МИНИСТЕРСТВО
ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ СССР

МОСКОВСКИЙ
ОРДЕНА ЛЕНИНА И ОРДЕНА ОКТЯБРЬСКОЙ РЕВОЛЮЦИИ
АВИАЦИОННЫЙ ИНСТИТУТ имени СЕРГО ОРДЖОНИКИДЗЕ

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ПОДСИСТЕМ САПР ЛА

Под редакцией д-ра техн. наук проф. О.Л. Смирнова

Учебное пособие

Утверждено
на заседании редсовета
21 июня 1982 г.

МОСКВА 1983

УДК:629.7.0I68I.3.06(075.8)

Авторы: А.В. Вестяк, В.А. Столярчук, В.Ф. Формалев,
В.М. Ягодин

Информационное обеспечение подсистем САПР ЛА: Учебное пособие / Вестяк А.В., Столярчук В.А., Формалев В.Ф, Ягодин В.М.; Под ред. О.Л. Смирнова. - М.: МАИ, I983. - 83 с., ил.

В учебном пособии рассматриваются следующие вопросы: основные концепции банков данных, способы организации банков данных, различные виды сортировок информационных массивов, методы поиска информации в банках данных, основные свойства банков данных СИОД, НСИ, ИНЭС-2, применение алгоритмического языка ПЛI для организации информационных систем.

Пособие предназначено для студентов старших курсов, аспирантов и слушателей СФПК, специализирующихся в области разработки и использования систем автоматизированного проектирования (САПР).

Рецензенты: В.В. Белов, В.М. Лихачев.

© Московский авиационный институт, I983 г.

А I8(075)
И 74I

ПРЕДИСЛОВИЕ

В последние годы в нашей стране и за рубежом интенсивно разрабатываются банки данных различного направления. Основная область применения банков данных – автоматизированные системы управления технологическими процессами, предприятиями, объединениями, отраслями и системы автоматизированного проектирования. В данном учебном пособии рассматриваются вопросы разработки информационного обеспечения САПР, включающего в себя совокупность специальным образом организованных данных, циркулирующих на различных уровнях САПР и находящихся под управлением системы управления базой данных (СУБД). При использовании баз данных сокращается время поиска данных благодаря специальным средствам организации сложных структур данных, наличию программных средств поиска и устранению избыточности данных при хранении и появляется возможность контролируемого доступа к данным с целью обеспечения защиты от несанкционированного доступа к ним, что является немаловажным для повышения эффективности, надежности и качества функционирования САПР. Важнейшими преимуществами использования концепции банков данных являются возможность легкого доступа пользователей к данным, гибкость в обращении с данными благодаря различным методам доступа, высокая производительность, быстрая обработка произвольных запросов (на данные), простота внесения изменений, возможность взаимодействия пользователя с системой на непроцедурном языке запросов и т.д.

Главы I и 4 написаны В.А. Столярчуком и В.М. Ягодиным, гл. 2-3 – А.А. Вестяком и В.Ф. Формалевым, гл. 5 написана В.М. Ягодиным.

I. ЭЛЕМЕНТЫ ИНФОРМАЦИОННОГО ОБЕСПЕЧЕНИЯ САПР

I.I. ОСНОВНЫЕ КОНЦЕПЦИИ БАНКОВ ДАННЫХ

В начале процесса проектирования технического объекта разработчик, как правило, располагает сравнительно небольшим количеством информации. В сущности, проектирование начинается с возникновения мысли о необходимости конкретного технического объекта, формализуемого в конечном итоге небольшим количеством желаемых (задаваемых) параметров. Постепенно в процессе проектирования количество получаемой информации увеличивается, причем часть ее отбрасывается как ненужная или не удовлетворяющая проектировщика по тем или иным соображениям. Однако общее количество информации все время возрастает, и конечным результатом процесса проектирования может быть весьма большое количество информации. Например, в результате проектирования такого сложного технического объекта, как самолет, появляется огромное количество чертежей, описаний, инструкций и т.п. документов, необходимых для создания первого опытного экземпляра изделия. Таким образом, процесс проектирования можно трактовать как процесс накопления информации, а систему проектирования – как систему накопления и обработки информации. Система автоматизированного проектирования, представляющая собой коллектив проектировщиков, взаимодействующих с комплексом средств автоматизации проектирования, отличается прежде всего режимом использования ЭВМ в процессе проектирования. Это отличие от традиционных средств решения задач с помощью ЭВМ заключается в наличии развитого диалогового режима для конечных пользователей, не владеющих (или слабо владеющих) какими-либо языками программирования.

Одной из причин внедрения диалогового режима является необходимость оперативного задания исходных данных и запоминания информации при работе с программным обеспечением САПР. Участие в процессе проектирования коллектива проектировщиков, каждому из которых должен быть обеспечен доступ к имеющейся информации, приводит к мысли о выделении накапливаемой информации в отдельную подсистему САПР. Таким образом, возникает идея об отделении дан-

ных от программ, приводящая к созданию банков данных, в которых должна храниться необходимая для проектировщиков информация. Но сложность задач, которые приходится решать проектировщику, и насущные потребности процесса проектирования требуют от банков данных значительно большего, чем обеспечение простого "склада" пусть даже упорядоченной информации. Например, одной из задач, которые приходится решать специалисту на начальном этапе проектирования самолета, является поиск некоторой статистической информации, причем не по всем самолетам, а только по самолетам одного класса с разрабатываемым. Отсюда и более сложная задача выборки информации, которую вряд ли имеет смысл решать программными средствами самого проектировщика. Еще пример. Традиционной задачей конструктора является подбор профиля из ограниченного (например, технологами) набора профилей, причем профиля, имеющего, допустим, заранее заданную площадь сечения. Понятно, что подобные требования к банку профилей требуют решения весьма специфичных задач, которые опять-таки вряд ли целесообразно поручать конструктору. Конечно, приведенные примеры не слишком сложны, если рассматривать отдельно банк профилей, отдельно банк материалов, отдельно банк прототипов и т.д. Но инженерные конструкции как объекты проектирования описываются множеством параметров, включающих:

- информационные и математические модели;
- библиотеки модулей типовых деталей, звеньев и механизмов;
- нормали и ГОСТ;
- библиотеки чертежей и т.д.

Очевидно, что такая информация характеризуется прежде всего колоссальным объемом и сложной структурой. Ориентирование в таком объеме информации, не говоря уже о выборе нужной с учетом связей между данными, представляет для конкретного пользователя при традиционных методах поиска весьма трудоемкую задачу, могущую если не остановить, то существенно замедлить процесс проектирования. Поэтому при автоматизации процесса проектирования становится необходимой и автоматизация поиска и выдачи требуемой информации с тем, чтобы не разрывать (не останавливать) цикла автоматизированного проектирования. Эту задачу решает специальная подсистема САПР - подсистема информационного обеспечения, включающая в себя не только некие "склады" специальным образом упорядоченной информации, но и системы управления базой данных, обеспечивающие решение огромного многообразия информационных задач, возникающих в

процессе проектирования. Итак, под информационным обеспечением САПР понимается совокупность цифровых, алфавитно-цифровых и графических данных, циркулирующих на различных уровнях САПР и отражающих состояние объекта проектирования в виде исходной, промежуточной или результирующей информации, организованной в автономные массивы данных и находящейся под управлением СУБД. Кроме того, в информационное обеспечение входят система документации, система кодирования, информации и классификации объектов проектирования, а также другие инструктивно-методические материалы.

Способы хранения, поиска и обработки информации в современных САПР приобретают первостепенное значение и оказывают определяющее влияние на структуру и принципы функционирования таких систем в целом. С помощью системы управления базами данных реализуются преимущества централизованного накопления и хранения больших объемов информации для многократного использования ее при решении задач проектирования. При этом отпадает необходимость дублирования данных в локальных базах данных.

Историческая справка. В истории развития систем представления данных во внешней памяти ЭВМ можно выделить несколько этапов.

Этап I. На этом этапе создание, управление и организация доступа к данным полностью обеспечивались программами пользователей, создавших для себя эти данные. Использование таких данных в программах других пользователей было крайне затруднено, да и сам процесс обмена информацией между оперативной и внешней памятью был малоэффективен.

Этап II. Управление данными и организация доступа к ним частично стали обеспечиваться операционными системами.

Этап III. В течение этого периода на основе опыта разработки узкоспециализированных систем хранения и обработки информации постепенно формировались требования к СУБД широкого профиля, появились первые универсальные СУБД.

Этап IV. (Начало 70-х годов.) Создаются интегрированные базы данных и развитые СУБД. На этом этапе произошло осмысливание накопленного опыта и окончательное формирование требований к универсальным СУБД. Были разработаны и реализованы различные подходы к представлению данных в базах данных.

В СУБД появились средства высокого уровня: языки описания данных (ЯОД) и языки манипулирования данными (ЯМД), дополняющие традиционные языки программирования, для организации больших баз

данных сложной структуры, управления ими и поиска необходимой информации.

Этап У. Создание распределенных банков данных коллективного пользования, специально разработанных для систем автоматизированного проектирования в качестве информационного обеспечения.

Сейчас еще нет универсального банка данных, удовлетворяющего всех без исключения пользователей, но разработано большое количество банков, имеющих различные области применения. Эти банки отличаются языками описаний данных, языками манипулирования данными, структурами хранения данных и в силу этого имеют различную эффективность. В этой связи следует отметить тенденцию к универсализации и стандартизации языков описаний данных и манипулирования данными, проявившуюся, например, в предложениях рабочей группы по базам данных Системного комитета КОДАСИЛ [21, 33].

Основные концепции универсальных банков данных. В развитии информационных систем можно выделить два подхода. Первый подход – специализация. По этому пути шло создание документальных и фактографических информационно-справочных систем. Характерным признаком таких систем является жесткое задание структур данных. Второй подход – универсализация. По этому пути пошли разработчики универсальных систем управления базами данных. Введем определения.

Банк данных – это система программных, языковых, организационных, технических средств и обслуживающего персонала, предназначенная для централизованного накопления и коллективного использования данных. Система управления базой данных – это совокупность языковых и программных средств, предназначенных для создания, ведения и коллективного использования данных.

Язык манипулирования данными используется для поиска данных в базе данных, а также для модификации данных, модификации структуры данных (реструктуризация). Существуют различные реализации ЯМД. В частности, ЯМД может быть автономным, не зависящим от конкретных языков программирования. Наибольшее распространение имеют ЯМД, реализованные в виде операторов обращения к процедурам, написанным на алгоритмическом языке, обеспечивающим операции обмена прикладной программы с банком данных.

Язык описания данных используется для описания структур данных и взаимосвязей между элементами данных. От выбора структуры данных в значительной мере зависит эффективность работы всего

банка данных. В банках данных процесс описания данных отделен от процесса обработки (поиска) их.

На рис. I.I представлена общая структура банка данных.

