, идет запись во внутренний массив, то оба формата используются следующим образом: выборка из внешнего массива идет по формату, указанному в операторе ВЫБРАТЬ, а затем идет запись во внутренний массив по формату, указанному в описании этого массива. В частном случае эти форматы могут совпадать. Если же они не совпадают, то могут быть необходимы преобразования типов и возможны усечения, подавления, сдвиги, которые сделают указанный процесс обмена необратимым. Если вообще отсутствуют форматы, т. е. они не

указаны ни в операторе обмена, ни в описании переменной (это может быть для числовых данных), то передача данных производится в принятой для конкретной машины форме представления чисел (с фиксированной или плавающей запятой).

Таким образом, форматы числовых переменных могут указываться либо в их описаниях, либо в операторах обмена, либо и там и там. Форматы строчных переменных обязательно указываются в описаниях и могут указываться в операторах обмена. Если формат числовой переменной (массива) указан только в операторе обмена, то значения этой переменной (массива) внутри машины должны храниться в форме, принятой для представления чисел данного типа в этой машине, и только при обмене преобразуются в соответствии с форматом, заданным в операторе обмена. Таким образом, форматы, указанные в операторах обмена, всегда относятся к внешнему массиву. По этим форматам идет или занесение значений во внешний массив или извлечение их из внешнего массива. Внутренний массив определяется всегда или форматом, заданным в его описании, или принятым в машине конкретным представлением числовых данных (для числовых величин, не имеющих форматов в описаниях). При обмене элементами данных в качестве таких элементов принимаются значения отдельных элементарных переменных (простых или с индексами).

Для составных переменных или массивов такими элементами являются их элементарные компоненты. При этом автоматически вводится дополнительное измерение, перечисляющее подряд все элементарные компоненты, входящие в их состав, независимо от их типов и вхождений в более крупные (тоже внутренние) составные переменные или массивы. Если, например, на некотором уровне в составную переменную в качестве ее компоненты входит массив (несоставной), то он представляет собой одну элементарную компоненту и ему ставится в соответствие один формат в списке форматов. Если же элементы этого массива сами являются составными (т. е. внутри составной переменной имеется составной массив), то такой составной массив развертывается в последовательность его компонент и каждая элементарная компонента, представляющая собой несоставной массив, рассматривается как отдельный член в списке внутренних массивов. Независимо от фактической глубины строения составных переменных каждая из них полностью развертывается в последовательность значений элементарных компонент, которые нумеруются подряд. При этом не учитывается наличие составных переменных различных промежуточных уровней, так как составные переменные форматов не имеют.

Отдельные выражения (переменные) или идентификаторы массивов (не составных), записанные без индексов, выступают как отдельные члены списка внутренних массивов и каждому из них соответствует один формат. При этом все элементы массива будут иметь этот формат, так как всему массиву в последовательности форматов соответствует один очередной формат. Если в операторе обмена указан идентификатор составного массива без индексов, то он развертывается как составная переменная в последовательность элементарных компонент и этой последовательности ставится в соответствие очередная последовательность форматов.

Типичным будет такой случай, когда будет задан список форматов для всех компонент составного массива и при переходах к очередным элементам этого массива (т. е. при изменении индексов) будет производиться циклическое повторение этого списка форматов. При этом однотипные компоненты во всех элементах составного массива будут иметь одинаковые форматы. Общим правилом является то, что каждому члену списка внутренних массивов соответствует один, очередной, формат в списке форматов. Если этот внутренний массив построен с помощью заголовка цикла и содержит в себе список внутренних массивов, то каждому внутреннему массиву из этого списка будет соответствовать один формат из списка форматов.

Соответствие форматов из списка форматов внутренним массивам из совокупности внутренних массивов всегда устанавливается путем счета слева направо с учетом повторителей элементов списка форматов. При этом если совокупность внутренних массивов исчерпывается раньше списка форматов, то остальными элементами списка форматов пренебрегают; если список форматов исчерпывается раньше совокупности внутренних массивов, то происходит переход к первому элементу списка форматов и т. д. до исчерпания совокупности внутренних массивов.

Все прочие элементы списка форматов задают соответствующие подготовительные операции перед выполнением редактирования по формату, перед которым они помещены.

Размер страницы определяет количество строк (значением левого индексного выражения) и количество столбцов (значением правого индексного выражения) печатаемой страницы. Если размер страницы отсутствует, это означает, что страница имеет стандартные размеры. Размер страницы

указывается только в операторе ПЕЧАТЬ.

Элемент вида X (⟨индексное выражение⟩) управляет относительным размещением элементов данных согласно следующим правилам ⟨W — значение индексного выражения⟩:

1) при выполнении оператора ВЫБРАТЬ пропускаются очередные W элементов обмена; в том случае, если внешний массив исчерпывается до заполнения внутреннего, результат выполнения оператора ВЫБРАТЬ по отношению к указанному внутреннему массиву не определен;

2) при выполнении операторов ПОСЛАТЬ или ПЕЧАТЬ в группу данных перед очередным элементом вставляются W пробелов. При печати в случае исчерпания строки (страницы) происходит автоматический переход на новую строку (страницу).

Указатели размещения представляют собой средства для управления соответствующими устройствами печати. Знак размещения ⊔ задает пробел, знак / задает переход к началу следующей строки, знак × задает переход к началу следующей страницы. Если перед каким-либо знаком размещения помещен повторитель, указанная операция повторяется соответствующее число раз. Значение индексного выражения после символа **строка** определяет номер строки, после символа **столбец** — номер столбца на странице. Указатели размещения **строка** (⟨ индексное выражение⟩) и **столбец** (⟨индексное выражение⟩) определяют абсолютное расположение очередного печатаемого символа на странице. В случае, если номер строки (столбца), задаваемый с помощью указателя **строка (столбец)**, превышает количество строк (столбцов), задаваемых размером страницы, происходит автоматический переход на новую страницу (строку).

Спецификации могут использоваться для указания типа оборудования, участвующего в операции обмена.

Операторы передачи записей. К числу этих операторов относятся операторы ПИСАТЬ, ЧИТАТЬ, УДАЛИТЬ, ПОДВЕСТИ.

Синтаксис.

⟨элемент записи⟩ : : = ⟨внешний массив⟩ **из** ⟨групповой параметр⟩ | ⟨внешний массив⟩ **из** ⟨групповой параметр⟩ **ключ из** ⟨выражение⟩ | ⟨внешний массив⟩ **ключ** ⟨выражение⟩ из ⟨групповой параметр⟩

⟨оператор ПИСАТЬ⟩ : : = ПИСАТЬ (⟨элемент записи⟩)

⟨элемент чтения⟩: : = ⟨внешний массив⟩ **в** ⟨групповой параметр⟩ | ⟨внешний массив⟩ **в** ⟨групповой параметр⟩ **ключ в** ⟨переменная⟩ | ⟨внешний массив⟩ **ключ** ⟨выражение⟩ **в** ⟨групповой параметр⟩

⟨оператор ЧИТАТЬ⟩ : : = ЧИТАТЬ (⟨элемент чтения⟩)

⟨список для удаления⟩ : : = ⟨внешний массив⟩ **ключ** ⟨выражение⟩ | ⟨список для удаления⟩, ⟨внешний массив⟩ **ключ** ⟨выражение⟩

⟨оператор УДАЛИТЬ⟩:: = УДАЛИТЬ (⟨список для удаления⟩)

⟨список для подвода⟩: : = ⟨внешний массив⟩ **пропуск** ⟨индексное выражение⟩ | ⟨список для подвода⟩, ⟨внешний массив⟩ пропуск ⟨индексное выражение⟩

⟨оператор ПОДВЕСТИ⟩ : : = ПОДВЕСТИ (⟨список для подвода⟩)

Примеры.

ЧИТАТЬ (Р в В)
ЧИТАТЬ (К **ключ** ’ПЕТРОВ’ в А)
ПИСАТЬ (Н **ключ** ’КАМА’ из С)
ПИСАТЬ (К **из** В **ключ из** ’А’ ← ’В’)
ПОДВЕСТИ (Р **пропуск** 10)

Семантика. Операторы ПИСАТЬ и ЧИТАТЬ имеют три следующие разновидности.

Первая разновидность:

ПИСАТЬ (⟨внешний массив⟩ **из** ⟨групповой параметр⟩)

ЧИТАТЬ (⟨внешний массив⟩ **в** ⟨групповой параметр⟩) Эти операторы определены только для последовательных массивов, имеющих характеристику **запись.** Производится передача (выборка) очередной записи во (из) внешний массив из группового параметра (в групповой параметр). После выполнения указанной операции внешний массив подготовлен к записи (чтению) следующей записи. Причем, если массив имеет характеристику **обратное,** под «следующей» записью понимается предыдущая (в смысле физического расположения записей в массиве).

Вторая разновидность:

ПИСАТЬ (⟨внешний массив⟩ > **из** ⟨групповой параметр⟩ **ключ из** ⟨выражение⟩)

ЧИТАТЬ (⟨внешний массив⟩ **в** ⟨групповой массив⟩ **ключ в** ⟨переменная⟩)

Эти операторы определены только для последовательных массивов записей с ключами. Ключ — это некоторый признак, относящийся к записи. При записи происходит запись данных группового параметра во внешний массив. Одновременно формируется ключ: его значением становится значение выражения, помещенного после символа **ключ из,** преобразованное, если это необходимо, к типу **строчный.** Длина ключа определяется соответствующей характеристикой в операторе ОТКРЫТЬ, осуществляющем отведение внешней памяти (т. е. в случае необходимости полученное значение усекается или пополняется справа пробелами). При чтении производится чтение очередной записи внешнего массива в групповой параметр, а значение ключа этой записи присваивается переменной типа **строчный,** помещенной после символа **ключ в.** После выполнения указанных операций внешний массив подготовлен к записи (чтению) следующей записи. Символ **обратное** имеет тот же смысл, что и выше.

Третья разновидность:

ПИСАТЬ (⟨внешний массив⟩ **ключ** ⟨выражение⟩ **из** ⟨групповой параметр⟩)

ЧИТАТЬ (⟨внешний массив⟩ **ключ** ⟨выражение⟩ **в** ⟨групповой параметр⟩).

Эти операторы определены для прямых массивов и для последовательных массивов с ключами. По ключу находится требуемая запись во внешнем массиве. Производится запись (чтение) данных из группового параметра (в групповой параметр) на место указанной записи (из указанной записи). Ключ записи (во внешнем массиве) сохраняется.

Оператор УДАЛИТЬ определен только для прямых массивов. Из внешнего массива удаляется запись, положение которой определяется ключом (т. е. значением выражения, помещенного после символа **ключ).**

Оператор ПОДВЕСТИ определен только для последовательных массивов. Если W есть значение индексного выражения, помещенного после символа **пропуск,** то:

1) если $W \geq 0$, выполняется прогонка W записей в прямом направлении;

2) если $W < 0$, выполняется прогонка W записей в обратном направлении.

Прямое и обратное направления устанавливаются с учетом наличия или отсутствия характеристики **обратное.**

Операторы передачи записей не осуществляют никаких преобразований данных (не редактируют данные).

Данные. *Синтаксис.*

⟨комментарий⟩ : : = ⟨переменная⟩ : =

⟨число с комментарием⟩ : : = ⟨комментарий⟩ ⟨число⟩

⟨логическая константа с комментарием⟩: : =⟨комментарий⟩ ⟨логическая константа⟩

⟨строка с комментарием⟩ : : = ⟨комментарий⟩ ⟨строка⟩ ⟨элемент данных⟩: : = ⟨число⟩ | ⟨логическая константа⟩ | ⟨строка⟩

⟨элемент данных с комментарием⟩: : = ⟨число с комментарием⟩ | ⟨логическая константа **с** комментарием⟩ | ⟨строка с комментарием⟩

⟨группа данных⟩: : = ⟨элемент данных⟩ | ⟨группа данных⟩, ⟨элемент данных⟩

⟨группа данных с комментариями⟩ : : — ⟨элемент данных с комментариями⟩ | ⟨группа данных с комментариями⟩, ⟨элемент данных с комментариями⟩

⟨данные⟩ : : = ⟨группа данных⟩; | ⟨группа данных с комментариями⟩;

Примеры групп данных.

25, 2.0, — 0.5, + 10.5_{10}—5;

101, 0000, 1100, 1001**,** 0.5, 500, —10;

Пример группы данных с комментариями.

A [1, 1] : = 10 , A [2, 1] : = 50, A [3, 1] : = 60;

Семантика. Группа данных образует значение внутреннего массива, подготовленное на внешнем носителе или выдаваемое на внешний носитель. При передаче группы Данных во внутренний массив (выборке из внутреннего массива) устанавливается взаимнооднозначное соответствие между элементами группы данных и элементами внутренних массивов согласно описаниям и форматам внутренних массивов и форматам, имеющимся в операторах ВЫБРАТЬ, ПОСЛАТЬ, ПЕЧАТЬ. Каждой группе данных соответствует один внутренний массив из совокупности внутренних массивов в

операторах ВЫБРАТЬ, ПОСЛАТЬ или ПЕЧАТЬ не сопровождаемой символом данные.

Группа данных с комментариями соответствует внутреннему массиву из совокупности внутренних массивов в операторах ВЫБРАТЬ, ПОСЛАТЬ или ПЕЧАТЬ, сопровождаемой символом **данные.** В этом случае группа данных состоит из самоидентифицирующихся элементов данных. Это означает следующее:

1) при передаче во внутренний массив данные представляют собой непрерывную последовательность элементов обмена и включают информацию (в виде комментариев), указывающую те элементы внутренних массивов, значениями которых должны стать эти элементы данных. Переменные, представляющие комментарии, в качестве индексных выражений могут содержать лишь числа. В качестве внутренних массивов могут выступать только групповые параметры;

2) при передаче данных во внешний массив выполняется обратный процесс, т. е. данные, взятые из совокупности внутренних массивов, помещаются во внешние массивы с комментариями, имеющими указанный Выше смысл. В качестве внутренних массивов допускаются только переменные.

14. Особенности описаний в языке АЛГЭМ-2

Описания служат для того, чтобы определить некоторые свойства величин, используемых в программе, и связать эти величины с идентификаторами.

Как уже говорилось, идентификаторы в АЛГЭМ-2 делятся на внутренние и внешние. Внешними называются идентификаторы, помещенные в описании после описателя **внешний.** Все остальные идентификаторы называются внутренними. Область действия внешнего идентификатора — совокупность блоков и процедур, в которых он описан как внешний (за исключением тех блоков и процедур, где он описан как локальный). Описание внутреннего идентификатора имеет силу только для блока (процедуры), в начале которого оно помещено. К моменту выхода из блока или процедуры (через конец или оператор перехода) все внутренние идентификаторы, которые описаны в блоке, теряют присвоенный им смысл. Очевидно, что описания внешних идентификаторов, помещенные в разных блоках и процедурах, не должны противоречить одно другому, так как они определяют одни и те же величины, используемые в разных блоках и процедурах во всей программе.

Все идентификаторы программы, за исключением меток, имен входов процедур, не являющихся функциями, стандартных функций и идентификаторов операторов обмена (описываемых в операторе ОТКРЫТЬ), должны быть описаны.

В АЛГЭМ-2 имеется пять видов описаний:

⟨описание⟩ : : = ⟨описание типа⟩ | ⟨описание массивов⟩ | ⟨описание составной переменной⟩ | ⟨описание составного массива⟩ | ⟨описание переключателя⟩

Кроме того, отдельный вид представляют описания в процедуре, синтаксис которых был рассмотрен при изложении процедур. В описаниях типа переменных в АЛГЭМ-2, так же как и в АЛГЭМ-1, могут указываться их форматы (для строчных переменных обязательно); описание форматов имеет по сравнению с АЛГЭМ-1 более сложный синтаксис с большими возможностями; это описание почти полностью взято из языка АЛГЭК. Дополнительным по сравнению с АЛГЭМ-1 в представлении форматов является: введение шпаций (групп пробелов), вставок шпаций или строк, подавляемых частей (П-часть и ∗-часть), масштабов чисел, «+» и «—» частей, указателя усечения У, указателя подразумеваемой точки Т, логического формата и снятие ограничений на число символов в значении порядка и общее число символов.

Синтаксис форматов переменных.

⟨повторитель⟩ : : = (⟨индексное выражение⟩)

⟨шпация⟩ : : = ␣ | ␣ ⟨повторитель⟩ | ⟨шпация⟩ ␣

⟨вставка⟩ : : = ⟨шпация⟩ | ⟨строка⟩ | ⟨пусто⟩

⟨П-позиции⟩ : : = П | П ⟨повторитель⟩

⟨П-часть⟩: : = ⟨П-позиции⟩ | ⟨П-часть⟩ ⟨П-позиции⟩|

⟨П-часть⟩ ⟨вставка⟩

⟨∗-позиции⟩ : : = ∗ | ∗ ⟨повторитель⟩ ⟨∗-часть⟩ : : = ⟨∗-позиции⟩ | ⟨∗-часть⟩ ⟨∗-позиции ⟩ |

⟨∗-часть⟩ ⟨вставка⟩

⟨подавляемая часть⟩ : : = ⟨П-часть⟩ | ⟨∗-часть⟩

⟨9-позиции⟩ : : = 9 | 9 ⟨повторитель⟩

⟨9-часть⟩ : : = ⟨9-позиции⟩ | ⟨9-часть⟩ ⟨9-позиции⟩ | ⟨9-часть⟩ ⟨вставка⟩

⟨1-позиция⟩ : : = 1 | 1 ⟨повторитель⟩

⟨1-часть⟩ : : = ⟨1-позиция⟩ | ⟨1-часть⟩ ⟨1-позиция⟩ | ⟨1-часть⟩ ⟨вставка⟩

⟨формат целого без знака⟩ : : = ⟨подавляемая часть⟩ | ⟨9-часть⟩ | ⟨подавляемая часть⟩ ⟨9-часть⟩ | ⟨вставка⟩ ⟨формат целого без знака⟩

⟨ряд М⟩ : : = М | М ⟨повторитель⟩

⟨масштаб⟩ : : = ⟨ряд М⟩ ⟨вставка⟩

⟨+позиция⟩ : : = + | + ⟨повторитель⟩

⟨+ часть⟩ : : = ⟨+ позиции⟩ | ⟨+ часть⟩ ⟨+ позиции⟩ | ⟨+часть⟩ ⟨вставка⟩

⟨— позиции⟩ : : = — | — ⟨повторитель⟩

⟨— часть⟩ : : = ⟨— позиции⟩ | ⟨— часть⟩ ⟨— позиции⟩ | ⟨— часть⟩ ⟨вставка⟩

⟨знаковая часть⟩ : : = ⟨пусто⟩ | ⟨вставка⟩ ⟨+ часть⟩ | ⟨вставка⟩ ⟨— часть⟩

⟨формат целого⟩ : : = ⟨знаковая часть⟩ ⟨формат целого без знака⟩

⟨указатель точки⟩ : : = . ⟨вставка⟩ |Т ⟨вставка⟩

⟨указатель округления⟩ : : = ⟨пусто⟩ | У ⟨вставка⟩

⟨формат мантиссы⟩ : : = ⟨указатель точки⟩ ⟨9-часть⟩ ⟨указатель округления⟩

⟨формат десятичного числа⟩ : : = ⟨формат целого без знака⟩ | ⟨формат мантиссы⟩ | ⟨формат целого без знака⟩ ⟨формат мантиссы⟩

⟨масштабированный формат десятичного числа⟩ : : = ⟨масштаб⟩ ⟨формат десятичного числа⟩

⟨формат порядка⟩ : : = $_{10}$ ⟨вставка⟩ ⟨9-часть⟩ |$_{10}$ ⟨вставка ⟩ + ⟨вставка⟩ ⟨9-часть⟩ | $_{10}$ ⟨вставка⟩ — ⟨вставка⟩ ⟨9-часть⟩

⟨формат числа без знака⟩ : : = ⟨формат десятичного числа⟩ | ⟨масштабированный формат десятичного числа⟩ | ⟨формат порядка⟩ | ⟨формат десятичного числа⟩ ⟨формат порядка⟩

⟨десятичный формат⟩ : : = ⟨знаковая часть⟩ ⟨формат числа без знака⟩

⟨формат числа⟩ : : = ⟨формат целого⟩ | ⟨десятичный формат⟩ | ⟨1-часть⟩

⟨логический формат⟩ : := ⟨1-часть⟩

⟨С-позиции⟩) : : = С | С ⟨повторитель⟩

⟨формат элемента строки⟩ : : = ⟨С-позиция⟩ | вставка⟩ ⟨формат строки⟩ : : = ⟨формат элемента строки⟩ | ⟨формат строки⟩ ⟨формат элемента строки⟩

⟨формат⟩ : : = ⟨формат числа⟩ | ⟨логический формат⟩ ⟨формат строки⟩

Примеры форматов.

ППППП 99 ⊔ 999 ⊔⊔

П(4)

ПП9 ⊔ (4) — (6)9Т 9 (3)У(3)

⊔(4) — 9(6)·9(5)$_{10}$—99

9(N + M) СС(3) ⊔ ⊔ С(2)

+ + +99

— П99 ⊔ ССС ⊔ ’РУБ.’ ⊔ СС ⊔ ’КОП.’

⊔(4)9 — П(6).9(5)$_{10}$—99

⊔ ⊔— —* * .9(3)$_{10}$ + 9

Редактирование целых и вещественных чисел и строчных значений в АЛГЭМ-2 производится так же, как и в АЛГЭМ-1. Индексное выражение, стоящее в повторителе, кроме повторителя ряда М, должно иметь целое положительное значение. Каждый из символов П, *, 9, а также символы «+» и «—» в знаковой части, кроме самого левого из них, представляют собой отдельную цифровую позицию получающегося числового значения. Символ 1 представляет отдельную двоичную позицию числового значения.

Буква П означает подавление (замену) незначащего нуля символом пробела; символ * означает подавление незначащего нуля символом *. Символы «+» и«—» в знаковой части означают подавление незначащих нулей в позициях, соответствующих знаковой части, символами пробела, кроме самого правого из этих нулей, который заменяется символом «+» или «—».

Позиция десятичной точки указывается в формате точкой, буквой Т или конструкцией (масштаб). В первом случае десятичная точка фактически помещается в соответствующий разряд получающегося числового значения, во втором она подразумевается в числовом значении, т. е. ничем не обозначается, и ее наличие и положение могут быть определены только с помощью формата.

Если числовой формат не содержит ни точки, ни буквы Т, то позиция десятичной точки подразумевается непосредственно справа от крайней правой цифровой позиции, не считая цифровых позиций порядка.

При редактировании нецелое число обычно округляется, т. е. заменяется ближайшим к нему числом, содержащим столько цифр, сколько предусмотрено их данным форматом. Если же в формате числа стоит буква У, то происходит «усечение» (отбрасывание младших разрядов), при котором последний сохраняемый разряд соответствует самой правой цифровой позиции, не считая цифровых позиций порядка.

Символы пробела ␣ в формате задают необходимую разгонку получившегося числового значения.

Редактирование значений строчных выражений производится путем установления соответствия символов С в формате, не принадлежащих вставкам, элементам значения строчного выражения слева направо. Символы С, не принадлежащие вставкам, заменяются соответствующими символами значения строчного выражения, и удаляются внешние кавычки строк, являющихся вставками. При выдаче на печать происходит редактирование с учетом вставок и масштабов. Значением индексного выражения в повторителе ряда М может быть как положительное, так и отрицательное целое число. Положительное значение индексного выражения указывает на сдвиг десятичной запятой, помещенной в формате, вправо на количество числовых разрядов, равное этому значению; при отрицательном значении происходит сдвиг влево на количество разрядов, равное абсолютному значению индексного выражения. При выходе десятичной запятой после такого сдвига вправо или влево за значащие разряды недостающие значащие разряды формата заменяются нулями слева или справа соответственно. Перед выдачей на печать происходит включение вставок между разрядами отредактированного значения в соответствии с форматом.

Редактирование логических значений заключается в установлении соответствия между символами 1 в формате и значениями двоичных разрядов логического значения слева направо. В случае необходимости логическое значение дополняется справа достаточным количеством нулей.

Значение целого или вещественного числа при его редактировании по формату 1-часть получается заменой 1-позиции соответствующими двоичными разрядами этих чисел. Перед редактированием происходит совмещение правого двоичного разряда числового значения с правым символом формата.

Описание типа. Описание типа в АЛГЭМ-2 аналогично описанию типа в АЛГЭМ-1, за исключением того, что список типа может содержать описатель **внешний:**

⟨список типа⟩ : : = ⟨список элементов⟩ | **внешний** ⟨список элементов⟩ | ⟨список элементов⟩ **внешний** ⟨список элементов ⟩

В АЛГЭМ-2 предусматривается 4 типа: **вещественный, целый, логический, строчный.** Переменные, которым предписан тип **логический,** принимают значения, являющиеся двоичными константами. Значениями переменных типа строчный могут быть открытые строки. В АЛГЭМ-2 в отличие от АЛГЭМ-1, допускаются арифметические выражения, включающие себя как целые, так и вещественные компоненты. В арифметическом выражении любая позиция, которая может быть занята переменной типа **вещественный,** может быть занята и переменной типа целый.

Описание массивов в АЛГЭМ-2 совпадает с описанием массивов в АЛГЭМ-1, за исключением того, что в качестве границ массивов, описываемых в процедурах, может указываться, кроме арифметических выражений, звездочка «∗». Идентификаторами таких массивов должны быть формальные параметры данной процедуры. Значение таких границ становится равным значениям границ массивов — фактических параметров. Так же как и в АЛГЭМ-1 в описаниях массивов могут указываться и форматы значений переменных. Формат распространяется на идентификатор массива, стоящий непосредственно перед форматом. Выражения для границ массивов могут зависеть от переменных и указателей функций, не локальных в блоке, для которого имеет силу описание массивов. Они могут зависеть от переменных и указателей функций, локальных в процедуре, если эти переменные и функции являются формальными параметрами данной процедуры. Значения выражений для границ вычисляются один раз при каждом входе в блок или процедуру. Во внешней процедуре могут быть описаны массивы:

1) с постоянными границами;

2) с границами, зависящими от формальных параметров данной процедуры;

3) с границами, заданными звездочками.

В последнем случае идентификатор описываемого массива должен быть формальным параметром данной процедуры.

Описания составных переменных и составных массивов. Описание составных переменных и составных массивов в АЛГЭМ-2 отличается от АЛГЭМ-1 введением переопределений и описателя **внешний:**

⟨элемент описания⟩ : : = ⟨описание типа⟩ | ⟨описание массива⟩ | ⟨описание составной величины⟩ | ⟨элемент описания ⟩ **или** ⟨элемент описания⟩

⟨составная величина⟩ : : = ⟨идентификатор составной переменной⟩ | **массив** ⟨идентификатор составного массива⟩ | ⟨список граничных пар⟩|

⟨структура составной величины⟩ : : = ⟨элемент описания⟩ **уровень** | ⟨элемент описания⟩; ⟨структура составной величины⟩

⟨описание составной величины⟩ : : = **составной** ⟨составная величина⟩; ⟨структура составной величины⟩ | **составной внешний** ⟨составная величина⟩; ⟨структура составной величины⟩

Примеры:

1) составной внешний А;
целый ШИФР, НОМЕР;
строчный М **вид** СС, НВ **вид** С(43);
составной массив П; **строчный** ФАМИЛИЯ
вид С(15),
ДОЛЖНОСТЬ **вид** С(25) **уровень уровень**

2) составной массив СВОДКА [1 : 30];
строчный ИЗДЕЛИЕ **вид** С(15);
составной ГРАФА; **целый** СТ1, СТ2, СТ3 **уровень**
или
целый массив ГР [1 : 3]
или
составной ДАТА; **целый** ДН, МЦ, ГД
уровень уровень

Основной символ **или** задает переопределение одного элемента описания другим, т. е. допускается одновременное существование на одном и том же рабочем поле двух и более величин разной структуры, описания которых вводятся посредством разделителя **или.**

Если составная переменная или составной массив описаны с использованием символа **внешний** этот символ автоматически приписывается всем величинам, входящим в их состав, независимо от наличия в их описании этого символа.

Глава 5

АССОЦИАТИВНОЕ ПРОГРАММИРОВАНИЕ

15. Сущность ассоциативного программирования и способы построения списков

Сущность решения многих информационно-логических задач сводится к так называемой категоризации объектов, т. е. к разделению объектов на виды, типы, классы и т. д., в зависимости от их свойств и признаков. При этом запись новой информации об объектах в память машины или поиск информации в машине осуществляются по значениям этих признаков. Примерами подобных задач могут служить учет и планирование материально-технического снабжения, учет кадров, задачи библиографического поиска, обработки и накопления научной информации, машинной медицинской диагностики заболеваний и т. д.

Для эффективного решения подобных задач необходима специальная организация работы запоминающих устройств машин, которая позволяла бы осуществлять поиск информации не только по ее адресам, но и по ее признакам.

Существует два основных пути организации машинной памяти, называемой ассоциативной и обеспечивающей быстрый поиск объектов в зависимости от их признаков.

Первый путь — схемный; он состоит в построении специальной памяти, в которой запоминающие ячейки (частично или полностью) обладают свойством выполнять операции сравнения. Для поиска информации в такой памяти вместо адресов задаются наборы признаков. В ячейках памяти производится сравнение хранящихся в них признаков с заданными и на выход ЗУ выдаются адреса ячеек, в которых хранится информация, обладающая заданными признаками; в некоторых случаях выдается сразу искомая информация. Такой схемный способ организации памяти можно назвать *схемно-ассоциативным.* При этом необходимые наборы признаков хранятся во всех ячейках памяти и поиск информации, обладающей заданным набором признаков, производится одновременно и независимо по всему объему памяти. Связь между отдельными ячейками памяти, в которых хранятся сходные признаки, т. е. прямая ассоциативная связь, отсутствует. Свойство ассоциативности такой организации памяти проявляется в возможности поиска объектов по их признакам, т. е. в характере связи между памятью и внешним потребителем информации. Ассоциативность заключается в том, что одновременно извлекается из всех ячеек информация об объектах, обладающих общим набором признаков. Таким образом может выявляться наличие общих свойств у данных, хранящихся в разных ячейках, т. е. наличие ассоциативных связей между данными.

Второй путь организации ассоциативной памяти можно назвать *программно-ассоциативным.* Он основан на использовании обычных адресных систем памяти и заключается в том, что ассоциативные связи между различными данными, хранящимися в памяти, осуществляются либо путем специального упорядоченного расположения этих данных (или их наименований) в памяти машины, либо путем объединения их в последовательные цепочки при помощи специальных адресов связи, коды которых хранятся в тех же ячейках памяти совместно с данными. Группы последовательно расположенных данных или данных, связанных адресами связи, называются *списками.* При этом необходимые признаки информации могут храниться либо во всех объединяемых членах, либо только в отдельных ячейках, помещаемых в началах списков.

Некоторые члены списков, в общем случае, могут указывать ответвления к другим спискам, так называемым *подспискам* и т. д. Любой список со всеми отходящими от него подсписками называется *списковой структурой.* Программно-ассоциативный способ, по-видимому, является более реальным для практического осуществления при больших объемах переменной информации, так как не требует разработки специальной ассоциативной памяти большого объема. Он обладает гибкостью и обеспечивает возможность изменения в широких пределах алгоритмов поиска информации, состава и характера признаков, по которым производится поиск, в том числе реализацию многоступенчатых и рекурсивных поисковых процессов. *Ассоциативным программированием* мы называем совокупность способов решения информационно-логических задач, основанных на программной реализации ассоциативных связей между данными, хранящимися в памяти машины. За рубежом этот круг вопросов известен под несколькими названиями: списковая обработка, узловая обработка, цепная адресация, метод управляющих слов.

Ассоциативное программирование удобно применять при логической обработке информации,

изменяющейся по своему составу и объему, когда процессы поиска и обработки имеют иерархический и рекурсивный характер.

В общем случае при ассоциативном программировании не требуется производить заранее жесткое распределение памяти. Распределение памяти осуществляется автоматически в ходе процесса обработки в соответствии с фактическим поступлением данных.

Ассоциативное программирование позволяет значительно (в сотни и тысячи раз в зависимости от количества просматриваемых или обрабатываемых данных) ускорить поиск данных, их анализ и обработку.

Способы построения списков. Рассмотрим следующие способы построения списков в машинной памяти:

1) последовательный,
2) цепной,
3) гнездовой,
4) узловой.

Последовательные списки. *При последовательном* способе построения списков ассоциативные (списковые) слова, представляющие отдельные члены данного списка, размещаются в последовательно расположенных ячейках машинной память

Достоинством последовательного способа построения списков является возможность быстрого поиска требуемых членов списков в тех случаях, когда эти члены располагаются в порядке монотонного изменения какого-либо признака (например, номера объекта). Тогда для поиска объекта с заданным номером может быть применен, например, способ последовательного деления списка пополам, обеспечивающий нахождение нужного члена списка приблизительно за $N = \log_2 n$ операций сравнения, где n — общее число членов списка. Недостатками последовательных списков являются:

а) необходимость заранее знать количество членов в каждом списке и соответственно распределять память машины для построения каждого списка. При этом память машины используется нерационально, так как ожидаемые размеры списков часто не совпадают с фактическими. Некоторые зоны памяти оказываются недостаточными для размещения списков, в то время как другие зоны используются не полностью;

б) сложность процедур включения в список новых членов и исключения старых членов в тех случаях, когда списки должны быть упорядочены по каким-то признакам. При этом для вставки нового члена в середину списка или исключения старого члена из середины списка требуется сдвиг всех последующих членов. Если включение и исключение членов списка производится случайно по отношению к их расположению в списке, то в среднем на каждую операцию включения или исключения одного члена списка приходится сдвиг половины списка.

Цепные списки. При цепном способе построения списков отдельные члены списка располагаются произвольно в машинной памяти и связываются между собой при помощи так называемых адресов связи. Адрес связи помещается совместно с данным членом списка и указывает положение следующего члена списка. Идея цепной адресации списков и подробная разработка методики построения и преобразования цепных списков была дана американскими учеными Невелом, Симоном и Шоу [15].

Обычно при обработке данных о некоторой совокупности объектов эти объекты распределяются между различными списками, причем один и тот же объект может находиться одновременно в нескольких списках. Для того чтобы многократно не повторять во всех списках, куда включен объект, всю информацию об этом объекте, поступают следующим образом. Выделяют определенную область памяти (обычно на магнитных лентах), в которой последовательными участками, так называемыми записями, размещается вся информация об объектах, причем каждому объекту соответствует отдельная позиция (одна запись) со своим адресом. Эта область памяти называется областью записей объектов.

Списки объектов, формируемые по различным признакам, строятся обычно в оперативной памяти или в промежуточной памяти (на магнитных барабанах или дисках).

Область памяти, в которой размещаются списки, называют списковой областью.

Возможны случаи, когда списки располагаются на магнитной ленте, а для обработки переписываются частями в оперативную память. При наличии отдельных записей, содержащих полную информацию об объектах, в самих списках объекты указываются только своими условными наименованиями. Обычно такими машинными наименованиями объектов являются адреса первых ячеек памяти, в которых хранятся записи для соответствующих объектов. Таким образом, адреса записей являются машинными наименованиями объектов; под этими наименованиями объекты фигурируют во всех списках, в которых

они участвуют.

Таблица 4

Адреса записей	Записи объектов
1000	N1, x1, yl, zl
1001	N3, x3, y3, z3
1002	N9,x9, y9, z9
1003	N6, x6, y6, z6
1004	N2, x2, y2, z2
1005	N11, xll, yll, zll
1006	N8, x8, y8, z8

В табл. 4 показан пример области памяти, содержащей записи для некоторых объектов. В данном случае все записи одного типа и содержат некоторый внешний номер объекта № и координаты объекта xi, yi, zi. Адреса записей, показанные в левой колонке (1000, 1001, 1002 и т. д.) являются машинными наименованиями объектов; они указываются в цепных списках, приведенных в табл. 5.

Каждая ячейка в списковой области памяти делится на две части: в одной части хранится машинное наименование объекта, являющегося членом данного списка, а в другой — адрес связи, указывающий положение следующего члена списка.

Таблица 5

	Адреса ячеек	Содержание списковых ячеек	
		Машинные наименования объектов	Адреса связи
Фиксатор свободных ячеек	100	3	110
Фиксатор списка объектов	101	6	108
	102	1003	105
	103	1004	107
	104	1005	КС
	105	1002	104
	106		109
	107	1001	102
	108	1000	103
	109		КС
	110		106

Для цепного способа построения списков характерной чертой является цепная организация свободных ячеек списковой области памяти. Все свободные ячейки этой области памяти (а в начальный момент, когда в памяти нет еще ни одного списка, то все ячейки списковой области памяти) завязаны в так называемый список свободных ячеек (СЯ). Одна ячейка в памяти (в нашем примере ячейка с адресом 100) постоянно закрепляется в качестве фиксатора (или указателя) свободных ячеек. В указателе свободных ячеек в первой половине ячейки хранится число имеющихся в наличии свободных ячеек, а во второй половине — адрес связи, показывающий положение первой ячейки из списка свободных ячеек. В первой ячейке имеется адрес связи, указывающий вторую ячейку, во второй — третью и т. д. В последней ячейке списка свободных ячеек (так же как и в последней ячейке любого цепного списка) вместо адреса связи стоит условный код (КС), показывающий конец списка. Список свободных ячеек служит резервом, из которого берутся ячейки для построения списков и в который возвращаются ячейки, освободившиеся из списков. Процессы взятия ячеек из СЯ и возвращения в СЯ не программируются программистами. Они должны выполняться автоматически либо стандартными подпрограммами, либо схемно.

Для построения каждого цепного списка программистом заранее выделяется одна ячейка, которая

будет играть роль фиксатора или указателя этого списка. Адрес ячейки известен программисту и указывается во всех командах, содержащих обращения к этому списку, в качестве адреса (машинного наименования) списка.

Таким фиксатором списка объектов в табл. 5 является ячейка с адресом 101. Ячейки для фиксаторов списков обычно берутся программистом не из СЯ; для них оставляется специальная группа ячеек, хотя, вообще говоря, эти ячейки можно брать также и из СЯ. Указатели списков могут строиться различными способами. Например, указатель списка, так же, как и указатель СЯ, может содержать две части: первую, указывающую число членов в данном списке, и вторую, указывающую адрес первого члена списка. Все члены списка соединяются между собой в цепь при помощи адресов связи так же, как это было описано для списка свободных ячеек.

В отличие от списка свободных ячеек, в котором первые части ячеек не используются, в списках объектов первые части ячеек содержат машинные наименования (адреса записей) объектов, являющихся членами данных списков.

В табл. 5 показан один цепной список, содержащий объекты (1, 2, 3, 6, 9, 11). Все остальные ячейки соединены в СЯ. Буквы КС означают условно код конца списка. Как видно из приведенного примера, ячейки с членами цепных списков могут размещаться в памяти в произвольном порядке; связи их между собой обеспечиваются адресами связи. При этом не требуется заранее выделять под каждый список определенное число ячеек. Эти ячейки берутся из общего резерва СЯ для всех списков по мере их нарастания. Таким образом обеспечивается большая гибкость в использовании памяти.

Другим важным достоинством цепного способа организации списков является удобство включения новых членов списков и исключения ненужных. Как включение членов, так и исключение их может производиться из любого места списка без перемещения всех остальных членов.

Включение нового члена в цепной список обычно связано с появлением нового объекта, для которого составляется запись и заносится в одну из свободных зон области памяти, отведенной под записи. Заметим, что в этой области памяти также должен быть организован цепной список свободных зон или ячеек, из которого берутся зоны (группы ячеек) для записей новых объектов и в который включаются освободившиеся зоны после вычеркивания из всех списков ненужных объектов. При занесении новой записи из списка свободных зон берется очередная свободная зона, в которую заносится запись нового объекта. Адрес этой записи становится машинным наименованием нового объекта. Затем по значениям признаков, входящих в запись этого объекта, определяется, к каким спискам он должен быть отнесен, и производится последовательное включение данного объекта в соответствующие списки. Наиболее простым является включение нового члена в начало списка, но можно включать новый член и после любого другого члена без перестановки всех остальных членов списка.

Для включения нового члена в любой список (и в любое место списка) сначала должна быть взята свободная ячейка из СЯ. В левую половину этой ячейки записывается машинное наименование (адрес записи) нового объекта. Затем процесс включения может идти двумя путями. Если новый объект включается в начало списка, то на место адреса связи в новую ячейку записывается значение адреса связи, взятое из фиксатора данного списка, а в фиксатор данного списка на место адреса связи записывается адрес новой ячейки. Если новый член включается куда-то в середину списка, то сначала находится предшествующий член списка, после которого требуется включить новый член, а затем производится замена адресов связи: в предшествующем члене ставится адрес связи, указывающий новый член, а в новом члене — адрес связи, взятый из предшествующего члена. В обоих случаях производится увеличение счетчика числа членов в фиксаторе данного списка на единицу. В фиксаторе СЯ (так же как и в фиксаторе свободных зон) при взятии одной ячейки из СЯ (или зоны из списка свободных зон) производится уменьшение на единицу соответствующих счетчиков.

В табл. 6 показаны те же списки (СЯ и список объектов), что и в табл. 5 только из СЯ взята одна ячейка (110), а в список объектов (в его начало) добавлен новый объект с машинным номером 1006, ранее не включенный в этот список.