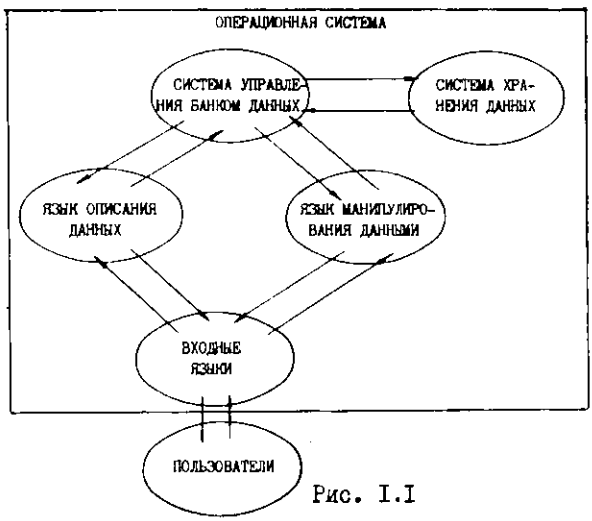


Рис. I.I

Перечислим основные подходы к описанию структур данных, хранящихся в банках данных.

Иерархический подход. В этих банках структуры данных представляются в виде иерархических структур. Недостатком такого подхода является то, что описание реальных информационных объектов в виде древовидных структур не всегда возможно (неадекватно информационной модели объекта).

Сетевой подход. В таких банках допускают описание произвольных связей между элементами сетевой структуры.

Реляционный подход. В работе [32] был впервые предложен теоретико-множественный подход к описанию информационных моделей объектов. Этот подход является основой реляционного способа представления данных. Реляционные базы данных определяются как совокупность прямоугольных информационных таблиц (ИТ), каждая из которых логически описывается в виде плоского двумерного файла с одним типом записи.

Важнейшей концепцией банков данных является независимость данных. Различают логическую и физическую независимость данных [I0, I9].

Логическая независимость данных. В системе создается (генерируется) набор физических баз данных. Каждый пользователь в соответствии со своей задачей выбирает нужный ему набор физических баз данных или их частей, т.е. на основе созданных физических баз данных можно построить любое количество логических баз данных (рис. I.2). Другими словами, каждый пользователь создает свою подмодель данных, которая является подмножеством модели данных. При таком подходе к формированию подмоделей данных существенно уменьшается избыточность данных, что очень важно при использовании больших баз данных многими пользователями.

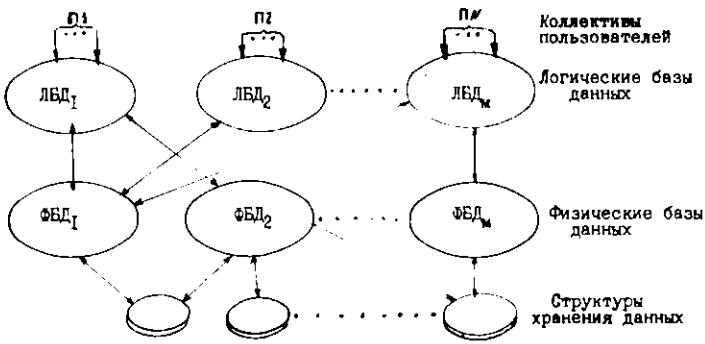


Рис. I.2

Логическая независимость достигается тем, что описания баз данных хранятся отдельно от прикладных программ. Это обеспечивает независимость логического уровня данных от прикладных программ и от физической организации данных на внешних носителях, от методов доступа, от характеристик внешних устройств. Логическую структуру данных можно изменять, не изменяя при этом прикладных программ. Пользователь отделен от структур хранения.

Физическая независимость подразумевает возможность изменять структуру хранения, не изменяя при этом ни прикладных программ, ни логического описания структур данных.

Защита данных - это обеспечение условий, предупреждающих несанкционированный доступ к данным. Можно выделить внешнюю и внутреннюю защиту данных. Внешняя защита предусматривает программные и организационные методы и средства контроля доступа к информации, внутренняя - обеспечивается СУБД.

Контроль за целостностью – предотвращение искажения данных и восстановление данных в случае системного сбоя.

Параллельное использование означает возможность нескольким пользователям обратиться к одним и тем же данным в один и тот же момент времени.

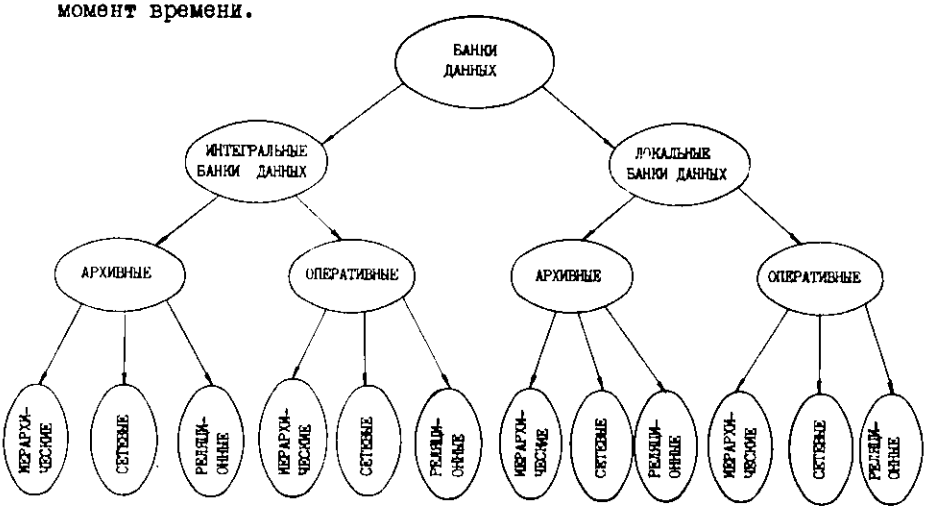


Рис. I.3

При выборе банка данных для САПР необходимо учитывать много факторов: объем данных, сложность структуры данных, сложность и частоту реструктуризации данных, оперативность доступа к данным, конфигурацию ЭВМ, версию операционной системы, количество видеотерминалов, степень секретности данных, необходимый уровень подготовки прикладных программистов, использующих данные, необходимый уровень подготовки системных программистов, поддерживающих работу банка данных, и т.д.

Одна из возможных классификаций банков данных приведена на рис. I.3. Банки данных можно разделить на интегральные и локальные. Под интегральным банком понимается банк, содержащий информацию, необходимую большому коллективу пользователей. Информацию, имеющую частный характер и представляющую интерес для узкого круга проектировщиков, можно хранить в локальном банке данных. Интегральные и локальные банки данных, в свою очередь, можно разделить по частоте обращения к информации на архивные и оперативные. Далее банки данных можно разделить по способам описания данных на иерархические, сетевые и реляционные.

Характеристики банков	Банки данных					
	ОКА	СИНБАД-2	ИНЭС-2	СЕДАН	БАНК-ОС	ПАРМА
Минимальная конфигурация	о/п-128 кбайт НМД-3 НМЛ-1	о/п-128 кбайт НМД-3	о/п-256 кбайт НМД-3	о/п-128 кбайт	о/п-128 кбайт НМД-3	о/п-256 кбайт НМД-3
Рекомендуемая конфигурация	о/п-512 кбайт НМД-4 НМЛ-2	о/п-256 кбайт НМД-3 НМЛ-2	о/п-512 кбайт НМД-4	-	о/п-256 кбайт НМД-3 НМЛ-2	-
Операционная система, версии	ОС-4.0 и выше (MVT, MET)	ОС-4.0 (MVT, MET)	ОС-4.0 и выше (MVT, MET)	ОС-4.0	ОС-4.0 и выше (MVT, MET)	ОС-4.0 и выше
Количество машинных команд	100 тыс.	64,4 тыс.	-	-	10 тыс.	80 тыс.
Режим функционирования	Пакетный	Пакетный	Пакетный Теледоступ	Пакетный	Пакетный	Пакетный
Допустимые логические структуры данных	Иерархическая. Ограниченная сетевая	Иерархическая. Ограниченная сетевая	Сетевые	Сетевые	Сетевые	Сетевые

В таблице приняты следующие обозначения: о/п - оперативная память ЭВМ, НМД - накопитель на магнитном диске, НМЛ - накопитель на магнитной ленте.

В настоящее время за рубежом эксплуатируется большое количество банков данных: TOTAL, IMS, IDMS, ADABAS и др. [24].

Наибольшее распространение получила СУБД TOTAL, которая может использоваться на ЭВМ IBM/360, IBM /370. Система разработана в 1969 г. Связь прикладных программ с базой данных в системе TOTAL осуществляется с помощью вызова подпрограммы. С базой данных можно работать на алгоритмических языках Кобол, ПЛ1, Фортран, Ассемблер. Язык описания структур данных, используемый в системе, отличается от соответствующих предложений, разработанных КОДАСИЛ.

Система IMS разработана фирмой IBM в середине 60-х годов. Система предназначена для работы на ЭВМ IBM/360, IBM /370. Структура данных - иерархическая. С базой данных можно работать на алгоритмических языках Кобол, ПЛ1, Фортран, Ассемблер. Языки описания данных и манипулирования данными отличаются от стандарта, предложенного КОДАСИЛ.

Система IDMS фирмы Cullinane Corp разработана в 1970 г. для ЭВМ IBM/360, IBM/370. ЯОД и ЯМД этой системы соответствуют рекомендациям КОДАСИЛ.

Система ADABAS разработана западногерманской фирмой SOFWARE AG. Эта система предназначена для ЭВМ IBM/360, IBM/370, работающих с операционными системами DOS, OS, VS. Система работает с прикладными программами, написанными на языках Кобол, ПЛ1, Фортран, Ассемблер.

В СССР получили распространение следующие универсальные СУБД, работающие под управлением ОС ЕС ЭВМ: ОКА [16], СИНБАД-2 [28], использующие иерархический подход к организации данных; ИНЭС-2 [3], СЕДАН [22], БАНК-ОС [28], СИОД-ОС [17], ПАРМА [18], использующие сетевой подход к организации данных. Некоторые сведения о перечисленных СУБД приведены в табл. на с. 11.

1.2. СРЕДСТВА ОБРАБОТКИ И ХРАНЕНИЯ ИНФОРМАЦИИ

С развитием систем автоматизированного проектирования будет увеличиваться объем информации, хранящейся в банках данных. Время поиска (ответа) в САПР является очень существенным фактором при диалоговом режиме работы проектировщиков с банком данных. Поскольку объем информации имеет устойчивую тенденцию к возрастанию, то компенсирование неизбежного увеличения времени поиска нужной информации, помимо совершенствования структуры самих банков данных, будет происходить за счет увеличения производительности ЭВМ.

Рост производительности ЭВМ и, следовательно, САПР тесно связан с увеличением быстродействия логических элементов, совершенствованием архитектуры ЭВМ и усовершенствованием внешних устройств, оборудования и линий связи, совершенствованием структуры памяти. Рассмотрим коротко некоторые из этих аспектов.

Быстродействие ЭВМ. При однопроцессорной конфигурации ЭВМ повышение быстродействия возможно главным образом за счет увеличения быстродействия ее логических элементов.

Исследования показали, что существенное увеличение быстродействия ЭВМ может быть достигнуто при реализации качественно новых архитектурных принципов, основным из которых является параллельность выполнения операций.

Одними из самых удачных концепций построения вычислительных систем являются концепции векторных и матричных ЭВМ. Основным обрабатывающим элементом в таких ЭВМ является векторный (матричный) процессор, который представляет собой систему "традиционных" процессоров. Главное отличие векторных процессоров от матричных состоит в том, что в векторных процессорах элементарные процессоры или не связаны между собой или же связаны через общие модули основной памяти. В матричных процессорах элементарные процессоры связаны между собой и образуют двумерную "решетку". В качестве примера может служить ЭВМ ILLIAC-IV с быстродействием до 200 млн опер./с, имеющая многопроцессорный комплекс, состоящий из 64 процессоров [35]. Исследуются также возможности высокопроизводительных конвейерных ЭВМ [24].

Память ЭВМ. Память ЭВМ можно разделить на оперативную и внешнюю. Время обращения к оперативной памяти минимально и составляет, например, для ЕС-1040 0,45 мкс. Но объем оперативной памяти ограничен. В ЕС ЭВМ система адресации допускает максимальный ее объем до 16777216 байт [12]. Поэтому для хранения больших объемов информации используют внешнюю память, где носителями информации обычно являются магнитные диски и магнитные ленты. Время обращения к магнитному диску превышает время обращения к оперативной памяти, зависит от конкретной модели ЭВМ и составляет, например, для серии ЕС ЭВМ от 25 до 180 мс [12]. Отсюда следует, что организация памяти ЭВМ, емкость накопителей и время считывания с них информации существенно влияют на эффективность вычислительного процесса в целом.

В вычислительных центрах СССР получили распространение накопители ЕС-5061, работающие с пакетами сменных дисков ЕС-5261. Емкость этого пакета 29,17 Мбайт [2]. В Народной Республике Болгарии для серии ЕС разрабатывается накопитель ЕС-5066, в котором в качестве носителя информации используется 12-дисковый пакет ЕС-5266. Емкость этого пакета сменных дисков равняется 100 Мбайт [2]. Такая емкость реализуется за счет увеличения числа дисков в пакете до 12, увеличения плотности записи до 160 бит/мм, увеличения количества дорожек до 404. Пакет дисков вращается со скоростью 3600 об/мин. Повышенная частота вращения носителя и плотности записи обеспечивают высокую скорость передачи данных до 806 кбайт/с[2].

В ЭВМ ЕС-1035, ЕС-1045, ЕС-1055 реализованы возможности работы с виртуальной памятью. Объем реальной оперативной памяти равен 256–512 кбайт, а объем виртуальной памяти – до 16 Мбайт. При организации виртуальной памяти в режиме *SVS* используется сегментно-страничный способ [25]. Размер страницы – 2 кбайта, размер сегмента – 64 кбайта. Страницы виртуальной памяти хранятся на магнитном диске в системном наборе данных, называемом страничным набором данных. Адреса, присваиваемые страницам виртуальной памяти при размещении в оперативной памяти, называются реальными адресами. Часть оперативной памяти, используемая для размещения страниц виртуальной памяти, называется реальной памятью. Переадресация виртуальных адресов в реальные обеспечивается системой динамической переадресации, использующей программно-организуемые таблицы переадресации: таблицы сегментов, таблицы страниц. Необходимые пользователю страницы перекачиваются (перелистываются) из внешней памяти в оперативную по мере необходимости.

В настоящее время проводятся исследовательские работы с целью создания принципиально новых средств и сред хранения информации. В ближайшем будущем можно ожидать замену магнитных дисков и лент на более совершенные запоминающие устройства, использующие эффекты в ферромагнитных, полупроводниковых, оптических и других материалах. Одним из наиболее перспективных является голографический метод хранения информации, обладающий большими потенциальными возможностями. Преимущества этого метода заключаются в высокой плотности записи, оцениваемой в 10^8 бит/см^3 для объемных голограмм, низкой удельной стоимости $10^{-3}...10^{-4}$ цент/бит, малом времени считывания страницы данных с объемом более 10^4 бит (1 мкс), высокой помехозащищенности. В объемной среде можно записывать большое ко-

личество голограмм путем их наложения и последующего селективного восстановления. Голографические методы позволяют записывать, хранить и восстанавливать информацию, представленную в виде картин (чертежей), пространственных изображений (например, внешний вид летательного аппарата) и т.д., что имеет большое значение в системах автоматизированного проектирования. Принципы построения когерентных оптических вычислительных машин и способы обработки информации на таких ЭВМ рассмотрены в работе [I].

2. СПОСОБЫ ОРГАНИЗАЦИИ ИНФОРМАЦИИ В БАЗАХ ДАННЫХ И ОБРАБОТКА ИНФОРМАЦИОННЫХ МАССИВОВ

2.I. СПОСОБЫ ОРГАНИЗАЦИИ ИНФОРМАЦИИ В БАЗАХ ДАННЫХ

Логическая организация баз данных

Рассмотрим основные определения при организации информации в базах данных.

Объекты – понятия реального мира, информация о которых сохраняется в базах данных. Объекты могут быть материальными (изделие, служащий, населенный пункт) и нематериальными (идеи, события).

Имя атрибута:	Таб. номер	ФИО	Пол	Дата рождения	Отдел	Код профессии	Должность	Зарплата
Форма представления	N4	AV	B1	N6	N3	N2	AV	N3
	1730	ПЕТРОВ А.И.	1	100335	102	02	ВЕД. ИНЖ	190
	0985	ИВАНОВА Н.А.	0	151139	203	57	СТ. ИНЖ	160
Запись:	2040	СИДОРОВ В.И.	1	080640	601	25	МОНТАЖНИК	200
	1273	ЕГОРОВА О.П.	0	190238	505	73	СТ. ТЕХНИК	110
	1532	ЯНИН А.В.	1	021148	803	45	СНС	200
	1970	ПЕЧНИКОВА М.А.	0	250862	705	92	ТЕХНИК	90
	1678	БОРИСОВ В.Ю.	1	050155	802	21	МНС	120

Идентификатор объекта (ключевой атрибут)

Значения атрибута «Дата рождения»

Рис. 2.I

При обработке данных часто имеют дело с совокупностью однородных объектов, называемых н а б о р о м о б ъ е к т о в. На рис. 2.I представлен набор объектов СОТРУДНИКИ.

Атрибуты – свойства объектов, представленные в базе данных в виде информации об объекте. Атрибутам присваиваются значения.

Элемент данных – простейшее неделимое значение атрибута, имеющее свое наименование.

Например, на рис. 2.I значения атрибута ДАТА РОЖДЕНИЯ - элементы данных, а значения атрибута ОТДЕЛ не являются элементами данных, так как эти значения могут иметь свои описания. В последнем случае значение атрибута называется а г р е г а т о м д а н н ы х.

В качестве идентификатора объекта принимается один из элементов данных, называемый к л ю ч е в ы м э л е м е н т о м д а н н ы х. Идентификатор объекта должен быть уникальным; никакой другой объект не может иметь то же значение элемента данных.

Запись - поименованная совокупность одного или более элементов данных или агрегатов данных. На рис. 2.I любая строка с набором элементов данных является записью сотрудника с уникальным ключевым элементом данных ТАБЕЛЬНЫЙ НОМЕР.

Файл - упорядоченный набор записей одинаковой структуры. Наиболее приемлемой формой файла в базах данных является двумерный (или плоский) файл, представленный на рис. 2.I. Все агрегаты данных, создающие объемный файл, ниже будут представлены в виде системы плоских файлов.

Уникальный элемент данных, являющийся ключевым, называется п е р в и ч н ы м к л ю ч о м. Однако можно использовать ключи, которые идентифицируют не уникальную запись, а группу записей, обладающих общим свойством. Такие ключи называются в т о р и ч н ы м и (и л и д е с к р и п т о р а м и). Например, в качестве вторичного ключа можно выбрать атрибут ПОЛ в задаче выбора записей сотрудников мужского (или женского) пола.

Таким образом, если с помощью первичных ключей можно строить прямые файлы, то с помощью вторичных ключей из прямых файлов можно строить инвертированные файлы, содержащие только записи с заданными значениями вторичных ключей.

Форма представления данных на рис. 2.I указывает на то, имеет ли элемент данных фиксированную длину или переменную, а если фиксированную, то какую. Например:

N4 - десятичное число с фиксированной запятой, состоящее из четырех цифр;

AV - буквенно-цифровой элемент данных переменной длины;

BI - двоичное число длиной в I бит.

Логические записи данных - это записи данных в представлении прикладной программы, прикладного программиста или пользователя.

Физические записи отражают реальные способы хранения информации на внешних носителях.

Например, логическими записями являются записи файла на рис. 2.1. В целях экономии машинного времени физическая запись может быть построена из нескольких логических записей (в виде блока записей), в результате чего обмен между основной и внешней памятью ЭВМ осуществляется физическими записями. В частном случае физической записью может быть одна логическая запись. Из требований, предъявляемых к базам данных, вытекают три типа организации данных [19] (рис. 2.2.

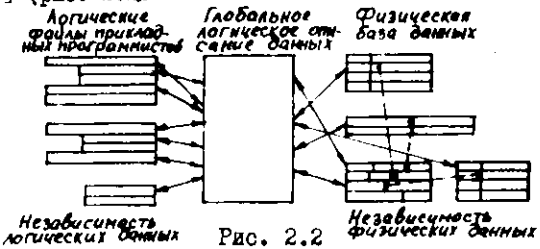


Рис. 2.2

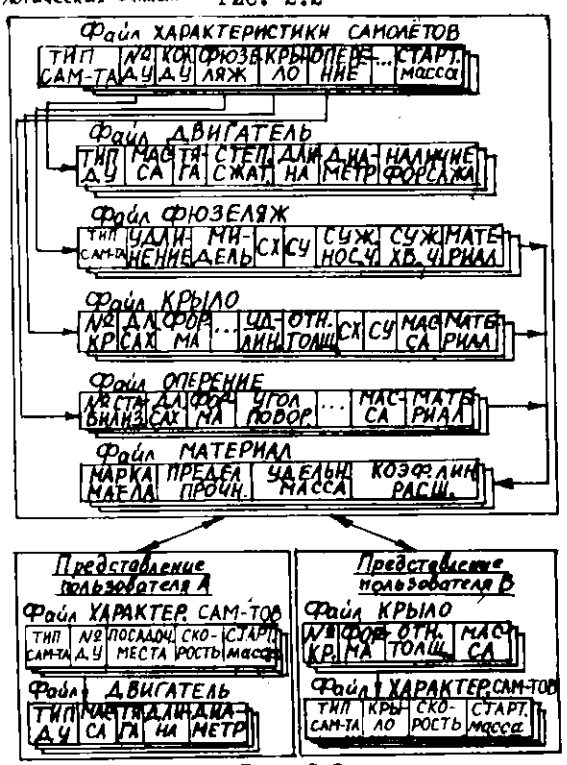


Рис. 2.3

Внешняя организация данных связана с таким представлением данных, каким его понимают прикладные программисты и конечные пользователи. Они описывают свое представление о файлах в прикладных программах.

В соответствии с внешней организацией данных существует таблица, описывающая ту часть данных, которая ориентирована на нужды одной или нескольких прикладных программ (внешние файлы). Такая таблица называется подсхемой.

Глобальная логическая организация данных - это общая организация или концептуальная модель базы данных, на основании которой могут быть получены различные внешние организации. В соответствии с ней существует глобальное описание логической структуры базы данных - таблица, логически описывающая всю базу данных.

Физическая организация - это физическое представление данных и расположение их на запоминающих устройствах. В соответствии с этим существует описание физической организации базы данных - таблица физического расположения данных на носителях информации.