Процесс исключения членов из цепных списков осуществляется также путем замены адресов связи. Находится член, предшествующий исключаемому (для первого члена это будет фиксатор списка), и в предшествующем члене адрес связи заменяется на адрес связи, взятый из исключаемого члена. Одновременно производится уменьшение числа членов в фиксаторе данного списка на единицу. Этот процесс был проиллюстрирован в предыдущем примере при исключении очередной ячейки из СЯ.

Таблица 6

Адрес ячеек	Содержимое ячеек	
	Машинное наименование объектов	Адрес связи
100	2	106
101	7	110
102	1003	105
103	1004	107
104	1005	КС
105	1002	104
106		109
107	1001	102
108	1000	103
109	—	КС
110	1006	108

Исключение некоторого члена из данного списка может быть связано либо с переносом его в другой список объектов, либо с полной ликвидацией данного члена. В последнем случае соответствующая списковая ячейка включается в СЯ. При полной ликвидации объекта соответствующая зона памяти с записью этого объекта включается в список свободных зон.

Таким образом, в списковых членах цепных списков в явном виде указываются два адреса: адрес следующего члена списка и адрес записи объекта, являющегося данным членом списка. Этот способ построения списков является достаточно гибким, но приводит к некоторому увеличению объема памяти за счет хранения в явном виде указанных адресов. Несколько более экономичным в этом отношении является гнездовой способ.

Таблица 7

Адреса ячеек	Списковые слова			
	П	КС		
100			105	Фиксатор свободных гнезд
101			108	Фиксатор списка объектов
102	0	0	1004	2-е гнездо
103	0	0	1003	
104	1	0	114	
105		0	111	Свободное гнездо
106				
107				
108	0	0	1000	1-е гнездо
109	0	0	1001	
110	1	0	102	
111		1	1	Свободное гнездо
112				
113				
114	0	1	1002	3-е гнездо (неполное)
115				
116				

Гнездовые списки. При гнездовом способе вместо указания явным образом в каждом члене списка адреса следующего члена этого списка списковые слова, представляющие члены одного списка, располагаются подряд в последовательных ячейках памяти. При этом в списковых словах указываются только машинные наименования (адреса записей) объектов, являющихся членами данного списка, и некоторые дополнительные признаки. Так как состав различных списков переменный, то невозможно заранее точно знать длины списков и выделить им соответствующие места в памяти. Поэтому данный вариант реализуется не в виде сплошных последовательностей списковых ячеек, относящихся к одному

списку, а в виде гнезд членов одного списка. Внутри гнезда члены размещаются подряд, а связь между гнездами осуществляется при помощи адресов связи. Отсюда происходит название «гнездовой» способ. Этот способ является сочетанием последовательного и цепного способов и обладает некоторым преимуществом перед указанным выше способом построения цепных списков в отношении расхода памяти и скорости просмотра списковых членов внутри гнезда. При гнездовом способе списковые слова содержат кроме машинных наименований объектов, являющихся членами списка, два признака: признак переходного слова (П) и признак конца списка (КС). Переходные слова служат для связи между гнездами одного списка. В табл. 7 и 8 показан пример гнездового списка. Так же как и при цепном способе, для каждого гнездового списка должен быть выделен фиксатор (указатель) списка, т. е. фиксированная ячейка, в которой хранится адрес, показывающий положение первой ячейки первого гнезда этого списка. Свободные ячейки при гнездовом способе объединяются не в цепи свободных ячеек, а в цепи свободных гнезд ячеек.

Таблица 8

Адреса записей	Записи объектов
1000	N1, xl, yl, zl
1001	N3, x3, y3, z3
1002	N9, x9, y9, z9
1003	N5, x5, y5, z5
1004	N4, x4, y4, z4

Список объектов в табл. 7 состоит из трех гнезд, причем в третьем гнезде занята всего одна ячейка. Список свободных гнезд состоит из двух гнезд. Свободные гнезда соединяются при помощи адресов связи, помещаемых в первых ячейках гнезд. Во втором свободном гнезде, в 1-й ячейке, адрес связи отсутствует, а признак КС равен 1, что указывает на конец списка свободных гнезд.

Узловые списки. Узловой способ построения списков является обобщением цепного способа и служит для образования многосписковых структур. В отличие от цепных списков, в которых от каждого члена списка может быть сделан переход только к одному следующему члену списка, в узловых списках от каждого члена списка могут быть сделаны переходы ко многим другим членам, т. е. каждый член является узлом пересечения многих списков. При узловом способе все списковые слова, представляющие один и тот же объект в разных списках, располагаются в памяти машины не независимо друг от друга, как это имеет место при построении простых цепных списков, а подряд. Обычно они располагаются совместно в одной ячейке памяти или в группе последовательно расположенных ячеек, образуя так называемый узел. Можно также сам узел представлять в виде цепного или гнездового списка. Описательная информация о данном объекте (запись объекта) может располагаться либо совместно с его узлом, либо в узле будет указываться машинное наименование объекта, т. е. адрес, определяющий положение записи объекта в памяти машины.

Совместное размещение узлов и записей объектов удобно при постоянном составе узлов и записей, раздельное их расположение — при переменном составе. В узловых списках (с объединенным или раздельным расположением узлов и записей) нет необходимости повторять в каждом списковом слове, входящем в состав узла, наименование данного объекта. В состав каждого узла входит одно слово, играющее роль заголовка узла, и нескольких списковых слов. В заголовке узла указывается наименование объекта (адрес записи) и некоторая дополнительная информация об объекте и самом узле (например, число списковых слов в узле), В списковых словах содержатся адреса связи, определяющие положение следующих членов соответствующих списков. Кроме того, могут указываться признаки членов списков, по которым производится их поиск, и некоторые дополнительные признаки (тип спискового слова, признак исключения объекта из данного списка и др.). Для каждого объекта образуется, свой узел и своя запись, содержащая информацию о данном объекте. Общие признаки всех членов данного списка могут указываться не в каждом члене списка, а только в указателе (фиксаторе) списка (ячейке, хранящей адрес первого члена списка и некоторые другие данные о списке).

Используя узловой способ, можно строить списковые структуры, отражающие наличие сложных ассоциативных связей между большим количеством различных объектов. Каждой отдельной цепочкой адресов связи объединяются в единые списки те объекты, у которых имеются одинаковые значения определенных признаков.

Таблица 9

	Адреса ячеек	Адрес первого члена списка	Номер спискового слова	Значение признака	Число членов в списке
Фиксатор списка *А*	001	108	1	ПП1	4
Фиксатор списка *Б*	002	114	2	ПП2	2

В табл. 9, 10, 11 показана схема построения узловой многосписковой структуры с раздельным расположением узлов и записей объектов. В данном примере в каждой ячейке размещается по одному списковому слову, состоящему из поискового признака (ПП) и адреса связи (АС). Каждый узел располагается в трех ячейках. Первые ячейки узлов отводятся под наименования объектов, представляющие собой адреса их записей. Поисковый признак служит для выбора нужного объекта при просмотре списков. Поисковый признак обычно состоит из двух частей: наименования признака и его значения. Наименование признака является общим для всех членов (узлов), входящих в данный список, а значения могут быть различными. Бывают случаи, когда в список объединяются только члены, имеющие одинаковое значение данного признака; тогда эти значения могут указываться только один раз в фиксаторе списка. В качестве кодов наименований могут использоваться адреса фиксаторов списков. Наименования признаков необходимо указывать

Таблица 10

	Адреса ячеек	Содержимое ячеек ассоциативных узлов	
		ПП	АС
Узел для 5-го объекта	105 106 107	1026 001 000	 114 000
Узел для 1-го объекта	108 109 110	1024 001 002	 111 КС
Узел для 3-го объекта	111 112 113	1028 001 000	 105 000
Узел для 8-го объекта	114 115 116	1030 001 002	 КС 108
	117		

в соответствующих списковых словах всех узлов в тех случаях, когда предусматривается оперативное исключение выбывающих объектов из всех списков, в которые он входит. При этом наименования признаков необходимы для того, чтобы, выполняя исключение некоторого объекта, иметь возможность по кодам наименований признаков, содержащимся в его узле (в соответствующих списковых словах), определить, к каким спискам относится этот объект. Если не требуется предусматривать возможность оперативного исключения объектов из списков, то коды наименований признаков можно не помещать в списковых словах, а указывать их только в фиксаторах соответствующих списков (или вообще не указывать, если этими кодами являются адреса фиксаторов).

В качестве адреса связи в списковом слове указывается адрес первой ячейки узла и номер спискового слова в узле, относящегося к данному цепному списку. Нумерация списковых слов идет без учета заголовка узла. Этот номер при записи на бумаге отделяется обычно от собственно адреса связи запятой. Если принято правило, что списковые слова, образующие один список, располагаются в узлах на одинаковых по порядку местах, то номер спискового слова в узле можно указывать только один раз в фиксаторе данного списка. Ассоциативные узлы могут быть переменного или постоянного состава. Постоянный состав узлов упрощает их обработку (перепись, включение и исключение), но приводит к излишнему расходу памяти, так как при этом, если какой-нибудь объект не входит в список,

соответствующая ячейка спискового слова не используется.

Таблица 11

Адреса ячеек записей	Записи объектов	
1024 1025	N1 x1 y1 zl	1-й объект
1026 1027	N5 x5 y5 z5	5-й объект
1028 1029	N3 x3 y3 z3	3-й объект
1030 1031	N8 x8 y8 z8	8-й объект

В табл. 10 показана многосписковая структура с постоянным составом узлов (по три ячейки в узле) и постоянным закреплением списковых слов за списками. Первое списковое слово во всех узлах в нашем примере закреплено за списком *А*, а второе списковое слово — за списком *Б*, поэтому номера списковых слов при адресах связи не указываются. В общем случае списковые слова, занимающие одно и то же место в разных узлах (например, первые списковые слова), могут использоваться для построения нескольких списков, если эти списки строятся для взаимно исключающих значений признаков. Ясно, что при этом один и тот же объект (один и тот же узел) может одновременно входить только в один из таких списков.

В табл. 11 показан участок списковой области памяти с записями объектов, а в табл. 9 — участок памяти с фиксаторами списков *А* и *Б.*

В каждом фиксаторе списка указаны адрес первого члена списка, порядковый номер списковых слов в узлах, соответствующих данному списку, общее для всех членов списка значение признака объектов, объединенных в список, и число членов в списке. В качестве кодов наименований признаков в данном примере приняты адреса фиксаторов списков, которые указываются в соответствующих списковых словах.

Список *А* содержит все объекты, причем они следуют в списке в порядке возрастания их номеров. Список *Б* содержит два объекта N8 и N1. Концы списков обозначены условными кодами КС.

Свободные списковые слова в узлах должны быть заполнены нулями. Это будет свидетельствовать о том, что данный объект не входит в соответствующий список. При практическом построении списковых структур рассмотренных типов (цепной, гнездовой, узловой) используются некоторые дополнительные приемы, в частности, применяются специальные слова (структурные слова), дающие возможность построения разветвлений в списках и отсылающие к фиксаторам подсписков. Строение фиксаторов может быть самым различным. С их помощью не только ведется учет числа членов в списках, но и осуществляются переходы к подспискам, обеспечивается возможность повторного использования ячеек гнездовых списков, освободившихся после исключения выбывших членов, просмотры и наращивания гнездовых списков и другие функции.

Способы адресации цепных списков. Можно указать четыре способа адресации цепных списков.

Полная прямая адресация. Адреса связи в списковых словах представляют собой полные прямые адреса ячеек ЗУ, используемого для размещения списков. При этом способе обеспечивается гибкость и простота программирования и упрощается построение схем адресации. Объекты имеют во всех списках одни и те же машинные наименования (свои полные прямые адреса), которые могут использоваться для их поиска. Недостатком такого способа является большая разрядность адресов связи, которая должна охватывать весь объем памяти, а отсюда и большой расход памяти под размещение списков.

Относительная плавающая адресация. При этом способе адрес связи в списковом слове указывает

разность адресов (с соответствующим знаком) данного члена и следующего члена списка. Если разность адресов двух соседних членов списка превышает отведенную для адресов связи разрядную сетку, то используется непрямая адресация. При этом адрес связи указывает адрес ячейки, в которой находится полный прямой адрес следующего члена списка. Так как для хранения всех непрямых адресов может быть выделена определенная область памяти (например, начальная зона), то для указания этих ячеек не потребуется большая разрядность адресов. Относительная плавающая адресация в сочетании с непрямой адресацией может использоваться также и для указания адресов условных и безусловных переходов в командах. Относительные и непрямые адреса в списковых словах различаются соответствующим признаком.

Достоинства данного способа:

— экономия объема памяти, расходуемого под адреса связи, так как относительные плавающие адреса могут иметь существенно меньшее количество разрядов (на 50% и больше);

— независимость адресов связи в списках и адресов условных и безусловных переходов в программах от расположения списков и программ в памяти машины.

Недостатки данного способа:

— наличие некоторых ограничений, налагаемых на расположение членов списков в памяти. Необходимо, чтобы члены одного и того же списка располагались в памяти по соседству (в одной и той же зоне), чтобы свести к минимуму число непрямых адресаций. В связи с этим возникает необходимость учитывать заполнение и освобождение отдельных зон памяти (нужны отдельные цепи свободных ячеек по зонам памяти);

— невозможность использования плавающих относительных адресов в качестве наименований объектов, так как адреса, обеспечивающие переходы к одному и тому же объекту из разных точек списка или разных списков, будут различными;

— усложнение схем адресации в связи с необходимостью сочетания непрямой и относительной адресации и преобразованием относительных адресов связи для выработки действительных (полных) адресов переходов.

Относительная индексная адресация. Адреса связи в списковых словах указывают относительные адреса следующих членов списка не по отношению к данному члену списка, а по отношению к некоторой постоянной (начальной) точке массива или зоны памяти. В этом случае для получения действительных адресов членов списков необходимо относительные адреса связи суммировать с некоторой постоянной для всего списка величиной, представляющей собой начальный адрес зоны (массива). Начальный адрес может устанавливаться на индексном регистре и автоматически складываться с относительными адресами.

При необходимости перехода к членам, находящимся за границами данной зоны, так же как и в предыдущем случае, должна применяться непрямая адресация. Достоинства способа те же, что и в предыдущем случае, и, кроме того, в пределах зон (в одной зоне располагаются, как правило, списки объектов одного и того же типа) относительные индексные адреса связи являются одинаковыми во всех списках, в которых они используются, поэтому эти адреса могут использоваться в качестве наименований объектов для их поисков. Недостатки способа те же, что и в предыдущем случае, за исключением упомянутой выше возможности использования адресов в качестве наименований объектов.

Кроме того, использование при программировании относительных адресов, имеющих жестко закрепленное начало и являющихся только положительными числами, проще, чем плавающих относительных адресов, которые могут иметь положительные и отрицательные значения.

Сравнивая относительную плавающую и относительную индексную адресации, необходимо заметить, что плавающая адресация обеспечивает в принципе возможность произвольного расширения области памяти машины (на которую распространяются ассоциативные связи) путем последовательных переходов от одних точек памяти к другим. При этом каждый раз переход совершается на шаг, не превышающий предельного размера относительного адреса.

В то же время индексная адресация жестко ограничивает область действия ассоциативных связей предельным размером относительного адреса, и для выхода из этой области необходима непрямая адресация.

Непрямая адресация. Выделяется специальная зона памяти для непрямой адресации с числом ячеек, равным максимальному числу всех объектов различных типов, на обработку которых рассчитывается система. Содержимыми этих ячеек будут полные прямые адреса первых ячеек записей (групп ячеек)

объектов, а также некоторые дополнительные признаки объектов (например, признак выбытия объекта). Непрямые адреса будут иметь существенно меньшую разрядность, чем полные прямые адреса записей объектов, за счет того, что общее число записей на один или два порядка меньше, чем общее число ячеек в памяти. Непрямые адреса могут служить непосредственно машинными наименованиями объектов и могут быть использованы в качестве адресов связей в цепных или гнездовых списках.

Указанный способ выгодно применять при совместном размещении ассоциативных узлов и записей объектов, так как при этом возрастает среднее число ячеек, занимаемых информацией об одном объекте.

Достоинства этого способа:

— уменьшение размеров адресов связей в списках;

— возможность записи в памяти машины информации об объектах различных типов в произвольном порядке, а не по зонам.

Недостатки данного способа:

— необходимость двойного обращения к памяти при адресации к записям объектов, что понижает скорость выборки и записи данных;

— необходимость рассчитывать разрядность адресов связи, исходя из общего числа объектов всех типов (а не одного типа, как при относительной адресации), что увеличивает размер адресов связи;

— необходимость иметь дополнительную память непрямой адресации, что уменьшает экономию в объеме памяти, получаемую за счет сокращения размера адресов связи.

16. Ассоциативные списковые структуры

Основные типы ассоциативных списковых структур.

Мы рассмотрели ряд способов построения списков: последовательный, цепной, гнездовой и узловой. Считалось, что членами этих списков являются отдельные объекты и каждому объекту соответствует одно списковое слово. Было также сказано, что членами списков могут быть другие списки, так называемые подсписки, которые вместе с основным списком образуют ассоциативную списковую структуру. В общем случае ассоциативной списковой структурой называется любая совокупность взаимосвязанных списков и подсписков.

Списковые структуры можно разделить на два основных типа: *объектные* и *признаковые.* В объектных структурах распределение объектов по спискам осуществляется относительно некоторых конкретных объектов, принимаемых за опорные точки системы деления или группирования объектов. Объектные структуры могут быть двух видов: включающие и исключающие.

Включающие объектные структуры используются для группирования однородных объектов, они строятся таким образом, что опорные объекты, от которых отходят подсписки, соответствующие определенным подразделениям группирования, сами входят в состав подразделений в качестве их первых членов. Причем от одного объекта может отходить несколько подсписков понижающихся уровней и объект будет одновременно членом всех подсписков. Исключающие объектные структуры используются для группирования разнородных объектов, они строятся таким образом, что опорные объекты сами не являются членами отходящих от них подсписков.

Примером включающей объектной списковой структуры может служить структура, возникающая при решении задач селекции некоторых подвижных объектов методом последовательного деления всей совокупности этих объектов на группы, когда один и тот же объект может входить в состав нескольких групп объектов различной степени укрупнения в качестве их опорного объекта, т. е. объекта, относительно которого идет формирование группы.

Примером исключающей объектной структуры может служить система деления некоторой населенной территории: сначала вся территория делится на автономные части, представляющие собой страны. Страны образуют список объектов верхнего уровня. Каждой стране может соответствовать свой список областей, являющийся списком второго уровня. Каждой области может соответствовать свой список районов — список третьего уровня и т. д. В этом случае опорные
объекты системы деления сами не являются членами отходящих от них списков.

Объектные ассоциативные структуры имеют древовидный или вообще разветвляющийся характер. Верхний уровень таких структур образует цепной список некоторых объектов, каждый из которых может быть точкой ответвления цепного подсписка (или нескольких подсписков) других объектов. Эти цепные подсписки будут представлять собой второй уровень данной списковой структуры; от них

могут ответвляться цепные подсписки третьего уровня и т. д. Поиск объектов в объектных ассоциативных структурах осуществляется путем просмотра сначала цепного списка верхнего уровня и отбора в нем по заданным признакам объектов и ответвляющихся от них подсписков второго уровня. Затем идет просмотр отобранного подсписка второго уровня и отбор в нем уже по другим признакам объектов и ответвляющихся от них подсписков третьего уровня и т. д. до тех пор, пока не будут исчерпаны либо все уровни структуры (по просматриваемой ветви), либо заданный набор поисковых признаков.

При цепном способе построения списков объектные ассоциативные структуры образуются при помощи списковых слов двух видов: объектных и структурных. Эти слова различаются дополнительным указателем *S*, который для объектных списковых слов равен нулю, а для структурных списковых слов — единице.

Объектные и структурные списковые слова обычно имеют одинаковые форматы; в простейшем случае каждое слово состоит из трех частей (слогов): указателя вида слова *S*, наименования члена списка *H*, адреса связи АС.

В объектных списковых словах *H* представляет собой наименование объекта, являющегося членом списка. Таким машинным наименованием служит обычно адрес записи (формуляра объекта), содержащей собственную информацию об объекте. В структурных списковых словах *H* означает адрес вершины подсписка, ответвляющегося от того члена объектного списка, который стоит непосредственно перед данным структурным словом.

Таким образом, включая после любого объектного спискового слова одно или несколько структурных слов, можно образовать один или несколько подсписков, ответвляющихся отданного объекта. На рис. 14 показан пример объектной ассоциативной структуры с двумя подсписками, ответвляющимися от основного списка. В основном списке находятся объекты N1, N2 и N4. От объекта N1 (после него) отходит подсписок, содержащий объекты N3 и N6. От объекта N4 отходит другой подсписок, в котором находится один объект N5.

Цифры, стоящие перед прямоугольниками, обозначают адреса соответствующих ячеек памяти. Символы КС на местах адресов связи обозначают, как обычно, концы списков. В структурных списковых словах адреса связи (АС)

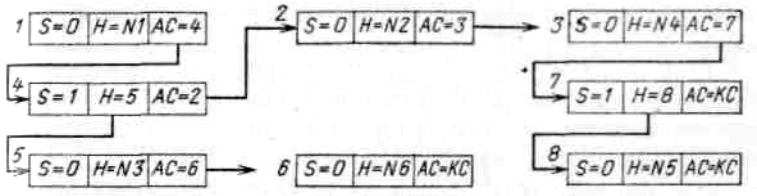


Рис. 17. Пример объектной ассоциативной структуры.

как бы заменяют собой адреса связи в предшествующих объектных списковых словах, от которых производится ответвление подсписков.

В признаковых структурах в качестве опорных точек системы классификации объектов используются не конкретные объекты, а определенные признаки или комбинации признаков, в общем не связанные с конкретными объектами. Признаком объекта называется любое свойство, используемое для поиска и характеризующееся наименованием и значением. Для каждого признака должно существовать одно наименование (код признака) и ряд значений, причем считается, что значения одного признака являются взаимно исключающими. Это означает, что один и тот же объект в одно и то же время не может характеризоваться несколькими значениями одного признака (т. е. он не может быть одновременно, например, белым и красным или большим и маленьким и т. д.). Значения признака могут носить качественный характер (цвет, вкус и т. д.), и тогда они представляются строчными величинами, а могут иметь количественный характер и представляться величинами типа целый или вещественный. Значения могут также представляться логическими величинами. В этом случае признаки называются альтернативными.

В общем случае ряд признаков может образовывать иерархическую систему и значения одного признака явятся точками разветвления для деления объектов по другому признаку. Например, сначала производится деление объектов по цвету, затем объекты одного цвета делятся по весу, затем объекты одного цвета и веса делятся по размеру и т. д.

Можно выделить два основных пути организации поиска объектов (или списков объектов) в

признаковых структурах:

А. *Структурный путь,* реализуемый в виде различного рода поисковых деревьев (древовидных признаковых структур).

Б. *Вычислительный путь,* реализуемый в виде различного рода алгоритмов вычисления адресов объектов (адресов их записей или формуляров) по значениям признаков этих объектов.

Рассмотрим структурный путь. В общем случае древовидные признаковые структуры могут состоять из двух частей: собственно поискового дерева, определяющего систему классификации объектов по заданному набору признаков, и совокупности списков объектов, объединяющих объекты, обладающие определенными значениями признаков.

Способ поисковых деревьев имеет большое количество различных модификаций. Мы рассмотрим две модификации этого способа:

а) позиционное поисковое дерево, структура которого определяется заранее в процессе программирования. При этом способе каждый признак представляется определенным разрядом (позицией) в некотором многоразрядном числе, заданном в позиционной системе счисления. Допустимый набор признаков и число значений каждого признака заранее фиксированы;

б) наращиваемое поисковое дерево, формируемое в процессе включения в систему новых объектов. При этом набор допустимых значений признаков заранее не фиксируется; он может произвольно изменяться в соответствии с фактическими значениями признаков, имеющимися у включаемых в списки объектов.

В качестве членов объектных списков в общем случае могут фигурировать также ассоциативные списковые структуры различных видов, в том числе древовидные признаковые структуры, объектные структуры, просто объекты и т. д.

Признаковые структуры с позиционными поисковыми деревьями. В данных признаковых структурах наборы значений признаков, которыми обладают те или иные объекты, представляются в виде многоразрядных чисел; при этом количество разрядов соответствует количеству признаков. За каждым признаком закрепляется определенный разряд, и количество значений этого разряда соответствует количеству значений соответствующего признака. В общем случае может получиться система счисления с различным количеством значений разных разрядов, т. е. с разными основаниями. Ясно, что наиболее просто оперировать с системой, имеющей одно основание, однако это может привести в некоторых случаях к неэкономной кодировке значений признаков, так как нужно будет выбирать основание системы счисления, исходя из признака, имеющего наибольшее число значений. При этом для других признаков эта возможность кодирования не будет использоваться в полной мере.

Позиционное поисковое дерево строится обычно следующим образом. В вершине дерева помещается один список, который будет списком верхнего (1-го) уровня, этот список осуществляет деление всех объектов по одному какому-нибудь признаку (например, цвету). От каждого члена этого списка отходят подсписки второго уровня, соответствующие делению объектов по другому признаку (например, весу). От членов этих подсписков отходят подсписки третьего уровня, соответствующие делению объектов по третьему признаку, и т. д. Членами подсписков самого нижнего уровня дерева будут объекты, обладающие теми значениями признаков, которые соответствуют всей цепочке членов подсписков всех уровней дерева, ведущей к данному члену нижнего подсписка. Вместо отдельных объектов в подсписках дерева нижнего уровня могут фигурировать списки объектов, обладающих заданным набором признаков. Такие списки объектов мы будем называть объектными списками. В общем случае членами подсписков нижнего уровня в поисковых деревьях могут быть не только объекты и объектные списки, но и другие списковые структуры (объектные и признаковые).

Позиционные поисковые деревья удобно использовать для поиска объектов в тех случаях, когда задается полный набор признаков и порядок расположения этих признаков соответствует порядку построения поискового дерева, т. е. следованию признаков по уровням дерева. Тогда сначала просматривается список верхнего уровня и в нем отыскивается член, имеющий значение, соответствующее значению первого признака (старшего разряда) в заданном наборе. В подсписке, отходящем от найденного члена, находится член, имеющий значение, соответствующее второму разряду числа, и т. д.

Если же для поиска задается неполный набор признаков и они заданы в другом порядке, то процесс поиска усложняется. При этом необходимо просматривать для каждого незаданного признака все его значения, т. е. вести поиск сразу по нескольким параллельным ветвям дерева. Для облегчения поиска объектов по неполным наборам признаков могут быть использованы поисковые деревья с

горизонтальными или поперечными связями. Как уже говорилось, поисковые признаковые деревья представляют собой иерархическую систему классификации объектов по заданному набору признаков. В этих деревьях в подсписках различных уровней кроме самого верхнего уровня могут повторяться однотипные подсписки, осуществляющие деление объектов по одному и тому же признаку. Например, если мы делим объекты по цвету и по весу и решили построить список верхнего уровня по цвету, то списки значений признака второго уровня, осуществляющие деление объектов по весу, могут частично или полностью повторяться. Аналогичным образом могут многократно повторяться и подсписки других уровней.

В поисковых деревьях с поперечными связями указанные однотипные подсписки или даже однотипные члены в этих подсписках объединяются в новые цепные списки при помощи дополнительных адресов связи; эти связи как бы пересекают древовидную структуру в горизонтальном направлении. Использование таких связей позволяет в некоторых случаях повысить эффективность поисков объектов по различным неполным комбинациям признаков по сравнению с обычными древовидными структурами. При этом каждый горизонтальный цепной список имеет свою вершину и кодом данного признака считается адрес ячейки, представляющей собой вершину этого горизонтального списка. Таким образом, любой признак будет иметь только один код, и, задавая комбинации признаков в виде последовательностей их кодов, можно однозначно находить соответствующие им списки объектов. Другим вариантом; построения признакового поискового дерева является построение его без поперечных связей, но с указанием в каждом члене подсписков поискового дерева кода наименования признака того подсписка, который отходит от этого члена; значения же этого признака будут указываться специальным кодом в каждом члене этого ответвляющегося подсписка. Таким образом, в каждом члене любого подсписка помимо двух адресов связи будут находиться два кода признака: код значения данного признака и код наименования следующего признака.

Поисковые деревья могут рассматриваться как ориентированные графы, имеющие одну начальную вершину и совокупность промежуточных и оконечных вершин. Вершины соединяются между собой ребрами. В каждую вершину может входить только одно ребро, а выходить может несколько. Все вершины дерева делятся на ряд уровней.

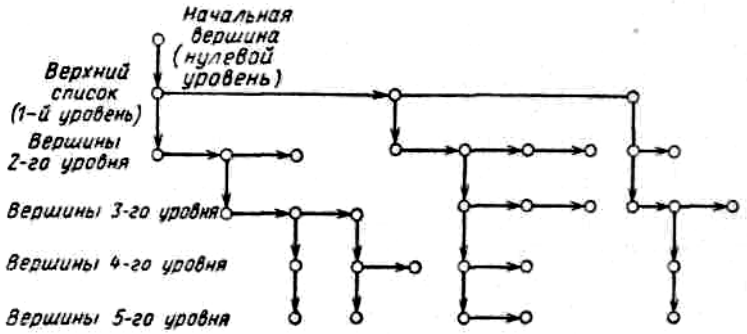


Рис. 18. Схема поискового дерева.

Вершины, из которых не выходит ни одно ребро, называются оконечными. Последовательность вершин и ребер, ведущая от начальной вершины к какой-либо оконечной вершине, называется путем. С точки зрения применения цепных списков для построения поисковых деревьев удобно представлять эти деревья графически следующим образом (рис. 18): будем считать, что все ребра дерева могут быть двух видов: вертикальные и горизонтальные. В списковых членах горизонтальным ребрам соответствуют одни адреса связи (например, первые), а вертикальным ребрам — другие адреса связи (например, вторые).

Из начальной вершины может выходить только одно вертикальное ребро. Переход от одной вершины к другой, осуществляемый с помощью вертикального ребра, будем считать переходом на другой уровень дерева (переход к подсписку), а переход от одной вершины к другой, осуществляемый при помощи горизонтального ребра, означает перемещение вдоль одного уровня дерева (вдоль одного подсписка).

Таким образом, совокупность вершин, связанных горизонтальными ребрами, соответствует одному подсписку. Номер уровня любой вершины определяется количеством вертикальных ребер, ведущих от начальной вершины к данной вершине.

Поиск объектов в признаковых структурах обычно осуществляется в две стадии: сначала по заданным признакам отыскивается с помощью поискового дерева нужный объектный список, а затем

производится поиск нужного объекта в данном списке (либо по другим признакам, либо по собственной информации об объекте). Поисковые деревья могут быть либо постоянными, либо переменными. Постоянные поисковые деревья строятся заранее в процессе программирования задачи и охватывают все возможные подразделения классификации объектов. Переменные поисковые деревья образуются в процессе обработки информации и включают только те подразделения классификации, которые относятся к объектам, фактически имеющимся в системе.

В качестве отдельных признаков могут использоваться и сочетания нескольких признаков. Тогда общее число значений такого комбинированного признака будет равно произведению количества значений элементарных признаков (если совместимы все сочетания значений этих признаков). Количество различных объектных списков, отходящих от поискового дерева, равно общему количеству подразделений классификации нижних уровней (нижние уровни для различных путей не обязательно должны быть одинаковы).

Значения поисковых признаков, имеющих количественный смысл, обычно задаются в десятичной системе счисления. В этом случае можно строить двоично-десятичные поисковые деревья, в которых каждому разряду десятичного числа соответствует двухуровневое поисковое поддерево. На верхнем уровне поддерева строится один подсписок с тремя членами (00, 01, 10), а на нижнем уровне — три подсписка, из которых, два первых имеют по четыре члена (00, 01, 10, 11), а третий имеет два члена (00, 01). Из подобных поддеревьев может строиться поисковое дерево для любых многоразрядных признаков, заданных в десятичной системе.

Пример позиционной поисковой структуры для поиска слов в словаре. Одна из типовых задач, встречающихся при обработке больших объемов информации, представленной на естественном языке (машинный перевод, поиск научно-технической информации и т. п.), заключается в нахождении слов в словаре. Если словарь составлен в алфавитном порядке и имеет неизменный характер, то здесь эффективно может быть применен вариант бинарного поиска (деление пополам). Этот метод требует проверки всего $\log_2 N$ членов словаря, где N— общее число членов в словаре. Однако такой словарь, представляющий собой, по существу, последовательный список, неудобен в тех случаях, когда требуется его часто изменять (исключать старые или добавлять новые члены). Как мы уже упоминали, если включаемые члены равномерно распределены по всему объему словаря, то на каждое включение будет приходиться в среднем N/2 сдвигаемых членов. Алфавитный словарь, организованный в виде простого цепного списка, удобен для изменения, но требует выполнения слишком многих операций при поиске.

Древовидная признаковая структура представляет собой такую форму организации данных, которая обеспечивает быстрый поиск (почти как при бинарном методе) и легкость изменения словаря.

Опишем вариант построения поискового дерева для такого словаря.

Это дерево строится по цепному способу из списковых слов, состоящих из трех частей (слогов):

1) наименования члена списка *Н*; в данном случае наименование представляет собой код некоторой буквы;

2) первого адреса связи АС1, представляющего собой адрес ответвляющегося подсписка;

3) второго адреса связи АС2, представляющего собой адрес следующего члена данного подсписка.

Ниже показан формат ассоциативного слова. Окончание цепных списков обозначается, как обычно, символом КС.

Н	АС1	АС 2

Верхний уровень поискового дерева представляется цепным списком, в котором каждый член соответствует одной букве алфавита, причем в этот список входят только те буквы, с которых могут начинаться слова (например, буквы Ы, Й, Ъ, Ь не войдут в список верхнего уровня). От каждого члена списка верхнего уровня ответвляется цепной подсписок, включающий в себя буквы, которые могут находиться на втором месте в словах, начинающихся с букв, указанных в соответствующих членах списка верхнего уровня. От каждого члена списка второго уровня отходит подсписок третьего уровня, содержащий буквы, которые могут стоять на третьем месте в словах, имеющих две первые буквы, указанные на соответствующих местах в списках первого, второго уровней и т. д.

На рис. 19 представлена часть списковой структуры для некоторых слов, начинающихся с буквы Д. Цифры, стоящие перед прямоугольниками, показывают адреса соответствующих ячеек памяти.

В данном примере показан фрагмент списковой структуры без указаний окончаний цепных списков

(подсписков) одного уровня. Вместо вторых адресов связи в тех списковых словах, на которых оборваны подсписки, оставлены пустые места, которые обозначают, что этот подсписок можно продолжить.

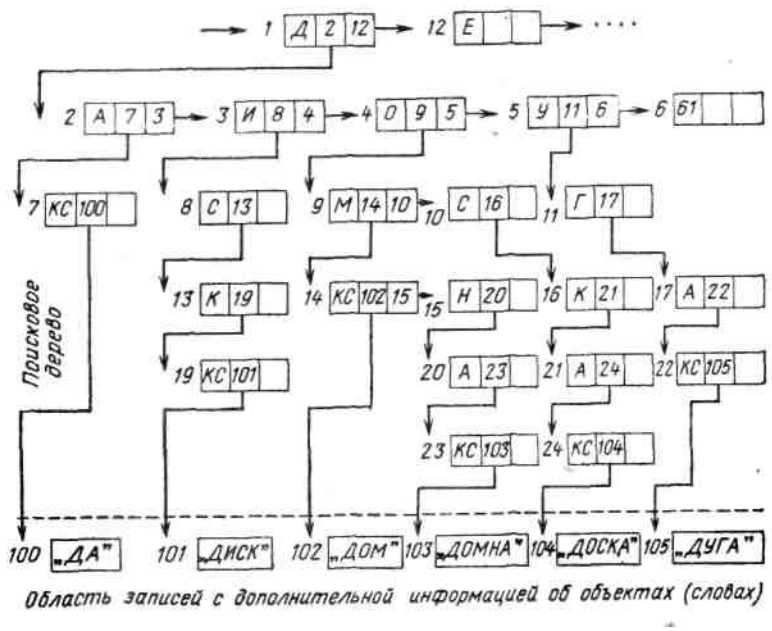


Рис. 19. Фрагмент ассоциативной структуры словаря.

В данном поисковом дереве различные ветви могут включать в себя общие участки (общие вертикальные звенья), причем ветви, соответствующие различным словам, имеющим одинаковые начальные группы букв, могут иметь различную длину. Например, ветви, соответствующие словам «дом» и «домна», имеют общую часть «дом», а все ветви, показанные на рис. 19, имеют общую вершину, соответствующую начальной букве Д.

Для указания окончания любой ветви используется так называемое списковое ассоциативное слово, в котором на месте наименования члена списка (кода буквы в нашем случае) ставится символ КС. В оконечных списковых словах АС1 используется для указания адреса записи в словаре, содержащей необходимую информацию о данном объекте (например, эквивалент данного слова на другом языке, грамматические сведения о слове и т. д.). Если это оконечное списковое слово используется для указания окончания слова, не являющегося самым длинным из данного семейства слов, то АС2 в этом списковом слове указывает адрес следующего члена в данном подсписке, который будет указывать следующую букву более длинного слова, имеющего окончившееся слово своей начальной частью. В оконечных списковых словах наиболее длинных ветвей АС2 не используется.

Цепная адресация, примененная в данной списковой структуре, обеспечивает легкость изменения как горизонтальных (путем изменения вторых адресов связи АС2), так и вертикальных подсписков (путем изменения первых адресов связи АС1).

Естественно, что все ячейки памяти, используемые для хранения членов подобных древовидных структур, могут располагаться в памяти произвольным образом. Скорость поиска объектов с помощью таких структур зависит от числа уровней в структуре и количества членов в горизонтальных подсписках.

К недостаткам подобных словарей следует отнести сравнительно большой расход памяти под поисковые деревья.

Этот расход зависит от общего количества слов в словаре, среднего размера слов, а также от степени гнездования слов (от доли слов, имеющих общие начальные части).

Некоторые соотношения, дающие общую ориентировку при выборе размеров подсписков и числа уровней в поисковых деревьях, будут приведены в следующем параграфе.

Для повышения скорости поиска слов в таком словаре целесообразно располагать буквы в подсписках дерева в порядке убывания их частостей встречаемости на соответствующих местах в словах. Однако такое расположение букв требует большой работы по анализу текстов той области науки, для которой составляется словарь, с целью выявления частостей встречаемости букв на определенных местах в словах и после определенных сочетаний букв.

Можно пойти и по другому пути: не проводить подобных исследований заранее, построить

описанный древовидный словарь сначала более или менее произвольным образом и обеспечить в процессе эксплуатации его «самообучение».

Для этого должен осуществляться учет частостей обращений к различным словам (буквам) в данном словаре и соответствующая его перестройка, т. е. изменение положений букв в подсписках деревьев.

Очевидно, что при использовании цепного способа построения деревьев такое изменение подсписков не вызовет затруднений.

Признаковая структура с наращиваемым поисковым деревом. Способ наращиваемых поисковых деревьев, предложенный В. И. Ландауером и Н. С. Прайсом [14[, обеспечивает возможность изменения поисковых деревьев в процессе работы в соответствии с фактическими значениями признаков рассматриваемых объектов.

Наращиваемые поисковые деревья служат для нахождения по заданному набору значений признаков (дескрипторов) адресов объектов (или вершин списков объектов), обладающих этими признаками.

Наращиваемое поисковое дерево может использоваться в качестве составной части в признаковых структурах в сочетании с объектными списками, построенными по различным принципам: последовательному, гнездовому, цепному и узловому. Во всех этих случаях поиск объектов осуществляется в два этапа — сначала при помощи поискового дерева находится нужный объектный список, а затем находится нужная позиция данных (объект) внутри этого списка. Наращиваемое поисковое дерево представляет собой совокупность вершин и ребер, расположенных по определенным уровням табл. 12. Число ребер, отходящих от каждой вершины, определяется из условия минимума времени поиска (см. ниже). Для простоты в примерах рассматривается дерево, у которого от каждой вершины отходит не более трех ребер. Вершины дерева в памяти машины представляются отдельными гнездами ячеек (группами последовательных ячеек N1, N2, N3 и т. д.). Каждое гнездо содержит списковые слова, размещаемые в отдельных ячейках. Таким образом, в каждом гнезде ячеек размещается последовательный подсписок поискового дерева.

Таблица 12

Номер вершины дерева	Адрес спискового слова	Списковое слово			
		Значение признака	КП	КД	Адрес связи
N 1	0100	П0	0	0	1022 — Адреса вершин дерева
	0101	П1	0	0	0135
	0102	П2	1	0	0511
N 2	0135	П5	0	1	0612 — Адреса вершин объективных списков
	0136	П1	0	1	2031
	0137	П8	1	1	5042
N 3	0511	П8	0	1	0256
	0512	П12	0	1	6011
	0513	П10	1	1	1531
N 4	1022	П15	0	1	2041
	1023	П7	0	1	2512
	1024	П6	1	1	0172

Число членов (списковых слов) в подсписке равно числу ребер, отходящих от данной вершины.

Адресом подсписка является адрес первой ячейки (первого спискового слова) гнезда. Как показано в табл. 12, каждое списковое слово содержит два слога: признаковый слог и адресный слог.

Признаковый слог представляет собой некоторый код, с помощью которого различаются отдельные вершины дерева; это идентификаторы вершин дерева. Наращиваемое поисковое дерево строится для одного поискового признака, имеющего количественный смысл; наименование этого признака является общим для всего дерева, и поэтому оно в признаковых слогах не указывается. В них указываются только значения этого признака.

Особенностью рассматриваемого дерева является расположение списковых слов в гнездах (подсписках) в порядке возрастания значений поискового признака (например, в гнезде N1 П0< П1<П2; в гнезде N2 П5<П1<П8 и т. д.). Кроме того, в деревьях рассматриваемого типа значения признака в списковых словах всех подсписков (гнезд), относящихся к одному уровню, также возрастают при переходе от гнезда к гнезду слева направо (например, для всех подсписков первого уровня: П5<П1<П8< <П9< П12<П10<П15<П7<П6).

Адресный слог содержит в себе собственно адрес связи и два служебных разряда КП и КД. Адреса

связи служат для указания переходов внутри поискового дерева от членов вышестоящих подписков к нижестоящим ответвляющимся подспискам.

Служебный разряд КП (конец позиции) является признаком окончания данного гнезда (подсписка, соответствующего определенной вершине дерева). Этот разряд имеет смысл в тех случаях, когда подписки поискового дерева имеют переменный состав. Служебный разряд КД является признаком окончания дерева; во всех подсписках, кроме подписков нижнего уровня, этот разряд равен нулю. В списковых словах подписков нижнего уровня этот разряд равен единице. Наличие единицы указывает на то, что адреса связи в подсписках нижнего уровня дерева являются уже не адресами нижестоящих подписков дерева, а адресами вершин объектных списков. Служебный разряд КД имеет смысл в тех случаях, когда число уровней в различных ветвях дерева различно.

Для поиска объектов задается значение признака, общее для всех объектов, объединенных в одном объектном списке. Поиск производится путем прослеживания вершин дерева сверху вниз и сравнения заданного значения признака со значением признака в просматриваемых подсписках. Дерево строится таким образом, чтобы в каждом подсписке, ответвляющемся от некоторого спискового слова вышестоящего подсписка, наибольшее значение признака (т. е. значение признака в последнем члене этого подсписка) было равно значению признака в списковом слове вышестоящего уровня. Таким образом, при прослеживании подписков дерева нужно переходить к нижестоящему подписку, ответвляющемуся от того спискового слова данного подсписка, в котором значение признака оказалось равным или большим заданного значения признака (при условии, что в предшествующем списковом слове этого подсписка значение признака было меньше заданного).

Таким образом, в наращиваемом поисковом дереве подписок самого верхнего уровня делит весь диапазон изменения значения признака на некоторые достаточно крупные интервалы. Каждый подписок следующего уровня, т. е. подписок, ответвляющийся от каждого спискового слова верхнего подсписка, делит соответствующий интервал на более мелкие интервалы. Подписки третьего уровня делят соответствующие подынтервалы на еще более мелкие подынтервалы и т. д.

Сбалансированным деревом называется такое дерево, у которого число уровней в разных ветвях отличается не больше чем на единицу. При этом образование нового уровня дерева начинается только после заполнения всех свободных списковых слов в подсписках предыдущего уровня. Сбалансированное дерево наращивается по мере добавления новых объектов и образования объектных списков с другими значениями признаков. При образовании нового подсписка дерева берется новое гнездо свободных ячеек памяти, т. е. группа ячеек, расположенных последовательно. Число ячеек в гнезде должно быть равно числу списковых слов, которые должны входить в новый подписок. Это число должно быть фиксировано заранее и известно в момент образования нового подсписка. В последнем списковом слове нового подсписка записывается значение признака, взятое из вышестоящего исходного спискового слова, а в предпоследнем списковом слове нового подсписка записывается новое значение признака, необходимость включения которого и обусловила образование нового подсписка. При этом несколько ячеек, находящихся в начальной части нового гнезда, остаются временно свободными. В дальнейшем при поиске таких свободных ячеек этот факт устанавливается путем проверки только первых ячеек гнезд.

При записи каждого нового объекта производится проверка наличия в поисковом дереве того значения поискового признака, которое характеризует записываемый объект. Если это значение признака уже имеется, то новый объект просто включается в соответствующий объектный список без каких-либо изменений в дереве. Если требуемого значения признака в дереве нет, то его включают в нижний уровень дерева в то место, которое соответствует условию монотонного возрастания значений признака. Если в соответствующем подсписке нет свободной ячейки, то производится поиск ближайшей свободной ячейки в других подсписках нижнего уровня и затем сдвиг вправо или влево всех промежуточных членов в подсписках нижнего уровня с корректировкой значений признака в подсписках вышестоящих уровней.

На рис. 20 дан пример указанной признаковой структуры. Часть, расположенная выше пунктирной линии, представляет собой наращиваемое дерево. В дереве каждый прямоугольник представляет один подписок, состоящий из трех членов (списковых слов). Для слов указаны только значения признака, а вместо адресов связи используются стрелки, ведущие от исходных слов к ответвляющимся подспискам. Под пунктирной чертой находится область объектных списков, содержащая данные об объектах с их значениями поискового признака, причем объекты с одинаковыми значениями признака объединены в общий список при помощи адресов связи. В данном случае допускается, что один объект может иметь

несколько значений поискового признака (это может быть, например, если под одним объектом понимать группу предметов), т. е. различные значения признака, для которого построено данное поисковое дерево, не являются взаимно исключающими.

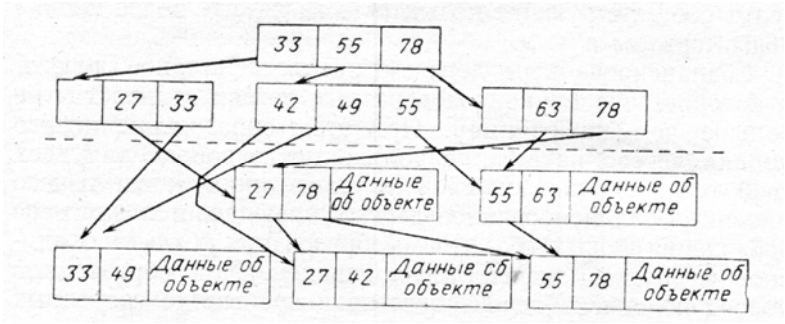


Рис. 20. Схема признаковой структуры.

В вычислительной системе с описанной организацией памяти [14] фактически образуются две ассоциативные структуры: одна для команд программы и вторая для информации об объектах.

Исходное программирование для этой системы ведется на псевдоязыке, состоящем из набора списковых операторов, который может расширяться по мере надобности. Каждый оператор реализуется при помощи машинной подпрограммы, хранящейся в виде одного подсписка. Работа машины осуществляется по интерпретативному способу, при котором операторы исходной программы последовательно исполняются интерпретирующей программой.

Достоинствами наращиваемого поискового признакового дерева является возможность произвольного деления интервала изменения признака в зависимости от фактического состава поступающих объектов и гибкость использования памяти, не требующая ее предварительного распределения, а также эффективность поиска данных за счет оптимального выбора числа ветвей и уровней в дереве. Недостатком этого способа является необходимость сдвигов значений признака при наращивании дерева.

Многосписковые ассоциативные структуры. Н. С. Прайсом и Х. И. Греем предложена организация списковой структуры, аналогичная рассмотренной в предыдущем разделе. Она также основана на цепной адресации. Изложим кратко эту систему, описанную в [12]. Вся память машины разделена на две области: область поисковых деревьев и многосписковую область или область ассоциативных узлов.

Для построения поисковых деревьев и ассоциативных узлов используются слова (называемые катенами) двух видов: слова данных и списковые слова. Оба вида слов имеют одинаковые размеры. Формат слова данных представлен на рис. 21, формат спискового слова — на рис. 22.

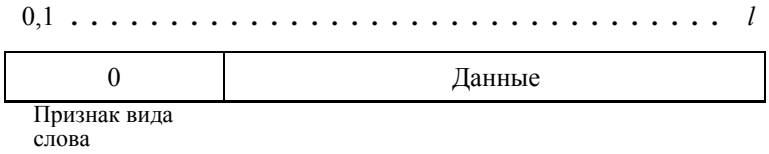


Рис. 21. Формат слова данных.

Из указанных слов формируются информационные позиции, т. е. отдельные записи, соответствующие определенным объектам. Эти позиции будут представлять собой в поисковом дереве вершины дерева, а в многосписковой области — ассоциативные узлы. Узлы могут иметь переменный состав, т. е. различное число слов. В каждой адресуемой ячейке памяти может размещаться фиксированное число слов; если требуемое число слов не помещается в n-й ячейке, то занимается следующая ячейка (n+ 1-я ячейка памяти); если же следующая ячейка окажется занятой, то в последнем слове, находящемся в

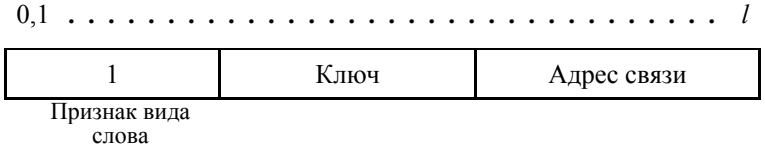


Рис. 22. Формат спискового слова.

данной n-й ячейке, на месте ключа пишется адрес, указывающий продолжение данной позиции (продолжение ассоциативного узла). В каждом слове выделяется еще дополнительный разряд, указывающий окончание позиции. При помощи списковых слов различные позиции (узлы)

объединяются в списки, обладающие одинаковыми ключами (признаками).

Каждый узел может входить во столько списков, сколько списковых слов имеется в узле.

Поиск объектов, обладающих заданным набором признаков, осуществляется в два этапа: сначала при помощи поискового дерева ключ, заданный в запросе, преобразуется в адрес начала списка объектов, обладающих данным ключом; затем просматривается этот список и отбираются объекты, обладающие всеми заданными в запросе ключами. В [12] рассматривается пример информационной системы для учета кадров, рассчитано па 1 млн. позиций. Каждое лицо характеризуется 15 признаками, причем каждый признак может иметь 10 значений; таким образом, информация о каждом лице представляет 15-разрядным десятичным числом.

В системе принят постоянный размер позиций, являющихся как вершинами поискового дерева, так и ассоциативными узлами. Каждая позиция содержит пять списковых слов. Для того чтобы представить в одной позиции (с помощью пяти слов) 15 признаков, эти признаки делятся на пять групп по три признака в группе.

Каждая группа из трех признаков образует один надпризнак, который может иметь 1000 различных значений, и, следовательно, одному надпризнаку может соответствовать 1000 объектных списков. Всем пяти надпризнакам соответствует 5000 различных объектных списков, которые могут использоваться в системе.

Поисковое дерево состоит из пяти и частично из шести уровней: на первом (верхнем) уровне имеется одна позиция — последовательный список, состоящий из 5 слов, от которого отходят пять ветвей к пяти подспискам второго уровня. Каждый подсписок второго уровня соответствует одному надпризнаку. От этих подсписков ответвляются подсписки третьего уровня, тоже состоящие из пяти членов. От подсписков третьего уровня отходят такие же подсписки четвертого уровня, а от подсписков четвертого уровня отходят подсписки пятого уровня.

В этих подсписках первые четыре члена указывают на вершины объектных списков, расположенных в многосписковой области, а последний член каждого подсписка пятого уровня указывает на подсписок шестого уровня, состоящий из четырех членов, каждый из которых содержит адрес вершины списка в многосписковой области.

Картотека учета кадров

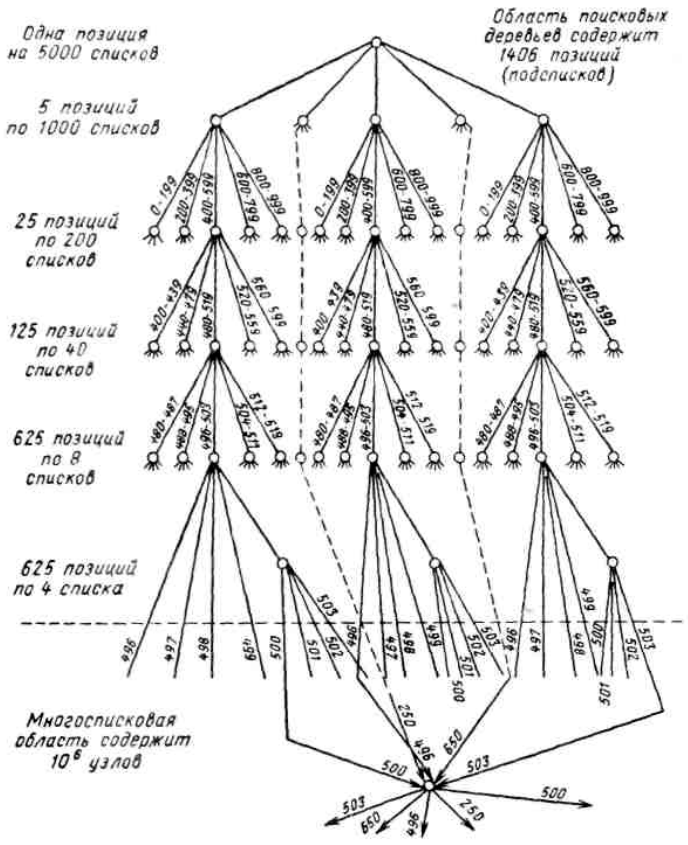


Рис. 23. Ассоциативная структура для учета кадров.

На рис. 23 приведена схема описанной ассоциативной структуры. Пунктирными линиями показаны связи данного ассоциативного узла с поисковыми поддеревьями, соответствующими второму и

четвертому надпризнакам. Чтобы не загромождать рисунок, связи показаны упрощенно. В действительности они имеют вид таких же ветвящих деревьев, как и связи для остальных надпризнаков.

Ассоциативный узел будет состоять из пяти списковых слов и нескольких слов данных; для узла, показанного на рисунке, ключи в списковых словах, представляющие собой надпризнаки, будут иметь следующие значения:

500	Адрес связи	250	Адрес связи	496	Адрес связи	650	Адрес связи	503	Адрес связи

Для оценки эффективности подобных ассоциативных систем Прайсом и Греем предложен комбинированный критерий, представляющий собой произведение времени поиска заданного объекта на требуемую общую емкость памяти [12].

Более подробно эти вопросы будут рассмотрены ниже.

Способ вычисляемых адресов. Способ вычисляемых адресов основан на преобразовании заданных наборов поисковых признаков либо в адреса записей с собственной информацией об объектах, либо в адреса ассоциативных узлов объектов или в адреса вершин списков объектов.

Естественными требованиями к таким вычислениям являются однозначность преобразования и соответствие получаемого диапазона изменения адресов размерам области памяти, отведенной для размещения искомых позиций. Различают вычисление прямых адресов, когда в результате получаются непосредственно адреса искомых позиций, и вычисление непрямых адресов, когда в результате получаются адреса ячеек, содержимым которых являются адреса искомых позиций. В общем случае значения признаков объектов могут изменяться произвольно, и многие объекты могут обладать одним и тем же значением данного поискового признака. Если считать, что в одной ячейке может быть записан только один член списка, то естественно все объекты с одинаковым значением поискового признака объединять в цепные списки, а при помощи вычислений определять адреса вершин таких списков.

Существует еще другой так называемый открытый способ размещения и поиска объектов, обладающих одинаковым значением поискового признака. При этом способе первый появившийся объект записывается в ячейку с адресом, полученным в результате преобразования значения признака в адрес, а каждый последующий объект записывается в ближайшую в сторону возрастания адресов свободную ячейку памяти. Поиск членов при этом происходит в два этапа. Сначала по заданному значению поискового признака определяется адрес первого объекта, а затем путем прямого просмотра в сторону возрастания адресов определяются остальные объекты, обладающие тем же значением этого признака. Как запись, так и поиск членов производятся циклично, т. е. после заполнения последней ячейки памяти снова используется первая ячейка и т. д.

Запись непрямых адресов может быть осуществлена двумя способами: для этого выделяются либо специальная группа ячеек памяти, либо определенная часть во всех ячейках памяти.

В последнем случае обеспечивается более широкий диапазон изменения адресов, но значительно возрастает расход памяти, так как во всех ячейках памяти необходимо выделять две адресные части: под вычисляемые (непрямые) адреса вершин списков и под адреса, используемые непосредственно для связи членов внутри цепных списков. Использование одной и той же адресной части в ячейке для обеих целей не представляется возможным, так как могут быть случаи, когда одна и та же ячейка должна указывать положение вершины некоторого цепного списка (т. е. содержать в себе непрямой адрес) и в то же время содержать член некоторого цепного объектного списка со своим адресом связи. Способ вычисляемых адресов удобно применять тогда, когда значения признака, по которому производится поиск объектов, изменяются монотонно, или когда их удается аппроксимировать монотонной функцией.

Например, этот способ можно применять вместо постоянных поисковых деревьев с одинаковым числом членов во всех подсписках, как показано ниже.

Пусть, например, мы имеем систему классификации объектов, включающую в себя шесть признаков: П1, П2, . . . П6, каждый из которых может иметь три значения. Сравним для этого случая способы поискового дерева и вычисляемых адресов. Строим три поисковых дерева, объединив в каждом из них два признака в один надпризнак.

На рис. 24 показана таблица значений признаков от П1 до П6. Объединение в надпризнаки произведено следующим образом: признаки П1 и П2 образуют первый надпризнак, которому соответствует поисковое дерево *I*; признаки П3 и П4 образуют второй надпризнак, которому

соответствует поисковое дерево *II*, и признаки П5 и П6 объединяются в третий надпризнак, которому соответствует поисковое дерево *III*.

н а д п р и з н а к и

I		*II*		*III*	
П1	*П2*	*П3*	*П4*	*П5*	*П6*
01	*01*	*01*	*01*	*01*	*01*
10	*10*	*10*	*10*	*10*	*10*
11	*11*	*11*	*11*	*11*	*111*

Рис. 24. Таблица значений признаков

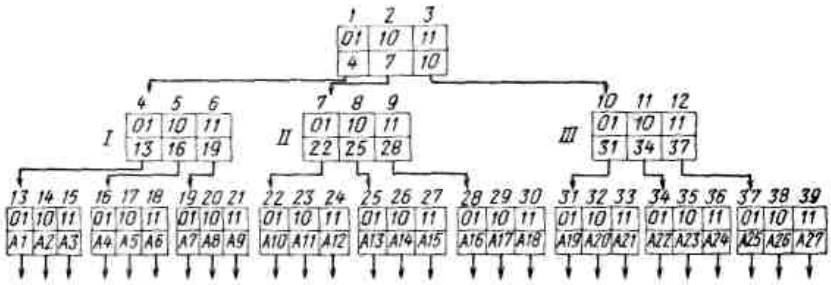


Рис. 25. Поисковые деревья.

На рис. 25 показаны три поисковых дерева *I*, *II*, *III*. Подсписки расположены в последовательных ячейках памяти с адресами от 1 до 39. В каждой ячейке имеется две части: значение признака и адрес с ответвляющегося подсписка. В подсписках нижнего уровня адресные части указывают положение вершин соответствующих объектных списков (с A1 по A27). Над клетками указаны адреса соответствующих ячеек, занимаемых членами подсписков поисковых деревьев.

Используя данные поисковые деревья (*I*, 77 и *III*), можно осуществлять поиск объектов следующим образом:

а) из заданного набора значений шести признаков (П1—П6) выбираем одну пару признаков (П1 и П2 или П3 и П4, или П5 и П6), которая будет поисковым надпризнаком, и соответствующее этому надпризнаку поисковое дерево. Адрес вершины этого поискового дерева определяем при помощи подсписка верхнего уровня по коду поискового дерева (*I*-01, *II*-10, *III*-11);

б) в выбранном поисковом дереве просматриваем список верхнего уровня и находим в нем член, имеющий то же значение, что и первый признак в выбранном нами поисковом надпризнаке;

в) по адресу связи, имеющемуся в этом члене, находим подсписок второго уровня;

г) просматриваем подсписок второго уровня и находим в нем член, имеющий то же значение признака, что и второй признак, входящий в поисковый надпризнак;

д) по адресу связи, имеющемуся в этом члене, находим искомый объектный список;

е) просматриваем найденный объектный список и отбираем в нем объекты, удовлетворяющие заданным значениям остальных четырех признаков (не вошедших в состав поискового надпризнака).

При этом придется в среднем просматривать половину членов подсписков первого и второго уровней и половину членов объектного списка (считаем, что все признаки равновероятны).

Поиск требуемого объектного списка (или отдельного объекта) при помощи поискового дерева может быть заменен вычислением адреса ячейки, в которой находится адрес вершины объектного списка. Для этого вместо трех поисковых деревьев *I*, *II*, *III* достаточно иметь группу из n последовательно расположенных ячеек памяти, в которых хранятся адреса вершин объектных списков (A1—An), где n — общее число объектных списков. Пусть $b0$ означает адрес ячейки, после которой располагаются ячейки указанной группы. При сделанных допущениях о системе кодирования признаков (П1—П6) и указанном расположении ячеек, содержащих адреса вершин объектных списков, формула для вычисления адресов вершин объектных списков по заданным значениям признаков n_i и n_{i+1} и номеру (коду) поискового дерева k (k = 1, 2 или 3) будет иметь вид

$$A = b0 + 3^2 (k — 1) + 3 (n_i — 1) + n_{i+1}.$$

Достаточно просто могут быть составлены формулы для вычисления адресов при других конфигурациях поисковых деревьев.

Вообще говоря, способ вычисляемых адресов во многих случаях позволяет получить экономию в количестве ячеек памяти и сокращение числа операций при поиске.

Число операций, необходимых для вычисления непрямого адреса, пропорционально числу признаков (т. е. числу уровней дерева) и не зависит от числа значений признаков (т. е. от числа членов в подсписках поискового дерева), в то время как число операций просмотров членов подсписков при поиске по поисковому дереву пропорционально произведению числа признаков на число значений признаков (числа уровней дерева на число членов в подсписках). Следует заметить, что при переменных поисковых деревьях и постоянных деревьях, но имеющих различную длину признаковых

подсписков, необходимы еще ячейки для хранения сведений о числе членов в признаковых подсписках, а алгоритм вычисления непрямых адресов усложняется за счет учета размеров подсписков.

Другим примером применения способа вычисляемых адресов может служить алгоритм, осуществляющий регистрацию некоторых объектов в зависимости от их положения на плоскости или в пространстве. Пусть положение объектов определяется прямоугольными координатами (x, y, z). При помощи некоторого заданного числа старших разрядов координат объектов все пространство может быть разбито на области, для которых все находящиеся в каждой области объекты должны объединяться в один список. Из выделенных старших разрядов координат x, y, z может быть образовано одно двоичное число, являющееся адресом ячейки, в которой хранится адрес вершины соответствующего списка. Недостатком этого способа является, очевидно, необходимость резервирования ячеек для непрямой адресации всех возможных таких списков, хотя в действительности многие списки могут быть пустыми.

Одной из распространенных модификаций способа вычисляемых адресов является способ поиска слов в словаре при помощи сверток, получаемых по кодам букв, образующих слово.

Пусть, например, все буквы алфавита имеют свои постоянные коды, представляющие собой шестиразрядные двоичные числа. Тогда для любого слова может быть получена его свертка, т. е. двоичное число заданной разрядности (например, 10 или 15 разрядов). Один из простых способов свертывания состоит в том, что коды букв, образующих слово, складываются поразрядно со сдвигом на один разряд вправо или влево. Сдвиг производится в пределах заданной разрядности кода свертки, после чего коды остальных букв складываются без сдвигов. Пусть, например, имеем следующие коды букв: *а* —000001, *в* — 000010, *е* — 001000, *к* —001010, *р*—100100, *с* —100101, *т*— 100110. Тогда десятиразрядная двоичная свертка слова «свертка» может быть получена следующим образом (рис. 26).

Основным недостатком способа сверток является возможность появления неоднозначности сверток, т. е. случаев, когда различные слова после свертывания будут иметь одинаковые коды сверток. Для исключения неопределенности в этих случаях добавляют вспомогательные признаки, введение которых, естественно, усложняет процесс поиска. Кроме того, использование кодов сверток слов в качестве адресов ячеек памяти, в которых хранится информация о данных словах, приводит к неравномерности в заполнении памяти, так как получаемые адреса, вообще говоря, имеют случайный характер.

Номера разрядов кода свертки

	1	2	3	4	5	6	7	8	9	10
с	1	0	0	1	0	1				
b		0	0	0	0	1	0			
е			0	0	1	0	0	0		
р				1	0	0	1	0	0	
т					1	0	0	1	1	0
к					0	0	1	0	1	0
а					0	0	0	0	0	1
	1	0	0	0	0	0	0	1	0	1

Коды отдельных букв

Рис. 26. Код свертки, полученный путем поразрядного (без переносов) сложения кодов букв.

Некоторые соотношения для ассоциативных списковых структур. Рассмотрим случай симметричного поискового дерева, т. е. такого дерева, в котором все ветви имеют одинаковое число уровней k и от каждого члена подсписков не нижнего уровня отходит подсписок с n членами. От каждого члена подсписков нижнего уровня могут отходить объектные списки с произвольным числом членов m. При указанном строении поискового дерева максимальное число объектных списков будет равно числу членов в подсписках нижнего уровня дерева:

$$N = n^k. \qquad (1)$$

Для нахождения какого-нибудь объектного списка по заданной последовательности k признаков необходимо просмотреть k подсписков и выполнить в среднем в каждом подсписке л проверок членов, где $\mu \leq 1$.

Может быть и такой случай поиска объектов, когда на каждом уровне дерева (в каждом подсписке не нижнего уровня) нужно проверять все n членов подсписка. Это может оказаться необходимым в случае объектных структур, в которых в одном подсписке находится несколько членов, удовлетворяющих условиям отбора, а по условиям поиска требуется отыскать все эти члены. В признаковых структурах в каждом подсписке может быть только один член с данным значением признакового кода, и поэтому просмотр каждого подсписка нужно вести только до встречи с членом, имеющим заданное значение проверяемого признака. Отсюда появляется множитель $\mu \leq 1$. Общее количество проверок членов поискового дерева, необходимых для нахождения требуемого объектного списка, будет равно

$$L_{\text{пд}} = \mu n k = \mu n \frac{\ln N}{\ln n}. \qquad (2)$$

Величина $L_{\text{пд}}$ при фиксированном N будет иметь минимум при n = е (2,71). Таким образом, если выбирать параметры поискового дерева, исходя из требования максимальной скорости просмотра этого дерева, то целесообразно брать ближайшие к е = 2,71 целые значения (n = 2 или 3).

Оценим ориентировочно количество ячеек памяти, занимаемых поисковым деревом. В подсписках нижнего уровня общее число членов будет равно общему числу объектных списков N. В подсписках предпоследнего уровня будет в n раз меньше членов, а в подсписках уровня, предшествующего предпоследнему, — в n^2 раз меньше членов и т. д. Таким образом, общее число членов в поисковом дереве будет определяться геометрической прогрессией:

$$\Phi = N + \frac{N}{n} + \frac{N}{n^2} + \ldots + \frac{N}{n^{k-1}} = N \frac{n\left(n^k - 1\right)}{(n-1)n^k}.$$

Заменяя в (1) n на N, получим

$$\Phi = \frac{n(N-1)}{n-1} \approx \frac{nN}{n-1}. \qquad (3)$$

Из формулы (3) видно, что, начиная с n = 2, для которого Φ = 2 (N — 1), величина Φ с ростом n уменьшается и стремится к N при $n = N$. Дальнейшее увеличение n не имеет физического смысла. Таким образом, видно, что с увеличением n, с одной стороны, уменьшается Ф,
т. е. объем памяти, расходуемой под поисковое дерево, а с другой стороны, увеличивается число операций поиска (при n > е). Интересно получить совместную оценку, учитывающую расход памяти и количество операций поиска. В качестве такого критерия иногда принимают [12]
произведение количества ячеек памяти, требуемых для построения поискового дерева, на число операций проверок членов поискового дерева, необходимых для нахождения
нужного объектного списка:

$$K = L\Phi = \mu n \frac{\ln N}{\ln n} \frac{nN}{(n-1)}. \qquad (4)$$

Величина K имеет минимум при $n = 4{,}2 \approx 4$.

Такое число членов в подсписках поискового дерева (при прочих равных условиях) будет наиболее выгодным как с точки зрения расхода памяти, так и с точки зрения скорости поиска.

Число различных объектных списков, в которых может находиться одновременно некоторый объект, определяется количеством списковых слов в ассоциативном узле объекта. Будем считать, что в состав спискового слова входят адрес связи и код признака (дескриптор).

Система классификации объектов, по которой строится поисковое дерево, в общем виде может быть представлена следующим образом. Каждый объект может характеризоваться некоторым количеством (S) признаков (свойств): цвет, вес, длина, координаты и т. д., причем объекты могут обладать не обязательно всеми S признаками. Каждый из признаков может принимать определенное число значений, например: для цвета — белый, красный и т. д. Обозначим число различных значений, которые может принимать i-й признак, через b_i. Различные значения одного и того же признака, как уже упоминалось, несовместны (взаимно исключающие). Различные признаки являются совместными, хотя некоторые объекты могут и не иметь всех признаков. Для подобной системы классификации могут быть построены различные варианты поискового дерева. Рассмотрим два крайних случая:

1. Для каждого значения каждого признака образуется свой отдельный объектный список, в который будут включаться все объекты, обладающие данным значением этого признака. Общее число объектных списков будет равно

$$N = \sum_{i=1}^{S} b_i . \qquad (5)$$

Каждый объект может входить в 5 списков одновременно, и, следовательно, в ассоциативных узлах объектов должно быть по S списковых слов.

2. Для каждой комбинации значений всех S признаков составляется свой отдельный объектный список, такой «список» будет иметь либо один член, либо вообще не будет иметь членов, если считать, что рассматриваемые объекты обязательно различаются между собой значением хотя бы одного признака.

Ясно, что ассоциативные узлы объектов в этом случае будут отсутствовать.

Общее число членов в подсписках нижнего уровня, т. е. число оконечных вершин дерева, будет равно

$$N = \prod_{i=1}^{S} b_i . \qquad (6)$$

Это число может значительно превышать фактическое число объектов, рассматриваемых в данной системе. Ясно, что в этом случае (при использовании взаимноисключающих значений признаков) каждый объект может относиться только к одной оконечной вершине дерева. В первом случае получается минимальное число объектных списков и поиск нужного объектного списка будет осуществляться наиболее просто, зато в каждом объектном списке будет большое количество членов, поиск которых усложняется. Также значительно возрастает расход памяти под списковые слова в ассоциативных узлах. Во втором случае вся ассоциативная структура вырождается фактически в одно поисковое дерево без объектных списков, которое может оказаться весьма разветвленным (при наличии большого числа признаков с большим числом значений). При этом весь процесс поиска объектов сводится к поиску по поисковому дереву нужной оконечной вершины. Весь расход памяти под ассоциативную информацию будет приходиться на поисковое дерево.

Промежуточные случаи между этими двумя крайними случаями могут быть получены путем разделения всех S признаков на несколько групп и представления каждой из таких групп признаков в виде одного так называемого надпризнака. Значением надпризнака будет комбинация значений признаков, входящих в его состав. Ясно, что общее число различных значений данного надпризнака будет равно произведению количеств значений входящих в его состав признаков. Так как различные значения каждого признака являются несовместными, то и различные значения одного надпризнака также будут несовместными.

Допустим, что мы разделили S признаков на l надпризнаков. Число признаков, объединяемых в i-м надпризнаке, обозначим через h_i. Очевидно:

$$\sum_{i=1}^{l} h_i = S . \qquad (7)$$

Число различных значений i-го надпризнака будет равно

$$N_i = \prod_{j=q_i}^{r_i} b_j , \qquad (8)$$

где $$q_i = \sum_{k=1}^{i-1} h_k + 1, \qquad r_i = \sum_{k=1}^{i} h_k .$$

Мы считаем, что признаки, объединяемые в один надпризнак, располагаются подряд. Это можно всегда сделать, так как никаких ограничений на перестановку признаков не налагается. Теперь мы можем построить для каждого надпризнака свое поисковое дерево, причем общее число оконечных вершин в этом дереве будет равно числу различных значений этого надпризнака. Если все S признаков разделены на l надпризнаков, то мы можем иметь l поисковых деревьев. Для каждого значения надпризнака образуется отдельный объектный список, в который будут включаться все объекты, обладающие этим значением данного надпризнака. Общее число объектных списков, относящихся к некоторому i-му надпризнаку, будет определяться формулой (8), а общее число всех объектных списков для всех l

надпризнаков будет равно их сумме:

$$N = \sum_{i=1}^{l} N_i \qquad (9)$$

Каждый объект может входить только в один из объектных списков, относящихся к данному надпризнаку; но естественно, что он может входить в несколько объектных списков, относящихся к разным надпризнакам. Максимальное число объектных списков, в которые может входить один объект, равно числу надпризнаков l, и поэтому в ассоциативных узлах объектов должно быть предусмотрено l списковых слов. В связи с тем что в описанной системе классификации может быть построено не одно, а l поисковых деревьев (по числу надпризнаков), данную ассоциактивную структуру мы будем называть «многодеревной ассоциативной структурой».

В общем случае число уровней в поисковых деревьях, принадлежащих разным надпризнакам, может быть различным; оно будет определяться числом различных значений соответствующих надпризнаков N_i. В ассоциативном узле объекта для каждого надпризнака выделяется одно списковое слово. В этом слове размещается адрес связи, с помощью которого образуется объектный список членов, обладающих данным значением надпризнака; кроме адреса связи в списковое слово входит признаковый код данного объектного списка. Разрядность этого кода выбирается таким образом, чтобы можно было различать N_i объектных списков, соответствующих N_i значениям данного надпризнака. Заметим, что признаковый код в списковых словах, с помощью которых образуются подсписки поискового дерева, должен обеспечивать возможность различать члены только в пределах одного подсписка ($n = 2 \div 5$).

Поиск объектов в описанной многодеревной ассоциативной узловой структуре осуществляется следующим образом. Задается совокупность значений всех S признаков искомого объекта. Если какие-либо из признаков не заданы, то поиск в общем случае может оказаться неоднозначным, т. е. может быть найдено несколько объектов, отвечающих требуемым условиям. Заданные S значений признаков делятся на l надпризнаков в соответствии с принятой для построения поисковых деревьев схемой деления. Затем из l надпризнаков выбирается один надпризнак, который мы будем называть поисковым надпризнаком. С помощью поискового дерева, относящегося к выбранному поисковому надпризнаку, производится поиск объектного списка, соответствующего заданному значению поискового надпризнака. После нахождения требуемого объектного списка производится поиск нужного объекта в этом списке путем последовательного просмотра его членов (ассоциативных узлов) и сравнения значений признаковых кодов в остальных списковых словах каждого узла с заданными значениями остальных надпризнаков (кроме поискового).

Член, у которого все значения признаковых кодов в списковых словах узла равны заданным, выдается в качестве результата. Таким образом, весь процесс поиска объекта расчленяется на три стадии:

а) выбор поискового надпризнака;

б) поиск объектного списка, соответствующего заданному значению выбранного надпризнака;

в) поиск объекта в объектном списке.

В каждом узле объектного списка необходимо проверить максимум l — 1 кодов надпризнаков. Если эти проверки производить последовательно, то в среднем придется проверять меньше чем l — 1 кодов, так как проверки нужно вести только до обнаружения первого несовпавшего кода. Для ускорения поиска все признаковые коды в каждом ассоциативном узле могут проверяться одновременно, т. е. за один такт работы машины.

Из изложенного видно, что при поиске объектов в многодеревной узловой структуре каждый раз в процедуре поиска участвует только одно из поисковых деревьев и один из объектных списков; в ассоциативных узлах объектов при этом используются: один адрес связи, входящий в списковое слово, соответствующее выбранному поисковому надпризнаку, и l— 1 кодов надпризнаков, входящих в состав списковых слов, соответствующих остальным надпризнакам. Таким образом, объем памяти, фактически используемой при каждом конкретном поиске объекта, значительно меньше общего объема памяти, занимаемого данной многодеревной ассоциативной структурой (не считая памяти, занятой дополнительной информацией об объектах — записями объектов). Из сказанного вытекает, что для осуществления поисков объектов по заданной системе признаков можно построить структуру, состоящую только из одного поискового дерева, выбрав в качестве поискового надпризнака некоторую постоянно закрепленную группу h признаков ($h < S$). При этом в ассоциативных узлах объектов достаточно иметь одно списковое слово, включающее в себя один адрес связи и один сложный

признаковый код, объединяющий в себе все остальные S — h признаков. Такие признаковые структуры мы будем называть однодеревными ассоциативными структурами.

Основное достоинство многодеревных структур состоит в возможности вести поиск объекта несколькими путями, выбирая в разных случаях в качестве поискового надпризнака любой из имеющихся l надпризнаков. При этом могут отбираться различные категории объектов путем задания неполных наборов их признаков. В качестве поискового надпризнака каждый раз может назначаться надпризнак, имеющий полный состав признаков, соответствующих отбираемой категории. Важным преимуществом многодеревных узлов структур является возможность выбора для поиска объекта наиболее короткого пути, т. е. такого поискового надпризнака, которому соответствует наиболее короткий объектный список. Это может быть сделано на основе учета вероятностей или частостей появления объектов с различными значениями надпризнаков. Оценка целесообразности построения однодеревной или многодеревной ассоциативной структуры должна производиться, исходя из характера поиска объектов (поиск одиночных объектов с полным набором признаков или поиск категорий объектов с неполными наборами признаков), а также исходя из распределения вероятностей появления объектов с различными значениями признаков (надпризнаков). Для выработки подхода к разрешению этого вопроса рассмотрим некоторые зависимости, определяющие процесс поиска в однодеревной ассоциативной структуре.

Допустим, что в качестве поискового надпризнака мы выбираем h первых признаков (из S). Остальные S — h признаков объединим в один надпризнак, который будем называть объектным. Так как никаких ограничений на перестановку признаков не налагается, то под h признаками можно понимать любые (фиксированные) h из S признаков.

Согласно (8) число объектных списков, соответствующих данному поисковому надпризнаку, будет равно

$$N = \sum_{i=1}^{h} b_i \ . \qquad (10)$$

Число операций проверок членов поискового дерева, необходимых для нахождения объектного списка, соответствующего заданному значению поискового надпризнака, будет согласно (2) равно

$$L_{\text{ПД}} = \mu n \frac{\ln N}{\ln n}. \qquad (2)$$

Эта величина имеет минимум при n = e, поэтому в подсписках поискового дерева целесообразно иметь по 2—3 члена. Коэффициент μ учитывает, что просматриваются не все члены подсписков поискового дерева, так как просмотр идет только до обнаружения требуемого значения проверяемого признака.

Будем считать, что число проверок членов выбранного k-го объектного списка, необходимых для нахождения одного заданного члена (объекта), пропорционально длине этого объектного списка (т. е. числу членов в нем):

$$L_{\text{ОС}_k} = \eta Q_{\text{ОС}_k} . \qquad (11)$$

Длину каждого объектного списка естественно считать пропорциональной вероятности появления объектов с соответствующим значением поискового надпризнака. (Очевидно, что вероятность появления объекта с данным значением объектного надпризнака на процент проверяемых членов Данного объектного списка влияния оказывать не будет).

Обозначим вероятность появления объектов с некоторым 6-м значением поискового надпризнака через p_k. Так как каждый из рассматриваемых объектов (общее число которых обозначим через P) должен находиться в одном и только в одном из объектных списков, относящихся к данному надпризнаку, то количества объектов, находящихся в объектных списках, соответствующих различным значениям поискового надпризнака, будут пропорциональны вероятностям появления этих значений надпризнаков:

$$Q_{\text{ОС}_k} = P_{P_k} . \qquad (12)$$

Считаем, что все S признаков взаимно независим» При этом вероятность появления определенного значения надпризнака будет равна произведению вероятностей появления значений признаков, образующих данный надпризнак:

$$p_k = \prod_{i=1}^{h} p_{i,j_i}, \qquad (13)$$

где p_{i,j_i} — вероятность появления j-го значения i-го признака.

Считаем, что для всех значений S признаков вероятности их появления заданы в виде таблицы, имеющей S колонок и в каждой i-й колонке по b_i строк (значений).