На рис. 2.3 приведено глобальное логическое описание (схема) базы данных ХАРАКТЕРИСТИКИ САМОЛЕТОВ и подсхемы двух прикладных программистов А и В, построенные на основе схемы с помощью средств программного обеспечения.

2.2. СТРУКТУРЫ ДАННЫХ

Совокупность данных, изображенных на рис. 2.1, описывает простейшую структуру в виде двумерного (плоского) файла. Каждая запись в этой структуре имеет одинаковый набор элементов данных или их агрегатов.

Наиболее широко распространена иерархическая организация данных в виде древовидных и сетевых структур.

Древовидная структура данных определяется как иерархия узлов с попарными связями, в которой самый верхний уровень имеет один узел, называемый корнем, а все остальные узлы связываются с одним и только одним узлом на более высоком уровне по отношению к ним самим.

Сбалансированное дерево - это такая древовидная структура данных, в которой каждый узел имеет одинаковое количество ветвей (рис. 2.4).

Несбалансированное дерево - древовидная структура, где каждый узел имеет неодинаковое количество ветвей (рис. 2.5). С пози-

ции физической организации сбалансированное дерево проще, чем несбалансированное.

Двоичное дерево - древовидная структура, в которой допускается не более двух ветвей (рис. 2.4).

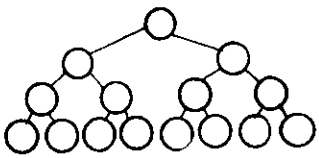


Рис. 2.4

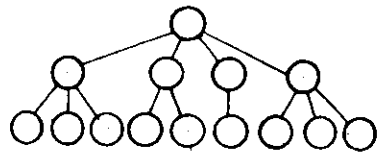


Рис. 2.5

Сетевая структура данных определяется как иерархическая структура с порожденными элементами, имеющими более одного исходного элемента (рис. 2.6).

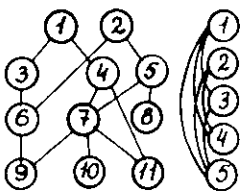


Рис. 2.6

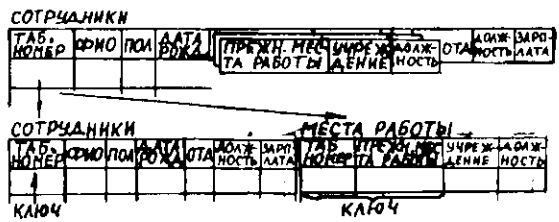


Рис. 2.7

Двумерный (плоский) файл. Совокупность данных, изображенных на рис. 2.1, образует простейшую структуру, называемую двумерным файлом. Каждая запись в этой структуре имеет одинаковый набор элементов данных. Все сложные структуры данных можно привести к двумерным путем введения избыточных элементов данных. Этот процесс называется нормализацией.

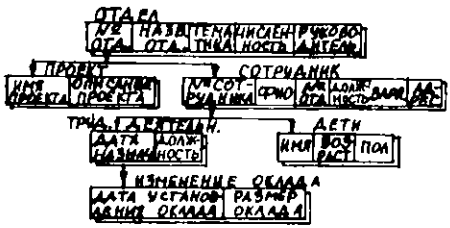
Рассмотрим двумерный файл на рис. 2.1 и предположим, что одним из атрибутов этого файла является агрегат данных, называемый повторяющейся группой, например:

Прежние места работы			
Учреждение	Должность	Учреждение	Должность

Тогда этот файл может быть нормализован путем выделения повторяющейся группы в отдельный двумерный файл (рис. 2.7). Полученному

таким образом новому файлу (таблице) присваивается собственное имя. Этот файл должен иметь ключи, однозначно определяющие любую запись.

Древовидная структура базы данных:

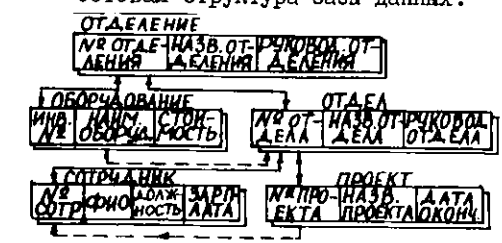


Нормализованная структура:

ОТДЕЛ (№, НАЗВ, ТЕМАТИКА, ЧИСЛЕННОСТЬ, РУКОВОД)
ПРОЕКТ (№ ОТД, ИМЯ ПРОЕКТА, ОПИСАНИЕ ПРОЕКТА)
СОТРУДНИК (№ СОТР, ФИО, № ОТД, ДОЛЖН, ЗАРПЛАТА)
ТРУД. ДЕЯТЕЛЬНОСТЬ (№ СОТР, ДАТА НАЗНАЧ, ДОЛЖН)
ДЕТИ (№ СОТР, ИМЯ РЕБЕНКА, ВОЗРАСТ, ПОЛ)
ИЗМЕНЕНИЕ ОКЛАДА (№ СОТР, ДАТА УСТАНОВ, РАЗМ)

Рис. 2.8

Сетевая структура базы данных:



Нормализованная структура:

ОТДЕЛЕНИЕ (№, НАЗВАНИЕ, РУКОВОДИТЕЛЬ)
ОБОРУДОВАНИЕ (ИНВ. №, № ОТДЕЛЕНИЯ, НАИМ. ОБ, СТОИМ)
ОТДЕЛ (№ ОТД., № ОТДЕЛЕНИЯ, НАЗВ. ОТД., РУКОВОД. ОТД)
СОТРУДНИК (№ СОТР, № ОТД, № ПРОЕК, ФИО, ДОЛЖН, ЗАРП)
ПРОЕКТ (№ ПРОЕКТА, № ОТДЕЛА, НАЗВ. ПРОЕК, ДАТА ОКОНЧ)
РАЗМЕЩЕНИЕ (№ ОТДЕЛА, ИНВ. № ОБОРУДОВ)

Рис. 2.9

В примере на рис. 2.7 можно заметить, что при нормализации увеличивается избыточность: элемент ТАБЕЛЬНЫЙ НОМЕР фигурирует в базе дважды. Однако эта избыточность наблюдается лишь на логическом уровне. На физическом уровне вовсе не требуется, чтобы некоторый элемент данных появлялся более чем в одной записи в качестве ключа.

На рис. 2.8 приведен пример нормализации четырехуровневой древовидной структуры базы данных путем удаления дуг соответствующих графов и добавления в полученный двумерный файл новых ключевых элементов данных (подчеркнуто). На рис. 2.9 приведен пример нормализации сетевой структуры данных. В нормализованном двумерном файле в записи введены вторичные ключи (подчеркнуты) для связи с записями данного уровня или других уровней. Кроме того, для отображения связей между файлами ОТДЕЛ и ОБОРУДОВАНИЕ введена новая запись (последняя).

Необходимо отметить, что работать с древовидными и сетевыми структурами чрезвычайно сложно в больших массивах. Особенно трудно разработать иерархические структуры на физическом уровне, так как значительно усложняется система индексации и адресации связанных записей.

Нормализованные структуры в виде двумерных файлов лишены указанных недостатков. Правда, эти структуры представляются в виде записей переменной длины, и необходимо в физическом описании данных каждой записи присваивать определенную длину. Наиболее важным достоинством нормализованных структур является независимость их внешних логических структур и физического представления от изменения глобальной логической структуры.

2.3. ФИЗИЧЕСКАЯ ОРГАНИЗАЦИЯ БАЗ ДАННЫХ

Физическое представление двумерных файлов, древовидных и сетевых структур можно реализовать, используя следующие методы: физически последовательное размещение, указатели, цепи и кольца, справочники, битовые отображения.

Двумерные файлы могут быть представлены в виде физически последовательных записей. Для более сложных древовидных и сетевых структур необходимо указывать иерархические связи между записями.

Физическое представление древовидных структур. Рассмотрим древовидную структуру логически организованных данных (рис. 2.10). На этом же рисунке древовидная структура представлена в виде физически последовательных записей, называемых левосписковой структурой, построенной следующим образом: выбираются узлы от вершины дерева вниз по самой левой ветви дерева; когда выбран узел самого нижнего уровня этой ветви, выбираются подобные узлы слева направо; процесс повторяется, причем выбранные узлы пропускаются.

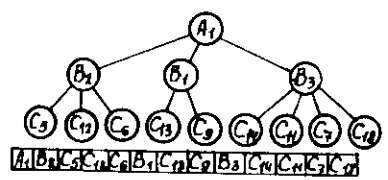


Рис. 2.10

При размещении в памяти физически последовательных записей можно указывать, к какому уровню иерархии они относятся, представляя список в следующем виде:

$$A_1 (B_2 (C_5 C_{12} C_6) B_1 (C_{13} C_9) B_3 (C_{14} C_{11} C_7 C_{18})) .$$

Последовательно организованные записи можно связывать с помощью системы указателей, встроенных непосредственно в запись (рис. 2.II).

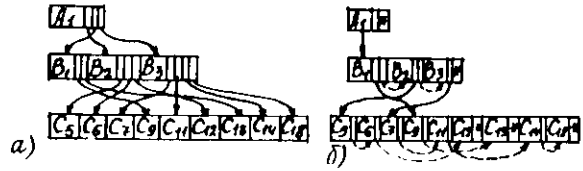


Рис. 2.II

На рис. 2.II,а приведены указатели на порожденные записи (так называемые иерархические структуры с явными уровнями [3I], а на рис. 2.II,б - указатели на подобные (пунктирные линии) и порожденные (сплошные линии) записи (так называемые иерархические структуры со связанными и явными уровнями) [3I].

Как видно из организации, представленной на рис. 2.II, непосредственно в записи встраиваются несколько полей для формирования в них множественных указателей на возможные связи. В первом случае возможны указатели на исходные записи (направление дуг графов снизу вверх).

Другой формой связи между записями древовидной структуры являются цепи (рис. 2.I2,а)и кольца (рис. 2.I2,б).

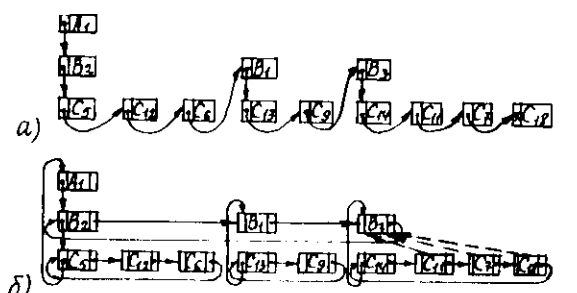


Рис. 2.I2

Цепь - это такая структура, которая представляет собой совокупность записей, разбросанных в одном или разных файлах и связанных между собой последовательностью указателей. Очень часто цепи называют списковой организацией. При работе с цепью необходимо знать, где она начинается и где заканчивается. Начало цепи должно

быть задано, конец может определиться либо по счетчику записей, либо по признаку конца, помещенного в последнюю запись цепи.

С п р а в о ч н и к – это файл, в котором хранится информация о связях между записями в других файлах.

На рис. 2.I3 древовидная структура рис. 2.I0 представлена с помощью справочника.

Вызванная в основную память часть справочника, касающаяся конкретного запроса, позволяет быстро отыскать физический адрес необходимой записи, что значительно быстрее прохождения цепочки указателей, встроенных в записи. На рис.2.I3 индекс справочника – это указатель на начало списков указателей различной структуры в справочнике. На этом рисунке приведено три вида различных справочников, соответствующих древовидной структуре рис. 2.I0.