Таким образом, индекс j для каждого i может меняться от 1 до b_i. В формуле (13) участвует некоторый фиксированный набор значений j, соответствующий 6-му значению поискового надпризнака. Заметим, что если некоторые признаки не являются независимыми, то они могут быть объединены в один признак (с большим числом значений — не надпризнак), который будет независим по отношению к остальным признакам, и таким образом случай зависимых признаков может быть сведен к рассматриваемому случаю независимых признаков. Частость обращений к данному объектному списку для поиска в нем объектов будет также пропорциональна вероятности появления данного значения поискового надпризнака p_k. Таким образом, количество проверок, приходящееся на долю данного объектного списка, будет пропорционально длине этого списка и частости обращения к нему:

$$L_{\mathrm{OC}_k} = \eta P \prod_{i=1}^{h} p_{i,j_i} M \prod_{i=1}^{h} p_{i,j_i}, \qquad (14)$$

где индекс j имеет некоторый фиксированный набор значений, соответствующий k-му объектному списку, а коэффициент M характеризует собой общее число реализованных поисков объектов. При этом $M\prod_{i=1}^{h} p_{i,j_i}$ представляет собой количество поисков, приходящихся на k-й объектный список. Из (14) получим

$$L_{\mathrm{OC}_k} = \eta MP \sum_{i=1}^{h} p_{i,j_i}^2. \qquad (15)$$

Общее количество проверок членов объектных списков, выполненных при всех M поисках объектов, будет равно сумме количеств проверок, приходящихся на все N объектных списков:

$$L_{\mathrm{oc}} = \sum_{k=1}^{N} L_{\mathrm{OC}_k} \qquad (16)$$

Эта сумма получится путем варьирования индекса j, который для каждого k принимает фиксированный набор значений. Сумма (16) получится следующим образом:

$$L_{\mathrm{oc}} = \eta MP \sum_{j_h=1}^{b_h} \ldots \sum_{j_2=1}^{b_2} \sum_{j_1=1}^{b_1} \prod_{i=1}^{h} p_{i,j_i}^2. \qquad (17)$$

Выражение (17) можно записать более компактно, если заменить многократную сумму произведений произведением сумм:

$$L_{\mathrm{oc}} = \eta MP \prod_{i=1}^{h} \sum_{j_i=1}^{b_i} p_{i,j_i}^2. \qquad (18)$$

Из (18) путем деления на M можно получить среднее количество проверок, приходящихся на один поиск объекта в объектном списке:

$$L_{\mathrm{oc}}^{(\mathrm{cp})} = \eta P \prod_{i=1}^{h} \sum_{j_i=1}^{b_i} p_{i,j_i}^2. \qquad (19)$$

Объединяя формулы (2) и (19), получим общее среднее количество операций проверок в поисковом дереве и объектном списке, необходимых для нахождения одного объекта:

$$L^{(\mathrm{cp})} = L_{\mathrm{пд}} + L_{\mathrm{oc}}^{(\mathrm{cp})} = \mu n \frac{\ln N}{\ln n} + \eta P \prod_{i=1}^{h} \sum_{j_i=1}^{b_i} p_{i,j_i}^2. \qquad (20)$$

Заменив N на $\prod_{i=1}^{h} b_i$ получим

$$L^{(\text{ср})} = \frac{\mu n}{\ln n} \ln \prod_{i=1}^{h} b_i + \eta P \prod_{i=1}^{h} \sum_{j_i=1}^{b_i} p_{i,j_i}^2 . \qquad (21)$$

Здесь μ и η — постоянные коэффициенты; n — число члене в подсписках поискового дерева (которое является постоянным); b_i — число различных значений i-го признака; P — общее число рассматриваемых объектов; $P_{i\ j_j}$. — вероятность появления j-го значения i-го признака.

Используя формулу (21), можно исследовать два вопроса

1) Для заданной таблицы распределения вероятностей p_{i,j_i} появления объектов с различными значениями признаков выбрать из S признаков такие h признаков (и определите само число $h \le S$), чтобы среднее количество операция проверок, необходимых для поиска объектов, было минимальным. Эта задача построения оптимальной с точки зрения времени поиска однодеревной ассоциативной структуры.

Выбор признаков должен производиться с учетом числа различных значений каждого признака и распределения вероятностей появления отдельных значений. По-видимому, обобщенной вероятностной характеристикой признака может служить энтропия различных значений этого признака.

2) Для заданной таблицы распределения вероятностей p_{i,j_i} появления объектов с различными значениями S признаков произвести разбиение S признаков на l поисковых надпризнаков таким образом, чтобы при поиске объектов каждый раз при помощи наиболее выгодного поискового надпризнака (т. е. надпризнака, который имеет более короткий объектный список для заданного сочетания признаков) общее количество операций проверок было минимальным. Это задача построения оптимальной с точки зрения времени поиска многодеревной ассоциативной структуры. Таким образом, имея таблицу распределения вероятностей появления объектов с различными признаками, можно оценить выгоду с точки зрения времени поиска от перехода к многодеревной узловой структуре и сравнить эту выгоду с дополнительным расходом памяти, связанным с этим переходом. Следует заметить, что приведенные формулы получены исходя из предположения о том, что различные признаки являются взаимно независимыми. Как уже говорилось, в случае, если некоторые признаки будут зависимыми, то их можно объединить в один признак (не надпризнак), имеющий большее число значений. Этот признак будет независимым по отношению к остальным признакам, и, таким образом, этот случай будет сведен к рассмотренному случаю независимых признаков.

Исследование формулы (21) аналитическим путем не представляется возможным, и поэтому упомянутые задачи можно решать численным методом (в частности, методом перебора с помощью электронной цифровой машины) применительно к конкретным значениям P, S, b_i и значениям таблицы p_{i,j_i}. Рассмотрим частный случай, когда все S признаков имеют одинаковое число значений ($b_1 = b_2 = \ldots = b_s = b$) и вероятности появления объектов с разными значениями признаков также одинаковы ($p_{i,j_i} = 1/b$). Тогда формула (21) примет вид

$$L = \mu n \log b^h + \eta P \frac{1}{b^h} . \qquad (22)$$

Определим, при каком h величина L имеет минимум, считая n = e:

$$\frac{dL}{dh} = \mu e \ln b - \eta P \frac{\ln b}{b^h} = 0. \qquad (23)$$

Отсюда

$$h = \log \frac{\eta P}{\mu e} = \frac{1}{\ln b} \ln \frac{\eta P}{\mu e}.$$

Например (полагая η= μ =0.5), при $P = 1024$ и $b = 2$ $h \approx 8$, а при $P = 10^6$ и $b = 10$ $h \approx 5$. При этом среднее число проверок в объектном списке будет равно ηe, т. е. оно будет таким же, как число проверок в каждом из подсписков поискового дерева. Отношение числа проверок, приходящихся на поиск в поисковом дереве, к числу проверок в объектном списке, будет определяться числом уровней

поискового дерева:

$$K = \ln \frac{\eta P}{\mu e}. \qquad (24)$$

Полагая η = μ = 0,5, получим $K = \ln P - 1$. Например, при $P = 10^3$ $K \approx 6$, при $P = 10^6$ $K = 13$.

Таким образом, если исходить из условия минимизации общего числа проверок, то наиболее выгодной оказывается структура, состоящая как бы из одного поискового дерева.

В этой структуре объектные списки в среднем должны иметь то же число членов, что и подсписки поискового дерева, но в отличие от них объектные списки будут иметь переменную длину. Основная часть проверок при этом приходится на долю поискового дерева, а на долю объектных списков остается незначительная часть.

Оценим для данного случая требуемый объем памяти под поисковое дерево и объектные списки. Если максимально допустимое число объектов P, а число всех признаков 5, то расход памяти под списковые слова в объективных списках будет равен

$$Q_{ac} \approx P(\log_2 P + \log_2 b^{s-h}), \qquad (25)$$

где первое слагаемое представляет адреса связи, а второе— признаковые коды.

Расход памяти под поисковое дерево будет равен

$$Q_{ac} \approx \frac{nN}{n-1}\lambda,$$

где $N = b^h$ — общее число объектных списков, а λ —число разрядов в списковом слове дерева*:

$$\lambda \approx 2 + \log_2 \frac{nN}{n-1} + \log_2 n.$$

Отсюда

$$Q_{ПД} \approx \frac{nb^n}{n-1}\left(2 + \log_2 \frac{nN}{n-1} + \log_2 n\right). \qquad (26)$$

Оценим соотношение в расходе памяти под списковые слова в объектных списках и поисковом дереве для некоторых конкретных случаев.

При $P = 10^3$, $b = 2$, $s = 16$, $h = 8$, $n = 3$ получим $q = Q_{ac}/Q_{пд} \approx 4$; при $p = 10^6$, $b = 10$, $s = 16$, $h = 4$, $n = 3$ получим $q \approx 200$. Вообще, расход памяти под объектные списки в основном зависит от числа объектов P, а расход памяти под поисковое дерево — от числа объектных списков $N = b^h$.

Так как число объектов P обычно значительно больше числа объектных списков N, то основной расход памяти приходится на объектные списки. Рассмотрим теперь частный случай построения ассоциативной структуры. Как и в предыдущем случае, будем считать, что все признаки имеют одинаковое число значений ($b_1 = b_2 = \ldots = b_s = b$) и равные вероятности появления объектов с разными значениями признаков ($p_{i,j_i} = 1/b$ для $i = 1, 2, \ldots, s$). S признаков делятся на l надпризнаков по h признаков в надпризнаке ($h = S/l$). В ассоциативном узле каждого объекта находится по l списковых слов, соответствующих отдельным надпризнакам. Будем считать, что все l признаковых кодов в узле проверяются одновременно.

Из формулы (21) получим (заменяя $p_{i,j_i} = 1/b$ и $h = S/l$)

$$L^{(ср)} = \mu n \log_n b^{\frac{s}{l}} + \frac{\eta P}{b^{\frac{s}{l}}}. \qquad (27)$$

Дифференцируя по l и приравнивая нулю, получим значение l, при котором число проверок будет минимальным:

---

*Считаем, что в списковом слове имеется два служебных разряда, адрес связи (относительный, он имеет примерно $\log_2 \frac{nN}{n-1}$ разрядов; считается, что дерево занимает сплошную зону ячеек) и признаковый код, служащий для различения членов в пределах одного подсписка (он имеет примерно $\log_2 n$ разрядов).

$$\frac{dL^{(\text{ср})}}{dl} = -\frac{\mu n S}{l^2}\log_n b + \frac{\eta P S}{l^2}\frac{\ln b}{b^{\frac{s}{l}}} = 0.$$

Отсюда (полагая $\eta \approx \mu$)

$$l = \frac{\ln b^s}{\ln P + \ln\ln n - \ln n}.$$

Если считать n = e, то получим

$$l = \frac{\ln b^s}{\ln P - 1}. \qquad (28)$$

Из формулы (28) видно, что оптимальное число надпризнаков, на которое должны быть разделены S признаков, увеличивается с увеличением b т. е. числа всех возможных комбинаций признаков, и уменьшается с увеличением числа рассматриваемых объектов P.

Например, для $P = 10^3$, $b = 2$, $S = 16$ получим $l \approx 2$;

для $P = 10^6$, $b = 10$, $S = 16$ получим $l \approx 3$.

Как уже говорилось выше, с увеличением числа надпризнаков растет число поисковых деревьев и ассоциативных слов в узлах, а следовательно, растет объем памяти, занимаемой ассоциативной информацией, причем основной расход памяти идет под списковые слова в узлах.

Рассмотренные частные случаи построения однодеревной ассоциативной структуры помимо непосредственного практического применения могут служить также для общей ориентировки при построении структур с переменным числом значений признаков и различными вероятностями их появления.

Для более точного анализа и выбора приемлемых параметров такого рода структур может быть непосредственно использована формула (21), которая должна исследоваться численным методом. Анализ показывает, что основная часть операций проверок, выполняемых при поисках объектов, приходится на просмотр поискового дерева, а основной расход памяти под ассоциативную информацию — на долю списковых слов в узлах объектов. Учитывая это, можно построить так называемую многодеревную ассоциативную структуру, которая будет наиболее эффективной как с точки зрения быстроты поиска объектов, так и с точки зрения сокращения расхода памяти. При этом все S заданных признаков объектов разбиваются на l надпризнаков и для каждого из надпризнаков строится свое поисковое дерево. Таким образом, всего в структуре будет l деревьев. Но в узлах объектов отводится место всего под одно списковое слово, которое будет включать в себя адрес связи и один объектный надпризнак. Следовательно, каждый объект может находиться только в одном из объектных списков, принадлежащих к одному из указанных поисковых деревьев. Выбор поискового дерева, т. е. поискового надпризнака, по которому должен производиться поиск объекта, осуществляется с учетом вероятностей появления объектов с различными значениями надпризнаков. Исходя из физического содержания задачи и имеющихся сведений о частостях появления объектов с разными значениями надпризнаков, при программировании заранее составляется таблица вероятностей появления объектов с различными значениями признаков (p_{i,j_i}). При поступлении очередного объекта по значениям его S признаков при помощи таблицы определяются вероятности появления каждого из l надпризнаков и выбирается в качестве поискового надпризнака тот, который имеет минимальное значение вероятности. По этому надпризнаку будет производиться поиск. При указанном способе в каждом объектном списке будут находиться не все объекты, обладающие данным значением надпризнака, а только те, у которых это значение имеет наименьшую вероятность появления по сравнению со значениями его других надпризнаков. Этот способ обеспечивает значительное сокращение расхода памяти под ассоциативные узлы за счет того, что в каждом узле будет не l, а одно списковое слово. Этот способ приводит и к ускорению поиска за счет того, что в каждом объектном списке в среднем будет находиться в l раз меньше объектов и объекты будут распределяться по объектным спискам более или менее равномерно. При этом поиск будет идти каждый раз оптимальным образом, т. е. по тому объектному списку, который имеет наименьшую вероятность появления объектов с данным значением надпризнака. Дополнительные операции, связанные с определением по таблице вероятностей нужного надпризнака, могут быть оформлены в виде стандартной подпрограммы либо реализованы схемно в виде одной специальной команды.

17. Методика ассоциативного программирования

Рассмотренные нами алгоритмические языки АЛГЭМ-1 и 2 позволяют описывать различные алгоритмы переработки информации на ЭВМ в тех случаях, когда объем и вид данных (количество и структура записей) заранее известны и могут быть точно описаны. Однако существуют задачи, в которых объем и состав информации заранее неизвестны. Более того, в ходе обработки структура информации меняется, что требует соответствующего перераспределения памяти. К числу таких задач относятся накопление и поиск научной и библиографической информации, машинный перевод, автоматическое программирование, опознавание образов, моделирование обучения и т. д.

Для эффективного описания подобных алгоритмических процессов необходим специальный алгоритмический язык. Работы в этом направлении ведутся уже в течение многих лет и привели к созданию ряда языков, предназначенных для так называемой неарифметической обработки данных (LISP, IPL-V, COMIT и др.).

В настоящее время определилась тенденция к унификации языков программирования и построению подобного языка на основе АЛГОЛа.

В настоящем параграфе рассмотрим некоторое расширение языка АЛГЭМ-1, описанного подробно в гл. 3, параграфе 10, предназначенное для более эффективного описания различных алгоритмов с гибкой системой организации данных. Это расширение сводится к введению двух адресных операций, представляемых двумя видами скобок (скобки адреса и скобки содержимого), описателя список, который служит для описания списковых величин.

Используя это дополнение и все остальные средства АЛГЭМ-1, а также метод процедур, можно записывать алгоритмы процессов переработки информации, включающей в себя как обычные (несписковые), так и списковые величины.

Основными средствами ассоциативного программирования являются:

1) использование адресов связи для построения списков различных видов, объединяющих объекты с общими признаками;

2) использование списковых структур, представляющих собой многоуровневые списки, т. е. списки с ответвляющимися от них подсписками, для предоставления иерархических систем организации данных;

3) использование так называемых продвигаемых списков (стэков или магазинов) для временного запоминания данных в определенном порядке и восстановления их в обратном порядке;

4) организация свободной памяти в виде цепного списка ячеек, обеспечивающая гибкость и полноту использования всего объема памяти и исключающая необходимость в ее детальном предварительном распределении.

Существует много конкретных способов и форм машинной реализации ассоциативного программирования.

Указанные идеи усиленно разрабатываются многими учеными различных стран. К числу первых и наиболее значительных работ в этой области относятся работы А. Ньювелла, Симона, Шоу, Тонге, Гелернтера. Заметим, что универсальный язык программирования PL-1, построенный на базе ФОРТРАНа, включает в себя средства для обработки списковой информации. Для обработки списков в известной мере может быть использована методика адресного программирования Е. Л. Ющенко. Кроме того, существует много конкретных работ, представляющих описания специальных языков программирования, связанных с определенной машинной реализацией (т. е. расположением данных в ячейках машинной памяти).

Известны также работы по созданию специализированных алгоритмических языков, пригодных для обработки списковой информации.

Приводимое ниже описание методики ассоциативного программирования следует рассматривать как дополнение АЛГЭМ-1 с включением в него раздела процедур АЛГОЛа.

Основной особенностью этой методики является возможность представления процессов обработки ассоциативной или списковой информации, имеющей переменный состав и произвольное размещение в памяти машины.

Язык АЛГЭМ удобен для описания процессов обработки информации, имеющей жесткую, заранее установленную структуру. По сравнению с АЛГОЛом в АЛГЭМе имеются дополнительные средства для описания составных величин, указания отдельных разрядов или символов, образующих величины, и, кроме того, введен тип строчных величин.

Адресные соотношения. В АЛГОЛе различаются следующие классы величин: простые (или, как их

лучше называть, одиночные) переменные, массивы, метки, переключатели и процедуры, т. е. объекты, которые могут иметь свои идентификаторы. Правда, метки могут быть представлены целыми числами без знака, и, таким образом, числа частично тоже попадают под определение величины.

Мы будем пользоваться более широким определением понятия «величина», включив в него кроме перечисленных объектов также числа, строки и списки. Можно дать следующее синтаксическое определение понятия «величина»:

⟨величина⟩ : : = ⟨число⟩ | ⟨метка⟩ | ⟨строка⟩ | ⟨простая переменная⟩ | ⟨переменная с индексами⟩ | ⟨массив⟩| ⟨список⟩ | ⟨переключатель⟩ | ⟨процедура⟩

Таким образом, величина — это некоторая единица информации (не количества информации), представленная либо своим идентификатором (наименованием), либо непосредственно своим значением (последнее относится к числам, строкам, меткам).

В машинах все величины представляются цифровыми кодами (как правило, двоичными кодами). Этими кодами представляются как значения, так и наименования величин. Коды хранятся в ячейках памяти и называются содержимыми. Каждой ячейке памяти ставится в однозначное соответствие некоторый постоянный код, который называется ее адресом.

Наличие двух видов машинных кодов (адресов и содержимых) является принципиальной особенностью машинной переработки информации, которую необходимо учитывать при построении алгоритмов решения информационно-логических задач.

Обычно понятия адреса и содержимого привязаны жестко к конкретным машинам. Для того чтобы при программировании на алгоритмическом языке, не связанном с конкретными машинами, можно было учитывать указанную особенность машинного решения задач, необходимо ввести в язык общие понятия адреса и содержимого, не привязывая их к конкретной машине.

Дадим следующие формальные синтаксические определения понятий адреса и содержимого:

⟨адрес⟩:: = ⟨арифметическое выражение⟩ | ⌈ ⟨величина⟩⌉

⟨содержимое⟩ : : = ⌊ ⟨адрес⟩ ⌋

Верхние уголковые скобки называются адресными; они показывают, что вместо величины, заключенной в эти скобки, должен быть взят ее адрес, т. е. некоторое целое число.

Нижние уголковые скобки называются скобками содержимого: они показывают, что вместо адреса, заключенного в эти скобки, должна быть взята величина, находящаяся в ячейке с данным адресом. В тех случаях, когда указанные символы (скобки адреса и скобки содержимого) используются в программах, написанных на алгоритмическом языке, без дополнительных описаний так называемых адресных соотношений (см. ниже), они имеют смысл буквально адресов и содержимых ячеек памяти той конкретной машины, на которой предполагается решение данной задачи. При этом предполагается, что каждая из величин, представляемых с использованием адресных скобок или скобок содержимого, размещается в отдельной ячейке памяти и располагается там стандартным для данной машины способом (с фиксированной запятой, с плавающей запятой или в виде набора алфавитно-цифровых символов). В тех же случаях, когда расположение величин в ячейках памяти не является стандартным (т. е. одна величина занимает несколько ячеек или в одной ячейке находится несколько величин), а также в тех случаях, когда в качестве содержимых, имеющих один адрес, желательно иметь сложные величины (массивы, списки, составные переменные, процедуры, переключатели), необходимо в состав описаний данного блока программы включать дополнительные описания адресных соотношений.

Описание адресного соотношения строится следующим образом.

Адресное соотношение состоит из двух частей, правой и левой, соединенных знаком равенства. В левой части в адресных скобках указывается величина (или несколько величин), являющаяся содержимым по данному адресу (если указывается несколько величин, имеющих общий адрес, то они разделяются запятыми). В правой части указывается величина, являющаяся адресом. В качестве адресов могут указываться числа, переменные или арифметические выражения. Например, адресом может быть сумма двух переменных, из которых одна представляет собой базисный адрес, а вторая — относительный адрес. Каждое описание адресного соотношения, как и любое другое описание, заканчивается точкой с запятой. Принимается правило, что перечисление нескольких величин через запятые означает их расположение в одной ячейке или в нескольких подряд идущих ячейках памяти; величины, перечисленные через запятые, представляют собой последовательный список, положение которого полностью определяется адресом его первого члена. Подобным образом могут указываться адресные соотношения для величин, являющихся массивами или элементами массивов, для составных переменных, а также для величин, входящих в состав списков. Рассмотрим этот вопрос подробнее.

Если величиной, адрес которой требуется обозначить, является массив переменных, то в адресных скобках указывается только идентификатор этого массива. Тогда содержимым, которое будет извлекаться по данному адресу, будет весь массив полностью, причем размеры этого массива будут определяться его основным описанием, задаваемым обычным способом.

Если же содержимым, которое должно извлекаться по заданному адресу, должен быть не весь массив, а его один элемент, то при описании адресного соотношения за идентификатором массива в адресных скобках нужно указать индексные (квадратные) скобки (пустые).

Аналогичным образом при задании адресных соотношений для составных переменных, если требуется, чтобы содержимым по Данному адресу считалась вся составная переменная, в адресных скобках должен ставиться только идентификатор этой составной переменной.

Если нужно, чтобы содержимым данного адреса являлась некоторая компонента составной переменной, то в адресных скобках указывается идентификатор этой компоненты с уточнением или без уточнения (если отсутствие уточнения не приводит к неопределенности).

Если в адресных скобках указан идентификатор списка, то содержимым данного адреса будет не весь список, а только его заголовок (см. ниже списковые описания). Если же нужно иметь в качестве содержимого по данному адресу какую-либо компоненту, входящую в состав заголовка списка, то это делается при помощи адресного соотношения, записываемого таким же образом, как для компоненты составной величины.

Адресные соотношения для величин, входящих в состав списковых членов, даются по правилам, аналогичным тем, которые указаны для компонент составных переменных. Если требуется устранить неопределенность, то идентификаторы этих величин указываются в адресных скобках совместно с уточнениями. В качестве уточнений могут выступать не только идентификаторы составных переменных, в которые входят данные компоненты, но и идентификаторы списков, в которые входят списковые члены, имеющие в своем составе данные компоненты.

Заметим, что в качестве адресов в правых частях адресных соотношений могут фигурировать не только простые переменные, но и элементы массивов, компоненты составных переменных, а также компоненты списковых членов. Во всех случаях, если не возникает неопределенности, эти величины указываются просто своими идентификаторами (например, адреса, являющиеся элементами массивов, указываются без индексных скобок, так как ясно, что адресом может быть только один элемент массива). В тех случаях, когда возникает неопределенность, эти величины указываются совместно с уточнениями, составленными по правилам АЛГЭМа.

Иногда адресом некоторой величины могут быть несколько переменных (или арифметических выражений); в этом случае все они перечисляются в адресном соотношении через знак равенства. Например, если адресом величины A будут величины i, j, k, то адресное соотношение имеет вид

$$\lceil A \rceil = i = j = k$$

При наличии описания адресного соотношения содержимым по данному адресу всегда является величина, указанная в адресных скобках в левой части описания.

В тех случаях, когда выдерживается стандартное расположение величин, согласно которому каждая величина помещается в одну ячейку, описания адресных соотношений приводить не нужно. Не нужно приводить описаний адресных соотношений и для величин, не использующихся при программировании совместно с адресными скобками и скобками содержимого, даже если они не имеют стандартного расположения.

Вообще, если в программе используются где-либо скобки содержимого без соответствующего описания адресного соотношения, то всегда содержимым в этом случае будет содержимое одной ячейки памяти той конкретной машины, на которой будет идти решение задачи, с адресом, равным значению величины, стоящей в скобках содержимого.

Синтаксическое определение описания адресного соотношения имеет вид:

⟨описание адресного соотношения⟩ : : = ⌈⟨величина⟩⌉ = ⟨арифметическое выражение⟩ | ⟨описание адресного соотношения⟩ = ⟨арифметическое выражение⟩

Примеры:

1) Пусть величины p, q, r, s должны помещаться в одну ячейку с адресом a. Описание адресного соотношения для этого случая будет иметь вид

$$\lceil p, q, r, s \rceil = a;$$

2) Пусть составная величина НАРЯД занимает несколько подряд идущих ячеек и адрес первой ячейки равен b. Адресное соотношение должно быть записано в виде

$$\lceil \text{НАРЯД} \rceil = b;$$

3) Пусть, например, адресами элементов массива «заголовок списка» являются соответственно элементы массива «дескриптор». Тогда адресное соотношение будет выглядеть следующим образом:

$$\lceil \text{заголовок списка} \rceil = \text{дескриптор};$$

4) Пусть имеется три величины b, m, n, которые образуют цепной список. Каждая из этих величин расположена в первой половине одной ячейки, во второй половине которой находится адрес связи, т. е. число, указывающее адрес ячейки, в которой расположена следующая величина данного списка. Во второй половине ячейки, в которой находится последний член данного цепного списка, помещается некоторый условный постоянный код, обозначаемый идентификатором КС и указывающий конец списка. Пусть адрес ячейки, в которой находится первый член этого списка, равен a, тогда адресное соотношение для указанного случая будет иметь вид

$$\lceil b, \lceil m, \lceil n, \text{КС} \rceil \rceil \rceil = a;$$

Понятия адреса и содержимого в общем случае можно интерпретировать как понятия наименования и значения некоторого объекта, не связывая их с машинной реализацией. Эти понятия носят относительный характер: одна и та же группа символов или один и тот же код может представлять собой наименование одной категории объектов и в то же время может быть значением одного объекта, являющегося членом категории более высокого уровня классификации.

В машинной интерпретации код адреса может рассматриваться как наименование той единицы информации, значение которой представлено кодом содержимого этой ячейки. Относительный характер понятий наименования категории объектов и значения объекта при машинной реализации проявляется в том, что содержимым некоторой ячейки может быть адрес другой ячейки и т. д.

Выделение компонент в содержимых. Адреса и содержимые могут участвовать в качестве операндов во всех операциях, предусмотренных в алгоритмическом языке, а также фигурировать в левых частях в операторах присваивания.

Адресные скобки и скобки содержимого могут применяться многократно, и, таким образом, будут получаться адреса и содержимые более высоких рангов.

К адресам и содержимым полностью применимы описанные в АЛГЭМе способы выделения компонент составных переменных и разрядов. Если для содержимых не дано описания вида, то подразумевается, что указание разрядов относится к двоичным разрядам ячейки машинной памяти той машины, на которой предполагается решение задач.

При этом считается, что все двоичные разряды ячейки пронумерованы подряд слева направо от 0 до некоторого *n* и указание разрядов определяет эти двоичные разряды ячейки.

Выделенные разряды переменной или выделенная компонента составной может использоваться в качестве адреса, т. е. заключаться в скобки содержимого. Заключение выделенной компоненты составной величины (или выделенных разрядов) в скобки содержимого означает, что эта компонента должна использоваться в качестве адреса независимо от ее исходного (т. е. до выделения) положения в составной величине. Таким образом, выделенная компонента как бы автоматически сдвигается в сторону младших разрядов адреса. Аналогичным образом используются выделяемые компоненты и при выполнении с ними арифметических операций, т. е. выделенная компонента всегда рассматривается как отдельная величина.

Имеют место следующие естественные соотношения между выделенными компонентами в правых и левых частях операторов присваивания. Если правой частью оператора присваивания является выделенная компонента, а для величины, стоящей в левой части, не указана компонента или разряды, то значение выделенной компоненты правой части присваивается всей величине, стоящей в левой части. Если же для величины, стоящей в левой части оператора присваивания, указаны разряды или компонента, то выделенная компонента правой части устанавливается так, чтобы совпали младшие разряды выделенных компонент в левой и правой частях оператора присваивания (это справедливо для целых переменных; в случае дробных или смешанных переменных выравнивание должно производиться по положению десятичной запятой).

В этом случае при выполнении оператора присваивания все остальные части переменной, указанной

в левой части оператора присваивания, кроме выделенной компоненты (или выделенных разрядов), остаются без изменения.

Если выделенная компонента правой части имеет большее число разрядов, чем выделенная компонента левой части, то не совпавшие разряды (лишние) компоненты правой части пропадают. Например, a .⌊ A ⌋: = C; показывает, что значение C присваивается величине a, представляющей собой компоненту составной величины, имеющей адрес, равный значению величины A.

В данном случае обозначение компоненты содержимого в виде a.⌊ A ⌋ имеет смысл идентификатора величины и присваивание идет так, как если бы в левой части стоял просто идентификатор величины a. Такой же смысл идентификатора имеет задание части ячейки при помощи указания разрядов.

Запись вида (i : j)⌊ A ⌋ : = C; определяет присваивание значения величины C группе разрядов ячейки, адрес которой равен значению величины A. Младший разряд принимающей группы имеет номер i, старший разряд — номер j. Если у величины, являющейся содержимым по адресу A, имеются описания адресного соотношения и вида, то левой частью оператора присваивания будет группа символов этой величины с номерами от i до j.

Если нужно указать, что выделенная компонента содержимого сама обозначает адрес ячейки, в которую посылается значение некоторой величины, то записывается следующее выражение:

$$\lfloor a.\lfloor A \rfloor \rfloor := C; \quad \lfloor (i: j)\ \lfloor A \rfloor \rfloor : = C;$$

Эти записи обозначают, что значение некоторой величины C присваивается содержимому ячейки, имеющей адрес, равный либо значению a.⌊ A ⌋, либо значению кода, находящегося в разрядах с i по j в ячейке с адресом, равным значению A.

Так как содержимым по некоторому адресу (в соответствии с описанием адресного соотношения) может быть составная величина, а содержимые могут фигурировать как в правых, так и в левых частях операторов присваивания, то тем самым допускается операция присваивания над составными величинами одинакового типа и структуры в правой и левой частях оператора присваивания.

Описания списков. Как известно, все величины (за исключением некоторых стандартных), используемые в алгоритмических программах, должны быть описаны в начале тех блоков, в которых они используются (или во внешних блоках). Это требование относится и к списковым величинам (спискам и списковым структурам), хотя назначание описаний списковых величин существенно отличается от назначения описаний несписковых величин. Описания обычных несписковых переменных (элементарных и составных) используются транслятором для: а) распределения памяти под эти величины; б) определения характера операций над величинами (с округлением или без округления и др.).

Списковые величины, как уже говорилось, не требуют для себя распределения памяти в обычном понимании этого термина. Размеры и структура списков могут меняться в процессе работы в широких пределах; они образуются автоматически в соответствии с фактическим составом обрабатываемой информации и располагаются на имеющихся свободных местах в памяти машины.

Однако, несмотря на указанную гибкость в использовании списковых величин, при программировании задач, в которых участвуют списковые величины, требуется соблюдение двух основных условий:

а) должна быть заранее определена структура (формат) списковых членов и расположение их относительно границ ячеек памяти машины;

б) должна быть произведена предварительная организация свободной области памяти, выделяемой под списковые величины. Не списковая область распределяется обычным образом, а списковая область организуется в соответствии с типом используемых списков (либо в виде общего цепного списка свободных ячеек, либо в виде цепного списка свободных гнезд ячеек фиксированного размера, либо в виде последовательного списка свободных зон на МЛ или в ОЗУ и т. д.).

Указанные ограничения вызваны тем обстоятельством, что процедуры обработки списковой информации привязаны к форматам списковых членов и способам организации свободной памяти машины.

Для описания списковых величин вводится описатель **список,** который вместе с символом **уровень** образует пару описательных скобок, подобную паре **составной... уровень.** Описание списка состоит из двух частей: из описания заголовка списка (фиксатора списка) и из описания структуры спискового члена. Эти две части описания разделяются точкой с запятой. Описание списка начинается символом **список,** после которого идет описание заголовка списка, а после него дается структура спискового

члена, которая заканчивается символом **уровень.** Как заголовок списка, так и списковый член в общем случае представляют собой составные переменные. В частном случае заголовок списка может вообще отсутствовать.

Так как в состав спискового члена может входить в качестве элемента другой список или даже несколько списков, то эту возможность необходимо предусмотреть соответствующим построением описания списка, а именно построением структуры спискового члена.

Структура спискового члена аналогична структуре составной переменной, за исключением того, что наряду с элементами описаний, принятыми в АЛГЭМе, в структуру спискового члена могут входить и описания списков.

Используя определение составной величины, можно построить следующее определение описания списка:

⟨идентификатор списка⟩ : : = ⟨идентификатор⟩

⟨описание заголовка списка⟩ : : = ⟨пусто⟩ | ⟨идентификатор списка⟩ | ⟨описание составной величины⟩

⟨элемент структуры спискового члена⟩ : : = ⟨элемент описания⟩ | ⟨описание списка⟩

⟨структура спискового члена⟩ : : = ⟨элемент структуры спискового члена⟩ | ⟨структура спискового члена⟩; ⟨элемент структуры спискового члена⟩

⟨описание списка⟩ : : = **список** ⟨описание заголовка списка⟩; ⟨структура спискового члена⟩ **уровень**

Следует сделать несколько пояснений относительно описания заголовка списка. Здесь воможны три варианта. Во-первых, описание заголовка списка может вообще отсутствовать. В этом случае сразу же за символом **список** будет стоять точка с запятой, за которой будет следовать описание структуры спискового члена. Во-вторых, описание заголовка списка может быть представлено просто идентификатором списка, за которым стоит точка с запятой. В этом случае идентификатор списка играет роль некоторого примечания, поясняющего название или назначение данного списка при чтении алгоритма людьми. Транслятором этот идентификатор списка не используется. В-третьих, описание заголовка списка может быть представлено в виде описания составной величины. В этом случае в качестве составной величины будет указываться идентификатор списка, который является фактическим идентификатором заголовка (фиксатора) данного списка. Структура этой составной величины будет представлять собой структуру заголовка списка. Здесь может указываться число членов в списке, адрес первого члена списка и другие данные, относящиеся к списку в целом.

При наличии такого описания заголовка списка транслятор выделит соответствующую ячейку (ячейки) в памяти машины для размещения фиксатора списка. Отсюда ясно, что фактический заголовок списка может быть представлен в описании списка только в виде составной переменной.

При этом за символом **список** должен сразу стоять символ **составной.** Таким способом могут описываться списки любого типа (цепные, узловые, гнездовые, последовательные).

Важно то, что для величины, заключенной между символами **список... уровень,** обычного распределения памяти производить не требуется, а заданная структура списков учитывается при предварительной организации свободной памяти и в процессе составления операторов программы.

Сравнение адресного и индексного способов работы со списками. Используя описания составных переменных и массивов и возможности выделения их компонент, предусмотренные в языках типа КОБОЛ, АЛГЭМ, АЛГЭК, PL-1 и др., можно программировать операции переходов по цепным спискам.

Операция получения содержимого по заданному адресу, вообще говоря, может быть представлена в виде операции обращения к элементу массива по заданному индексу. Для этого необходимо ввести в описание данных в соответствующем блоке программы массив, элементами которого будут члены цепных списков. Каждый член списка будет представлен составной величиной, в которую входит адрес связи. Этот адрес связи будет указывать порядковый номер элемента в массиве, т. е. являться его индексом.

Для иллюстрации двух указанных возможностей обращения к списковым величинам рассмотрим пример.

Пусть требуется осуществить просмотр некоторого цепного списка, расположенного в оперативном запоминающем устройстве (ОЗУ), и отбор членов, у которых признак П равен заданному значению (ЗП). Введем условный массив ОЗУ [1 : N], охватывающий некоторую зону памяти из N ячеек, в которой будет располагаться список. Будем считать, что каждый член списка расположен в одной ячейке и состоит из трех компонент: признака П, адреса связи АС и признака конца списка КС.

Адрес связи АС указывает положение следующего члена списка относительно начала данной зоны ОЗУ

(т. е. это относительный, а не абсолютный адрес). Индекс элемента в условном массиве ОЗУ [1 : N], представляющего собой первый член списка (вершину списка), обозначим через Ф. Он является заданным. Отобранные члены будем записывать в массив Т [1 : n]; значение индекса этого массива будет указывать число отобранных членов.

Величины ЗП, Ф, Т [i], i, n будем считать глобальными по отношению к рассматриваемому блоку программы. Адреса связи АС являются порядковыми номерами ячеек той зоны ОЗУ, которая выделена для размещения списка.

```
начало...;
составной массив ОЗУ [1 : N];
целый П вид 1 (10), АС вид 1 (20);
логический КС уровень
целый К, j;...;
. . . . . .
К: =Ф;  i:=1;
для  j : = К пока ⌉КС . ОЗУ [j], АС.  ОЗУ  [j] цикл
начало если П. ОЗУ [j] = ЗП то
начало Т [i] : = ОЗУ [j]; i : = i + 1 конец
К : = АС. ОЗУ [j] конец
. . . . . .
конец
```

Этот же участок программы, записанный с использованием адресных символов и описателя **список,** будет иметь вид

```
начало . . .;
список цепь; составной элемент, целый П вид 1 (10),
    АС вид 1 (20); логический КС уровень
уровень;
целый j; . . .;
  ⌈ элемент ⌉ = АС;
. . . . . .
. . . . . .
i:=1;  j: =Ф;
для АС : = j пока ⌉КС.⌊ АС ⌋, АС.⌊ АС ⌋ цикл
начало если П.⌊ АС ⌋ = ЗП то
начало Т [i] : =⌊АС ⌋; i : = i + 1 конец;
j : = АС.⌊ АС ⌋ конец
. . . . . .
конец
```

Здесь идентификатором всего списка является «цепь» (этот идентификатор фактически не используется), а идентификатором члена списка —«элемент».

Сравнение обоих вариантов записи показывает, что они примерно одинаковы; второй вариант содержит в описаниях блока дополнительное описание (а именно адресное соотношение); это описание в данном случае может быть опущено, так как согласно условию каждый член списка находится в отдельной ячейке.

Рассмотренная в гл. 3 система автоматизации программирования АЛГЭМ может использоваться при индексном способе программирования процессов обработки списков, так как возможно представление членов списков в виде составных величин, компонентами которых являются адреса связи — индексы элементов составных массивов. Однако использование адресных символов для описания цепных списков является удобным по следующим причинам:

1) они естественны для машинного представления и наглядны для программирования, так как соответствуют существу машинных операций при обращении к списковым величинам;

2) скобки содержимого могут нести дополнительные функции, выполняя роль обычных (круглых)

скобок для выделения определенных выражений.

Описание списковых процедур. При обработке списковых величин может применяться метод процедур. В виде процедур можно оформлять различные типовые списковые процессы: включение нового члена в список, исключение члена из списка, объединение списков, разделение списка на части, поиск заданных членов в списке, копирование списка и т. п.

Эти процедуры оформляются по правилам АЛГОЛа с учетом добавлений, сделанных в АЛГЭМ-1, а также с использованием введенных в данной главе адресных соотношений и описательных скобок **список... уровень.**

Эти средства используются не только в теле процедуры, оформляемом в виде блока, но и в заголовке процедуры, в разделе спецификаций.

Подобно тому как в АЛГОЛе предусматривается обязательная спецификация формальных параметров, входящих в список значений, при программировании списковых процедур обязательна спецификация формальных параметров, являющихся списковыми величинами. Спецификации списковых величин представляют собой описания списков и описания адресных соотношений, составляемые в полном соответствии с приведенными выше правилами построения таких описаний в блоках. На списковые процедуры полностью распространяются все правила локализации величин в блоках и в процедурах, действующие в АЛГОЛе. В частности, если какой-нибудь формальный параметр процедуры обозначен таким же идентификатором, как и некоторая величина, являющаяся глобальной по отношению к данной процедуре, то эта величина не может быть использована в данной процедуре. При построении спецификаций списковой процедуры (адресных соотношений) могут использоваться величины, являющиеся глобальными для данной процедуры и не являющиеся в ней формальными параметрами.

В описаниях списковых процедур могут использоваться все другие описания, входящие в состав описаний данного блока программы (или какого-нибудь внешнего блока), если это не нарушает общих правил локализации величин в блоках.

В адресных соотношениях, находящихся в составе описаний в блоке, являющемся телом процедуры, могут участвовать как величины, описанные в самом блоке, так и величины, описанные в разделе спецификаций этой процедуры, а также величины, глобальные по отношению к данной процедуре.

Полная спецификация списковых величин в процедурах необходима в связи с тем, что каждая конкретная процедура составляется для определенной структуры списков, и поэтому использовать эти процедуры можно только при обработке списков, имеющих такую же структуру списков и соответствующую организацию свободной списковой памяти.

Перечислим общие дополнительные описания, используемые при ассоциативном программировании:

а) описание списковых процедур, в том числе специальной процедуры организации свободной списковой памяти;

б) описание адресных соотношений;

в) описание списковых величин.

18. Примеры процедур, используемых при ассоциативном программировании

Ниже проводятся описания типовых процедур, используемых при ассоциативном программировании. В качестве примера более сложной программы приводится программа копирования списковой структуры.

Описания списковых процедур мы приводим не изолированно, а в виде перечня описаний процедур, входящих в состав описаний некоторого блока программы. В начале этого блока стоит символ начало, для которого отсутствует соответствующий ему символ **конец,** так как этот блок не является полным. Он содержит в себе описания нескольких величин (ЦЕПНОЙ СПИСОК, ЧС, КС, СЯ, i), глобальных по отношению к приводимым ниже описаниям процедур, а также описания процедур. На этом указанный блок программы как бы обрывается. Распределение разрядов в форматах элементарных величин, входящих в состав спискового слова, дано с таким расчетом, чтобы списковое слово полностью помещалось в ячейку машины «Минск-22», имеющую 37 двоичных разрядов.