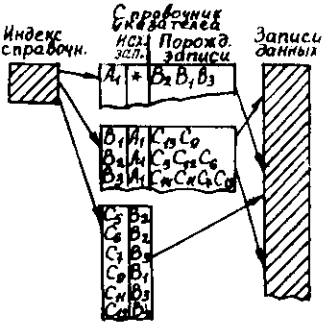


Рис. 2.I3

На рис. 2.I4 приведена матрица двоичных чисел, находящихся на пересечении строк и столбцов, в которых расположены ключи связанных записей, если двоичное число равно единице, и несвязанных записей, если двоичное число равно нулю. Такие указатели в виде матрицы двоичных чисел называются битовыми отображениями.

В случае, когда древовидные структуры на рис. 2.I0 имеют большое количество уровней иерархии, в матрице битового отображения двоичные коды, соответствующие определенным уровням иерархии, разделяются пустыми столбцами (символом "пробел"). На рис. 2.I4 эти области заштрихованы.

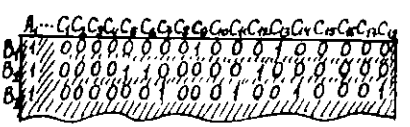


Рис. 2.I4

Одним из недостатков цепей является значительное время, затрачиваемое на поиск нужной записи, так как необходимо просматривать все записи. Существует метод, называемый фильтрованием, с помощью которого минимизируется время поиска записей. Сущность его заключается в перемещении к началу цепи наиболее часто используемых записей. Например, если запись использовалась заданное количество раз, то она перемещается на одну позицию ближе к началу цепи, а счетчик фильтрования устанавливается на нуль.

Кольцевые структуры являются замкнутыми цепями. Очень часто кольцевые структуры создаются с дополнительным указателем на заголовок цепи, находящийся в каждой записи. С помощью этого указателя всегда можно восстановить заголовок при обрыве цепи.

На рис. 2.I2,б указатели образуют кольца подобных записей и кольца, замыкающие исходные и порожденные записи. В каждой записи более нижнего уровня иерархии можно сформировать указатели на исходные записи (штриховые стрелки). Это дает возможность создавать двусторонние кольца. В этом случае число указателей удваивается.

Справочники хранятся отдельно от записей. Во всех остальных структурах указатели были встроены в записи.

Физическое представление сетевых структур. Сетевые структуры могут быть простыми в случаях, когда существуют связи от одного исходного элемента ко многим порожденным, и сложными со связями от многих исходных элементов ко многим порожденным (рис. 2.I5).

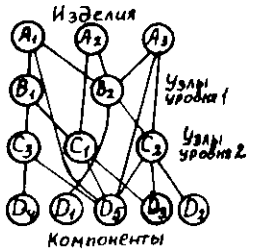


Рис. 2.I5

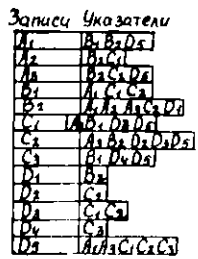


Рис. 2.I6

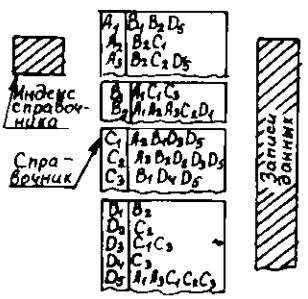


Рис. 2.I7

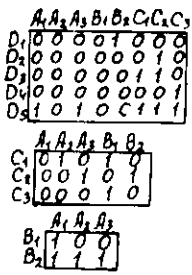


Рис. 2.I8

Для простых и сложных сетевых структур используются те же методы физического представления, что и для древовидных. Однако программное обеспечение, разработанное для древовидных структур, будет непригодно для сетевых, поскольку во всех методах физической организации записи будут иметь переменное число указателей и, следовательно, переменную длину. Таким образом, физическое представление сетевых структур использует организацию файлов переменной длины.

На рис. 2.16 – 2.18 представлена сложная многоуровневая сетевая структура, изображенная на рис. 2.15 с помощью метода указателей переменной длины (рис. 2.16), встроенных в записи, метода справочников (рис. 2.17) и метода битовых отображений (рис. 2.18).

На рис. 2.18 четырехуровневая сетевая структура представлена в виде трех битовых отображений. На первом битовом отображении приведены связи между первыми тремя уровнями и последним, на втором – – связи между первыми двумя уровнями и третьим и, наконец, на третьем – между первыми двумя уровнями.

Из всех пяти методов наиболее перспективными по быстродействию являются метод справочников и метод битовых отображений.

Способы адресации. Основная проблема адресации состоит в следующем: как по первичному ключу определить местоположение записи на физическом носителе?

Существуют следующие способы адресации записей: последовательная, блочная, индексно-последовательная, индексно-произвольная, адресация преобразованием ключа в адрес, статистическая. Все эти способы адресации характеризуются методом локализации (поиска) записи с заданным ключом.

Последовательная адресация файла предполагает последовательное размещение записей, отсортированных или неотсортированных по ключевым элементам данных. При поиске записей в последовательной организации осуществляется последовательное сканирование файла путем просмотра ключа каждой записи и сравнения его с заданным ключом.

Блочная адресация предполагает, что файл отсортирован по ключу. Тогда для локализации достаточно просматривать ключ записи, с которой начинается каждый блок, и в случае отыскания блока, в котором хранится искомая запись, осуществлять последовательное сканирование записей всего блока.

Адресация преобразованием ключа в адрес и статистическая адресация будут рассмотрены ниже в разделах, связанных с соответствующим видом сортировки и поиска.

Рассмотрим подробнее индексно-последовательную и индексно-произвольную адресацию.

Индексно-последовательная адресация предполагает сканирование, т.е. последовательную адресацию в узкой области, локализованной каким-либо образом. Если файл упорядочен по ключам, то при адресации используется таблица, называемая индексом. При обращении к таблице задается ключ искомой записи, а результатом обращения является относительный или абсолютный адрес более узкой области (по сравнению со всем файлом), в которой заключена запись с искомым ключом.

Индекс – это таблица, с которой связана процедура, воспринимающая на входе информацию о некоторых значениях атрибутов и выдающая на выходе информацию, способствующую быстрой локализации записи (записей), имеющей заданные значения атрибутов. В индексе хранятся ссылки на блоки записей. Наличие индекса существенно ускоряет поиск записи по ключу (по сравнению с последовательным сканированием всего файла). Но при этом приходится хранить таблицу значительных размеров. Для уменьшения этой таблицы (индекса) вводится еще один индекс, так называемый индекс индексов.

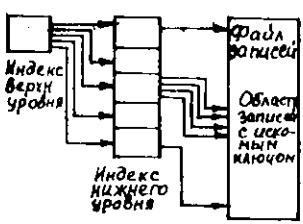


Рис. 2.19

На рис. 2.19 приведен пример двухуровневого индекса. В индексе нижнего уровня приведены начальные и конечные значения адресов и соответствующие им ключи записей для областей, на которые разбит файл. Поскольку индекс k-го уровня может быть большим, то он сам разбивается на области, а границы относительных или абсолютных адресов этих областей и ключи записей, соответствующих этим областям, сведены в индексе верхнего уровня. Таким образом, если известен ключ записи, то при обращении к индексу верхнего уровня определяются границы области адресов в индексе нижнего уровня. В этих границах заключена искомая запись. Индексно-последовательная адресация возможна и с большим количеством уровней индексов.

Индексно-произвольная адресация предполагает наличие неотсортированного по ключу файла. В этом случае каждой записи в файле должен соответствовать в индексе элемент. Следовательно, сам индекс по длине будет равен длине файла. Индексно-последовательный файл более экономичен и более эффективен по быстродействию, чем

индексно-произвольные файлы. Однако индексно-произвольная адресация применяется для адресации файлов по вторичным ключам и упорядоченных по первичным. Для одного и того же файла могут быть организованы различные индексы, соответствующие различным ключам. Для упорядоченных ключей индекс будет содержать по одному элементу на блок, а для неупорядоченных ключей индексы более длинные, так как содержат по одному элементу для каждой записи.

2.4. ИНВЕРТИРОВАННЫЕ ФАЙЛЫ

Инвертированный файл – это файл, структура которого обеспечивает возможность быстрого произвольного поиска информации, не специфицированной в запросах заранее. В файле имеются независимые списки или индексы, упорядоченные по ключам, которые соответствуют значениям определенных полей в записях файла.

Если рассмотреть плоский двумерный файл, упорядоченный по первичному ключу (назовем его прямым файлом), то форма простейшего запроса будет иметь вид $A(E)-?$, т.е. какое значение имеет атрибут A в объекте E. В данном случае идентификатор объекта E выступает в качестве первичного ключа. Если же файл упорядочим по значениям вторичных ключей (или по значениям произвольных атрибутов), то получим простейший инвертированный файл (рис. 2.20). Из одного прямого файла можно получить множество инвертированных, причем в каждом случае информационные записи не перемещаются, а создаются специальные списки указателей.

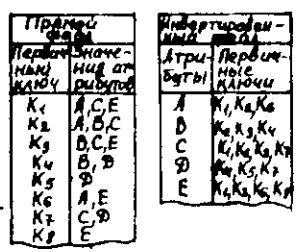


Рис. 2.20

Для инвертированных файлов форма запроса на поиск информации имеет вид $A(?)\overset{\neq}{<}V$, т.е. какой объект E имеет значение атрибута A, равное (не равное, меньшее, большее) V.

Инвертированные файлы находят применение в системах следующих трех типов:

со вторичными ключами (подобно рассмотренной на рис.2.19). В этих системах записи расположены в соответствии с первичным ключом, а непосредственно в записях имеются вторичные ключи;

частично инвертированных файлов. В них первичный индекс отсутствует, вторичные ключи (инвертированные индексы) используются для прямой адресации записей, что существенно облегчает корректи-

ровку файлов, так как допускается размещение новых записей в любом свободном участке файла;

полностью инвертированных файлов. Здесь предусмотрены отдельные файлы, содержащие значения атрибутов, входящих в состав записей. Значения элементов данных, составляющих конкретную запись, размещаются в памяти произвольно.

Для ускорения поиска в полностью инвертированных файлах вводятся два индекса (таблицы): индекс документов (индекс нижнего уровня) и индекс экземпляров (индекс верхнего уровня). С использованием этих индексов поиск можно осуществлять настолько быстро, что системы полностью инвертированных файлов можно использовать в диалоговом режиме при поиске документов и обработке текстов.

На рис. 2.21 приведены структура памяти в системе полностью инвертированных файлов и структура самого файла.