Идентификатор СЯ обозначает заголовок списка свободных ячеек, идентификатор i — некоторую промежуточную (рабочую) переменную целого типа, ЧС — член списка. Последние три примера программ следует рассматривать как блоки, входящие в состав данного блока:

начало
список ЦЕПНОЙ СПИСОК; **составной** ЧС; **логический** S; **целый** И **вид** 1 (18), АС **вид** 1 (18)
уровень
уровень;
целый КС **вид** 1 (18), СЯ **вид** 1 (18), i **вид** 1 (18); ⌈ЧС⌉ = АС = i = СЯ;
процедура ПОДГОТОВКА (АН, ЧЯ, СЯ);

примечание производится соединение в цепной список свободных ячеек группы ячеек памяти, начиная с адреса АН, количеством ЧЯ. Адрес вершины этого списка присваивается величине СЯ. Величины СЯ, i, АС, КС являются глобальными по отношению к этой и последующим процедурам;
целый АН, ЧЯ; ⌈ЧС⌉ = АН + ЧЯ — 1;
начало
СЯ : = АН;
для i : = АН **шаг** 1 **до** АН +ЧЯ — 2 **цикл**
АС.⌊ i ⌋ : = i + 1;
АС. ⌊АН + ЧЯ — 1 ⌋ : = КС;
конец
процедура ЯЧЕЙКА (К);

примечание обеспечивается получение свободной ячейки из списка свободных ячеек с присваиванием адреса этой ячейки величине К и соответствующая корректировка списка свободных ячеек;
целый К;
если СЯ = КС **то** на ОСТАНОВ **иначе начало** К : = СЯ; СЯ : = АС. ⌊СЯ⌋ **конец**
процедура ВСТЭК (К, В, Н);

примечание в вершину СТЭКа, которым может быть любой список, заданную адресом К, записываются значения логического признака В и наименования Н на соответствующие им места в списковом слове. Прежние значения признака S и наименования И в этом слове переносятся на соответствующие места в свободную ячейку, взятую из списка свободных ячеек, которая включается в тот же список после вершины. При этом происходит как бы сдвиг всех прежних членов списка. В обращениях к этой процедуре обычно первый параметр является константой, которая фиксирует положение вершины СТЭКа, т. е. указывает ячейку, в которой находится первый член СТЭКа. Поэтому последующие члены СТЭКа вставляются за этой ячейкой;
логический В; **целый** Н, К; ⌈ЧС⌉ = К;
начало
ячейка (i);
И. ⌊i⌋: = И.⌊K⌋;S.⌊i⌋ : =S. ⌊K⌋;
И.⌊K⌋ : = H; S.⌊K⌋ : = B;
АС. ⌊i⌋ : = АС. ⌊K⌋;
АС.⌊K⌋ : = i;
конец
процедура ИЗСТЭКА (К, В, Н);

примечание из СТЭКа, заданного адресом вершины К, исключается верхний член, значение признака S, наименования И которого присваиваются логической величине В и величине Н. Второй член СТЭКА пересылается на место первого члена, т. е. в ячейку К, остальные члены СТЭКа при этом как бы поднимаются на одну ступень вверх. Ячейка, в которой находился второй член СТЭКа, возвращается в список свободных ячеек;
целый Н, К; **логический** В; ⌈ЧС⌉ = К;
начало
В : = S.⌊K⌋;
Н : = И. ⌊K⌋;
i : = АС. ⌊K⌋;
⌊K⌋ : =⌊АС. ⌊K⌋⌋;
⌊i⌋ : = СЯ;

СЯ : = i;
конец
процедура ПОИСК (К, Н, В, Д, М);

примечание производится поиск в списке с адресом вершины К объекта, имеющего наименование И и логический признак S равными соответственно заданным параметрам Н и В. Адрес первого найденного объекта присваивается величине Д и управление передается следующему оператору программы. Если не будет найден ни один объект, то величине Д присваивается адрес последнего члена списка и управление передается оператору с меткой М;

логический В; **целый** Н, К, Д;
метка М; ⌈ЧС⌉ = К;
N : **если** И.⌊К⌋ = Н ∧ S.⌊К⌋ = В **то** Д : = К **иначе**
начало если АС. ⌊К⌋ = КС **то**
начало Д : = К; **на** М **конец иначе**
начало К : = АС.⌊К⌋ **на** N **конец конец**

процедура ВСТАВКА (К, В, Н);

примечание заданные значения логического признака В и наименования Н заносятся на соответствующие места в ячейку и эта ячейка вставляется в список после ячейки с адресом К;

целый К, Н**; логический** В; ⌈ЧС⌉ = К;
начало
ЯЧЕЙКА (i);
И.⌊i⌋ : = Н; S.⌊i⌋ : = В;
АС.⌊i⌋ : = АС.⌊К⌋;
АС.⌊К⌋ : = i;
конец

процедура СВЯЗЬ (К, Ч);

примечание значение адреса связи в ячейке К устанавливается равным величине Ч, являющейся адресом некоторой ячейки. Если первая ячейка была последней в одном списке, а вторая — первой в другом списке, то при помощи этой процедуры оба списка будут объединены в один;

целый К, Ч; ⌈ЧС⌉ = К;
АС.⌊К⌋ : = Ч;

Для связи двух списков, заданных адресами первых членов, к первому из них сначала применяется процедура ПОИСК с Н = 0, а затем связь (Д, Ч).

процедура СТИРАНИЕ (К);

примечание все ячейки списка, начинающегося с ячейки с адресом К, вставляются в список свободных ячеек. Заметим, что здесь тоже можно применять две процедуры: поиск по списку СЯ, а затем *связь* (Д, К);

целый К; ⌈ЧС⌉ = К;
начало
ЯЧЕЙКА (i);
СЯ : = К;
N : **если** АС.⌊К⌋ = КС **то на** М **иначе**
К : = АС.⌊К⌋;
на N;
М : АС.⌊К⌋ : = i;
конец

Ниже приводятся три примера программ, использующих указанные процедуры.

Пример 1.

Исключить из списка, адрес вершины которого равен А, все члены, у которых логический признак S и наименование И равны соответственно логической величине В и строчной величине С. Присвоить величине J число исключенных членов.

Символ * (звездочка) обозначает любую неиспользуемую ячейку.

начало

целый J, A; **логический** B; **строчный** C;
⌈ЧС⌉ =A;
J : =0;
M : ПОИСК (A, C, B, A, N);
ИЗСТЭКА (A, *, *);
J : =J + 1; **на** M;
N : **конец**

Пример 2.

Произвести обмен наименованиями И между первым и последним членами списка, начинающегося с ячейки с адресом A. **начало строчный** I ; **целый** A, J; ⌈ЧС⌉=A = J;
I : =И.⌊A⌋; J: = A;
M : **если** AC.⌊A⌋ ≠ КС **то**
начало A : = AC.⌊A⌋; **на** M **конец,**
И. ⌊J⌋ : =И.⌊A⌋; И. ⌊A⌋ : =I;
конец

Пример 3.

Рассмотрим пример записи на языке ассоциативного программирования более сложной программы, осуществляющей копирование некоторой ассоциативной списковой структуры. Эта программа в качестве основной части имеет программу копирования цепного списка, но так как списковые структуры представляют собой в общем случае совокупности списков и подсписков различных уровней, то полная программа копирования структуры должна обеспечивать возможность многократного вызова собственно программы копирования цепного списка, т. е. она должна иметь рекурсивный характер. Такой вызов должен осуществляться в каждой точке разветвления прослеживаемого списка (подсписка). При этом необходимо каждый раз запоминать место возврата в данный список для того, чтобы после окончания копирования подсписка продолжить копирование ранее прослеживаемого списка. Такое запоминание точек возврата осуществляется с помощью СТЭКа I, в который засылаются адреса членов копируемой структуры (i), от которых происходит ответвление подсписков.

При ответвлении от данного списка некоторого подсписка в СТЭКЕ I запоминается адрес члена списка, от которого произошло ответвление; при ответвлении другого подсписка от данного подсписка в этот же СТЭК засылается адрес члена данного подсписка, от которого произошло ответвление, и т. д.

В программе используется второй СТЭК J для запоминания текущих адресов (j) членов структуры — копии, от которых ответвляются подсписки копии. Оба СТЭКа I и J работают синхронно и их, вообще говоря, можно было бы заменить одним СТЭКом. Использование двух раздельных СТЭКов I и J делает процесс более наглядным.

Кроме этих двух СТЭКов в данной рекурсивной программе используется еще и третий СТЭК M. Этот СТЭК служит для запоминания меток возврата при прерываниях хода программы и повторных обращениях к началу программы собственно копирования ассоциативной структуры. Такие прерывания осуществляются каждый раз при встрече в основной (копируемой) структуре точки ответвления. После окончания копирования очередного подсписка происходит восстановление СТЭКа M. В эти же моменты происходит и восстановление СТЭКов I и J.

Естественно, что при работе данной программы используется список свободных ячеек, из которого при помощи процедуры ЯЧЕЙКА (j) берутся свободные ячейки для построения ассоциативной структуры-копии.

Для начала работы программы копирования задается величина A — адрес вершины (первого члена, а не заголовка) списковой структуры-оригинала. B — это переменная, которой должен быть присвоен адрес вершины списковой структуры-копии; i — адрес текущего копируемого члена структуры-оригинала; j — адрес текущего члена в структуре-копии; k — промежуточная (рабочая) переменная.

Заметим, что так как в СТЭКе M должны храниться метки возврата, а не значения (числовые) некоторой переменной величины, то в качестве третьего фактического параметра в соответствующих обращениях к процедуре ВСТЭК указаны метки в виде строчных величин, т. е. метки, заключенные в кавычки. При *i* обращении к СТЭКу M при помощи оператора процедуры ИЗСТЭКа третий параметр этой процедуры L принимает значения соответствующих меток, т. е. величина L должна быть строчной величиной. В качестве второго параметра в операторах процедур ВСТЭК и ИЗСТЭКА фигурирует

постоянно единица, так как этот параметр не используется, но указывать его нужно для общего счета числа параметров.

```
    начало
    целый i, j, К, I, J, M; строчный L вид A9;
    ⌈ЧС⌉ = i = j = К;
    начало
    i: =A;
    ЯЧЕЙКА (B);
    j: =B;
    ВСТЭК (M,  1,  M3);
M1  : если ⌈S. ⌊i⌋ то
начало S. ⌊j⌋ : = 0; И. ⌊j⌋ :  = И. ⌊i⌋ если AC. ⌊i⌋ : = КС то начало AC.⌊j⌋ : = КС; на M2 конец
закончилось копирование последнего члена подсписка;
K:=j; ЯЧЕЙКА (j); AC. ⌊K⌋ : = j;  i : = AC ⌊i⌋ ; на M1 конец закончилось копирование очередного
объектного члена списка;
S. ⌊j⌋ : =1; К: = j; ВСТЭК (J, 1, j);
    ЯЧЕЙКА (j); И. ⌊K⌋ : = j;
    ВСТЭК (I, 1, i); i: = И.⌊i⌋ ; ВСТЭК (M, 1), 'M4'); на M1;
    M4: ИЗСТЭКА (I, 1, i); i : = AC.⌊i⌋;
    ИЗСТЭКА (J, 1, К); ЯЧЕЙКА (j);  AC. ⌊K⌋ : = j;
    на M1;
M2 : ИЗСТЭКА (M, 1, L);
на;
M3 : ОСТАНОВ;
    конец
    конец
```

Поиск документов в ассоциативно-адресной дескрипторной поисковой системе. Общий принцип действия дескрипторных поисковых систем был рассмотрен во введении. Сейчас рассмотрим один из способов машинной реализации такой системы и алгоритм поиска документов в ней, который запишем на алгоритмическом языке.

Для простоты будем считать, что вся информация помещается в оперативной памяти машины, и поэтому вопросы обмена данными между ОЗУ и МЛ или МБ рассматривать не будем.

Для построения дескрипторной поисковой системы необходимо иметь словарь дескрипторов, т. е. фиксированный набор терминов-определений и массив поисковых образов документов. Поисковый образ документа это группа дескрипторов, характеризующих содержание документа; у разных документов эти поисковые образы могут иметь различное число дескрипторов (в среднем по 5—10 дескрипторов). Будем использовать узловой способ организации массива поисковых образов документов. Все документы, учитываемые в системе, так же как и все дескрипторы, имеют свои постоянные коды. Считаем, что такими кодами являются порядковые номера.

При прямом способе организации массива поисковых образов документов за каждым кодом документа перечисляются все коды дескрипторов, которыми обладает этот документ.

Каждый документ в памяти машины будем представлять в виде одного ассоциативного узла, в котором число списковых слов соответствует числу дескрипторов, имеющихся у документа. Таким образом, каждому дескриптору будет соответствовать одно определенное списковое слово в узле. Ясно, что одни и те же дескрипторы будут фигурировать в поисковых образах многих различных документов.

Представление поисковых образов документов в виде ассоциативных узлов позволяет с помощью списковых слов связывать в единые цепные списки все документы, имеющие одни и те же дескрипторы. Очевидно, что один и тот же документ будет входить во столько различных цепных списков, сколько дескрипторов он имеет в своем поисковом образе.

В памяти машины выделяется специальная зона (группа последовательных ячеек) под массив фиксаторов или заголовков списков. Каждому дескриптору соответствует один цепной список и один заголовок списка; каждый заголовок списка размещается в одной ячейке. Адрес этой ячейки является

кодом данного дескриптора; зная код дескриптора, можно обратиться к нужному заголовку списка. Вопрос о том, как от словесного представления дескриптора перейти к его коду, мы рассматривать не будем. Это может быть сделано, например, методом сверток.

Заголовок списка включает в себя пять элементов, т. е. он является составной величиной. Первым элементом, входящим в состав заголовка, будет адрес связи начала цепного списка (АС) документов, имеющих в своем поисковом образе данный дескриптор. Этот адрес указывает первый ассоциативный узел, а точнее, первую ячейку узла, представляющего первый документ в этом списке документов. Так как каждый документ характеризуется несколькими дескрипторами, каждому из которых соответствует свое списковое слово в его ассоциативном узле, то для следования по цепному списку необходимо указывать не только адрес очередного узла, но и номер того спискового слова в узле, которое соответствует данному списку (т. е. данному дескриптору). Для этой цели служит второй элемент в заголовке списка, называемый номером слова начального (НС). Оба элемента АС и НС образуют слово ССН — списковое слово начальное.

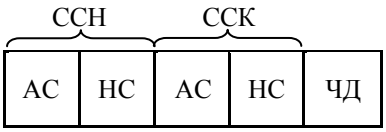
Третий элемент заголовка представляет собой адрес связи, указывающий конечный (последний) ассоциативный узел в данном цепном списке, соответствующий последнему документу в данном списке (АС — адрес связи конца).

Четвертый элемент указывает номер спискового слова в ассоциативном узле, соответствующем последнему документу (НС — номер слова конечного). Эти АС и НС образуют ССК — списковое слово конечное — и используются при включении в систему новых документов. Каждый новый документ, вводимый в систему, должен включаться в те цепные списки, которые соответствуют имеющимся в его поисковом образе дескрипторам. Новые документы включаются в концы списков, а просмотр списков при поиске производится с начала списка.

При таком порядке включения новых документов документы в цепных списках будут располагаться в хронологической последовательности, соответствующей порядку их поступления в поисковую систему. Можно было бы не иметь в заголовках списков элементов для концов списков, но тогда для нахождения последних ассоциативных узлов в цепных списках (т. е. концов этих списков) пришлось бы каждый раз при включении нового документа просматривать полностью соответствующие цепные списки, что привело бы к излишней затрате машинного времени. Кроме того, хранение в заголовках списков адресов двух граничных точек списков (начала и конца) полезно с точки зрения контроля работы системы.

Возможен также такой вариант, когда используется адрес только одного (последнего) узла и запись узлов идет в обратном порядке, т. е. конца МЛ к ее началу. Этот способ более подробно будет рассмотрен в следующей главе.

Пятый элемент заголовка списка представляет собой число членов в данном списке, т. е. число документов (ЧД), введенных в систему и обладающих данным дескриптором. Этот элемент необходим для периодической проверки размеров списков, соответствующих разным дескрипторам. Знание размеров списков необходимо для корректировки словаря дескрипторов. Кроме того, знание размеров списков может быть использовано для построения более рациональных запросов и организации оптимального процесса поиска документов. Ниже показан формат заголовка списка.



Рассмотрим теперь строение ассоциативного узла. В первой ячейке узла размещается составная величина, которую мы назовем заголовком узла. В состав заголовка узла входят три элемента. Первым элементом будет указатель документа (УД), представляющий собой некоторый код документа, например его порядковый номер, с помощью которого этот документ может быть найден в хранилище.

Следующим элементом заголовка узла является так называемая дополнительная информация о документе (ДИ), которая содержит дополнительные сведения о документе и сама может являться составной величиной. Конкретное строение ДИ зависит от конкретных условий применения системы, и мы его рассматривать не будем. Сюда могут входить, например, признаки вида документа (книга, журнал, отчет и т. д.), год и место издания, язык, на котором написан документ, и т. д.

Третьим элементом заголовка узла является число списковых слов в узле (ЧС). Оно необходимо для указания общего размера узла (при вводе документов в систему и построении узлов, а также при

переносе узлов с места на место). Этим элементом заканчивается заголовок узла.

Во второй и последующих ячейках узла располагаются списковые слова, по одному слову в ячейке памяти. Каждое списковое слово состоит из двух частей: адреса связи (АС), указывающего первую ячейку другого узла, представляющего следующий документ в данном цепном списке, и номера слова (НС), указывающего, какое списковое слово в следующем узле соответствует данному дескриптору. Окончания цепных списков обозначаются специальным кодом КС, который ставится на месте адресов связи. Заметим, что код КС в соответствии с алгоритмом поиска, который описывается ниже, должен быть больше по своей величине любого из адресов связи АС. Таким кодом может быть код, состоящий из единиц во всех разрядах адреса связи.

На рис. 27 приведен формат ассоциативного узла.

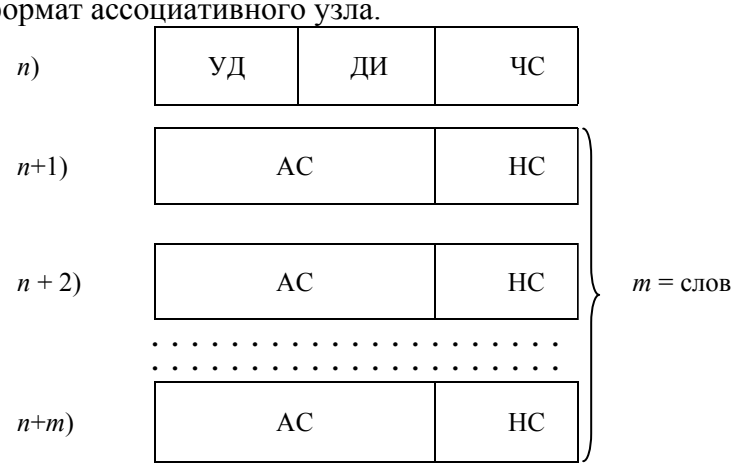


Рис. 27. Ассоциативный узел.

Для пояснения общего принципа работы системы на рис. 28 приведен фрагмент ассоциативно-адресной структуры, включающий в себя словарь из пяти дескрипторов (пять заголовков списков, размещенных в ячейках с адресами от 107 до 111) и пять ассоциативных узлов, соответствующих пяти документам. В рассматриваемой ассоциативно-адресной системе все цепные списки имеют направленный характер, т. е. адреса связи в них при движении вдоль по спискам монотонно возрастают. Это связано с тем, что расположение ассоциативных узлов на магнитной ленте соответствует порядку ввода документов в систему.

В общем случае в цепных или узловых списках члены списков будут располагаться в памяти машины произвольным образом, и поэтому адреса связи у них могут меняться при переходе от одного члена списка к следующему как в сторону возрастания, так и в сторону убывания. В приведенном фрагменте ассоциативно-адресной системы выдерживается свойство монотонного возрастания адресов связи при перемещении вдоль списков.

Монотонное изменение адресов связи является существенным достоинством ассоциативно-адресного способа построения дескрипторной поисковой системы, так как позволяет производить просмотр цепных списков, расположенных на магнитной ленте, за один прогон ленты без реверсов и повторных прогонов.

При реализации подобной ассоциативно-адресной поисковой системы на магнитных лентах возникают некоторые особенности в программировании и распределении памяти, которые мы сейчас кратко рассмотрим.

Оперативная память машины под все несписковые величины распределяется обычным образом (т. е. выделяются зоны констант, рабочих ячеек, зона для размещения программы и т. д.). Из остающегося свободного объема ОЗУ выделяется зона под обрабатываемый списковый блок (например, размером в 2048 ячеек). В соответствии с размером этой рабочей списковой зоны, выделенной в ОЗУ, производится запись спискового массива на магнитной ленте в виде блоков данных по зонам, размер которых равен размеру рабочей списковой зоны в ОЗУ. В процессе работы системы списковый массив будет переписываться для обработки по блокам в рабочую списковую зону ОЗУ. Так как размеры ассоциативных узлов для разных документов могут быть различными, а размеры всех зон на МЛ устанавливаются одинаковыми, то ясно, что в разных зонах может быть различное число ассоциативных узлов, а в конце зон могут оставаться свободные ячейки. Для контроля заполнения МЛ списковым массивом в ОЗУ должны быть выделены постоянно две ячейки: в одной, называемой фиксатором зоны, должен храниться адрес очередной заполняемой зоны МЛ, а во второй — фиксаторе ячейки — адрес очередной свободной ячейки в этой зоне (практически это может быть один код). Эти

фиксаторы используются только при включении новых документов в систему. При необходимости включить новый документ в систему сначала по значению фиксатора зоны определяется нужная зона, а затем по значению фиксатора ячейки определяется адрес ячейки внутри зоны. Производится проверка, достаточно ли свободных ячеек в этой зоне для записи нового ассоциативного узла, и если достаточно, то эта зона переписывается с МЛ в ОЗУ и в нее вносится новый ассоциативный узел с включением его во все необходимые списки. После этого значение фиксатора ячеек уменьшается на соответствующее количество ячеек.

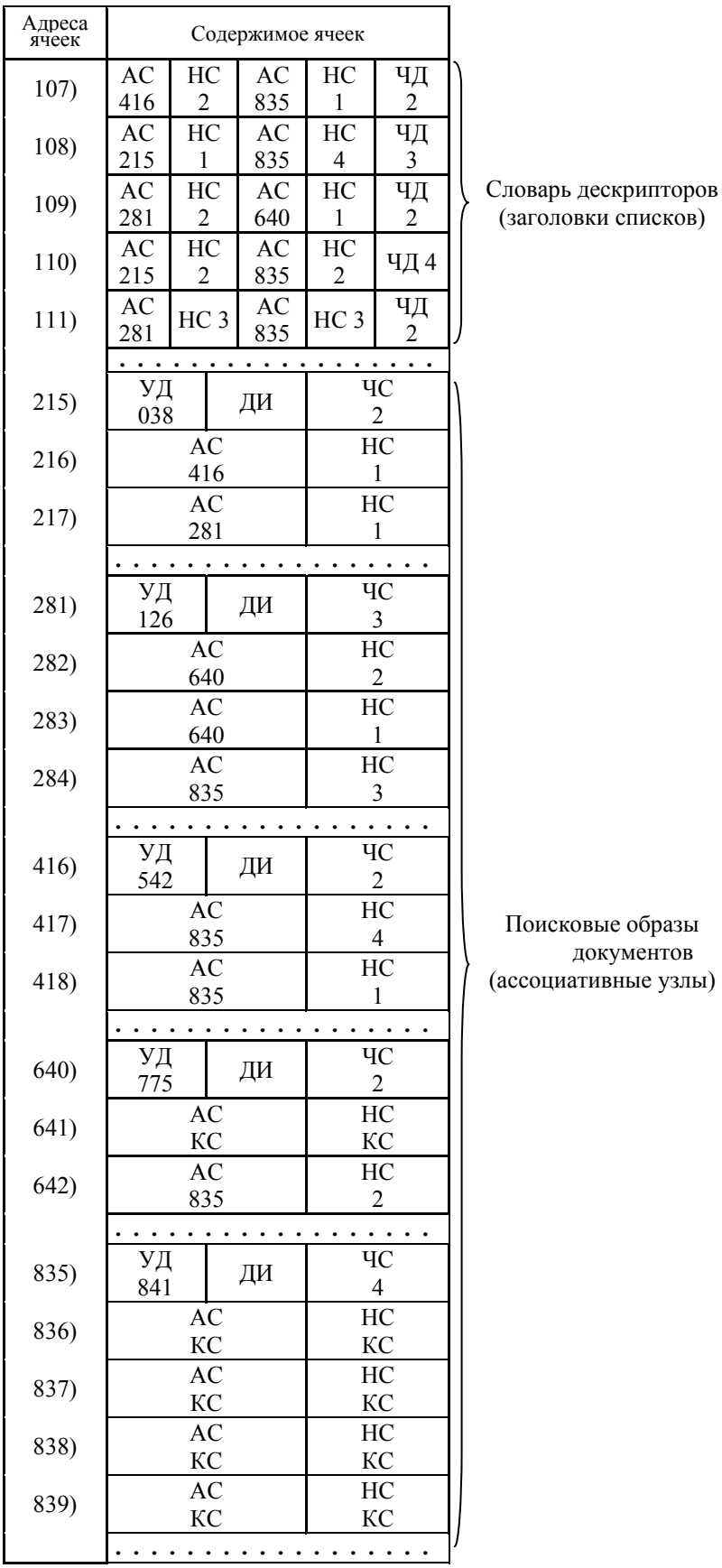


Рис. 28. Фрагмент ассоциативно-адресной поисковой системы.

Если количество свободных ячеек в данной зоне окажется меньше требуемого для записи нового узла, то запись нового узла производится в следующую зону; при этом значение адреса текущей зоны в фиксаторе зоны увеличивается на единицу, а значение фиксатора ячеек сбрасывается в нуль. При поиске нужных документов весь списковый массив поочередно по блокам переписывается с МЛ в рабочую списковую зону ОЗУ. Ясно, что при записи в ОЗУ ассоциативные узлы каждого блока попадают на определенные места (в ячейки) внутри рабочей списковой зоны ОЗУ, и поэтому адреса связи, используемые в узлах, должны соответствовать адресам, которые будут иметь требуемые узлы в рабочей списковой зоне.

Ниже приводится запись на алгоритмическом языке алгоритма поиска документов в ассоциативно-адресной дескрипторной поисковой системе. Для простоты предполагается, что вся информация помещается в оперативной памяти машины, и поэтому описание распределения памяти не приводится. Алгоритм оформлен в виде процедуры. В качестве формальных параметров фигурируют: массив заголовков списков (СЛОВАРЬ), массив дескрипторов запроса (ЗАПРОС) и массив отобранных документов (ОТВЕТ). Все эти формальные параметры конкретизируются наименованием, т. е. вместо их идентификаторов при обращении к процедуре будут подставляться идентификаторы соответствующих фактических массивов. Основной списковый массив документов, в котором производится поиск, не фигурирует в качестве формального параметра, так как заданием массива заголовков списков (словаря дескрипторов) однозначно определяется и этот массив документов. Ясно, что в спецификациях составного массива СЛОВАРЬ и целых массивов ЗАПРОС и ОТВЕТ границы изменения индексов не указываются. Они будут заданы при конкретизации этих массивов фактическими параметрами.

К формальным параметрам относится, кроме того, простая переменная, представляющая собой число дескрипторов в запросе. Эта переменная, определяющая размер массива ЗАПРОС, должна быть введена в качестве отдельного параметра, так как она используется в теле процедуры независимо от массива ЗАПРОС. Для составного массива СЛОВАРЬ, являющегося формальным параметром, в соответствии с правилами АЛГЭМ-1 дается подробная спецификация, определяющая структуру этой составной величины. В теле процедуры, оформленном в виде блока, описан ряд величин, являющихся локальными по отношению к данному блоку. Сюда относятся целые простые переменные i, j, играющие роль параметров циклов, величина МАКС АС, используемая в качестве промежуточной (рабочей) величины (для запоминания максимального значения адреса связи). Здесь же описана структура основного спискового массива документов, в котором производится поиск. Последнее описание необходимо для построения операторов тела процедуры. После перечисленных описаний величин идут описания адресных соотношений, используемые при построении операторов тела процедуры. В данном примере используется простейший критерий смыслового соответствия для поиска документов: полное вхождение дескрипторов запроса в поисковый образ документа. Поиск осуществляется по одному запросу.

Сначала идет оператор цикла, осуществляющий выборку необходимых заголовков списков из словаря дескрипторов. Используя элементы массива ЗАПРОС в качестве адресов элементов массива СЛОВАРЬ, выбираем *n* значений величины ССН; эти значения присваиваются элементам рабочего массива ССР (списковое слово рабочее). Затем идет оператор цикла, осуществляющий проверку на совпадение адресов связи (АС) в выбранных словах ССН.

Совпадение адресов связи будет свидетельствовать о том, что все адреса связи указывают на один и тот же ассоциативный узел, т. е. на один и тот же документ, который должен быть выдан в составе ответа. Здесь сначала проверяется наличие среди адресов связи символа КС (конца списка). Если среди проверяемых адресов связи имеется этот символ, то дальнейшие поиски бесполезны, так как среди дескрипторов запроса имеется дескриптор, у которого в цепном списке вообще нет документов. Если окажется, что среди АС нет символа КС и, кроме того, не все АС совпадают между собой, то необходимо осуществить переход вдоль цепных списков, соответствующих заданным дескрипторам запроса. Для этого сначала выбирается из всех адресов связи (АС), имеющихся в составе элементов массива ССР, максимальный адрес связи (МАКС АС); затем производится движение вдоль по всем цепным спискам, соответствующим заданным дескрипторам, до тех пор, пока текущие адреса связи в них не станут равными или большими этого МАКС АС. Очевидно, что в силу монотонности изменения адресов связи при движении вдоль списков, совпадения адресов связи не произойдет ни для одного значения адреса связи, меньшего, чем МАКС АС. При наличии на каком-то этапе прослеживания списков совпадения всех текущих адресов связи процесс движения по спискам

приостанавливается и производится присваивание заголовка данного узла очередному элементу массива ОТВЕТ (фиксируется найденный документ).

После этого делается один шаг по всем прослеживаемым цепным спискам, затем осуществляется переход к оператору с меткой ПРОВЕРКА НА СОВПАДЕНИЕ.

Шаг по списку, т. е. переход от данного члена цепного списка к следующему, указываемому адресом связи, содержащимся в данном члене списка, производится путем замены соответствующего элемента массива ССР списковым словом с адресом АС + НС, где АС — адрес ассоциативного узла, а НС — номер спискового слова в узле.

Описанный процесс движения по цепным спискам и проверки адресов связи на совпадение продолжается до тех пор, пока не окончится какой-нибудь из прослеживаемых цепных списков.

После этого производится выдача на печать массива ОТВЕТ, содержащего заголовки узлов отобранных документов. Дополнительные детали алгоритма поиска указываются в виде примечаний в самом алгоритме, представленном на алгоритмическом языке.

Естественно, что в этом алгоритме, описывающем только процесс поиска документов по заданному набору дескрипторов (запросу), отсутствуют какие-либо сведения, касающиеся образования как основного спискового массива документов, в котором производится поиск, так и массивов словаря дескрипторов и дескрипторов запроса, используемых при поиске.

Основной списковый массив документов образуется автоматически путем последовательного включения в этот массив новых документов. Это осуществляется при помощи специальной процедуры включения документа. Для каждого включаемого документа задается заголовок узла (УД, ДИ, ЧС) и набор его дескрипторов.

Сначала производится запись нового ассоциативного узла на очередное свободное место в памяти машины (при этом используются упомянутые выше фиксатор зоны и фиксатор ячейки); в этом новом узле на местах адресов связи (АС) во всех списковых словах ставится сразу символ КС (конец списка), так как этот узел, естественно, будет последним во всех списках, в которые он будет включен. Заметим, что для правильной работы рассмотренного нами алгоритма поиска необходимо, чтобы этот символ КС был больше по своему значению любого возможного значения адреса связи (АС). После записи нового ассоциативного узла производится его включение во все списки, соответствующие имеющимся у него дескрипторам. Включение производится поочередно для каждого заданного дескриптора.

По коду дескриптора находится в словаре дескрипторов соответствующий заголовок списка; в этом заголовке выбирается вторая часть, а именно ССК, и по АС. ССК находится последний узел в данном списке, а по НС. ССК — номер спискового слова в этом узле. Ясно, что в качестве АС в этом списковом слове должен стоять КС.

Затем на место АС в это слово записывается адрес нового ассоциативного узла, а на место НС в это слово записывается порядковый номер дескриптора в наборе дескрипторов нового документа.

Эти же значения АС и НС записываются в ССК данного заголовка списка, так как теперь уже последним членом в списке будет ассоциативный узел нового документа. На этом включение нового узла в один список заканчивается. Таким же образом производится включение нового узла во все остальные списки, соответствующие остальным дескрипторам нового документа. Словарь дескрипторов вводится в систему отдельной процедурой один раз в начале работы поисковой системы. При этом в начальный момент, когда в систему не включен еще ни один документ, на местах всех АС в ССН и ССК заголовков списков должны стоять символы КС. Это необходимо для правильного выполнения описанного выше алгоритма поиска. Затем по мере образования списков документов символы КС будут заменяться соответствующими АС. Заметим, что можно для повышения контроля работы системы ввести еще обратные адреса связи, т. е. в последних членах списков (ассоциативных узлах) на местах АС ставить не КС, а адреса соответствующих заголовков списков в словаре дескрипторов; для указания же концов, списков ставить символ КС (все единицы) на месте НС в этом же списковом слове. Кроме того, факт окончания списка будет устанавливаться и по равенству текущего адреса связи в прослеживаемом цепном списке адресу заголовка этого списка. Следует заметить также, что в описанной процедуре поиска остается открытым вопрос о вводе в систему дескрипторов запроса.

Это должно делаться специальной процедурой (или оператором) ввода. При обращении же к процедуре поиска считается, что этот массив дескрипторов конкретного запроса уже введен в машину и с его помощью производится конкретизация наименованием формального параметра ЗАПРОС в данной процедуре. Также считается заданной и величина n — число дескрипторов в запросе, которая должна

вводиться при вводе запроса.

процедура ПОИСК (СЛОВАРЬ, ЗАПРОС, n, ОТВЕТ);
значение n; **целый** n;
составной массив СЛОВАРЬ; **составной** ССН;
целый АС **вид** 1 (20), НС **вид** 1 (5) **уровень;**
составной ССК; **целый** АС **вид** 1 (20), **НС вид** 1 (5)
уровень;
целый ЧД **вид** 1 (12) **уровень;**
целый массив ЗАПРОС, ОТВЕТ;
начало
целый i, j, k, МАКС АС **вид** 1 (20);
составной массив ССР [1 : n]; **целый** АС **вид** 1 (20),
НС **вид** 1 (5) **уровень;**
список МАССИВ ДОКУМЕНТОВ;
составной ЗАГОЛОВОК УЗЛА; **целый** УД **вид** 9 (5),
ДИ **вид** 9 (6), ЧС **вид** 99 **уровень;**
список ГРУППА АССОЦИАТИВНЫХ СЛОВ;
составной АССОЦИАТИВНОЕ СЛОВО; **целый** АС
вид 1 (20), НС **вид** 1 (5) **уровень уровень уровень;**
⌈СЛОВАРЬ[ ]⌉ = ЗАПРОС [ ];
⌈ЗАГОЛОВОК УЗЛА⌉ = АС;
⌈АССОЦИАТИВНОЕ СЛОВО⌉ = АС + НС;
ВЫБОРКА ЗАГОЛОВКОВ СПИСКОВ:
для i: = l **шаг** 1 **до** n **цикл**
ССР [i]: = ССН.⌊ЗАПРОС [i]⌋; k : = 0;
ПРОВЕРКА НА СОВПАДЕНИЕ:
для i : = 1 **шаг** 1 **до** n — 1 **цикл**
начало если АС. ССР [i] = КС то **на** ОКОНЧАНИЕ;

примечание случай АС. ССР [n] = КС будет обнаружен при выполнении оператора с меткой ВЫБОР МАКС АС;

если АС. ССР [i] ≠ АС. ССР [i + 1] **то**
на ВЫБОР МАКС АС **конец;**
k : = k + 1;
ОТВЕТ [k]:=⌊ AG ССР [l]⌋;

примечание этот оператор осуществляет фиксацию найденного документа, а следующий оператор цикла осуществляет один шаг по всем прослеживаемым цепным спискам;

для i: = 1 **шаг** 1 **до** n **цикл**
ССР [i] : =⌊АС. ССР [i] + НС. ССР [i]⌋,
на ПРОВЕРКА НА СОВПАДЕНИЕ;
ВЫБОР МАКС АС:
МАКС АС: = АС. ССР [i];

примечание i сохранило полученное значение, а все предшествующие АС равны i-му АС;

для j : = i + 1 **шаг** 1 **до** n **цикл**
начало если АС. GCP [j] = КС то **на**
ОКОНЧАНИЕ;

примечание все предшествующие АС уже проверены на равенство КС;

если МАКС АС < АС. ССР [j] **то**
МАКС АС : = АС. ССР [j] **конец;**

примечание при этом МАКС АС не может оказаться равным КС. Следующая часть программы осуществляет движение по всем прослеживаемым цепным спискам. Исходное положение характеризуется тем, что все текущие АС не равны КС и среди них выбран МАКС АС. Проверка на КС

при движении по спискам будет производиться автоматически за счет того, что нами принято условие, что значение КС больше любого АС;

```
i: = 1;
ШАГ ПО СПИСКУ:
если АС. ССР [i] <МАКС АС то
начало ССР [i] : =⌊ АС. ССР [i] + НС. ССР [i]⌋;
на ШАГ ПО СПИСКУ конец иначе
начало i : = i + 1; если i > n то
на ПРОВЕРКА НА СОВПАДЕНИЕ иначе
на ШАГ ПО СПИСКУ конец
ОКОНЧАНИЕ:
конец
```

Глава 6

СТРУКТУРА И АЛГОРИТМЫ ИНФОРМАЦИОННО-ПОИСКОВЫХ СИСТЕМ

19. Фактографическая И ПС для учета оборудования предприятий

В первой главе при рассмотрении состава и содержания задач отраслевой автоматизированной системы управления были описаны задачи учета оборудования и планирования его поставок, замены, ремонта и т. д. Эти задачи решаются на основе фактографической информации о состоянии оборудования предприятий, накапливаемой в главном вычислительном центре министерства, в специальной информационно-поисковой системе фактографического типа (ФИПС).

Рассмотрим принципы машинной реализации указанной системы, ее основные алгоритмы и способы организации информационных массивов. В основу ее построения положены методы ассоциативного программирования, рассмотренные в гл. 5 и предусматривающие организацию информации в виде списков и списковых структур 4-х типов (последовательные, цепные, гнездовые и узловые). Информация об оборудовании, используемая в ФИПС, по своему функциональному назначению делится на следующие виды:

1) информация, относящаяся к предприятиям в целом (общее число рабочих, размер производственных площадей и т. д.);
2) информация о конкретном составе оборудования каждого предприятия;
3) информация о резервах оборудования в отрасли;
4) информация о классификации оборудования и его технических характеристиках.

Информация первых двух видов образует первое структурное дерево, а остальная информация входит во второе структурное дерево (рис. 29 и 30). Верхний уровень первого структурного дерева представляется последовательным списком Т1, каждый член (ячейка) которого соответствует одному главному управлению министерства. Как упоминалось выше (гл. 1), каждое министерство делится на

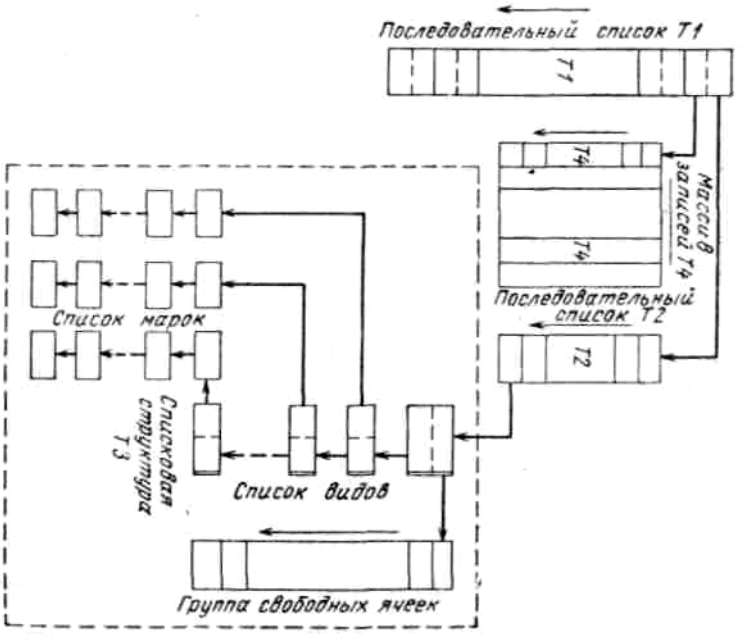


Рис. 29. Структурное дерево № 1.

ряд главных управлений (ГУ) и каждое ГУ имеет в своем ведении группу предприятий. В соответствии с этим учет оборудования в отрасли должен быть построен по предприятиям, а предприятия должны объединяться в группы по главным управлениям. Машинный шифр предприятия формируется из номера ГУ (старшие цифры) и номера предприятия (младшие цифры). Первая строка списка Т1 играет роль фиксатора; она содержит адрес свободной области на МЛ (АСО) и адрес свободных ячеек в ОЗУ (АСЯ). Затем идут списковые слова, содержащие признак конца списка Т1 (ПРК — один разряд), признак резервной (свободной) ячейки (ПРР — один разряд), начальный адрес массива на МЛ записей Т4, относящихся к предприятиям данного ГУ (НАТ4 — 23 разряда), плавающий относительный адрес начала гнезда списка Т2 фиксаторов списковых структур Т3 предприятий, относящегося к данному ГУ (ПОАСТ2 — 12 разрядов). Прибавление ПОАСТ2 к адресу ячейки, в которой он хранится, дает адрес

начала гнезда списка Т2. В последней ячейке списка Т1 хранится контрольная сумма. Строение списка Т1 показано на рис. 31. Адрес спискового слова в списке Т1, относящегося к ГУ с номером NГУ, будет получаться в виде суммы: АГУ [N] =НАТ1 + NГУ; где НАТ1 — начальный адрес списка Т1. Новые члены списка Т1 записываются либо на места исключенных членов (ПРР = 1), либо в конец списка.

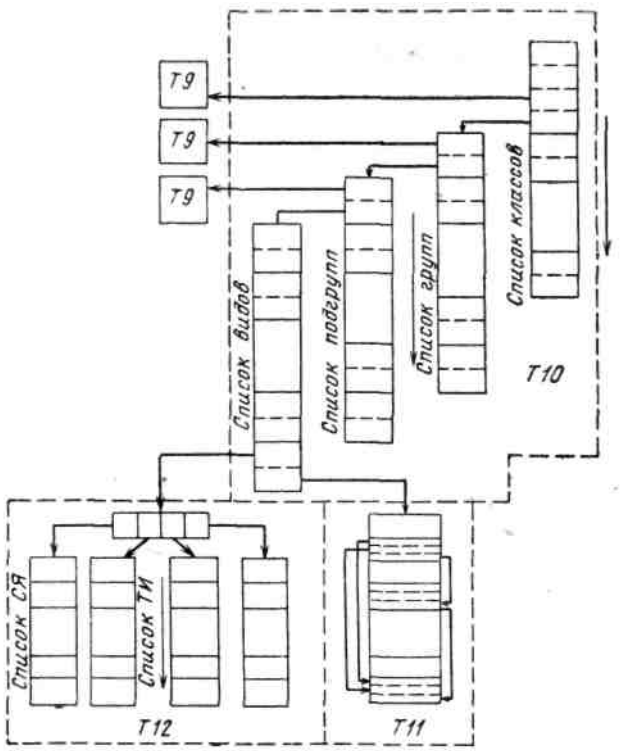


Рис. 30. Структурное дерево № 2.

Записи Т4 имеют одинаковое строение и размер для всех предприятий. Каждому предприятию соответствует одна запись Т4; на МЛ они располагаются подряд. Если К — число ячеек, занимаемых записью Т4, то адрес записи,

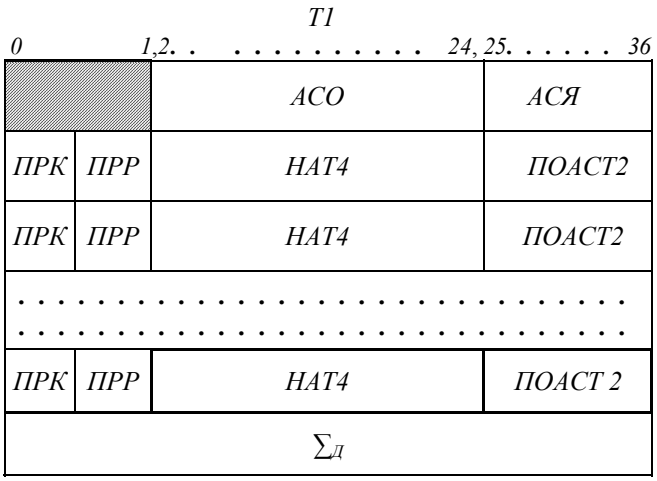


Рис. 31. Строение списка Т1.

относящейся к предприятию с номером NП (нумерация предприятий идет в пределах ГУ), будет равен:

$$АТ4 = НАТ4.\lfloor НАТ1 + NГУ \rfloor + К \times (NП — 1)$$

Здесь NГУ — номер главного управления. В записи Т4 указываются такие общие характеристики предприятия, как число станочников, размер производственных и вспомогательных площадей, стоимость основных фондов, коэффициенты сменности (общий и по видам оборудования), различные производственные и экономические показатели предприятия. Структура записи Т4 может иметь произвольный вид.

Каждому предприятию соответствует одна списковая структура Т3, содержащая подробные сведения о составе оборудования данного предприятия по видам, маркам, возрастным группам (NВГ) и

физическому состоянию (износу— ФИ). Фиксаторы всех списковых структур Т3 сгруппированы в гнездовой список Т2 (рис. 32); каждое гнездо этого списка соответствует одному ГУ, а каждая ячейка внутри гнезда — одному предприятию. В ячейке списка Т2 хранятся четыре величины: начальный адрес на МЛ (НАТ3 —23 разряда) и количество ячеек (КТ3—12 разрядов) списковой структуры Т3, относящейся к данному

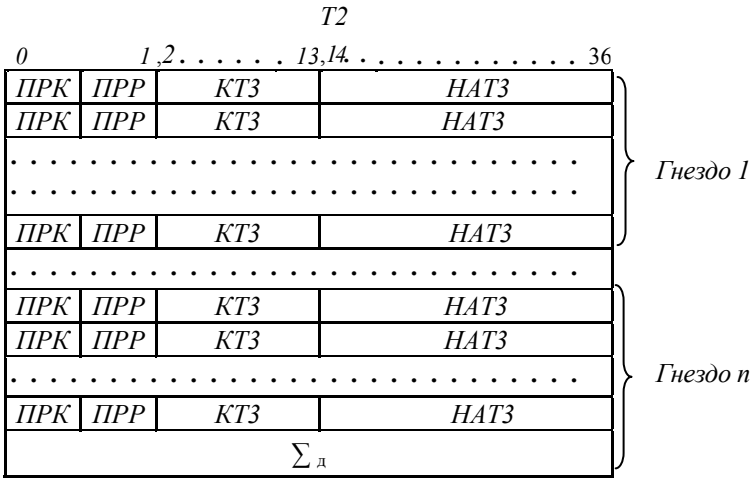


Рис. 32. Строение списка Т2.

предприятию, признак конца гнезда (ПРК—1 разряд) и признак свободной ячейки гнезда (ПРР—1 разряд). Адрес списковой структуры Т3 для предприятия с номером NП, относящегося к ГУ с номером NГУ, будет определяться следующим образом:

$$АТ3 = НАТ3.\lfloor АТ2 \rfloor + NП$$

или

$$АТ3 = НАТ3.\lfloor НАТ1 + NГУ + + ПОАСТ2.\lfloor НАТ1 + NГУ \rfloor \rfloor + NП$$

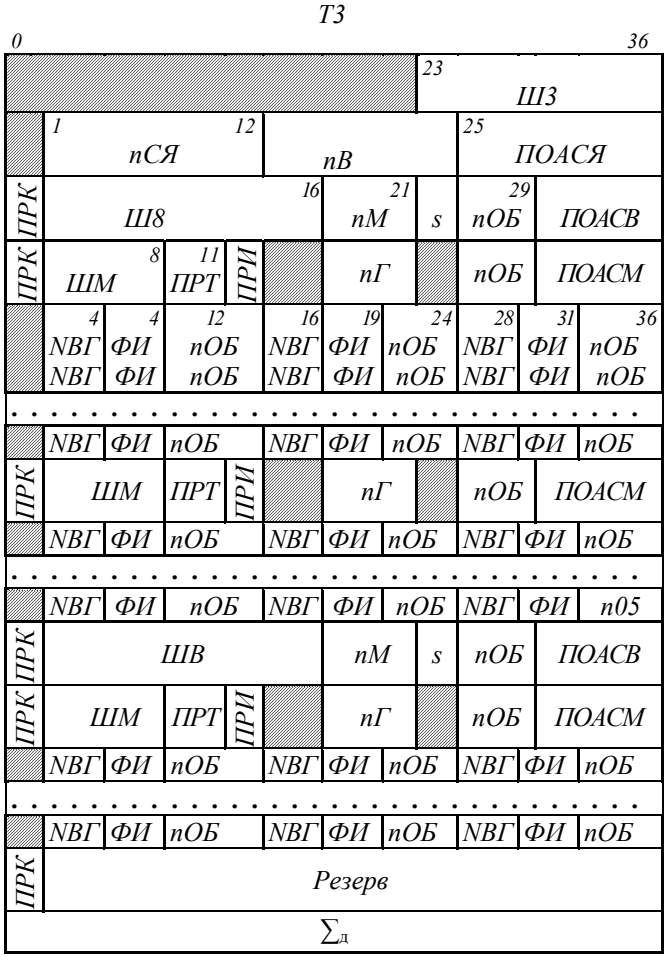


Рис. 33. Списковая структура Т3.

На рис. 33 показано строение списковой структуры Т3. Она включает в себя списки трех уровней: цепной список видов оборудования, от каждого члена которого ответвляется цепной подсписок марок, от каждого члена которого в свою очередь ответвляется последовательный подсписок групп оборудования, характеризующихся определенным годом выпуска (NВГ) и физическим износом (ФИ).

Первые две ячейки структуры Т3 играют роль заголовка. В первой ячейке помещается шифр завода Ш3, используемый для контроля обращения к данной списковой структуре. Во второй ячейке находятся три величины: nСЯ — количество свободных ячеек в участке памяти, отведенном под эту списковую структуру Т3, ПОАСЯ — плавающий относительный адрес начала группы свободных ячеек, nВ — количество членов в списке видов. Затем идет группа ячеек, в которых располагается первый член списка видов (одна ячейка), первый член подсписка марок (одна ячейка) и несколько ячеек с членами последовательного подсписка групп оборудования (по три члена в ячейке). Член списка видов состоит из пяти величин: шифра вида оборудования (ШВ), числа членов в подсписке марок (nМ), числа ячеек, занимаемых данным членом списка видов (s — в данном случае одна ячейка), общего количества оборудования данного вида (nОБ), плавающего относительного адреса следующего члена списка видов (ПОАСВ). Член подсписка марок состоит из шести величин: шифра марки (ШМ), признака точности оборудования (ПРТ — имеет 4 значения), признака импортного оборудования (ПРИ), количества членов в подсписке групп (nГ), общего количества оборудования данной марки (nОБ) и плавающего относительного адреса следующего члена списка марок (ПОАСМ). Концы списков видов и марок обозначаются признаком ПРК. = 1, размещаемым в знаковом разряде ячейки.

Каждый член последовательного списка групп оборудования занимает 12 разрядов и состоит из трех величин: номера возрастной группы (NВГ — может быть до 15 групп), характеристики физического износа (ФИ — может быть до 7 значений степени износа), количества оборудования данной группы (nОБ).

В общем шифре оборудования первые четыре цифры представляют шифр вида, а последние две цифры — шифр марки оборудования. При поиске сведений об определенном оборудовании сначала по шифру вида находят нужный член в списке видов, а затем по шифру марки — нужный член в ответвляющемся подсписке марок. После этого путем последовательного перебора просматривают (если нужно) возрастные группы.

Исключение членов из списков видов и марок производится путем записи нулевых кодов на места соответствующих шифров (ШВ = 0, ШМ = 0). Запись новых членов в эти списки осуществляется либо на места исключенных членов, либо образованием нового члена в свободной области памяти и включением его в соответствующий цепной список (по общим правилам работы с цепными списками).

Новые члены последовательных списков групп ставятся на свободные места; при отсутствии свободных мест данный член списка марок исключается полностью (весь список групп) и включается заново с размещением его в свободной области. При исчерпании всех свободных ячеек в участке памяти, отведенном под запись структуры Т3, производится ее уплотнение путем исключения свободных ячеек в составе списков с помощью специальной программы. Если это не поможет, то вся структура Т3 переписывается на новое место, где ей отводится больший участок памяти. Общий вид структурного дерева № 1, включающего в себя последовательные списки Т1 и Т2 записи Т4 и списковые структуры Т3, показан на рис. 29.

Структурное дерево № 2 содержит информацию о классификации оборудования, его технических характеристиках, а также сводную информацию по видам оборудования независимо от его принадлежности к тем или иным предприятиям.

Общая схема структурного дерева № 2 показана на рис. 30. В этом дереве объединены записи типов Т9 и Т12 и списковые структуры Т10 и Т11. Каждая запись Т9 представляет собой полное наименование класса, группы или подгруппы оборудования; эти записи используются при выдаче на печать требуемой информации в словесной форме (а не в виде шифров оборудования). Запись Т12 содержит сводную информацию по одному виду оборудования.

Списковая структура Т10 (рис. 34) состоит из последовательных списков четырех уровней, соответствующих делению оборудования на классы, группы, подгруппы и виды. Каждый из этих списков имеет заголовок, в котором указывается количество членов в списке (nЧ), количество оставшихся резервных позиций в данном разделе классификатора (nР). В списках классов, групп и подгрупп в ячейках, следующих за заголовками, располагаются члены списков (по одному в ячейке); каждый член списка содержит: адрес соответствующей записи Т9 на магнитной ленте (АТ9), количество ячеек в этой записи (КТ9) и плавающий относительный адрес начала ответвляющегося

подсписка (ПОАСПП).

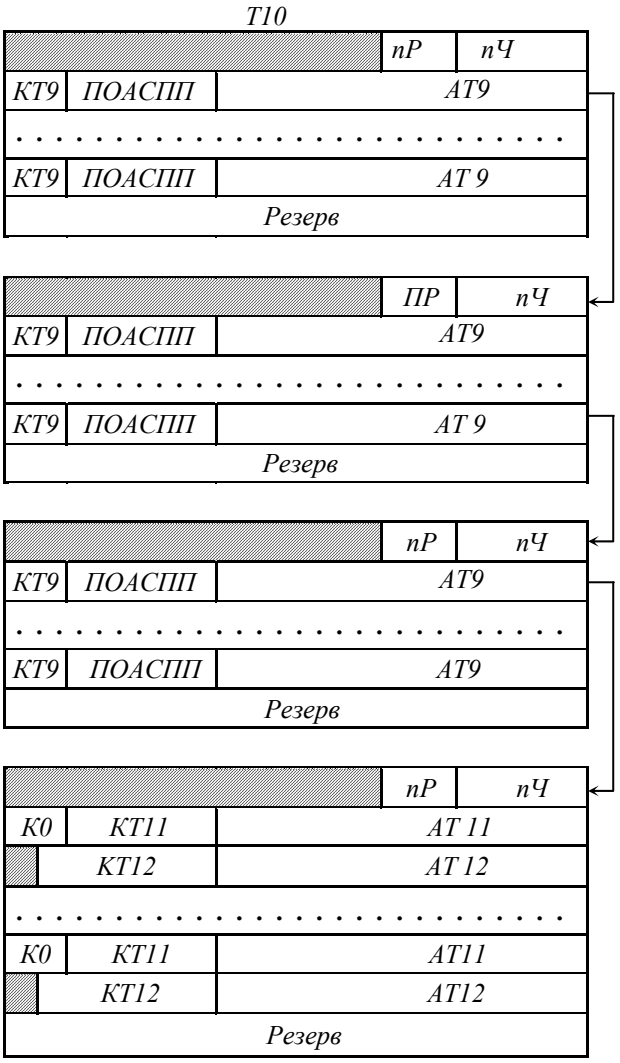


Рис. 34. Списковая структура Т10.

После каждого списка оставляется группа резервных ячеек. Каждый член в списках видов располагается в двух ячейках. В первой из них находятся: адрес на МЛ списковой структуры Т11, соответствующей этому виду (АТ11), количество ячеек в этой структуре (КТ11) и количество ячеек, занимаемых в этой структуре Т11 наименованием вида и фиксаторами узлов (КО). Во второй ячейке указывается адрес на МЛ записи Т12, содержащей сводную информацию по данному виду оборудования (АТ 12), и количество ячеек в этой записи (КТ12).

Адреса записей Т9 для заданного шифра класса, группы или подгруппы могут быть вычислены по следующим формулам:

Для класса: АКЛ = НАТ10 + N КЛ

Для группы: АГР = АКЛ + ПОАСПП.⌊ АКЛ ⌋ + N ГР

Для подгруппы: АПГР =АГР + ПОАСПП. ⌊АГР⌋ + + NПГР

Для вида: АВ = АПГР + ПОАСПП. ⌊АПГР⌋ + + 2NВ — 1

где NКЛ, NГР, NПГР, NВ — номера по классификатору класса, группы, подгруппы и вида оборудования соответственно. Списковая структура Т11 содержит словесное наименование данного вида оборудования, а также информацию о марках, входящих в этот вид, и их технических характеристиках.

Схема этой структуры показана на рис. 35. Первая ячейка играет роль заголовка; она содержит шифр вида (ШВ), используемый для контроля, количество марок, входящих в вид (nM), количество резервных позиций марок (nP), количество ячеек, занимаемых наименованием вида (s), и количество ячеек, занимаемых ассоциативным узлом каждой марки (m).

Затем идет группа ячеек (s), в которых записано словесное наименование вида оборудования. После этого располагается группа ячеек — фиксаторов списков, соединяющих узлы марок с одинаковыми техническими характеристиками. Конец этой группы указывается значением ПРК = 1.

При построении данной списковой структуры принято допущение, что каждый вид оборудования может иметь не более четырех основных технических характеристик (ТХ), например: мощность, диаметр стола и т. д. Каждая из характеристик имеет заранее установленное количество диапазонов возможных значений (градаций). Марки оборудования с характеристиками, лежащими в пределах одних и тех же градаций, считаются взаимозаменяемыми.

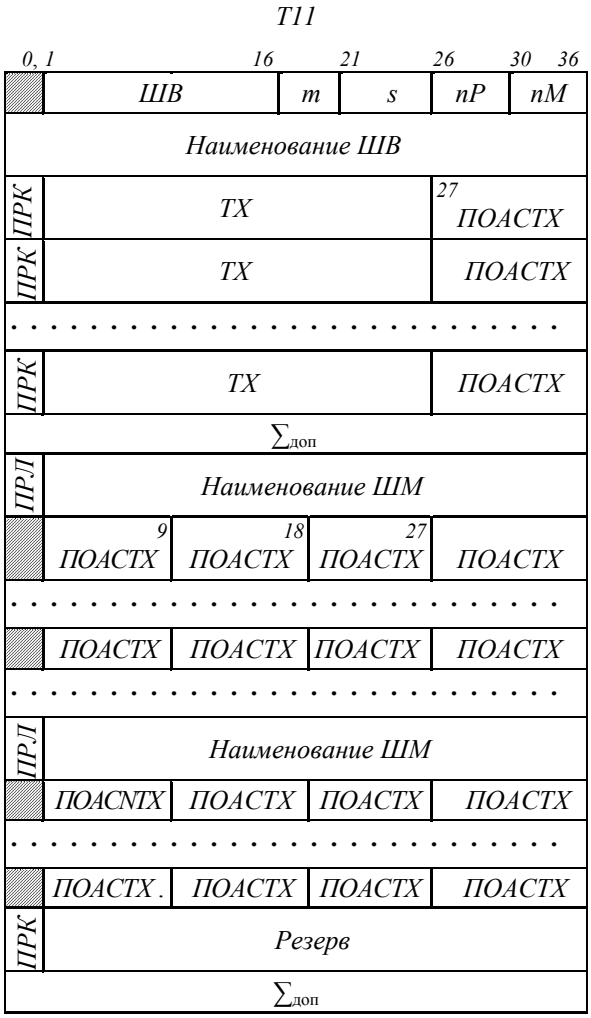


Рис. 35. Списковая структура Т11.

Фиксаторы цепных списков и ассоциативные узлы марок строятся для отдельных градаций технических характеристик. Устанавливается жесткий порядок размещения фиксаторов цепных списков и списковых слов (в данном случае списковое слово состоит только из плавающего относительного адреса связи — ПОАСТХ) в ассоциативных узлах. Каждый фиксатор занимает одну ячейку. Первый фиксатор соответствует первой градации 1-й характеристики; в этой ячейке записывается максимальное значение характеристики для этой градации (ТХ) и плавающий относительный адрес связи (ПОАСТХ), указывающий ячейку в ассоциативном узле некоторой марки с 1-й характеристикой, соответствующей этому диапазону.

Следующим идет фиксатор цепного списка для первой градации 2-й характеристики, затем — для первой градации 3-й характеристики и наконец — для первой градации 4-й характеристики. После этого идут 4 фиксатора для вторых градаций 4 характеристик, затем — для третьих градаций этих характеристик и т. д., пока не исчерпается максимально возможное число градаций каждой характеристики. Некоторые характеристики могут иметь меньшее число градаций, в этих случаях соответствующие ячейки фиксаторов не используются.

Некоторые виды и марки оборудования могут иметь меньше 4 характеристик; в этих случаях остаются неиспользованными ячейки в массиве фиксаторов. За массивом фиксаторов идут ассоциативные узлы марок. Каждой марке соответствует один узел; он состоит из словесного наименования марки (с признаком ПРЛ, показывающим, русскими или латинскими буквами записано это наименование) и группы ячеек со списковыми словами. В каждой ячейке ассоциативного узла данной марки содержится четыре списковых слова (ПОАСТХ), порядок размещения которых в ячейке

слева направо соответствует четырем характеристикам (с 1-й по 4-ю).

Если какая-либо градация какой-либо из 4 характеристик не используется (или вообще какая-либо из 4 характеристик отсутствует), то на поле, соответствующем этому цепному списку, ставятся нули. Поля списковых слов в ячейках жестко закреплены за цепными списками отдельных градаций 4-х характеристик. Если какой-либо цепной список заканчивается, то на месте его ПОАСТХ в последнем узле ставится код КС (все единицы). Следует заметить, что под наименование марки и под весь ассоциативный узел любой марки отводится одинаковое число ячеек. Для контроля в списковой структуре Т11 имеются две контрольные суммы: одна для заголовка, наименования вида и массива фиксаторов, а вторая—для всей структуры Т11 вместе с резервными ячейками.

Возможны два варианта поиска в структуре Т11:

1. По заданному шифру оборудования (ШО) находится наименование марки. Для этого вычисляется адрес ассоциативного узла марки по следующему соотношению:

$$АМ = НАТ11 + КО + m \times (ШМ — 1)$$

где НАТ11 — начальный адрес структуры Т11, КО — количество ячеек, занимаемых наименованием вида и массивом фиксаторов (это число указано в структуре Т10), ШМ — шифр марки, представляющий собой порядковый номер марки внутри вида; в этом же порядке должны располагаться ассоциативные узлы марок внутри структуры Т11.

2. По заданному набору значений 4-х (или меньше) характеристик находится марка (марки) оборудования, обладающая этими характеристиками.

Для этого путем просмотра массива фиксаторов и сравнения заданных значений характеристик с их граничными значениями для каждой градации находятся фиксаторы цепных списков, соответствующих требуемым градациям заданных характеристик. Затем по номерам найденных фиксаторов определяются места полей в ассоциативных узлах для списковых слов, соответствующих этим фиксаторам (для этого порядковый номер фиксатора делится на 4). После этого начинается поиск путем прослеживания одного цепного списка и проверки в каждом ассоциативном узле наличия отличных от нуля значений ПОАСТХ для остальных характеристик в определенных для них полях узлов. Если эти ПОАСТХ отличны от нуля и не равны коду конца списка, то проверяемая марка удовлетворяет условиям поиска. Если хотя бы по одной из заданных характеристик встретится код конца списка, поиск прекращается.

Записи Т12 служат для получения сводных данных по видам оборудования в масштабе отрасли промышленности (или главного управления). Все эти данные, вообще говоря, можно было бы получить путем просмотра ранее описанных списковых структур, в которых оборудование распределено по предприятиям, и выполнения сводных расчетов. Однако этот путь требует большего времени, и для наиболее часто запрашиваемых сводных показателей удобно иметь заранее заготовленные ответы, сгруппированные в записи Т12.

Вообще говоря, состав данных в записях Т12 может быть произвольным; для описываемой системы учета оборудования в эти записи включается несколько групп данных, расположенных в записи отдельными подмассивами. Одну группу образуют данные об излишнем оборудовании с указанием его марок, состояния, количества, стоимости и шифра заводов, где оно находится. В следующую группу входят данные о неустановленном оборудовании с указанием аналогичных сведений.

Отдельную группу образуют данные об отказах предприятий от заказанного оборудования.

Таким образом, с помощью второго структурного дерева осуществляется поиск и выдача данных по техническим характеристикам оборудования, его наименованиям и разнообразным сводным данным. Следует отметить широко развитые в описываемой ФИПС методы логического контроля: с помощью контрольных сумм для всех массивов и подмассивов, повторных проверок шифров оборудования (как правило, совпадающих с аргументами обращения к соответствующим таблицам), дополнительных величин, указывающих количество элементов в списках массивах и т. д., которые совместно с признаками концов фиксируют концы этих структур.

Основные алгоритмы ФИПС. Работа любой ФИПС включает в себя два режима:

1) накопление данных;

2) выдача данных.

Выдача данных ФИПС может происходить в трех видах:

1) в качестве ответов на вопросы;

2) в качестве стандартных сообщений выдаваемых в заранее определенные моменты времени;

3) в качестве стандартных сообщений, выдаваемых при выполнении заранее установленных условий

накопления информации.

Наиболее распространенным режимом является режим ответов ФИПС на вопросы человека. Этот режим мы и будем в основном рассматривать. Алгоритм такого процесса включает в себя

а) прием и расшифровку вопроса;

б) поиск и подготовку данных ответа;

в) формирование и выдачу ответа.

Основой такого процесса является стандартизация формы вопросов и ответов и использование машинного словаря терминов (дескрипторов). В качестве такого словаря в рассматриваемой ФИПС используется классификатор объектов (оборудования, предприятий) и их характеристик. Определяется множество допустимых вопросов, которые делятся на подмножества (типы вопросов — ТВ)

В зависимости от типа записей, в которых находятся искомые сведения Т3, Т4, Т12.

Стандартная форма вопроса предусматривает четыре составные части:

Ш3—шифр предприятия, по которому запрашиваются сведения;

ШО — шифр интересующего оборудования;

ОГР — ограничители, т. е. признаки, уточняющие вопрос (возраст, стоимость, состояние, сменность и т. д.). Ограничители должны обязательно входить в состав данных той записи, к которой относится вопрос.

ТВ = N — тип вопроса, указываемый его номером.

Допускаются вопросы, в которых один или несколько элементов, кроме типа вопроса, отсутствуют (прочеркнуты). В этих случаях ответы формируются по всем возможным значениям прочеркнутого элемента, например, по всем заводам, если отсутствует Ш3, или по всем видам оборудования, если не указан ШО.

В системе предусмотрена групповая последовательная обработка вопросов, что обеспечивается локальной для алгоритма поиска программой-диспетчером. Это программа регулирует очередь вопросов, переносит очередной вопрос на рабочее поле МОЗУ, включает в работу подпрограмму, соответствующую типу вопроса.

Сам локальный диспетчер включается в работу главным диспетчером ФИПС — программой, имеющей шифр А1, к которой передается управление после окончания работы любой из частных программ ФИПС. Локальный диспетчер проверяет правильность (допустимость) вопроса и в случае его некорректности печатает отказ.

Кроме локального диспетчера (шифр А2) в состав общей программы алгоритма поиска входят следующие подпрограммы: две подпрограммы ответов по дереву № 1, имеющие номера 01 и 02; две подпрограммы ответов по дереву № 2, имеющие номера 03 и 04; пять общих стандартных блоков подпрограмм, используемых указанными подпрограммами. Эти блоки имеют шифры СБАТI—2, СБАТ3, СБАТ10, СБШО, СБШ3.

При включении в работу СБАТ1—2, используя списки Т1 и Т2, определяет по заданному шифру предприятия (Ш3) адреса соответствующей записи Т4 и списковой структуры Т3.

СБАТ3 работает со списковой структурой Т3 и определяет по заданному шифру оборудования (ШО = ШВ + ШМ) адрес соответствующего элемента в структуре Т3. Сначала проверяется наличие в МОЗУ списковой структуры Т3 для заданного предприятия и при ее отсутствии производится перепись ее с МЛ. Затем идет двухступенчатый поиск — по списку видов оборудования и по ответвляющемуся подсписку марок оборудования. В процессе поиска ведется контроль правильности переходов по цепным спискам. В случае сбоя поиск прекращается, а на АЦПУ печатаются сведения о причине сбоя. Подобный контроль имеется во всех стандартных блоках и подпрограммах ФИПС. СБАТ10 осуществляет поиск в списковой структуре Т10 и по заданному шифру оборудования находит адрес* класса, группы, подгруппы или вида оборудования; сам адрес и его содержимое записываются в стандартные ячейки.

СБШО использует результаты работы СБАТ10 и осуществляет обращение либо к записям Т9, содержащим наименования подразделений оборудования, либо к списковой структуре Т11, из которой выбираются необходимые данные по видам и маркам оборудования.

Подпрограмма 01 обеспечивает выдачу ответов по общезаводской информации, хранящейся в записях Т4. Очередной вопрос должен находиться в фиксированном буфере из 3-х ячеек. В вопросе должно быть указано: тип вопроса (ТВ = 1), признак автора вопроса (человек или программа), шифр предприятия (Ш3) и номера показателей в записи Т4, которые должны быть выданы. Для определения расположения показателей в записях Т4 по их номерам служит специальная справочная таблица Т5, в

которой каждому номеру показателя соответствует одна ячейка. В этой ячейке указывается относительный номер ячейки в записи Т4 и номера первого и последнего разрядов в этой ячейке, соответствующих нужному показателю. В этой же таблице Т5 содержится полное наименование каждого показателя. В подпрограмме 01 предусматривается, что вопрос может содержать до шести номеров показателей. Эта программа может выдавать данные не только по одному предприятию, но и по главному управлению или министерству в целом. В процессе работы проверяется правильность (допустимость) заданных в вопросе Ш3 и показателей. Подпрограмма 01 занимает 380 ячеек ЭВМ «Минск-22». Приводимые ниже размеры программ относятся к этой машине.

Подпрограмма 02 служит для выдачи информации из списковых структур Т3, содержащих подробные сведения об оборудовании каждого предприятия. Вопросы, обрабатываемые этой подпрограммой, должны иметь стандартную форму и содержать тип вопроса, шифр оборудования и перечень номеров показателей с указанием по каждому показателю пределов изменения его значений. Программа выдает суммарные данные по оборудованию, характеристики которого лежат в заданных в вопросе пределах. Для поиска нужной структуры Т3 по шифру Ш3 данная подпрограмма использует стандартные блоки СБАТ1—2 и СБАТ3. Для контроля заданных аргументов (ШО, Ш3, пределов значений признаков) используются также стандартные блоки. Для выдачи данных по группам предприятий в подпрограммах 01 и 02 предусматриваются соответствующие циклы. В функции подпрограммы 02, так же как и подпрограммы 01, входит печать ответов на АЦПУ, а также сообщений о сбоях. Подпрограмма 02 занимает 520 ячеек.

Подпрограмма 03 служит для выдачи ответов по записям Т12, каждая из которых относится к определенному виду оборудования. Если вопрос предусматривает выдачу сведений по более крупному разделу классификатора (классу, группе или подгруппе), то подпрограмма 03 осуществляет циклический просмотр всех видов оборудования (записей Т12), входящих в заданный в вопросе раздел. Из каждой записи Т12 выбираются данные, относящиеся к заданной группе (неустановленное оборудование, резервное и др.), в соответствии с теми условиями, которые указаны в вопросе. Программа 03 занимает 710 ячеек.

Подпрограмма 04 осуществляет поиск сведений об оборудовании, заданном его характеристиками. Она использует для контроля те же стандартные блоки, для поиска в структурах Т3 — подпрограмму 02, а в записях Т12 — подпрограмму 03. В подпрограмме 04 предусмотрены циклы по заданным Ш3 предприятий и по заданным ШО видам оборудования. Отобранные сведения выдаются на печать. Подпрограмма 04 занимает 304 ячейки.

Рассмотрим вторую группу подпрограмм ФИПС, образующих программу корректировки списков, списковых структур и записей ФИПС. Необходимость корректировки обусловлена постоянными изменениями состава оборудования на предприятиях, его характеристик, изменениями классификатора оборудования, а также изменениями состава и характеристик самих предприятий. Эта программа имеет важное значение для ФИПС с динамическим характером информации. Указанная программа состоит из локального диспетчера А3 и управляемых им 5 подпрограмм: подпрограмм корректировки К1, К2, К3, К4, подпрограммы уплотнения списков К5, и ряда стандартных блоков. Локальный диспетчер А3 вводит информацию об изменениях с перфоносителей (с контролем), определяет по типу введенных данных соответствующую подпрограмму корректировки и передает ей управление. После окончания работы корректирующей подпрограммы управление передается локальному диспетчеру А3, который может передать управление другой корректирующей подпрограмме или главному диспетчеру А1.

Подпрограмма К1 вводит изменения в информацию, хранимую в записях Т4. Данные об изменениях подготавливаются в стандартной форме и содержат ряд записей, каждая из которых относится к одному предприятию. Запись состоит из шифра предприятия (Ш3) и последовательности пар данных, включающих в себя номер изменяемого показателя в записи Т4 и его новое значение. Положение изменяемого показателя в ячейках записи Т4 определяется подпрограммой К1 с помощью упомянутой раньше справочной таблицы Т5.

В соответствии с заданным числом записей изменений подпрограмма К1 вносит их во все записи Т4, имеющие указанные Ш3.

Подпрограмма К2 вносит изменения в структуры Т3, дополняя их новыми элементами или вычеркивая ненужные. Форма задания изменений также представляет собой последовательность записей изменений, относящихся к одному предприятию. Запись начинается шифром предприятия Ш3, за которым указывается шифр оборудования ШО с признаком, определяющим, в какую сторону должно быть сделано изменение: увеличение или уменьшение. Затем идет ряд ячеек, в каждой из

которых указано два числа: номер изменяемого показателя и величина изменения.

Подпрограмма К2 по шифру Ш3 находит нужную структуру Т3 и в ней по шифру ШО отыскивает нужное гнездо. Если такое гнездо найдено и требуется произвести увеличение показателей, то производится прибавление заданных значений к определенным показателям. Если требуемое гнездо ШО не найдено, то формируется новый элемент в структуре Т3 либо в качестве новой марки в соответствующем виде оборудования, либо в качестве нового вида оборудования, если он отсутствует. Новый элемент записывается либо на место вычеркнутой марки, либо в резервные ячейки. При отсутствии резервных ячеек вызывается уплотняющая подпрограмма К5, которая пополняет резерв ячеек. Если требуется уменьшить показатель и соответствующий элемент структуры Т3 найден, то производится вычитание и разность записывается в качестве нового значения соответствующего показателя. При этом может случиться так, что вычитаемое будет больше уменьшаемого и разность получится отрицательной. Тогда данный элемент структуры Т3 вычеркивается из этой структуры и делается попытка найти в структуре Т3 другой аналогичный элемент и его уменьшить на полученную разность. Если же вообще не будет найден в выбранной структуре Т3 элемент, соответствующий заданному ШО и другим признакам, то программа печатает отказ по данному изменению.

После внесения всех изменений в одну структуру Т3 (т. е. для одного предприятия) подпрограмма К2 переходит к следующей структуре Т3. Эта подпрограмма занимает 475 ячеек.

Подпрограмма К3 служит для корректировки записей Т12. Исходная информация об изменениях задается в виде массива записей изменений, относящихся к различным записям Т12. Этот массив предварительно упорядочивается по возрастанию ШО с целью сокращения времени обработки записей Т12. Записи Т12 поочередно вызываются в МОЗУ, в них находятся нужные разделы оборудования, в которых по заданным Ш3 и ШМ определяются элементы, подлежащие корректировке. При этом проверяется совпадение и ряда других признаков.

Если нужный элемент найден, то производится увеличение или уменьшение требуемых показателей аналогично тому, как это делается подпрограммой К2. Если искомый элемент не найден, то либо печатается отказ от корректировки (если корректировка заключается в уменьшении показателей), либо формируется новый элемент. Место для него определяется так же, как это описано для подпрограммы К2. Подпрограмма К3 занимает 512 ячеек.

Подпрограмма К4 служит для корректировки классификатора оборудования, представленного списками Т9, Т10 и Т1l. Исходная информация содержит шифр оборудования ШО, за которым указывается его наименование и за ним характеристики оборудования $ТХ_1$ . . . $ТХ_n$. Если ШО = 0, требуется корректировать записи Т9 и структуру Т10. Если Ш $\neq$ 0, то требуется корректировать структуру Т11. Если ШО $\neq$ 0, но ШМ =0, то это значит, что требуется внести в классификатор данные о новом классе, группе, подгруппе или виде оборудования, что осуществляется путем занесения новой записи Т9 на свободное место на МЛ и соответствующей корректировки структуры Т10. Если ШМ $\neq$ 0, то это значит, что классификатор нужно пополнить новой маркой оборудования. Для этого сначала рассчитывается адрес размещения новой марки в структуре Т11, затем записывается по этому адресу название марки и она включается во все цепные списки, соответствующие значениям ее характеристик (ТХ). Подпрограмма К4 может только пополнять классификатор, но не исключать из него какие-либо разделы. Эта подпрограмма занимает 267 ячеек.

Подпрограмма К5 обеспечивает уплотнение структур Т3 и записей Т12 за счет исключения ненужных элементов. Эта подпрограмма включается в работу подпрограммами К2 и К3 при отсутствии у них резервных ячеек. Остальные структуры уплотнять не требуется в связи с принятым в них порядком заполнения свободных ячеек.

Подпрограмма К5 последовательно просматривает структуру Т3 (если она была включена подпрограммой К2) и воспроизводит ее на новом месте в МОЗУ. При этом автоматически обходятся исключенные члены цепных списков и все элементы ответвляющихся подсписков марок располагаются подряд за своими фиксаторами, которые являются членами списка видов.

После воспроизведения одной структуры Т3 оценивается достигнутая экономия ячеек. Если такая экономия получена и резервная группа ячеек образована, то структура Т3 возвращается на МЛ. Если же уплотнение не произошло, то увеличивается резервное поле данной структуры и она записывается на МЛ на новое место и вносятся изменения в список Т2. Таким же образом происходит уплотнение записей Т12 при вызове подпрограммы К5 подпрограммой К3. Отличие состоит в том, что просматриваются и воспроизводятся списки разделов оборудования с пропусками пустых элементов. Подпрограмма К5 занимает 385 ячеек.

Третья группа подпрограмм ФИПС образует программу первоначальной организации списковых структур. Эта программа построена также по блочному принципу; она имеет свой локальный диспетчер (подпрограмму А4), подпрограммы В1 и В2, формирующие структурное дерево №1 (Т1, Т2, Т3, Т4), и подпрограммы В3 и В4, формирующие структурное дерево №2 (Т9, Т10, Т1l, Т12). Эти подпрограммы используют ряд стандартных блоков.

Подпрограмма-диспетчер второго уровня А4 получает управление от подпрограммы-диспетчера первого уровня А1, когда нет заявок на включение других локальных диспетчеров (А2 и А3). Она вводит информацию с перфоленты (с контролем) и по признакам введенных данных определяет, какую из формирующих подпрограмм включить в работу. Исходная информация для первоначального построения списковых структур задается в такой же стандартной форме, как и для корректировки этих структур, и выбор нужной формирующей подпрограммы (В1, В2, В3, В4) подпрограмма-диспетчер А4 производит по тем же правилам, что и подпрограмма А3 при выборе корректирующих подпрограмм. Для определения адреса свободной области МЛ используется стандартный блок СБАСО, который учитывает и необходимость перехода на другой лентопротяжный механизм, если для вводимого массива не хватает места на данной МЛ.

Для подсчета контрольных сумм и контрольной печати массивов используется отдельный стандартный блок. Расположение данных в записях Т4, Т9, Т12 и в структурах Т3, Т11 задается с помощью соответствующих справочных таблиц, рассмотренных раньше. Изменяя эти таблицы, можно изменять состав и количество данных в записях и структурах. Однако каждая такая таблица должна быть постоянной в течение всего времени работы данной ФИПС.

Ввод исходной информации в данную ФИПС всегда осуществляется в алфавитно-цифровом десятичном коде и значение каждого показателя при вводе записывается в отдельную ячейку.

Подпрограмма В1 формирует записи Т4. Считывается с МЛ список Т1, в который на место НАТ4 записывается начальный адрес записей Т4 для данного главного управления. Затем показатели каждого предприятия заносятся в запись Т4, заданную справочной таблицей Т5. При этом осуществляется логический контроль значений показателей. Подпрограмма В1 занимает 124 ячейки.

Подпрограмма В2 формирует списки Т2 и Т1 и структуры Т3. Ввод исходной информации при этом должен производиться порциями — по предприятиям, и за один вызов подпрограмма В3 формирует одну структуру Т3. Исходная информация по одному предприятию сортируется по возрастанию шифров оборудования (ШО) и данные, относящиеся к одному ШО, объединяются в одну запись.