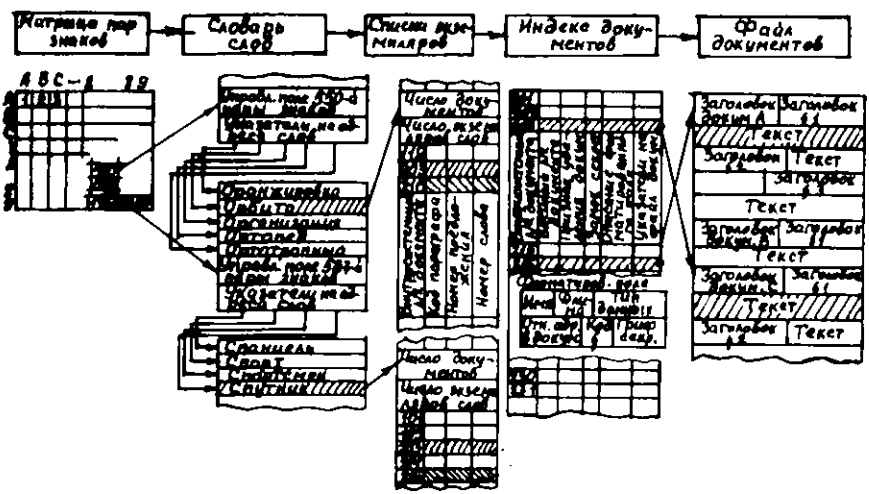


Рис. 2.21

В левой части этого рисунка приведен словарь слов вместе с матрицей всевозможных пар символов, встречающихся в словах на первых двух позициях. Эта матрица является в сущности индексом словаря слов ввиду большой длины последнего. Каждой паре знаков поставлен в соответствие указатель на блок словаря, содержащий группу слов, начинающихся с этих знаков.

Таким образом, в матрице содержится 37·37=1369 пар знаков.

Каждому слову поставлен в соответствие указатель на список экземпляров, являющийся перечнем документов, в которых встречается данное слово. Каждый список экземпляров содержит заголовок, из которого можно узнать число экземпляров (количество повторений) слова во всем файле документов, а также число документов, в которых это слово встречается.

Система присваивает документу уникальный внутрисистемный номер. В списке экземпляров, соответствующем какому-либо слову, содержатся внутрисистемные номера всех документов, в которых оно встречается.

В состав документов могут входить заголовки документов, имена авторов, аннотации, параграфы различных типов, заголовки параграфов и тексты. Все эти составляющие могут входить в критерий отбора. Пользователь может получить информацию о количестве документов, удовлетворяющих критерию отбора. После этого он может вывести на дисплей перечень авторов, названий, аннотаций или текстов найденных документов последовательно, страница за страницей.

Внутрисистемный номер документа является ключом к индексу документов. Этот индекс содержит адреса соответствующих документов в памяти.

В принципе можно хранить эти адресные указатели в списке экземпляров, но это нецелесообразно, так как список экземпляров содержит тысячи или сотни тысяч ссылок на документы, и поэтому использование в качестве ссылки адреса, а не номера документа существенно увеличит объем списка и время поиска. Поэтому индекс документов хранится отдельно. Индекс документов, кроме адресов, содержит внешний номер документа, признак удаления, перечень пользователей, допущенных к данному документу (гриф секретности). Все вспомогательные индексы и словарь слов в полностью инвертированных файлах, как правило, занимают больше места, чем сами документы.

2.5. Обработка информационных массивов

2.5.1. Виды сортировок

К основным процедурам обработки информационных массивов относятся следующие процедуры: формирования, сортировки, корректировки, информационного поиска, контроля. Рассмотрим основные алгоритмы сортировки и информационного поиска.

Известно, что к отсортированным по ключевым элементам данных информационным массивам можно осуществлять существенно более

быстрый доступ, чем к неотсортированным. Однако необходимость сортировки информационных массивов обусловлена не только эффективностью поиска. Неотсортированные массивы практически невозможно откорректировать. Все виды сортировки основаны на естественной упорядоченности 256 символов, имеющихся в распоряжении ОС ЕС [12, 13, 26].

Рассмотрим основные виды сортировки, используемые на практике.

Сравнительные, к которым относится обменная сортировка и сортировка Шелла, и распределительные, к которым относится сортировка поразрядным группированием и поразрядно-обменная сортировка.

Кроме этого существует сортировка вычислением адреса.

Обменная сортировка основана на сравнении пары соседних величин и их перестановке в требуемом порядке.

На рис. 2.22 приведен пример обменной сортировки. По грубой оценке этот вид сортировки требует $N \times (N-1)/2$ сравнений, а время ее работы приблизительно пропорционально N^2.

Сортировка Шелла (по имени автора) в основе своей похожа на обменную, однако перестановка пар ключей при первом просмотре осуществляется на расстоянии d, далее сравниваются ключи, расположенные в следующих позициях на том же расстоянии d, и т.д., пока не переберется весь массив.

При втором просмотре расстояние d, на котором сравниваются и переставляются пары, уменьшается вдвое, т.е. $d_{i+1} = \frac{d_i + 1}{2}$, где i – номер просмотра.

Если при заданном d_i упорядочены не все ключи, то с этим d_i осуществляются просмотры до тех пор, пока ключи не будут упорядочены.

На рис. 2.23 приведен пример сортировки Шелла с тем же массивом, что и в примере на рис. 2.22. Экспериментально показано, что время, требуемое на сортировку Шелла, примерно равно $B \cdot N \cdot (log_2 \cdot N)^2$, где B – эмпирический коэффициент, значительно меньший единицы. Таким образом, сортировка Шелла для умеренных N (порядка 100) значительно быстрее обменной сортировки.

Сортировка поразрядным группированием начинается с анализа самого младшего значащего символа ключевого слова. Все ключевые слова с одинаковыми младшими цифрами объединяются в группы. Содержимое групп располагается в порядке возрастания значения анализируемого разряда. Затем аналогичный процесс повторяется для более старшего разряда и т.д.

Исход. массив ключей	1-й	2-й	3-й	4-й	5-й	6-й	7-й
19	13	05	05	01	01	01	01
13	05	13	01	05	05	05	02
05	19	01	13	13	13	02	05
27	01	19	19	16	02	09	09
01	26	26	16	02	09	11	11
26	27	16	02	09	11	13	13
31	16	02	09	11	16	16	16
16	02	09	11	19	19	19	19
02	09	11	21	21	21	21	21
09	11	21	26	26	26	26	26
11	21	27	27	27	27	27	27
21	31	31	31	31	31	31	31

Рис. 2.22

Дано	$d_1=6$	$d_2=3$ 1-й просмотр	$d_2=3$ 2-й просмотр	$d_2=2$ 1-й просмотр	$d_2=2$ 2-й просмотр	$d_4=1$	Результат
19	19	*09	09	02	02	*01	01
13	13	*01	01	01	01	*02	02
05	*02	02	02	09	09	*05	05
27	*09	19	19	* 05	05	*09	09
01	01	*13	11	11	11	11	11
26	21	*05	05	** 13	13	13	13
31	31	*27	27	* 21	*16	16	16
16	16	*11	*13	*19	19	19	19
02	05	*21	21	** 16	*21	21	21
09	27	*31	31	* 26	26	26	26
11	11	*16	16	*27	27	27	27
21	16	26	26	*31	31	31	31

* – одна перестановка ** – две перестановки

Рис. 2.23

Исходн. массив ключей	Группирование по младшему разряду	Объединение	Группирование по старшему разряду	Объединение
19	0)	01	0) 01, 02, 05, 09	01
13	1) 01, 31, 11, 21	31	1) 11, 13, 16, 19	02
05	2) 02	11	2) 21, 26, 27	05
27	3) 13	21	3) 31	09
01	4)	02	4)	11
26	5) 05	13	5)	13
31	6) 26, 16	05	6)	16
16	7) 27	26	7)	19
02	8)	16	8)	21
09	9) 19, 09	27	9)	26
11		19		27
21		09		31

Рис. 2.24

Исходный массив	Двоичн. предст.	1-й просмотр	2-й просмотр	3-й просмотр	4-й просмотр	5-й просмотр	Итог. массив
19	10011	*01011	*00010	00010	*00001	00001	1
13	01101	01101	*00001	00001	*00010	00010	2
05	00101	00101	00101	00101	00101	00101	5
27	11011	*01001	01001	01001	01001	01001	9
01	00001	00001	*01101	*01011	01011	01011	11
26	11010	*00010	*01011	*01101	01101	01101	13
31	11111	11111	*10101	*01111	*10000	10000	16
16	10000	10000	10000	10000	*10011	10011	19
02	00010	*11010	*10011	*10101	10101	10101	21
09	01001	*11011	11011	11011	11011	*11010	26
11	01011	*10011	*11010	11010	11010	*11011	27
21	10101	*10101	*11111	11111	11111	11111	31

Рис. 2.25

На рис. 2.24 приведен пример поразрядного группирования. Наряду с преимуществом в простоте этот тип сортировки имеет следующий недостаток: оба процесса - сортировка и слияние требуют значительного количества дополнительной памяти. Среднее время этого вида сортировки оценивается выражением $C \cdot N \cdot log_p K$, где K - максимальное количество символов в ключевом слове, а p - максимальное (по всем ключевым словам) количество различных символов среди всех разрядов.

Поразрядно-обменная сортировка осуществляется путем составления групп ключей с M одинаковыми старшими битами и упорядочения внутри этих групп в соответствии со значением $(M+1)$-го бита. Упорядочение групп по $(M+1)$-му биту осуществляется просмотром сверху вниз до обнаружения единичного бита и снизу вверх до обнаружения нулевого бита. Соответствующие ключи меняются местами, и просмотр продолжается. На рис. 2.25 приведен пример на поразрядно-обменную сортировку. Среднее время работы алгоритма поразрядно-обменной сортировки составляет $D \cdot N \cdot log_2 N$. Так же как и сортировка поразрядным группированием, поразрядно-обменная сортировка требует для своей работы дополнительной памяти для хранения групп с одинаковыми разрядами.

Необходимо отметить, что все рассмотренные выше виды сортировки без труда допускают произвольную символику ключевых слов из состава 256 символов в ОС ЕС, т.е. необязательно ключи должны состоять из числовых символов. В них могут включаться и текстовые символы.

Сортировка вычислением адреса осуществляется путем преобразования ключа в адрес следующим образом: максимальное значение ключа делится на общее число элементов массива:

$$n = \left\lceil \frac{max\,(ключа)}{N} \right\rceil .$$

Значения ключей делятся на это число, в результате чего получается вычисленный адрес:

$$Адрес = \left\lfloor \frac{ключ}{n} \right\rfloor .$$

В этих выражениях скобки, замыкающиеся вверху, указывают на то, что это большее целое, а замыкающиеся внизу - меньшее целое.

На рис. 2.26 приведен пример сортировки вычислением адреса. В этом примере

$$n = \frac{31}{12} = \lceil 2,6 \rceil = 3 .$$

Для нескольких значений может быть вычислен одинаковый адрес. В этом случае для решения конфликтной ситуации осуществляется линейный поиск, т.е. сравниваются значения ключей, "претендующих" на один и тот же адрес.