Затем в списке Т2 выделяются (если необходимо) резервные ячейки для данного главного управления и по шифру предприятия определяется адрес ячейки в списке Т2, куда записывается НАТ3. Формирование цепных списков видов и марок оборудования в Т3 производится путем циклического просмотра групп данных, относящихся к определенным ШО. Необходимые данные о видах и марках располагаются в ячейках с помощью справочных таблиц. После просмотра всех групп данных, расположенных по ШО, определяется число резервных ячеек в Т3 (примерно 25% от числа заполненных ячеек в Т3), рассчитывается контрольная сумма, определяется nСЯ и ПОАСЯ, которые записываются в заголовок Т3, а в ранее определенную ячейку списка Т2 записывается число ячеек в Т3 (КТ3).

После обработки данных по одному предприятию управление передается диспетчеру А1. Программа В2 занимает 320 ячеек.

Подпрограмма В3 формирует записи Т12 и записывает их адреса и размеры в структуру Т10. Эта подпрограмма должна работать после того, как сформированы структуры Т10. Расположение данных в записи Т12 задается справочной таблицей Т28, которая предусматривает три группы данных в Т12 согласно трем подразделениям учитываемого оборудования (излишнее, неустановленное, резервное).

Объем исходной информации не должен превышать 4096 ячеек; если требуется ввести больший массив, то его разбивают на части. Первая часть объемом 4096 ячеек вводится и обрабатывается подпрограммой первоначального формирования В3, а остальные части добавляются с помощью подпрограммы корректировки КЗ.

Аналогичный прием может применяться и в отношении других структур и записей. При этом следует иметь в виду, что подпрограммы формирования являются более эффективными по сравнению с подпрограммами корректировки в отношении затрат машинного времени и занимаемого объема памяти. Исходная информация должна быть предварительно рассортирована так, чтобы все марки одного вида оборудования находились в одной группе.

Подпрограмма В3 производит сортировку введенной информации по возрастанию показателей,

заданных в специальной справочной таблице. Такими показателями являются шифр оборудования (ШО), тип подразделения оборудования (ТИ), шифр предприятия (Ш3), номер возрастной группы (NВГ), физическое состояние (ФИ) и др. В этой таблице указывается сначала общее число показателей, по которым должна проводиться сортировка, а затем последовательно указываются показатели в том порядке, в котором должна идти по ним сортировка. О каждом показателе задаются такие сведения: номер ячейки в исходном сообщении, содержащей этот показатель, номер ячейки с константой, необходимой для выделения этого показателя, константа сдвига этого показателя после выделения.

После сортировки производится объединение частей сообщения с одинаковыми значениями заданных показателей в единое сообщение с суммарными количественными значениями. Затем подсчитывается количество элементов в соответствующем подразделении оборудования (ТИ) и записывается в фиксатор данного подразделения.

В конце списка ТИ ставится признак КС и определяется адрес перехода к другому подразделению. После формирования списков всех подразделений ТИ определяется число резервных ячеек, которое заносится в фиксатор в заголовке записи Т12, запись Т12 записывается на МЛ, а ее адрес (АТ12) и размер (КТ12) — в структуру Т10 по адресу, определяемому ШВ. Подпрограмма В3 занимает 380 ячеек.

Подпрограмма В4 служит для ввода и формирования классификатора оборудования, представляемого записями Т9 и структурами Т10 и Т11. Исходная информация должна быть предварительно упорядочена по видам, группам и классам оборудования и вводится порциями не более 4096 ячеек. Сначала располагаются данные о классах (шифр класса и его наименование), затем идут такие же данные о всех группах, входящих в состав первого класса, затем данные о всех подгруппах первой группы, затем данные о видах первой подгруппы, объединенные с данными о марках каждого вида. Данные о видах и марках содержат кроме шифра и наименования и их характеристики.

После данных о всех видах (с марками) одной подгруппы располагаются данные о видах (с марками) другой подгруппы и т. д., пока не закончится полная группа. Затем идут данные о следующей группе оборудования в таком же порядке. После окончания одного класса аналогичным образом вводятся данные по следующему классу и т. д.

При первоначальном формировании классификатора какие-либо из его категорий могут отсутствовать; в дальнейшем они могут быть дополнительно введены с помощью подпрограммы корректировки К4. Подпрограмма В4 поочередно просматривает введенные данные и для встречающихся классов, групп и подгрупп формирует записи Т9 с фиксацией их в структуре Т10, а для данных о видах и марках формирует структуры Т11. Первая ветвь подпрограммы В4 служит для обработки данных о классах, группах и подгруппах. Если очередное сообщение относится к имеющемуся уже в Т10 подсписку, то оно включается в этот подсписок. Определяется по шифру категории адрес ячейки в Т10, куда должен быть записан адрес АТ9, по которому будет записана сформированная запись Т9. Если очередное сообщение является первым в своем подсписке в Т10, то сначала оформляется предыдущий подсписок (определяется и фиксируется для него резерв, примерно равный 20%, ставятся адреса связи к следующему подсписку той же категории). В списковое слово подсписка более высокого уровня записывается адрес связи для данного ответвляющего подсписка. Вторая ветвь подпрограммы В4 служит для обработки данных о видах и марках и формирования по ним структур Т11.

Если очередное сообщение о виде является первым в своем подсписке, то сначала заканчивается оформление подсписка, к которому относилось предыдущее сообщение, а затем образуется новый подсписок. Для этого рассчитывается ПОАСПП, который заносится в списковое слово вышестоящего списка, в фиксаторе подсписка в Т10 увеличивается количество видов, и в ячейку, соответствующую данному виду, заносится длина сообщения КО и его адрес на МЛ.

Подсчитывается контрольная сумма, которая заносится в конец сообщения, записывается длина наименования вида s и все сообщение переписывается в рабочие ячейки для формирования структуры Т11. После сообщения о виде должны следовать данные о марках. Наименование марки переносится в соответствующие ячейки, в которых формируется структура Т11 и производится включение данной марки в цепные списки, соответствующие ее характеристикам (ТХ). Если требуемый список отсутствует, то в ячейке, отведенной под фиксатор такого списка, записывается адрес связи, отсылающий к данной марке (к соответствующей ячейке в ней), в которой на месте адреса связи данной характеристики ставится признак КС. При включении каждой новой марки в фиксаторе вида увеличивается на единицу количество марок. После обработки введенных сообщений и включения их в структуру Т10 подсчитывается ее контрольная сумма и структура Т10 переписывается на МЛ.

Рассмотренные подпрограммы трех типов (поиска, корректировки и формирования) обеспечивают возможность построения информационных массивов ФИПС и поддержание их в рабочем состоянии, а также выполнение основной функции ФИПС — поиск данных и выдачу ответов на вопросы о составе и свойствах оборудования и его распределении по предприятиям.

Указанная фактографическая информационно-поисковая система может быть использована для накопления и поиска фактографических данных о других объектах, например о научных центрах, темах работ, методах исследований и т. п. В сочетании с документальной (библиографической) ИПС и специализированной ИПС для учета кадров описанная ФИПС позволяет создавать интегрированные информационные системы различного вида.

20. Информационно-поисковая система (ИПС) «Сетка-5»

Данная ИПС предназначается для поиска документов по запросам; она построена на основе дескрипторного описания документов. Поисковый массив записывается в порядке поступления документов на МЛ позонно. Каждому документу соответствует один ассоциативный узел, представляющий собой поисковый образ документа (ПОД).

Узел состоит из двух частей: заголовка и набора списковых слов, образующих дескрипторную часть узла. Количество списковых слов соответствует количеству дескрипторов в ПОД и не превышает 20.

Каждое списковое слово размещается в одной ячейке и состоит из двух частей: адреса связи (24 двоичных разряда) и кода дескриптора (12 двоичных разрядов). Наличие в явном виде кодов дескрипторов в списковых словах позволяет производить поиск документов путем прослеживания только одного цепного списка, имеющего наименьшую длину, и обеспечивает достаточно надежный контроль адресных переходов. При этом в каждом узле прослеживаемого цепного списка производится проверка наличия остальных дескрипторов, имеющихся в запросе. При этом используются коды дескрипторов, записанные в списковых словах. Следует заметить, что, хотя для кодов дескрипторов принято 12 двоичных разрядов, в ИПС «Сетка-5» предусматривается возможность использовать всего до 2048 различных кодов дескрипторов и объем словаря дескрипторов в ИПС «Сетка-5» не превышает 2048. Это существенно упрощает все алгоритмы и программы, так как все таблицы словаря дескрипторов (частоты встречаемости, начальных адресов связи, конечных адресов связи) помещаются по одной в зону МЛ. Однако небольшой объем словаря ограничивает возможность системы. В ИПС «Сетка-5» поиск осуществляется одновременно по группе запросов, поэтому на поиск в массиве 1,0 тысяч документов, занимающих одну МЛ, затрачивается 4—5 *мин* машинного времени ЭВМ «Минск-22».

Выдача ответов может производиться в двух видах: сокращенном и полном. Сокращенные ответы представляют собой порядковые (инвентарные) номера документов в хранилище, которые печатаются на цифровом печатающем устройстве (БПМ).

Полные ответы содержат библиографические описания документов, выдаваемые в том виде, в котором они были предварительно записаны на отдельную магнитную ленту. Полные ответы выдаются на АЦПУ-128. Возможен также вариант совместной работы ЭВМ «Минск-22» и машины «Поиск», при котором ЭВМ выдает номера документов, являющихся кодами кадров для машины «Поиск», а последняя выдает библиографические описания или рефераты по заданным номерам документов.

Критерием поиска является полное вхождение дескрипторов запроса в поисковый образ документа. Поисковые образы документов в памяти машины «Минск-22» записываются следующим образом. В первой ячейке каждой зоны МЛ записывается полный инвентарный номер документа, ПОД которого записывается первым в данной зоне. Затем идут ПОДы, представляемые в виде ассоциативных узлов. ПОД каждого документа занимает группу последовательных ячеек, образующих узел. В первой ячейке узла хранится относительный инвентарный номер документа, представляющий собой разность между фактическим инвентарным номером и инвентарным номером первого документа зоны. Для самого первого документа зоны эта разность, естественно, будет равна нулю. Применение относительных инвентарных номеров документов уменьшает разрядность чисел, представляющих инвентарные номера документов. В первой же ячейке узла записывается адрес библиографического описания документа, хранящегося обычно на другой МЛ.

В следующей ячейке узла хранится адрес первой ячейки следующего узла, т. е. другого ПОДа. Он используется при последовательном просмотре всех ПОДов подряд, что бывает иногда необходимо для корректировки поискового массива. В последующих ячейках располагаются списковые слова, по

одному в ячейке. Каждое слово содержит код дескриптора (КД) — в младших 12 разрядах, и адрес связи — в старших 24 разрядах. В знаковых разрядах первой и последней ячейки узла указывается знак минус. В ячейках, соответствующих списковым словам, в знаковом разряде ставится плюс. Последняя ячейка каждого узла содержит сжатую запись (в виде позиционного кода) ряда типовых признаков документа, относящихся обычно к библиографическому описанию. К. таким дескрипторам относятся: вид документа (журнал, книга и др.), на каком языке написан, наличие перевода и т. д. Каждый подобный признак отмечается указанием единицы или нуля в постоянно закрепленном разряде последней ячейки узла (ПОДа).

Адреса связи представляют собой полные адреса ячеек на МЛ, т. е. содержат в себе номера шкафа НМЛ, лентопротяжного механизма, зоны МЛ и адрес ячейки в зоне. Адреса связи в одной зоне МЛ идут в возрастающем порядке. При переходах из одной зоны к другой зоне той же или другой МЛ адреса связи могут изменяться произвольно.

В ИПС «Сетка-5» возможно одновременное использование не более 16 МЛ, т. е. весь поисковый массив может содержать порядка 100—120 тысяч документов (если считать в среднем по 10—15 ячеек на ПОД). Каждый цепной список поискового массива соответствует одному дескриптору, и для каждого дескриптора в специальных таблицах записываются начальный адрес цепного списка (АСН), конечный адрес цепного списка (АСК) и частоты встречаемости этого дескриптора в документах и запросах. Как уже говорилось, ассоциативный узловой способ построения поискового массива, использованный в ИПС «Сетка-5», сочетает в себе преимущества прямого и инверсного способов организации поисковых массивов. Рассмотрим основные алгоритмы ИПС «Сетка-5»:

1) Контроль правильности перфорации кодов дескрипторов на перфоленту. Для ввода новых документов в ИПС на перфоленту набиваются для каждого ПОДа инвентарный номер документа (со знаком минус) и коды дескрипторов (со знаком плюс). Максимальная величина вводимого массива равна одной зоне МЛ.

Указанная программа выявляет пропуски признаков «запись», неправильную набивку признака «Код адресный» и превышение объема вводимого массива допустимых размеров. Выявленные ошибки выдаются на печать для повторной перфорации и исправления.

2) Формирование поискового массива после ввода массива новых ПОДов. По этой программе выполняются следующие действия:

а) добавление в таблицу длин цепных списков для соответствующих дескрипторов чисел, показывающих, сколько раз эти дескрипторы встречаются во вводимом массиве ПОДов. Тем самым длины цепных списков увеличиваются на количество документов, включаемых в эти списки при данном пополнении;

б) запись в пополняемую зону МЛ, содержащей поисковый массив, вводимых ПОДов и формирование последних позиционных строк ПОДов на основе библиографических дескрипторов, содержащихся во вводимых данных.

Выделение библиографических дескрипторов из общей массы дескрипторов производится автоматически на основе сравнения их кодов. При этом заранее устанавливается значение граничного (минимального) кода библиографического дескриптора;

в) включение новых ПОДов (вернее, соответствующих им ассоциативных узлов) в цепные списки соответствующих
дескрипторов. При помощи таблицы конечных адресов связи находятся узлы (и списковые слова в них), являющиеся последними в данных цепных списках. В этих списковых словах ставятся адреса связи, отсылающие к соответствующим списковым словам вновь введенных узлов, эти же адреса указываются и в таблице конечных адресов связи в качестве новых конечных адресов. Во вновь введенных узлах в качестве адресов связи указывается код КС (конец списка — все семерки). В случае, если для некоторых дескрипторов производится первое включение соответствующих списковых слов, то их адреса заносятся также и в таблицу начальных адресов связи (АСН). Указанные процессы формирования поискового массива усложняются необходимостью определения, в каких зонах МЛ находятся концы цепных списков, и многократной переписью этих зон в МОЗУ и обратно на МЛ.

3) Контроль правильности перфорации библиографических описаний на перфоленту. Эта программа используется для выявления ошибок перфорации с целью их исправления с помощью аппарата «СТ-2М». Программа выявляет ошибки следующих типов: обнаруживает вводимый массив, превышающий по своему объему место, предусмотренное для него в МОЗУ, обнаруживает библиографические описания, превышающие максимально допустимый размер, а также отсутствие некоторых служебных

признаков.

4) Запись библиографических описаний на МЛ. Библиографические описания документов представляют собой алфавитно-цифровые тексты, выдаваемые из машины с помощью АЦПУ. Адреса соответствующих библиографических описаний и их размеры (количество ячеек) определяются в результате поиска в поисковом массиве. Программа осуществляет следующие действия:

а) вводит с перфоленты библиографические описания, представленные в коде М-2;

б) перекодирует массив из кода М-2 в код АЦПУ;

в) определяет и выдает на БПМ информацию об адресах и размерах библиографических описаний. Эти данные заносятся также (отдельной программой) в заголовки узлов;

г) записывает на МЛ подготовленную зону с библиографическими описаниями.

5) Запись заголовка ассоциативного узла. Программа записывает в первые ячейки ассоциативных узлов, вновь включенных в поисковый массив, относительные инвентарные номера документов, адреса их библиографических описаний и размеры этих описаний. Кроме того, она формирует вспомогательную позиционную таблицу, содержащую полные инвентарные номера документов, соответствующих первым ассоциативным узлам в каждой зоне магнитной ленты. Эти инвентарные номера, как уже говорилось, находятся в первых ячейках зон магнитной ленты. Позиционная таблица инвентарных номеров документов необходима при вычеркивании устаревших документов для того, что бы по номеру документа определить, в какой зоне магнитной ленты находится его ассоциативный узел. При этом, естественно, предполагается, что инвентарные номера документов изменяются монотонно как внутри зон, так и при переходах между зонами МЛ. Так как при работе ИПС запись инвентарных номеров документов и запись сведений об их библиографических описаниях могут быть разделены во времени, в данной программе предусматривается три варианта работы:

а) запись инвентарных номеров;

б) запись сведений о библиографических описаниях;

в) запись того и другого одновременно.

Включение в работу того или иного варианта программы
осуществляется с помощью ключей №1 и №2 на пульте управления ЭВМ «Минск-22».

6) Поиск документов по запросам в поисковом массиве. Поиск может вестись одновременно по группе запросов, содержащей не более 36 запросов, причем общее число' дескрипторов в группе запросов не должно превышать 128. В запросах могут быть как поисковые (тематические) дескрипторы, так и библиографические (стандартные), причем последние вводятся в ЭВМ со знаком минус.

Процесс поиска состоит из следующих этапов. Сначала производится ввод и сортировка дескрипторов запросов, регистрация их в таблице частоты встречаемости дескрипторов в запросах, перепись с МЛ в МОЗУ таблицы длин цепных списков и выбор для каждого запроса дескриптора с наименьшей длиной цепного списка. Запросы, у которых наименьшая длина цепного списка оказалась равной нулю, аннулируются. При этом на печать выдаются номера таких запросов для переиндексации. В процессе сортировки дескрипторов библиографический дескриптор в каждом запросе ставится в конец группы дескрипторов этого запроса. На этом же этапе вводится в МОЗУ с МЛ сначала таблица начальных адресов цепных списков, а затем таблица конечных адресов цепных списков и для дескрипторов с минимальными длинами списков определяются их начальные адреса. Производится проверка, не является ли выбранный начальный адрес в запросе меньше какого-либо конечного адреса цепных списков всех дескрипторов запроса, и если это так, то запрос аннулируется. В противном случае определяется для каждого запроса наименьший конечный адрес цепных списков для дескрипторов запроса. Этот адрес будет ограничивать поиск для соответствующего запроса по длине МЛ. Затем производится непосредственный поиск документов в поисковом массиве путем прослеживания цепных списков, имеющих минимальное число членов и ограниченных наименьшим для всех дескрипторов запроса конечным адресом.

Среди отобранных начальных адресов цепных списков всех запросов выбирается наименьший и в МОЗУ переписывается соответствующая ему зона МЛ. В этой зоне прослеживается цепной список с данным начальным адресом, а также другие списки, имеющие начальные адреса, относящиеся к этой зоне.

При прослеживании списков производится проверка каждого члена списка (узла) на полное вхождение в ПОД всех дескрипторов соответствующего запроса. Номера документов (и сведения об их библиографических описаниях), удовлетворяющих этому условию, записываются в таблицу ответов; при этом адрес связи из узла, соответствующего найденному документу, заменяет ранее находившийся

адрес связи в таблице начальных адресов связи. Процесс поиска заканчивается по каждому запросу по достижении граничного конечного адреса списка. Затем выдаются на печать результаты поиска: либо только номера найденных документов (на БПМ), либо их номера и библиографические описания (на АЦПУ). При выдаче ответов печатаются также номера запросов, к которым они относятся.

В ИПС «Сетка-5» предусмотрена возможность проведения поиска не во всем поисковом массиве, а в его части, включенной за какой-то последний период, что можно использовать при избирательном распределении информации. Для этого необходимо вместо начальных адресов списков использовать те значения конечных адресов списков, которые они имели перед пополнением поискового массива.

Кроме перечисленных основных алгоритмов в ИПС «Сетка-5» имеется еще ряд вспомогательных алгоритмов, служащих, например, для вычеркивания в поисковом массиве устаревших документов (без перенумерации и с перенумерацией остающихся), корректировки массива библиографических описаний и др.

Описанная система «Сетка-5» с определенными доработками была использована для построения ИПС для учета кадров.

Основная доработка была связана с динамическим (переменным) характером кадровой информации, а также необходимостью не только производить поиск объектов по их признакам, но и осуществлять статистическую обработку имеющейся информации.

В течение 1968—1970 гг. в Московском энергетическом институте на кафедре вычислительной техники совместно с Вычислительным центром МЭИ была разработана на базе ЭВМ «Минск-22» информационно-поисковая система для учета кадров и проведено ее внедрение. Данные о каждом человеке, вводимые в ИПС, делятся на две группы: поисковые и справочные (анкетные). Поисковые данные представляются в виде ассоциативной узловой структуры, в которой каждому лицу соответствует один узел.

Имеется словарь дескрипторов, отражающий специфику кадровых задач и состоящий из следующих разделов:

1) район, откуда прибыл человек (республика, область и т. д.),
2) точное место, откуда прибыл обучающийся,
3) категория, к которой относится обучающийся (из школьников, из рабочих и т. д.),
4) год поступления,
5) ожидаемый год окончания,
6) курс,
7) город, в котором происходит обучение,
8) учебное заведение,
9) специальность,
10) группа специальности,
11) вид обучения (очное, заочное, студент, аспирант и т.д.),
12) причина отчисления и дата,
13) пол.

Справочные данные представляют собой сведения анкетного характера, поиск по которым не ведется, но которые могут быть выданы по требованию после нахождения нужного лица по его поисковым признакам.

Справочные данные группируются в ИПС в качестве фактографических описаний. К числу справочных данных относятся следующие 19 пунктов:

1) фамилия, имя и отчество обучающегося,
2) год рождения,
3) национальность,
4) образование до поступления в данное учебное заведение,
5) домашний адрес,
6) номер паспорта,
7) социальное положение,
8) состояние здоровья,
9) общественная работа (заполняется ежегодно),
10) владеет ли языками (исключая русский язык),
11) номер приказа о зачислении,
12) номер приказа об отчислении,

13) номер приказа перевода,
14) номер приказа об академическом отпуске,
15) номер приказа о переводе на курс,
16) номер приказа о материальном обеспечении,
17) обеспеченность стипендией,
18) успеваемость (средний балл, полученный на экзаменах, и количество двоек; заполняется каждый семестр),
19) отметка, полученная на защите дипломного проекта, или средний балл, полученный на госэкзаменах.

Перечисленная группа данных об учащихся, представленная в ИПС, может быть расширена в зависимости от потребностей.

Поиск может быть проведен по любому запросу, содержащему поисковые признаки в любой их комбинации.

Например:

1) Сколько студентов, прибывших из Якутии в заданном году, учатся в вузах г. Москвы на 3 курсе?
2) Выдать номера по списку аспирантов, прибывших из Узбекистана.
3) Выдать анкеты стажеров, прибывших из Грузии и окончивших стажировку в заданном году.
4) Сколько учащихся техникумов окончит обучение в заданном году по специальности «химик-технолог по синтетическим волокнам»?

Ответы на запрос могут быть выданы либо в виде количества, либо в виде номеров учащихся в общем списке, либо в виде развернутых анкет лиц, обладающих некоторыми одинаковыми признаками, отраженными в запросе. Причем форма желаемого ответа также должна быть отражена в запросе.

Так, например, на 1 и 4-й вопросы ответ будет выдан в виде количества, на 2 вопрос — в виде списка номеров учащихся, на 3 вопрос — в виде развернутых анкет.

Кроме того, в составе данной кадровой ИПС имеются программы составления машиной различных типовых форм отчетности, существующей в настоящее время.

Эффективность указанной информационно-поисковой системы, помимо собственно справочной службы, подтверждается также возможностью широкой статистической обработки хранимых данных.

Реализованная на ЭВМ «Минск-22» информационно-поисковая система учета кадров обладает следующими характеристиками:

1) одновременно может производиться поиск по 36 запросам,
2) время поиска по группе запросов 5—6 *мин*,
3) время поиска по одному запросу 14 *сек*,

время печати ответа на один запрос в виде номеров документов (при ответе, содержащем около 20 номеров) 1 *сек*,

4) время печати ответа на один запрос в виде анкет учащихся (при ответе, содержащем примерно 20 анкет) 10— 15 *сек.*

Таким образом, за одну смену с помощью ЭВМ «Минск-22» может быть обработано 300—400 запросов. Это превышает существующие потребности в справках подобного рода. Следовательно, ЭВМ может быть использована для решения статистических и плановых задач, необходимых для оптимального планирования и управления подготовкой кадров в различных аспектах этого дела.

21. Документальная (библиографическая) ИПС для больших массивов документов и большого словаря дескрипторов

Принципы построения указанной ИПС мы рассмотрим на примере ИПС для медицины. В мире существует свыше пяти тысяч медицинских и биологических журналов, в которых ежегодно публикуется около полмиллиона статей. В медицине используются десятки тысяч различных терминов, в том числе большое количество синонимов. Большой объем массива документов, среди которых необходимо осуществлять поиск, и большой объем словаря дескрипторов, а также тесная связь и взаимное проникновение различных разделов медицины и биологии предъявляют ряд специфических требований к построению ИПС, особенно если при ее реализации ориентироваться на сравнительно небольшую ЭВМ типа «Минск-22».

Основной задачей рассматриваемой ИПС является ретроспективный (т. е. полный) поиск информации по узким тематическим запросам, представляемым в виде наборов дескрипторов из

заданного словаря.

В общем случае библиографические информационные фонды включают в себя три вида информации: первичную (литературные публикации, научные отчеты, патенты и т. п.), вторичную (рефераты, аннотации, информационные карты) и третичную (машинные поисковые образы документов — ПОДы и библиографические описания документов).

Технической базой для разработки рассматриваемой ИПС являлась ЭВМ «Минск-22», имеющая ОЗУ в 8196 ячеек по 37 разрядов и 16 ЛПМ* по 100 тыс. ячеек.

Структура и основные принципы построения ИПС. В основу построения библиографической информацинно-поисковой системы для медицины положены следующие принципы:

1) Трехступенчатая система поиска информации в ретроспективном медицинском массиве по текущим запросам. Такой подход обусловлен, с одной стороны, достаточно большим объемом словаря дескрипторов (медицинского тезауруса) и весьма большим объемом медицинской информации, среди которой должен осуществляться поиск, и, с другой стороны, ограниченным объемом оперативной и внешней памяти ЭВМ «Минск-22», на которой реализуется данная ИПС.

С целью обеспечения трехступенчатого поиска весь массив медицинской информации делится на ряд тематических подмассивов, соответствующих различным разделам медицины, а поисковые образы документов строятся с использованием элементов грамматики. При поиске сначала выбираются подходящие подмассивы, затем производится основной поиск внутри отобранных подмассивов по грубому критерию смыслового соответствия и, наконец, проводится логический анализ найденных документов с использованием грамматики ИПЯ и окончательный их отбор по точному критерию. Последний этап проводится только в том случае, если на втором этапе поиска будет найдено достаточно большое число документов (больше 100).

2) Сочетание ассоциативного способа организации главного поискового массива и поиска, осуществляемого на втором этапе, со способом логических шкал, используемым для предварительного отбора тематических подмассивов на первом этапе. Этот вопрос будет подробнее рассмотрен ниже.

3) Однократная перфорация и ввод исходной для наполнения ИПС информации, представляющей собой сочетание библиографических описаний документов с их поисковыми образами (ПОДами), с последующим автоматическим формированием кодированных ПОДов документов и библиографических описаний. Эта исходная информация используется при решении всех трех основных задач ИПС (ретроспективный поиск, ПРИ, подготовка библиографических указателей).

4) Модульный принцип построения всех программ ИПС и использование специальной операционной системы (программы-диспетчера) для управления работой ИПС и взаимодействия с операторами. Модульность построения алгоритмов и программ ИПС проявляется прежде всего в том, что три основных этапа поиска (выбор подмассивов, основной поиск документов, логический анализ и отбор) выполняются автономными программами, которые могут использоваться независимо друг от друга и даже в автономных ИПС. Автономными являются также программы формирования различных информационных массивов и ряд контрольных (служебных) программ и программы выдачи (печати) результатов поиска. Кроме того, основные программы поиска и формирования поисковых массивов в библиографической ИПС могут также использоваться и в фактографических ИПС для этапа поиска.

К числу основных компонент автоматизированной ИПС кроме технических средств относятся:

1) *математическое обеспечение,* включающее в себя комплекс алгоритмов и программ для формирования и корректировки информационных массивов, осуществления поиска информации, выдачи ответов на печать, выполнения контрольных и вспомогательных (служебных) операций;

2) *медицинский тезаурус,* представляющий собой словарь основных и наиболее употребительных в медицинской литературе терминов. В этом словаре все термины, являющиеся синонимами, объединены в так называемые группы эквивалентности и в каждой такой группе выделен основной термин, который называется *дескриптором.* При вводе библиографических описаний, поисковых образов документов и запросов возможно использование любых синонимов из числа учтенных в тезаурусе, так как все эти синонимы в конечном счете в памяти машины будут представлены одним и тем же числовым кодом дескриптора (КД). При выдаче результатов поиска документов из машины (ПОДов) в них будут фигурировать только основные дескрипторы. В библиографических описаниях, выдаваемых из машины, могут фигурировать также и синонимы, так как машина выдает библиографическое описание в том виде, в котором оно было введено в машину. Каждому дескриптору

* ЛПМ—лентопротяжный механизм.

приписывается свой числовой код (обычно порядковый номер). В тезаурусе указываются также родовидовые отношения, т. е. для данного дескриптора указываются дескрипторы, являющиеся более общими (родовыми по отношению к нему), и дескрипторы, являющиеся частными (видовыми) по отношению к данному дескриптору.

В медицинском тезаурусе имеется около 10—12 тысяч дескрипторов. Они разбиты на ряд полей (общенаучная терминология, здравоохранение, общая биология, молекулярная биология, организмы, анатомические термины, болезни, химические вещества и медикаменты, методы и оборудование, гигиена и эпидемиология, и др.). Каждое поле в свою очередь делится на ряд подполей, а в пределах подполя дескрипторы идут в алфавитном порядке (кроме синонимов, которые следуют за основным дескриптором). Таким путем образуется предметный вариант тезауруса. Кроме того, все дескрипторы (из всех полей и подполей) располагаются в едином алфавитном порядке, образуя алфавитный вариант тезауруса здесь синонимы попадают в различные места словаря в соответствии с их буквами, но все они имеют одинаковый числовой код. Предметный вариант употребляется при построении и корректировании тезауруса, а также при индексации документов и запросов, когда нужно подыскать подходящий дескриптор для выражения того или иного понятия. Алфавитный вариант используется при ручном кодировании дескрипторов, когда нужно для заданного дескриптора определить его числовой код. Кроме того, имеется и номерной вариант тезауруса, в котором все дескрипторы располагаются в порядке возрастания их числовых кодов. Здесь синонимы отсутствуют, и фигурируют только основные дескрипторы. Этот вариант нужен для того, чтобы по числовому коду дескриптора найти его полное словесное представление;

3) *массив логических шкал,* служащий для предварительного отбора тех тематических поисковых подмассивов, в которых могут находиться документы, отвечающие запросу. Строение и функции этого массива подробнее рассматриваются ниже. Следует заметить, что найденные на этапе предварительного отбора тематические подмассивы могут и не содержать искомых документов, что будет выявлено на втором этапе поиска;

4) *поисковые массивы,* включающие в себя поисковые образы документов. ПОД состоит из машинного кода документа (его инвентарного номера) и набора дескрипторов, описывающих содержание этого документа в основных аспектах. Машинный код документа может являться адресом его библиографического описания и адресом местонахождения его реферата, а также адресом (инвентарным номером) оригинала документа в библиотеке или в архиве. Возможен также такой вариант, когда машинный код документа является только адресом его библиографического описания на МЛ, а все остальные адреса (в том числе инвентарный номер) указываются в библиографическом описании. В ПОД входит также краткая библиографическая справка стандартного вида: тип документа (статья, монография и т. п.), язык, год издания и т. д. Поисковые массивы состоят из главного и текущего поисковых массивов. Главный поисковый массив содержит ПОДы, собранные в системе за все время ее существования, с учетом обновлений. Текущий поисковый массив состоит из ПОДов документов, поступивших в систему только за последний фиксированный период (месяц). Все новые ПОДы включаются сразу в оба поисковых массива. Текущий поисковый массив используется в конце текущего периода для решения задач избирательного распределения и подготовки библиографических аннотированных указателей. После этого текущий поисковый массив уничтожается и начинается формирование нового текущего поискового массива. Главный и текущий поисковые массивы состоят из ряда тематических поисковых подмассивов, соответствующих различным разделам медицины. В главном поисковом массиве тематические подмассивы располагаются на отдельных МЛ. Считая средний размер ПОДа в 10 ячеек (6—8 дескрипторов), получим, что в каждом тематическом подмассиве может быть до 10 000 документов. Тематические подмассивы, входящие в текущий поисковый массив (месячный), естественно, будут значительно меньше по своему объему и могут занимать одну или несколько зон магнитной ленты;

5) *справочные массивы библиографических описаний* включают в себя два массива: главный и текущий (месячный). Первый состоит из библиографических описаний всех документов, включенных в главный поисковый массив. При решении задачи ретроспективного поиска этот справочный массив используется для выдачи библиографических описаний по тем документам, которые были найдены ЭВМ по запросам. Выдача производится по машинным адресам, которые указаны в ПОДе для соответствующего описания. При решении задачи избирательного распределения и подготовки указателей используется текущий (месячный) массив библиографических описаний;

6) *информационный массив рефератов* (или аннотаций, информационных карт и т. п.) состоит из

микрофотокопий текста документов, зафиксированных на кинопленке. На каждом кадре кинопленки фиксируется адрес (семизначным десятичным числом). Кроме того, для группы последовательно идущих кадров может быть указан еще групповой адрес, так называемый адрес зоны (также семизначным числом).

При необходимости просмотра или получения фотокопии реферата адрес для поиска выдается ЭВМ. Этот адрес набирается на клавиатуре специализированного устройства, которое находит соответствующий кадр для последующего копирования. Этот массив используется в основном при решении задачи ретроспективного поиска;

7) *служебные массивы*, необходимые для эффективной организации процесса накопления, обработки и поиска информации в ИПС. Сюда относится массив наименований и кодов журналов и других изданий, массив наименований и кодов учреждений, специальностей и ряд других вспомогательных массивов. Эти массивы позволяют хранить в ПОДах и библиографических описаниях документов только коды наименований, а при выдаче на печать — формировать полные наименования.

Принципы машинной реализации ИПС. Машинная реализация ИПС для медицины предусматривает возможность трехступенчатого поиска документов по запросам. Первая ступень — предварительный отбор тематических поисковых подмассивов, в которых могут находиться требуемые документы. Этот этап реализуется при помощи логических шкал, фиксирующих использование дескрипторов в различных тематических поисковых подмассивах. Для каждого дескриптора отводится ячейка памяти, в которой размещается его логическая шкала. Каждому из разрядов этой шкалы поставлен в соответствие один тематический поисковый подмассив; если в этом подмассиве имеется хотя бы один документ, в ПОДе которого фигурирует данный дескриптор, то в этом разряде ставится единица, в противном случае — ноль. Так как общий объем тезауруса около 12 000 дескрипторов, то требуется 12 000 логических шкал, которые занимают 6 зон магнитной ленты машины «Минск-22». Очевидно, что при данной разрядности ячейки машины «Минск-22» целесообразно иметь в ИПС до 36 подмассивов (хотя можно иметь и 72 и больше подмассивов). На первом этапе поиска для каждого поискового образа запроса (ПОЗа) выбираются логические шкалы его дескрипторов и путем их логического умножения определяются тематические подмассивы, в которых могут быть документы, отвечающие данному запросу. Так как поиск идет обычно одновременно по большому числу запросов (20—30), то такая операция проводится сразу для всех ПОЗов. Затем производится сортировка ПОЗов по тематическим подмассивам. После этого поочередно проводится поиск внутри каждого тематического подмассива, отобранного на первом этапе. При этом поиск идет только по тем ПОЗам, которые относятся к данному тематическому поисковому подмассиву.

На втором этапе осуществляется поиск документов внутри тематических подмассивов. Для машинной реализации поисковых подмассивов принят ассоциативный узловой метод, сочетающий в себе преимущества прямого и инверсного способов организации поисковых массивов. Поисковый образ каждого документа в этом подмассиве представляется ассоциативным узлом, в котором каждому дескриптору ПОД соответствует одно списковое слово, состоящее из кода тематического дескриптора, кода смысловой связи и адреса связи. Адреса связи объединяют в единые цепные списки документы, обладающие одинаковыми дескрипторами. Каждому тематическому дескриптору соответствует один цепной список на данной МЛ. Поиск заданных документов ведется путем прослеживания одного цепного списка, соответствующего тому дескриптору из числа имеющихся в ПОЗе, который содержит наименьшее число документов в своей цепочке. При прослеживании данного цепного списка производится проверка каждого члена этого списка на вхождение в ПОД этого члена остальных дескрипторов, имеющихся в ПОЗе. В результате поиска определяются единые инвентарные номера (адреса библиографических описаний) тех документов, ПОДы которых удовлетворяют заданному критерию смыслового соответствия. В случае, если на этом этапе поиска обнаружено слишком большое количество документов, производится дополнительный отбор среди них тех документов, которые наиболее полно отвечают содержанию запроса. Для дополнительного отбора используется большее количество дескрипторов ПОЗ и смысловые связи (указатели отношений между дескрипторами), а также более тонкие критерии смыслового соответствия.

В качестве первичной информации для формирования поисковых и библиографических массивов перфорируются и вводятся в машину «Минск-22» поисковые образы и библиографические описания документов. Эти данные после частичного кодирования записываются в массивы ПОДов и библиографических описаний (на соответствующие магнитные ленты). Кодированию подвергаются дескрипторы и наименования журналов; кроме того, стандартные дескрипторы упаковываются в одну

ячейку в виде логической шкалы. Такое кодирование сокращает размер записи ПОДов и библиографических описаний на МЛ. Такой однократный ввод исходной информации позволяет сократить объемы перфорации и время, затрачиваемое на ввод документов в ИПС.

Собственно библиографическое описание представляет собой текст произвольного размера и содержания, ограниченный с начала и с конца специальными разделительными знаками. Оно содержит фамилию автора, заглавие и другие библиографические данные. ПОД представляет собой набор дескрипторов (обязательно взятых из тезауруса и выраженных в кодированной форме). Перфорация библиографических описаний документов (статей) производится непосредственно с журналов (их оглавлений) без переписи на бланки.

Перфорация ПОДов производится на перфоленту в виде кодов. Каждый ПОД и каждое библиографическое описание имеют общий инвентарный номер, по которому ЭВМ устанавливает их соответствие.

Перфорация библиографических описаний на перфоленту осуществляется в словесной (текстовой) форме (без предварительного кодирования), за исключением кодов журналов. При выдаче данных на печать по специальной программе из ПОДов формируются фразы, которые присоединяются к библиографическому описанию.

Возможен ввод дескрипторов ПОД и в словесной форме с последующим автоматическим кодированием. Для каждого тематического дескриптора машина формирует его двоичную свертку путем суммирования кодов букв словесного выражения дескриптора со сдвигом кода каждой буквы на один разряд. Одновременно производится подсчет числа букв в словесном выражении дескриптора. Затем производится обращение к разделу словаря (цепному списку), содержащему дескрипторы с данным числом букв, и в этом разделе находится код свертки дескриптора, а рядом с ним — фактический числовой код дескриптора (КД). Полученные таким образом коды дескрипторов используются при формировании ПОДов. Аналогичным образом производится и автоматическое кодирование дескрипторов при вводе в ЭВМ ПОЗов. Использование сверток непосредственно в качестве кодов дескрипторов неудобно по следующим соображениям. Во-первых, синонимы одного дескриптора будут иметь разные свертки, а следовательно, и разные коды, что недопустимо. Во-вторых, при использовании медицинского тезауруса в качестве международного (например, для стран социалистического лагеря) переводы дескрипторов на разные языки будут иметь различные свертки, что также приведет к различию в кодировании документов. При использовании же независимых числовых кодов поисковые образы документов и запросов будут одинаковыми на разных языках независимо от наличия синонимов.

На основе стандартных дескрипторов машина формирует стандартную справку, входящую позиционной строкой в ПОД (в последнюю ячейку узла). Первой строкой (первой ячейкой узла) является заголовок ПОДа, содержащий инвентарный номер документа. Первая и последняя ячейка каждого ассоциативного узла (ПОДа), содержащие заголовок и позиционную строку стандартных дескрипторов документа, выделяются записью в знаковых разрядах этих ячеек минуса. Внутренние ячейки каждого ассоциативного узла, содержащие списковые слова, имеют в знаковом разряде плюс. Списковое слово включает в себя следующие части: код дескриптора, код адреса связи, код вида смысловой связи и код направления смысловой связи. Всего может быть до 8 различных видов смысловых связей: **и, или, не, при помощи, в результате (-после), на основе, при наличии** (факторов); кроме того, имеется определительная связь. Указатель направления связи показывает номер того дескриптора в ПОДе, с которым связан данный дескриптор этой смысловой связью.

Машинная реализация медицинского тезауруса. Медицинский тезаурус (словарь дескрипторов) в памяти ЭВМ используется для трех целей:

1) для формирования формализованных фраз при выдаче данных на печать (по кодам дескрипторов и смысловых связей, имеющихся в ПОДах),

2) для учета родо-видовых и ассоциативных отношений между дескрипторами запросов и соответствующей модификации ПОЗов при выполнении различных вариантов поиска,

3) для обеспечения начальной адресации цепных списков в ассоциативных узловых структурах при работе с поисковыми массивами.

В соответствии с этими функциями тезаурус в памяти ЭВМ фигурирует в двух видах: 1) в виде грамматического (лексического) машинного словаря дескрипторов, содержащего сведения об образовании родительного падежа и сокращенной формы прилагательных; 2) в виде ряда таблиц фиксаторов цепных списков, соответствующих тематическим поисковым подмассивам.

Кроме того, может использоваться третий вид тезауруса, служащий для автоматического кодирования словесных представлений дескрипторов. Вход в этот словарь осуществляется с помощью сверток дескрипторов или их синонимов. Выходом являются коды дескрипторов и дополнительные сведения о родо-видовых или ассоциативных отношениях данного дескриптора с другими. Объем машинного словаря в несколько раз превышает общее число дескрипторов в тезаурусе за счет наличия в словаре большого числа синонимов. Особенностью программ работы с машинным словарем является необходимость минимизации количества обращений к магнитной ленте. Это достигается групповой обработкой вводимых запросов или поисковых образов документов, при которой одновременно обрабатываются все дескрипторы, относящиеся к одной зоне магнитной ленты, независимо от того, к каким запросам или ПОДам они относятся.

Таблицы фиксаторов цепных списков также записываются на магнитных лентах. Для каждого тематического подмассива имеется своя таблица фиксаторов цепных списков. Каждый фиксатор соответствует одному дескриптору, используемому в ПОДах данного подмассива. Так как заранее неизвестно, какие дескрипторы будут использоваться в том или ином подмассиве, то в каждой таблице фиксаторов предусматривается место для полного словаря дескрипторов. Так как тезаурус содержит около 12 тысяч дескрипторов, то для размещения каждой таблицы фиксаторов отводится 6 зон магнитной ленты. Ясно, что в каждом тематическом подмассиве используется только часть дескрипторов (приблизительно 15—20%) и в таблицах фиксаторов большая часть ячеек оказывается свободной. Однако выделение под таблицы фиксаторов полного объема памяти на МЛ упрощает обращение к фиксаторам списков, так как в этом случае код дескриптора служит адресом соответствующего фиксатора списка. Все свободные ячейки зон, занятых таблицами фиксаторов, соединяются в цепные списки свободных ячеек по зонам МЛ. Эти свободные ячейки могут использоваться для вспомогательных целей, в частности для указания родо-видовых и ассоциативных отношений межу дескрипторами данного тематического подмассива.

Таблицы фиксаторов могут размещаться на тех же МЛ, на которых находятся их тематические подмассивы, а могут находиться на отдельных МЛ. В последнем случае на одной МЛ ЭВМ «Минск-22» смогут разместиться таблицы фиксаторов для 6—8 тематических подмассивов. При общем количестве тематических подмассивов, приблизительно равном 36, требуется 6—7 магнитных лент ЭВМ «Минск-22» под все таблицы фиксаторов. Эти магнитные ленты, так же как и главная магнитная лента, содержащая грамматический машинный словарь, обратный машинный словарь (в котором по коду дескриптора определяется его словесное выражение), таблицу логических шкал и ряд вспомогательных таблиц, будут находиться в ЭВМ в процессе решения задачи ретроспективного поиска. Остающиеся лентопротяжные механизмы (8—9 ЛПМ) занимаются сменяемыми лентами с тематическими поисковыми подмассивами. Сменные МЛ выбираются из машинной библиотеки тематических подмассивов в соответствии с номерами, определенными на первом этапе поиска с помощью логических шкал. При установке сменных МЛ на определенные ЛПМ в ЭВМ вводится соответствующее число перфокарт, определяющих соответствие между тематическими поисковыми подмассивами и занимаемыми ими ЛПМ. При необходимости увеличить число тематических подмассивов (и соответственно МЛ в машинной библиотеке) можно вместо одной ячейки в логической шкале дескриптора использовать две и больше ячеек.

В соответствие с тематическими поисковыми подмассивами в машинной библиотеке должно быть и такое же число тематических библиографических подмассивов, каждый из которых будет занимать несколько МЛ. Тематические поисковые подмассивы строятся так, что каждый из них занимает одну МЛ.

Смысловые связи и машинный грамматический словарь используются в ИПС для построения формализованных фраз аннотации на основе ПОДов. При этом одни и те же ПОДы позволяют выдавать аннотации на различных языках, используя различные машинные грамматические словари. При разработке рассматриваемой ИПС были реализованы алгоритмы выдачи фраз, помимо русского языка, также на английском и немецком языках.

Ассоциативная узловая структура с обратным размещением узлов на магнитной ленте.

Каждый цепной список имеет фиксатор списка, который размещается в одной ячейке; адрес фиксатора является кодом данного дескриптора; зная код дескриптора, можно обратиться к фиксатору его цепного списка. Фиксаторы цепных списков в рассматриваемом варианте ассоциативной узловой структуры включают в себя три элемента. Первым элементом является адрес связи начала (АСН) цепного списка документов, имеющих в своих поисковых образах данный дескриптор. Этот адрес

указывает первый ассоциативный узел в данном списке, а точнее, ячейку в этом узле, соответствующую данному списковому слову. АСН используется при поиске документов в поисковом массиве для ограничения процесса поиска. Так как в данном варианте ассоциативной узловой структуры поиск ведется с конца цепных списков к их началу, то признаком окончания поиска будет являться достижение максимального из АСН цепных списков, отвечающих дескрипторам, используемым при поиске. Вторым элементом фиксатора цепного списка является число документов в этом списке (ЧД). Это число необходимо для выбора при поиске наиболее короткого списка для его прослеживания. Третьим элементом фиксатора является адрес конца списка (АСК), который указывает адрес ячейки со списковым словом в последнем включенном в поисковый массив документе, обладающем данным дескриптором. Этот адрес является исходным для просмотра цепных списков при поиске и для включения новых документов в систему.

При размещении трех указанных величин (АСН, ЧД и АСК) в одной ячейке, имеющей 37 двоичных разрядов, учитывается следующее: АСН должен обеспечивать адресацию в пределах одной МЛ, для чего нужно 18 двоичных разрядов. 9 разрядов отводится под ЧД, что позволяет иметь в списке до 512 членов. Так как списки образуются только в пределах каждой МЛ, где всего может быть до 10 тысяч ПОДов, следует признать указанное ограничение числа членов в каждом списке вполне приемлемым. Под АСК отводятся остающиеся 10 двоичных разрядов, что позволяет осуществлять адресацию в пределах одной МЛ с точностью до поля в 128 ячеек. Учитывая, что этот адрес служит для ограничения процесса поиска, следует признать эту точность вполне достаточной.

Каждый новый документ, включаемый в систему, должен включаться в те списки, которые соответствуют имеющимся в его поисковом образе дескрипторам. Включение новых документов в данном варианте производится в концы списков; просмотры списков при поиске документов также производятся, начиная с концов цепных списков. При этом заполнение магнитной ленты ассоциативными узлами ведется в обратном порядке, начиная с конца ленты. При таком порядке включения новых документов документы в цепных списках будут располагаться в хронологической последовательности, обратной порядку их поступления в поисковую систему.

При поиске документов первыми просматриваются ассоциативные узлы тех документов, которые включены в систему последними, что тоже представляет определенное преимущество, так как поиск начинается всегда с самых новых документов. Так как расположение ПОДов вдоль магнитной ленты и при этом способе соответствует порядку поступления документов в систему (только в обратном направлении), то и в этом случае, если необходимо ограничиться выдачей документов, поступивших в систему за определенный период (например, за последний год или два года), можно прекратить просмотр ленты при достижении фиксированной границы, соответствующей заданному периоду времени. Включение новых объектов в списки существенно упрощается за счет того, что при этом способе не требуется изменять адреса связи в узлах разных объектов, которые будут находиться, как правило, в разных зонах МЛ и для нахождения которых необходимо было бы осуществлять многократные переписи разных зон МЛ в МОЗУ.

В данном варианте включение нового объекта сводится к тому, что меняются адреса связи только в фиксаторах цепных списков, соответствующих заданным дескрипторам (там ставятся в качестве адресов связи концов цепных списков — АСК — адреса соответствующих списковых слов во вновь сформированном узле). В списковых словах нового ассоциативного узла теперь ставятся уже не коды концов списков (как в ранее описанном варианте), а адреса связи, взятые из отобранных фиксаторов цепных списков. Так как все фиксаторы цепных списков находятся в одной зоне (или в нескольких подряд идущих зонах), то эти изменения адресов делаются сразу для всех дескрипторов без многократных обращений ко многим зонам МЛ, в которых могли бы находиться концы списков при прежнем варианте.

Возможен и третий вариант, при котором запись узлов на МЛ ведется с начала МЛ, а соединение узлов в цепные списки идет в обратном направлении, от последнего записанного узла к началу ленты. Этот вариант схематично показан на рис. 36. Недостатком его является необходимость в процессе поиска многократно (при просмотре каждой требуемой зоны дважды) изменять направление движения ленты.

На рис. 37 дано схематическое представление расположения ассоциативных узлов на МЛ при втором варианте, который является наилучшим. Здесь запись узлов идет
от конца ленты к ее началу, а адреса связи в цепных списках направлены от начала ленты к ее концу, т. е. от узлов, записанных позже, к узлам, записанным раньше. Поэтому

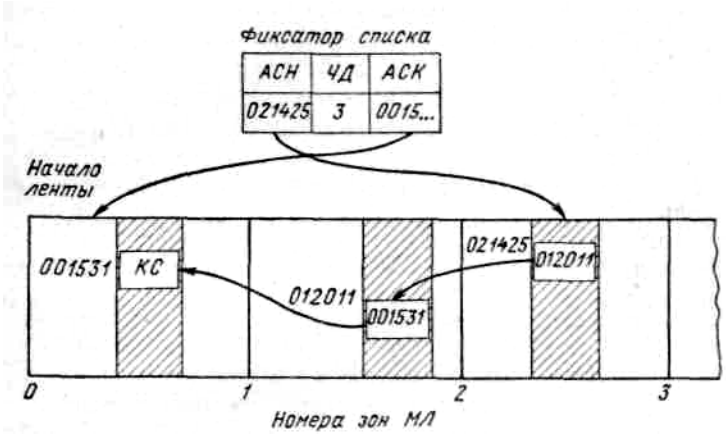


Рис. 36. Схема записи ассоциативных узлов от начала к концу МЛ.

в этом варианте все цепные списки имеют направленный характер, т. е. адреса связи в них при движении вдоль по спискам монотонно возрастают.

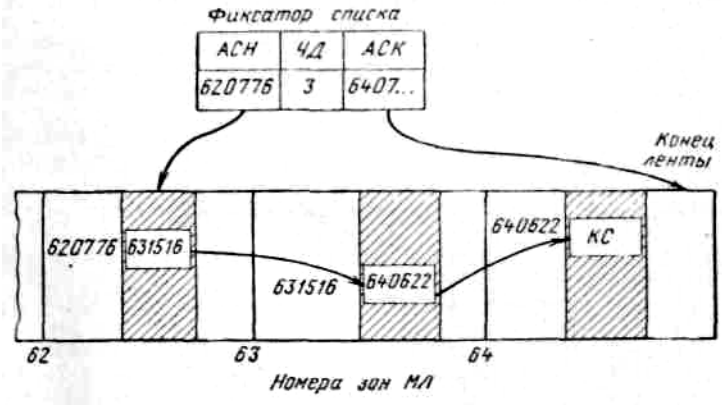


Рис. 37. Схема записи ассоциативных узлов от конца к началу МЛ.

На рис. 36 и 37 ассоциативные узлы показаны штриховкой. В них прямоугольниками выделены по одному списковому слову. Слева от прямоугольников указаны адреса ячеек, в которых находятся эти списковые слова, а внутри прямоугольников показаны адреса связи, отсылающие к следующим членам цепных списков. В общем случае в цепных списках члены списков могут располагаться в памяти машины произвольным образом, и поэтому адреса связи у них могут меняться при переходе от одного члена списка к следующему как в сторону возрастания, так и в сторону убывания. Для списков, которые целиком помещаются в МОЗУ, это значения не имеет. Так как в ЭВМ «Минск-22» поиск данных на МЛ может происходить только путем переписи в МОЗУ целых зон или их частей, то для сокращения числа обращений к МЛ и числа реверсов МЛ необходимо, чтобы адреса связи в цепных списках изменялись монотонно в соответствии с направлением движения ленты.

22. Принципы построения ассоциативной фактографической информационно-логической системы (АФИЛС)

Общие сведения об АФИЛС. АФИЛС предназначается для накопления фактографических данных о различных объектах (веществах, материалах, приборах, людях, болезнях и т. д.) и выдачи по запросам как адресов первичных источников информации (документов, статей, отчетов и др.), так и конкретных фактических сведений.

Учитывая большие объемы информации, подлежащей хранению, и динамичный характер использования подобных систем, целесообразно создавать подобные системы для узких разделов знаний. Отдельные системы могут содержать необходимые отсылки к другим специализированным системам, образуя интегрированную систему, охватывающую достаточно широкие области.

Основу программной реализации рассматриваемой системы составляют ассоциативные поисковые деревья и ассоциативные узловые структуры, обеспечивающие быстроту поиска, удобство ввода, накопления и вывода данных, возможность построения гибких перекрестных связей и многоаспектную классификацию и поиск данных.

Наполнение АФИЛС информацией связано с критическим анализом вводимых данных, т. е. представляет собой процесс, требующий творческого участия квалифицированных специалистов в данной области знаний.

При вводе каждого нового сообщения в АФИЛС эти специалисты должны запрашивать, как правило, АФИЛС о наличии в ней аналогичных данных, сопоставлять новые сведения с уже имеющимися в АФИЛС, производить необходимые корректировки в старых сведениях, а новые данные вводить с учетом того, что уже имеется в системе.

АФИЛС должна являться комплексной человеко-машинной информационной системой, в которой функции творческого анализа, отбора и оценки информации выполняются людьми (во всяком случае, в ближайшем будущем), а функции накопления, хранения, поиска, обработки и выдачи информации выполняются машиной.

В АФИЛС должны постепенно накапливаться в систематизированном виде полные и точные данные по соответствующим областям знаний, и этой информацией смогут эффективно пользоваться широкие круги специалистов. Таким образом, большой труд по отбору, анализу и вводу в АФИЛС информации будет многократно окупаться высокой скоростью, точностью, полнотой и широкой доступностью указанной информации. Одной из основных форм взаимодействия АФИЛС с людьми является метод «диалога — собеседования» ЭВМ с людьми, в процессе которого ЭВМ не только выдает ответы на поставленные вопросы, но и помогает ставить вопросы, постепенно уточняя тему поиска и предлагая для выбора человеку различные возможные варианты постановки вопросов.

Рассмотрим основные элементы системы.

Первым элементом является информационный машинный язык, включающий в себя словарь дескрипторов, используемых в данной отрасли знаний, и фиксированный набор смысловых связей, определяющих смысловые отношения между дескрипторами-терминами, входящими в состав сложных понятий. Словарь дескрипторов и набор смысловых связей составляются в основном еще до поступления конкретных сведений об объектах в данную информационную систему, на основе анализа текстов в этой области знаний.

Словарь строится с учетом синонимии и наличия иерархических и ассоциативных отношений между различными терминами. Для удобства пользования словарь формируется в трех вариантах: систематическом, алфавитном и древовидном (иерархическом).

В состав словаря дескрипторов включаются наиболее употребительные термины, определения, наименования объектов, свойств, процессов, единиц измерений и т. д.

Наряду с фиксированными в словаре основными терминами, представляемыми кодами, в АФИЛС допускается для уточнения основных терминов в справках использование «свободных» узкоспециальных терминов, представляемых текстом.

Естественно, что в связи с развитием науки, появлением новых терминов и устареванием терминов словарь терминов будет периодически обновляться, причем обновление будет идти, в основном, за счет включения употребительных «свободных» терминов.

Вторым элементом АФИЛС является массив фактических сведений об объектах. Собственно накопление этих сведений и является основной целью создания указанной системы. Этот массив состоит из трех частей:

1) поискового массива (массива ПОДов);

2) массива справочных данных об объектах (массива справок);

3) массива дополнительных данных об объектах.

Поисковая и справочная информация хранится в памяти машины, как правило, на магнитных лентах.

Под объектом следует понимать любой предмет или явление; однородные объекты объединяются в классы и описываются однотипным образом. Примерами объектов могут быть: вещество, прибор, схема, машина, метод расчета, процесс и т. д.

Дополнительная информация (в том числе схемы, чертежи, фотографии и т. д.) хранится вне АФИЛС, например на микрофильмах. Справка состоит из фиксированного списка пунктов, характерных для данного класса объектов. Каждый пункт может иметь одно или несколько сообщений, представляемых либо кодами терминов, либо формализованными фразами, либо текстом. В сообщении указывается всегда источник сведений и дата их ввода в АФИЛС.

Третьим элементом системы является набор алгоритмов и программ, обеспечивающих включение и исключение данных об объектах, их поиск, обработку и различные вспомогательные процессы, в частности выдачу на печать готовых ответов, прием словесных запросов и их перевод на внутренний язык АФИЛС, а также обеспечивающих специальные виды обработки: поиск аналогов, статистические оценки, выяснение корреляционных связей и т. д.

Справочная информация из АФИЛС может выдаваться в разных режимах: целиком справка, по отдельным пунктам, с заменой кодов наименованиями или без замены и др.

К числу типовых вопросов, на которые должна отвечать АФИЛС; могут быть отнесены следующие:

1) Сообщить, существует ли объект (вещество, схема, прибор, машина, метод расчета, технологический процесс, болезнь и т. д.), обладающий такими-то характеристиками, и выдать все его данные (или некоторые данные).

2) Выдать все или некоторые характеристики объекта (вещества, схемы, узла и т. д.), имеющего такой-то шифр (наименование).

3) Произвести статистический анализ объектов определенной категории.

4) Выяснить зависимость между определенными свойствами объектов и произвести их классификацию (категоризацию) в соответствии с этими свойствами.

5) По заданным наборам признаков найти аналоги в заданной группе приборов, машин или материалов.

6) Сообщить характеристики оборудования, необходимого для выполнения таких-то работ.

Последняя группа вопросов может использоваться при решении задач планирования экспериментов, материально-технического снабжения, определения нормативов на вновь выпускаемую продукцию и т. д.

Основными этапами работы АФИЛС будут являться:

1) Кодирование терминов (дескрипторов), заданных в запросах или поисковых образах объектов, с помощью специального машинного словаря путем свертывания их буквенных представлений. В результате должны получаться коды вводимых дескрипторов.

2) Опознавание понятий по ассоциативному словарю понятий и выдача адресов, указывающих положение заголовков отобранных ассоциативных узловых структур.

3) Прослеживание цепных списков в узловых структурах и отбор объектов, отвечающих заданным требованиям, с выдачей адресов записей их справок.

4) Выборка (или, наоборот, включение) требуемых данных из записей справок и выдача их на печать.

Основной частью алгоритма работы АФИЛС является опознавание вводимых фраз (путем их разложения на термины и анализа смысловых связей) и образование ответных фраз (синтез ответов).

В отличие от алгоритмических языков, служащих для описания процессов переработки информации, информационный машинный язык служит для описания свойств объектов или явлений, информация о которых хранится в машине.

Например, простейший информационный машинный язык, так называемый дескрипторный язык без грамматики, учитывает один вид отношений, **а** именно отношение эквивалентности (синонимии) между различными терминами.

В более развитых информационных машинных языках учитываются отношения подчиненности дескрипторов, а также смысловые связи между дескрипторами.

Основные трудности, возникающие при введении смысловых связей в информационный машинный язык, обусловлены необходимостью обеспечения однозначности представления смысловых отношений между дескрипторами при индексировании исходных документов и запросов.

Примером информационного машинного языка является язык, описанный в § 21, использующий 8 указателей смысловых связей. В литературе описан (например, ИПС «БИТ» [21]) более сложный язык и возможности его использования для описания свойств объектов.

В состав рассматриваемого информационного машинного языка входят помимо упомянутого словаря дескрипторов и набора смысловых связей также ряд разделителей (круглые скобки, запятая, точка с запятой), используемых при построении поисковых образов объектов и запросов.

Кроме того, вводится ряд правил (синтаксис языка), определяющих порядок построения сложных понятий из дескрипторов, смысловых связей и разделителей.

Помимо рассмотренных раньше 8 смысловых связей в состав информационного машинного языка могут входить, например, следующие связи: **быть субъектом, быть объектом, быть средством, быть причиной, быть позже, быть раньше, быть одновременно, иметь материалом, иметь средой, иметь объектом, иметь частью, иметь значением, иметь местом, иметь свойством** и т. п.

Некоторые из указанных смысловых связей в определенных контекстах являются эквивалентными или близкими по значению.

Несколько дескрипторов, соединенных знаками смысловых связей, образуют выражение, представляющее собой сложное понятие. Выражение заключается в круглые скобки и выступает в дальнейшем как одна величина. При перечислении нескольких несвязанных между собой величин (понятий) они разделяются запятыми.

В конце списка величин, относящихся к одному предложению, ставится точка с запятой, что означает конец предложения. Величины, в том числе и сложные (заключенные в скобки) могут связываться между собой смысловыми связями в пределах предложения.

Предложение является автономной информационной единицей; различные величины могут рассматриваться только в пределах предложения; различные предложения независимы друг от друга.

Смысловые связи всегда связывают два дескриптора (или выражения) и направлены от первой компоненты ко второй; при этом первая компонента играет роль, указанную заданной связью по отношению ко второй компоненте. Если необходимо связать несколько величин с другой величиной (или несколькими другими величинами) одинаковой связью, то соответствующие величины заключаются в/круглые скобки; внутри скобок они разделяются запятыми или также смысловыми связями (если это необходимо).

Введение перечисленных смысловых связей преследует три цели:

а) сокращение объема словаря за счет исключения из него словосочетаний и сложных понятий (кроме устойчивых),

б) обеспечение возможности более точного и четкого представления поисковых образов объектов и запросов,

в) обеспечение возможности полной, точной и единообразной записи фактографических сведений.

Ассоциативный словарь понятий (АСП). Рассмотрим некоторый способ организации поисковой информации об объектах, называемый ассоциативным словарем понятий (АСП).

Назначением АСП является опознавание понятий, представленных на информационном машинном языке, и выдача кодов этих понятий или (что то же самое) адресов вершин узловых списков, соответствующих опознаваемым понятиям. Опознаваемые понятия должны представлять собой последовательности следующих символов, принимаемых для АСП в качестве основных:

1) Коды дескрипторов.

2) Коды смысловых связей.

3) Коды закрывающей и открывающей круглых скобок, которые используются для ограничения выражений.

4) Код запятой, используемый для разделения независимых величин (дескрипторов или выражений), входящих в состав одного более сложного выражения.

5) Код точки с запятой, используемый для указания конца предложения.

Понятие в АСП строится по единообразной схеме и представляет собой совокупность, состоящую из главного члена и дополнений, относящихся к этому главному члену. Главный член пишется первым и может представлять собой дескриптор или выражение. Если главный член понятия является выражением, то он должен быть заключен в круглые скобки. За главным членом указывается символ смысловой связи, а за ним дескриптор (выражение), связанный этой связью **с** главным членом. Затем ставится снова символ смысловой связи и следующий дескриптор, причем этот символ связи и дескриптор относятся также к главному члену, и т. д. Таким образом, каждая пара — символ связи и дескриптор — образует одно дополнение главного члена. Если какое-либо дополнение само представляется не дескриптором, а выражением, то это выражение должно заключаться в круглые скобки; внутри же этих круглых скобок само это выражение также должно строиться по описываемой общей схеме

Последовательность дескрипторов (или выражений) одного уровня (по отношению к круглым скобкам) соединяется символами смысловых связей. При этом все эти связи и стоящие за ними дескрипторы (выражения) связывают первый (главный) член ПОНЯТИЙ с каждым из последующих. Устанавливается определенный порядок приоритетности (старшинства)

смысловых связей, в котором они должны следовать при перечислении дополнений, относящихся к одному главному члену.

В последовательности дополнений, относящихся к одному главному члену, естественно, могут отсутствовать многие смысловые связи, но имеющиеся там связи должны располагаться в порядке приоритетности. Этот порядок необходим для упрощения алгоритма опознавания понятий.

В поисковый образ объекта будут входить, как правило, несколько предложений (понятий); при записи они разделяются точками с запятой; при поиске каждое из понятий опознается в АСП независимо и дает отсылку к своему объектному цепному списку.

Перейдем к вопросу машинной реализации АСП.

Для каждого дескриптора, который является главным членом некоторого понятия, в АСП строится вершина в дереве, точнее в поддереве, все эти поддеревья в общем связаны в одно дерево. Каждое поддерево начинается заголовком, т. е. отдельной ячейкой, адрес которой равен коду главного члена определяемого понятия. Формат заголовка понятия (т. е. структура ячейки) имеет следующий вид:

A_o	$A_в$	$A_п$

A_o — адрес определения, т. е. адрес вершины поддерева, в котором один из путей является определением главного члена данного понятия, если этот член сам является выражением; если главный член—простой дескриптор, на месте A_0 ставится код этого дескриптора.

$A_в$ — адрес вертикальный, т. е. адрес, отсылающий к вершине ответвляющегося подсписка, содержащего коды символов, которые могут следовать за данным главным членом понятия. Как правило, это будет подсписок, содержащий коды смысловых связей;

$A_п$ — адрес перехода, т. е. адрес, отсылающий к узловому списку, соответствующему главному члену определяемого понятия.

От заголовка ответвляется поддерево, представляющее собой семейство понятий, начинающихся с одного и того же главного члена. Отдельные пути в этом поддереве представляют отдельные понятия. Каждое понятие — каждый путь должен заканчиваться фиксатором, т. е. ячейкой, имеющей такую же структуру как и заголовок. Вообще каждый заголовок понятия может являться одновременно фиксатором (т. е. конечной ячейкой) другого понятия, представляющего собой главный член данного понятия. Таким образом, одна и та же ячейка является фиксатором для пути дерева, начинающегося с некоторого более общего понятия, и является заголовком для нижестоящего поддерева, представляющего собой конкретизацию данного понятия.

Очевидно, что другого отдельного поддерева, которое бы также имело данное понятие своим главным членом в рассматриваемом словаре понятий, быть не должно.

При построении понятий считается, что все дополнения, находящиеся на участке пути данного дерева, заканчивающемся данным фиксатором, образуют одно единое понятие, которое будет главным членом понятий, образуемых продолжениями этого пути (через $A_в$ фиксатора).

Рассмотрим подробнее назначение адреса $A_п$. Каждое понятие, представленное в АСП некоторым путем (или участком пути), играет роль сложного дескриптора, используемого для описания или классификации объектов. Каждому такому дескриптору должен соответствовать свой цепной список в ассоциативной узловой структуре. Ассоциативные узлы располагаются на магнитных лентах; каждый узел соответствует одному объекту и характеризуется адресом начальной ячейки узла и количеством ячеек в узле.

Для указания начала (вершины) цепного объектного списка нужно задать адрес начальной ячейки первого узла в этом списке и номер спискового слова в этом узле, соответствующего данному цепному объектному списку или просто задать полный адрес соответствующего спискового слова в первом ассоциативном узле данного списка. Это делается с помощью адреса $A_п$.

Следует заметить, что описываемые детали построения списковых слов, заголовков и других величин, так же как и процессы работы АФИЛС (алгоритмы поиска и опознавания понятий и др.) являются одним из возможных вариантов организации подобных систем и здесь имеется большое поле для различных модификаций и усовершенствований таких систем. Данный материал излагается с некоторой ориентировкой на ЭВМ «Минск-22», на которой проводились эксперименты с подобной АФИЛС.

Для формирования ответвляющихся подсписков, образующих дополнения для главных членов понятий, используются списковые слова (ячейки) двух типов:

1) операционные слова, содержащие коды смысловых связей;
2) дескрипторные слова, содержащие коды дескрипторов или сложных дескрипторов.

Заметим, что сложный дескриптор и выражение в данном случае представляют одно и то же.

Операционное слово (ячейка) состоит из следующих частей:

а) кода смысловой связи (КСС, 12 разрядов);
б) кода адреса вертикального, т. е. адреса вершины ответвляющегося подсписка ($A_в$, 12 разрядов);
в) кода адреса горизонтального, т. е. адреса продолжения данного подсписка ($A_г$, 12 разрядов);
г) кода признака операционного слова (ПС, 1 разряд, равный нулю).

Дескрипторное слово (ячейка) состоит из следующих частей:

а) кода дескриптора (КД, 12 разрядов);
б) кода адреса вертикального, т. е. адреса вершины ответвляющегося подсписка ($A_в$, 12 разрядов);
в) кода адреса горизонтального, т. е. адреса продолжения данного подсписка ($A_г$, 12 разрядов);
г) кода признака дескрипторного слова (ПС, 1 разряд, равный единице).

Концы списков обозначаются кодами КС (все единицы) на местах соответствующих адресов связи.

При вертикальном следовании, т. е. сверху вниз по цепочке вершин дерева, соответствующих сложному понятию, могут образовываться промежуточные понятия. Для их фиксации используются фиксаторы, т. е. слова типа заголовков.

Сущность алгоритма опознавания понятий с помощью АСП.

Общий алгоритм опознавания понятий с помощью АСП состоит из двух основных частей.

1) алгоритма опознавания понятий при вводе в систему новых сообщений;

2) алгоритма опознавания понятий при поиске объектов по запросам.

Оба эти алгоритма имеют общую основу и начальные участки, но различаются своими вторыми половинами. Рассмотрим их общую основу, которой является алгоритм прослеживания дерева понятий.

Как при вводе в систему новых сообщений, так и при вводе запросов на поиск данных всегда должен быть задан поисковый образ объекта, который представляет собой набор законченных понятий — предложений. Все дескрипторы, смысловые связи и разделители, входящие в поисковый образ объекта, заменяются кодами, и, таким образом, этот образ вводится в ЭВМ в виде последовательности кодов.

Первым символом (не считая открывающих скобок — одной или нескольких) будет являться код дескриптора. По этому коду, как по адресу, обращаемся к ячейке, являющейся вершиной поддерева семейства понятий, для которых данный дескриптор является главным членом. В этой ячейке выбираем адрес $A_в$ и по нему обращаемся к ответвляющемуся подсписку, который будет подсписком смысловых связей. Выбираем второй символ поискового образа и сравниваем его последовательно с символами смысловых связей в подсписке, пока не находим совпавший символ. В найденной таким образом операционной ячейке выбираем $A_в$ и по нему обращаемся к дескрипторному подсписку, ответвляющемуся от данной смысловой связи.

Выбираем следующий символ в поисковом образе (который будет кодом дескриптора) и сравниваем его последовательно с кодами дескрипторов (понятий) в данном подсписке, пока не находим совпавший и т. д. При отсутствии в заданном поисковом образе внутренних скобок указанный процесс движения по дереву будет идти таким образом до исчерпания данного предложения в поисковом образе, или, до окончания (обрыва) вертикальной прослеживаемой ветви (пути) в дереве, или до того момента, пока при прослеживании какого-нибудь подсписка (операционного или дескрипторного) не произойдет совпадения очередного сравниваемого символа поискового образа ни с одним членом этого подсписка.

Эти три случая (окончание рассматриваемого предложения в поисковом образе, обрыв ветви дерева и отсутствие очередного символа поискового образа в прослеживаемом подсписке дерева) будут иметь различные продолжения для случая включения новых сообщений и для случая поиска данных по запросу, и мы их рассмотрим отдельно ниже.

Сейчас рассмотрим общую часть алгоритма для случая, когда в процессе опознавания понятия в поисковом образе будут встречаться внутренние скобки. Это означает, что в качестве некоторого дополнения к главному члену выступает не дескриптор, а сложное понятие, которое мы будем называть условно внутренним понятием. Внутри этого внутреннего понятия могут встретиться также скобки, т. е. свои внутренние понятия, и т. д.

При встрече каждого внутреннего понятия, т. е. внутренней открывающей скобки, процесс опознавания переходит на новый уровень рекурсии. Это значит, что алгоритм опознавания начинает работать сначала. Для этого необходимо запомнить (занести в СТЭК) адрес места останова, т. е. адрес вершины дескрипторного подсписка в дереве, который должен был бы сейчас прослеживаться [*], а затем начать сначала весь процесс опознавания понятия, но уже для внутреннего понятия.

По коду главного члена этого внутреннего понятия, как по адресу обращаемся к вершине соответствующего поддерева понятий и начинаем прослеживать это поддерево, используя последовательность основных символов внутреннего понятия. Если внутри этого внутреннего понятия будет снова встречено свое внутреннее понятие (т. е. опять открывающая скобка), то опять нужно перейти на новый уровень рекурсии, т. е. снова запомнить место останова и начать процесс опознавания для этого нового внутреннего понятия и т. д. В конце концов для самого внутреннего понятия удастся дойти до первой закрывающей скобки (мы рассматриваем сейчас случай полного совпадения символов в данном понятии поискового образа и прослеживаемой ветви поддерева), т. е. закончить опознавание самого внутреннего понятия. При этом будет получен код этого внутреннего понятия, представляющий собой, как об этом говорилось раньше, адрес ячейки, являющейся фиксатором этого внутреннего понятия. После этого необходимо осуществить возвращение на предыдущий уровень рекурсии. Полученный код опознанного внутреннего понятия вставляется на место этого внутреннего понятия в поисковый образ, т. е. вместо соответствующего выражения в круглых скобках, а из СТЭКа берется самый верхний адрес останова (т. е. последнего останова).

Используя этот адрес, начинаем прослеживать этот дескрипторный подсписок, используя в качестве сравниваемого символа образа код найденного внутреннего понятия. Предположим, что такой код будет найден в прослеживаемом подсписке и от соответствующего дескрипторного слова будет совершен следующий вертикальный переход к операционному подсписку. В этом подсписке будет найден очередной (для поискового образа) символ смысловой связи и процесс будет продолжен до тех пор, пока не будет встречена следующая (вторая) закрывающая скобка. Это значит, что будет опознано второе (считая от самого внутреннего) внутреннее понятие, и после этого делается переход на предшествующий уровень рекурсии и т. д. Обратные рекурсивные переходы будут совершаться до тех пор, пока не будет исчерпан весь СТЭК и не будет опознано самое внешнее понятие; адрес $A_п$ в фиксаторе этого понятия даст адрес заголовка соответствующего объектного списка.

Этот рекурсивный процесс прослеживания поддеревьев понятий будет общим как для процесса включения новых сообщений, так и для поиска данных, поскольку и в том и в другом случае требуется сначала отыскать (или включить) соответствующие объекты по их поисковым образам.

После того как произведено опознавание всех понятий (предложений), входящих в состав поискового образа объекта, процесс может идти по двум направлениям:

а) при запросе производится прослеживание найденных для опознанных понятий объектных списков и поиск нужных

[*] Внутренние скобки могут встретиться только после символа смысловой связи или после скобки.

узлов (объектов);

б) при вводе в систему нового объекта производится включение нового узла во все найденные объектные списки и формирование новой записи со справочной информацией об объекте.

При вводе новых данных об объекте, уже имеющемся в системе, производится поиск узла этого объекта, как в пункте *а,* и затем добавление (или исключение) сведений в запись этого объекта.

Различные варианты продолжения алгоритма будут иметь место при поиске и при включении в случае несовпадения символов сравниваемых понятий в каком-то месте процесса опознавания.

При включении нового объекта, если произошло в каком-то подсписке (операционном или дескрипторном) несовпадение символов, в простейшем случае производится просто наращивание дерева за счет очередных символов поискового образа объекта и образование нового объектного списка, в который и включается этот объект (наряду с включением его в другие уже имеющиеся объектные списки).

При поиске по запросу случай отсутствия очередного символа запроса в прослеживаемом подсписке требует выполнения более сложных процедур.

Здесь можно различать три случая:

1) отсутствие символа смысловой связи;
2) отсутствие символа дескриптора;
3) отсутствие символа внутреннего понятия.

При отсутствии символа связи делается попытка заменить этот символ в запросе через эквивалентный, который может указываться автором запроса или определяться автоматически самой машиной при помощи таблицы эквивалентности смысловых связей.

При отсутствии дескриптора:

а) делается попытка заменить его одним из эквивалентных (если они также указаны автором запроса);

б) делается попытка привести этот термин запроса к главному члену предшествующего (ближайшего внешнего) понятия, используя таблицу эквивалентных преобразований двух последовательных смысловых связей в одну связь;

в) если это приведение не дает результата, т. е. нет совпадения и на предшествующем уровне дерева, то делается попытка привести этот термин к главному члену следующего внешнего понятия. Это выполняется на основе таблицы приведения связей. Процесс идет до тех пор, пока этот термин не будет приведен непосредственно к самому внешнему главному члену. Если и здесь не будет найден его эквивалент, то этот термин (понятие) рассматривается как самостоятельный дескриптор, который даст начало своему объектному списку, и попытка его учета будет сделана при прослеживании объектных списков.

После всех приведений связей должна остаться связь между этим понятием и самим объектом, т. е., по существу, не смысловая связь между двумя понятиями, а указатель роли этого понятия в описании объекта. В ассоциативном узле соответствующего объекта также может стоять дополнительный код такого (или более общего) указателя роли. Очевидно, что по мере приведения связей эти связи приобретают все более общий и более слабый характер. Аналогичный процесс опознавания будет иметь место и для случая отсутствия символа понятия в прослеживаемом подсписке.

Подобный процесс эквивалентных преобразований смысловых связей и приведения понятий может быть использован и при включении в систему новых объектов. В этом случае (при несовпадении некоторых символов) сначала должна делаться попытка сблизить эти понятия путем указанных преобразований и в случае неудачи — производится наращивание поискового дерева понятий и образование нового объектного списка.

Как уже говорилось, поисковые образы объектов строятся в виде набора нескольких автономных предложений, каждое из которых выражает законченное понятие. Все понятия объединены описанным выше способом в ассоциативный словарь понятий. Кроме АСП в поисковый массив входят еще ассоциативные узловые структуры. Каждое списковое слово в каждом узле соответствует какому-либо понятию из АСП, и, таким образом понятия АСП играют роль сложных дескрипторов в обычных ассоциативно-адресных ИПС дескрипторного типа. В ассоциативных узлах каждое из списковых слов может иметь при себе также указатель роли, определяющий роль этого понятия по отношению к объекту. Здесь могут использоваться те же символы смысловых связей.

Основным критерием разделения поискового образа объекта на понятия является четкая фиксация различных аспектов его описания: каждое понятие должно соответствовать определенному аспекту описания объекта. Естественно, что любое поисковое дерево, как и вообще любая система классификации, строится по одному какому-то аспекту; поэтому одно поисковое дерево может служить для формирования одного понятия, описывающего данный объект. Использование ассоциативных узлов позволяет объединять различные понятия, необходимые для описания данного объекта в различных аспектах и представленные различными путями в АСП.

Литература

1. Ляпунов А. А. О логических схемах программ. «Проблемы кибернетики», 1958, вып. 1, стр. 46—74.
2. Бэкус Дж. и др. Сообщение об алгоритмическом языке АЛГОЛ-60. Журнал вычислительной математики и математической физики, 1961, т. 1, № 2, стр. 308—342.
3. Ледли Р. Программирование и использование цифровых вычислительных машин. Пер. с англ. Изд-во «Мир», 1966.
4. Китов А. И. и КриницкийН. А. Цифровые вычислительные машины и программирование. Физматгиз, 1961.
5. Ющенко Е. Л. Адресное программирование. Государственное изд-во технической литературы УССР, 1963.
6. Лавров С. С. Универсальный язык программирования. Изд-во «Наука», 1964.
7. Ингерман П. Синтаксически-ориентированный транслятор. Пер. с англ. Изд-во «Мир», 1968.
8. Китов А. И. Программирование информационно-логических задач. Изд-во «Советское радио», 1967.
9. Китов А.И. Основные принципы построения информационно-поисковой системы (ИПС) для медицины. В сб. «Цифровая вычислительная техника и программирование», № 6, изд-во «Советское радио», 1970.
10. Бусленко Н. П. Теория больших систем. Изд-во «Наука», 1969.
11. Bobrov D. G. and Raphael B. A comparison of list-processing computer languages. Communications of the A.C.M. 1964, v. 7, No. 4.
12. Prywes N. S. and Gray H. J. The organisation of a multilist-type assotiative memory. IEEE Trans., 1963, Sept.
13. Cooper D. C and Whifie1d H. Computer Journ. 1962, v. 5, No 1 ALP: An autocide list-processing language.
14. Landauer W. I. and Prywes N. S. A growing tree for descriptor language translation. Sambol language Data Process, New-York-London, Cordon and Breach, 1962, p. 153-172.
15. Newe11 A. Tonge F. M. An introduction to the IPL-V. Communications A.C.M., 1960, v. 3, No 4.
16. McCartthy J. Recursive of symbolic expressions and their computation by machine, P.I. Communications A.C.M. 1960, v. 3, No. 4.
17. Ge1ertner H. and oth. A fortran-compiled list-processing language. Journ. Assotiation for Computing Machinery, 1960, v. 7, No. 2.
18. Шиллер Ф. Ф. Алгоритмический язык описания экономико-математических задач — АЛГЭМ. В сб. «Цифровая вычислительная техника и программирование», № 1, Изд-во «Советское радио», 1967.
19. Королев М. А. и др. Алгоритмический язык для экономических расчетов — АЛГЭК. Кибернетика, 1966 г. № 4.
20. Бородулина Н. Г. и др. Система автоматизации программирования АЛГЭМ. Под ред. А. И. Китова. Изд-во «Статистика», 1970.
21. Информационно-поисковая система «БИТ» АН УССР, Киев, 1968