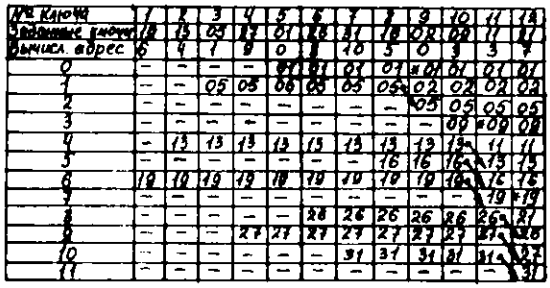


Рис. 2.26

Изложим алгоритм вычисления адреса в случае, когда ключевые слова составлены не из числовых, а из текстовых символов. При этом необходимо текстовые ключи преобразовать в числовые, сохраняя старшинство символов. Процедура преобразования осуществляется с помощью операций логического сложения двоичных разрядов по следующей схеме:

ОП1	ОП2	ОП. (результат)
I	I	I
I	0	I
0	I	I
0	0	0

с использованием двоичного представления всех символов в коде *EBCDIC* [26]. В младших двоичных разрядах каждого текстового символа кода *EBCDIC* содержится шестнадцатиричный код из диапазона 0...9. Следовательно, необходимо преобразовать старшие четыре разряда любого символа в шестнадцатиричную цифру $F = IIII$, так как любая цифра представляется одним байтом, в первой половине которого стоит двоичный код цифры F, а во второй - двоичный код искомой цифры. Например, машинный код цифры 7 в зонированном формате представляется так

$$F7 = \underbrace{1111}_{F}\ \underbrace{0111}_{7}.$$

В качестве примера рассмотрим преобразование пятисимвольного ключа НОМЕР в числовое значение. Для этого необходимо использовать

маску в виде пятисимвольного кода, состоящего из зонированных нулей $F0\,F0\,F0\,F0\,F0$. Тогда в результате логического сложения ключа и маски получается числовой ключ, представленный на рис.2.27. Таким образом, ключу НОМЗР соответствует число 86457. Этот числовой ключ уже несложно преобразовать в адрес с помощью изложенного выше алгоритма. В случае, если ключи смешанного типа, т.е. состоят из текстовых и числовых элементов, то, как правило, их можно упорядочить отдельно по числовым и текстовым символам.

Текстовой ключ	Символьн.	H	O	M	E	P
	Шестнад.	C8	D6	D4	C5	D7
	Двоичный	11001000	11010110	11010100	11000101	11010111
Маска	Символьн.	0	0	0	0	0
	Шестнад.	F0	F0	F0	F0	F0
	Двоичный	11110000	11110000	11110000	11110000	11110000
Числовой ключ	Двоичный	11001000	11010110	11010100	11000101	11010111
	Шестнад.	C8	D6	D4	C5	D7
	Символьн.	8	6	4	5	7

Рис. 2.27

Сравнение сортировок. Рассмотрим эффективность сортировок по быстродействию и по количеству требуемой дополнительной памяти. Эти критерии для каждой из сортировок сведены в следующую таблицу:

Тип сортировки	Оценка среднего времени	Дополнительная память
Обменная	$A \cdot N^2$	Не требуется
Шелла	$B \cdot N \cdot (Log_2 N)^2$	Не требуется
Поразрядным группированием	$C \cdot N \cdot Log_p \cdot K$	$N \cdot P$
Поразрядно-обменная	$D \cdot N \cdot Log_2 N$	$K+1$
Вычислением адреса	$E \cdot N$	$\sim 2,2 \cdot N$

Здесь N – размер массива; K – максимальный размер ключа в байтах (максимальное по всем ключевым элементам количество различных символов в разрядах); A, B, C, D, E – постоянные времени, куда входит время сравнения пары ключей, занесение в буфер и извлечение из него и т.д. Наиболее простые сортировки — обменная и поразрядным группированием. Наиболее быстродействующая сортировка – **поразрядным группированием, но она и наиболее неблагоприятна по объему требуемой дополнительной памяти.**

2.5.2. Методы поиска информации

Основной процедурой математического и информационного обеспечения САПР является процедура автоматического поиска информации, сформированной на внешних носителях. Необходимо отметить, что методы формирования (адресации) и поиска идентичны: какие описатели информации и действия были заложены при формировании, такие же действия и те же описатели будут использоваться при поиске.

К основным методам поиска относятся: линейный, блочный, двоичный, случайный, индексно-последовательный, индексно-прямой, преобразованием ключа в адрес.

Индексно-последовательный и индексно-прямой методы рассмотрены, например, в работе [19]. Принцип преобразования ключа в адрес рассмотрен в разд. 2.5.1.

В данном разделе будут рассмотрены четыре первых метода поиска.

Линейный поиск является простейшим методом поиска. Он заключается в том, что информация последовательно (линейно) просматривается до нахождения записи с данным ключевым элементом.

Этот метод поиска применяется в следующих случаях: для небольшого списка записей ($N \leq 50$–100); неотсортированного массива записей; для поиска в ленточных массивах.

Время поиска в этом алгоритме пропорционально $N/2$ и описывается соотношением $T_n = t_{cp} \cdot N/2$, где t_{cp} – время операции сравнения (сюда же входит и время чтения записи, если она находится на внешнем носителе); N – количество записей; $N/2$ – среднее количество записей, просматриваемых до нахождения искомой.

Блочный поиск соответствует блочной адресации. При блочном поиске локализация записи с заданным ключом осуществляется просмотром не каждого ключа, как в линейном поиске, а каждого первого ключа в каждом блоке (рис. 2.28) в последовательности возрастания ключей. На этом рисунке K_s – ключ искомой записи. Следовательно, записи должны быть отсортированы по ключу. При отыскании блока, где находится запись с заданным ключом, осуществляется линейный поиск. При уменьшении размера блока приходится проводить большое количество просмотров головных ключей. При увеличении размера блока значительное количество сравнений осуществляется внутри блока. Поэтому должен быть оптимальный размер блока.

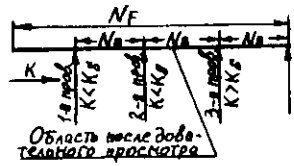


Рис. 2.28

Пусть N_F – количество записей в файле, N_B – количество записей в блоке. Тогда имеется $\lceil N_F/N_B \rceil$ блоков, где скобки обозначают наибольшее целое.

Среднее число блоков, которое надо просмотреть, чтобы найти искомый блок, равно

$$\sum_{K=1}^{\lceil N_F/N_B \rceil} K \cdot P_K ,$$

где P_K – вероятность того, что K-й анализируемый блок содержит искомую запись. Если считать, что искомая запись равновероятно находится в любом блоке, среднее число просмотренных блоков будет

$$\sum_{K=1}^{\lceil N_F/N_B \rceil} K \cdot \frac{1}{\lceil N_F/N_B \rceil} = \frac{\lceil N_F/N_B \rceil + 1}{2} .$$

В найденном блоке выполняется поиск нужной записи. Первая запись была проанализирована, следовательно, необходимо просмотреть записи от *1* до $N_B - 1$. Среднее число анализируемых записей в блоке будет

$$\sum_{i=1}^{N_B-1} i \, P_i = \sum_{i=1}^{N_B-1} i \cdot \frac{1}{N_B} = \frac{N_B - 1}{2} .$$

Пусть N_p – общее среднее число просматриваемых записей:

$$N_p = \frac{\lceil N_F/N_B \rceil + 1}{2} + \frac{N_B - 1}{2} = \frac{1}{2}\left(\lceil N_F/N_B \rceil + N_B \right) .$$

Число записей в блоке, при котором достигается минимальное значение N_p, определяется выражением

$$\frac{dN_P}{dN_B} \approx -\frac{N_F}{2N_B^2} + \frac{1}{2} = 0 , \quad N_B = \sqrt{N_F} .$$

Общее число записей, которые должны быть просмотрены, если в каждом блоке содержится $\sqrt{N_F}$ записей, будет

$$N_P = \frac{1}{2}\left\lceil \frac{N_F}{\lceil \sqrt{N_F} \rceil} \right\rceil + \frac{\lceil \sqrt{N_F} \rceil}{2} = \sqrt{N_F} .$$

Таким образом, оптимальное число записей в блоке равно корню квадратному из числа записей в файле. Точно такое же среднее число просматриваемых записей.

Двоичный поиск предполагает, что массивы информации заранее отсортированы. Тогда появляется возможность деления всего массива ключей пополам и анализа, в какой из половин находится заданный ключ.

Таким образом, двоичный поиск осуществляется в следующих случаях:

- для отсортированной по ключевым элементам информации;
- для информации, расположенной на носителях, допускающих адресацию (например, на томах прямого доступа или в оперативной памяти).

Алгоритм двоичного поиска следующий: весь массив ключей делится пополам и отбрасывается та половина, в которой нет искомого ключевого элемента данных; затем оставшаяся половина ($N/2$) делится пополам ($N/4$) и отбрасывается та четверть массива, которая не содержит заданного ключа, и т.д.; деление осуществляется до нахождения заданного ключевого элемента данных.

На рис. 2.29 приведен список ключевых элементов данных, среди которых методом половинного деления отыскивается ключ F.

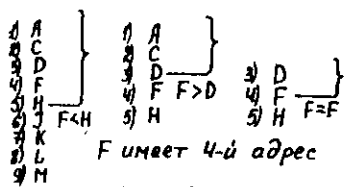


Рис. 2.29

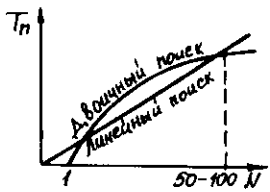


Рис. 2.30

Время поиска этого алгоритма составляет

$$T_n = t_{ср} \cdot \log_2 N,$$

где $\log_2 N = K$ – количество делений списка ключей ($N = 2^K$ – для четного числа N и $N = 2^K + 1$ – для нечетного).

Необходимо отметить, что для этого случая не только время сравнения может быть больше времени сравнения в линейном поиске, но и потребное количество оперативной памяти также будет больше, так как каждый раз необходимо помнить границы предыдущего деления. Однако для большинства массивов двоичный поиск выгоднее по времени, что видно из диаграммы сравнения, приведенной на рис. 2.30. При малых N выгоднее линейный поиск, а при больших – двоичный. Для ЭВМ серии ЕС точка пересечения линейного и двоичного поиска лежит в интервале $N = 50...100$. Однако если времена сравнения в обоих алгоритмах поиска одинаковы, то точка пересечения лежит при $N = 4$: $\frac{N}{2} = \log_2 N$.

Таким образом, существенное отличие значений $N = 4$ и $N = 50...100$ для ЕС ЭВМ достигается за счет времени различных дополнительных операций при двоичном поиске.

Случайный поиск использует псевдослучайные числа при вычислении адреса, по которому должен быть помещен ключ записи, т.е. метод случайного поиска базируется на методе случайного занесения. При этом общее время поиска может быть значительно ниже суммарного времени сортировки и двоичного поиска. Методом случайного поиска можно искать нужную информацию и в несортированных массивах.

Искомый адрес вычисляется с помощью ключевого элемента данных и равномерно распределенных в интервале [0; I] псевдослучайных чисел.

Таким образом, в методе случайного поиска существуют две проблемы:

- генерация псевдослучайных чисел (чисел, получаемых каким-либо математическим путем, в отличие от случайных чисел, получаемых физическим путем), основанная на значениях ключей;

- создание процедуры, которой следует придерживаться, когда первый подготовленный элемент попал в заполненный участок.

Для генерации псевдослучайных чисел по значению ключа можно использовать два метода (есть и другие методы [I3, 27]):

- первый заключается в том, что для данной группы из M ключевых элементов данных последние делятся на длину N таблицы, избыточную по длине по отношению к количеству ключей M: остатки от деления должны быть равномерно распределены в интервале [0; I] ;

- второй способ заключается в том, что ключевое слово рассматривается как двоичная дробь (например, $I2 \rightarrow 0,I2 = 0\cdot2^{-1}+0\cdot2^{-2}+0\cdot2^{-3}+I\cdot2^{-4}+0\cdot2^{-5}+I\cdot2^{-7}$), которая затем умножается на другую заданную дробь в виде дробного иррационального числа (например, $\pi/I0 \rightarrow 3\cdot2^{-1}+I\cdot2^{-2}+4\cdot2^{-3}+\ldots$); это можно осуществить с помощью следующих процедур Ассемблера:

```
L   1, SYMBOL
M   Ø, RHO    ,
```

где в адресе *SYMBOL* находится ключевое слово в виде двоичной дроби, а в адресе *RHO* – иррациональное дробное число.

В регистрах 0,I будет 64-битовое произведение. Младшие 3I бит (в RI) рассматриваются как двоичная дробь. Составленные таким образом двоичные дроби для каждого ключа дадут равномерно распределенные в интервале [0; I] числа. Если эти числа умножим на количество выделенных адресов N , получим величины, равномерно распределенные на интервале $\emptyset$,..., $N-1$. После того как сгенерированы случайные числа, определен адрес, необходимо разработать алгоритм включения числа в таблицу, а затем аналогичный алгоритм поиска ключа. Простейший алгоритм решения конфликтной ситуации заключается в том, что когда первая попытка дает k-ю позицию из N и эта позиция занята, то проверяется следующая $(k+1)$-я позиция и т.д. до тех пор, пока не будет найдена свободная позиция (линейный поиск).

Если поиск выходит за пределы массива N, то он возобновляется с его начала (т.е. считается, что таблица циклическая).

Пример. Предположим, что таблица имеет 17 позиций, в которых должны быть размещены, а затем найдены следующие двенадцать чисел (M=12): 19,13,05,27,01,26,31,16,02,09,11,21. Методом случайного поиска могут распознаваться только массивы, сформированные этим методом. Для массива ключей в количестве M=12 выделяется избыточная память N = 17. Генерируем псевдослучайные числа, равномерно распределенные на интервале [0; 1], по первому способу, разделив содержимое всех ключей на N=17. Остатки дадут равномерно распределенные числа на интервале [0; 1] с плотностью вероятности 1/17=0,0688. Если эти остатки разделим на полученную плотность вероятности (или, что одно и то же, умножим на N = 17), получим равномерно распределенные позиции в интервале [0; 16].

Ключ (M)	Деление на N=17	Дробная часть (дел.)	Плотность распределения дробного остатка 1/17=0,0588	Адрес =17*(дроб. часть)	Распределение по позициям (адресам)		Число проверок на обнаружение		Число проверок на отсутствие	
19	1,12	0,12		2	0	12-11		2	1	4
13	0,765	0,765		13	1-01	13-13	1	1	6	3
05	0,294	0,294		5	2-19,02*	14-31	1	1	5	2
27	1,588	0,588		10	3-02	16	2		4	1
01	0,0588	0,0588		1	4-21	16-16	1	1	3	1
26	1,53	0,53		9	5-05		1		2	
31	1,825	0,825		14	6				1	
16	0,942	0,942		16	7				1	
02	0,118	0,118		2	8				1	
09	0,53	0,53		9	9-26,09*		1		7	
11	0,648	0,648		11	10-09/26*		2		6	
21	1,235	0,235		4	11-22-11*		2	Σ=16	5	Σ=54

Рис. 2.31

Результаты приведены на рис. 2.31. Как видно из рисунка, распределение по позициям осуществлено необязательно в порядке возрастания ключевых элементов данных. В случае, если дробные части ключей оказываются одинаковыми, вычисляется одинаковый адрес. При этом в вычисленной позиции стоит тот ключ, который первым занял ее, а остальные сдвигаются в соседние свободные позиции. Число попыток на обнаружение равно числу попыток для нахождения данного ключа. Отыскание ключа осуществляется так же, как и включение в таблицу (т.е. искомый ключ делится на длину таблицы, а затем дробная часть умножается на эту длину). В случае конфликтных ситуаций (не совсем равномерное распределение), поступаем, как показано на рисунке. Число попыток на установление отсутствия элемента в таблице определяет отсутствие или присутствие данного

ключа в таблице. Считается, что число отсутствует, когда встречается пустая позиция. Например, поиск числа 54 (54/17=3,177; 0,177x17=3) дал бы адрес 3, но там его нет, и, чтобы убедиться, что его вообще нет, необходимо дойти до пустого адреса, так как все числа с одинаковыми вычисленными адресами по алгоритму случайного поиска должны стоять рядом.

С помощью данного примера оценим эффективность случайного поиска:

Длина таблицы	$N = 17$
Количество ключей	$M = 12$
Средняя плотность запоминания	$\rho = \frac{12}{17} = 0,705$
Число попыток на заполнение таблицы	$T_{запол} = 16$
Среднее число проверок на обнаружение величины	$T_{обн} = \frac{16}{12} = 1,33$
Среднее число проверок на отсутствие величины	$T_{отс} = \frac{54}{16} = 3,37$
	$\Sigma = 20,7$

Для сравнения приведем условные времена для упакованной (число позиций равно числу ключей) таблицы, использующей поразрядно-обменную сортировку и двоичный поиск:

Количество попыток для заполнения и сортировки	$T_{заполн} = M + M \cdot \log_2 M = 55$
Среднее число проверок на обнаружение величины	$T_{обн} = \log_2 M = 3,58$
Среднее число проверок на отсутствие величины	$T_{отс} = \log_2 M = 3,58$
	$\Sigma = 62,16$

Таким образом, метод случайного поиска дает значительный выигрыш во времени (20,7 вместо 62,16) в три раза. Однако ему присущи следующие недостатки: размер таблицы примерно на 50% больше количества ключей; таблица не может быть уплотнена после ее первоначального заполнения; трудно удалить какую-либо величину из таблицы, так как это нарушает цепочку адресов. Это один из самых серьезных недостатков, так как делает практически невозможной один из видов коррекции.

3. БАНКИ ДАННЫХ СИОД И НСИ

3.1. БАНК ДАННЫХ СИОД

Банк данных СИОД предназначен для формирования баз данных под управлением ДОС АСВТ, ДОС ЕС с приставкой для ОС ЕС [17].

СИОД позволяет автоматизировать: ввод информации и создание баз данных, управление банком данных, защиту данных, поиск информации.

СИОД удовлетворяет требованиям:

- независимости - БД представляет собой самостоятельный элемент информационной системы, независимый от специальных случаев применения;

- отсутствия избыточности - каждый элемент информации в банке данных встречается только один раз;

- удобства в использовании - пользователь имеет возможность достаточно просто обращаться к БД;

- гибкости структуры - обеспечения возможности изменения внутренней структуры данных без существенных затрат.

С помощью СИОД пользователь может: создать банк данных, ввести в банк данных новые данные, осуществить корректировку (изменение) данных в БД, произвести по запросам выборку данных из БД.

Структуры данных в СИОД. В СИОД определены следующие типы структур: аспекты, повторяющиеся группы, логические записи или просто записи, датотеки, банк данных.

Аспект - наименьшая структурная единица данных, состоящая из имени и значения. Имя характеризует общее свойство, присущее информационным объектам, для описания которых используется данный аспект, а значение отражает индивидуальное свойство аспекта.

Пример последовательности аспектов приведен в табл. 3.1.

Таблица 3.1

Фамилия	Должность	Оклад
Иванов	Инженер	130
Петров	Ст. инженер	150

Повторяющиеся группы представляют собой набор аспектов, которые могут повторяться несколько раз. Для всего набора указывается обобщенный аспект или псевдоаспект, имя которого служит для идентификации всего набора.

Пример повторяющихся групп приведен в табл. 3.2.

Таблица 3.2

Прежние места работы			
Учреждение	Должность	Учреждение	Должность
ИНЭУМ	Инженер	ВИНИТИ	Ст. инженер

Здесь имя повторяющейся группы или псевдоаспекта "Прежние места работы" состоит из двух имен аспектов: "Учреждение" и "Должность".

Логическая запись – объединение аспектов и повторяющихся групп, связанных с определенным значением порядкового аспекта. Интеграция аспектов и повторяющихся групп в записи осуществляется самим пользователем в момент ввода в БД. Это означает, что структура логических записей заранее не фиксируется, а отражается в программах пользователя. Порядковые аспекты позволяют однозначно идентифицировать логические записи, обеспечивая их различимость.

Датотека – логическое объединение логических записей, обладающих одинаковыми аспектами и повторяющимися группами (например, таблица). Значением порядкового аспекта можно дополнительно идентифицировать принадлежность записи к определенной датотеке.

Банк данных – наивысшая структурная организация информации, представляющая собой объединение нескольких датотек, связанных общим именем.

Например, банк данных с именем "Отрасль" может содержать следующие датотеки:

"Кадры", записи которой содержат аспекты "Фамилия", "Должность", "Оклад";

"Организации", записи которой содержат аспекты "Название", "Адрес", "Министерство".

Значения порядковых аспектов занимают шесть байтов, причем первые два байта отводятся под символы, идентифицирующие принадлежность записи датотеке. Например, в примере с отраслью в датотеке "Кадры" порядковые аспекты могут принимать значения от КДØØØI до КД9999, а в датотеке "Организации" – от ОРØØØI до ОР9999. Первые два символа в СИОД называются групповыми значениями. Имена аспектов в СИОД могут быть символическими и вербальными. Символическое имя – трехсимвольное слово, состоящее из букв латинского алфавита и цифр и начинающееся обязательно с буквы. В имена аспек-

тов не включаются цифра Ø и буквы I и O. Диапазон символических имен AAA... 99. Символическое имя устанавливается пользователем и необходимо для внутрисистемных функций.

Для удобства оперирования аспектами вне системы могут быть использованы в е р б а л ь н ы е и м е н а (словесные) аспектов, служащие синонимами символических имен. Они могут быть представлены любыми символами длиной не выше 3I байта. Для этого необходимо установить соответствие символических и вербальных имен. Значения аспектов могут быть как числовыми, так и текстовыми. Каждому аспекту ставится в соответствие множество значений. Характер значений (числовые или текстовые), размещение значений, способ использования и т.д. задаются с помощью описаний формата аспектов.

Формат аспектов. Формат значений аспектов определяется следующими параметрами:

I) положением значения аспекта в записи, определяемым местом соответствующего аспекта в зафиксированной последовательности аспектов данной датотеки (номер);

2) длиной значений аспекта ($\leqslant$ 256 знаков); при описании формата аспекта необходимо указать максимальную длину, которую может принимать значение данного аспекта;

3) положением десятичной запятой (точки), указываемым в виде десятичного числа от ØØ до I5;

4) выравниванием, осуществляемым в связи с тем, что длина конкретного значения аспекта может быть меньше указанной в п. 2 и для экономии памяти необходимо указать выравнивание "слева" или "справа"; значение порядкового аспекта всегда выравнивается слева;

5) методом поиска, который может быть "прямым", когда просматриваются только записи, содержащие указанное в запросе значение данного аспекта, и "последовательным", при котором просматриваются все записи;

6) нулем как значением аспекта, определяющим, может ли "Ø" быть значением аспекта; если нуль не является значением аспекта, то все незначащие нули в поле значений данного аспекта подавляются и в БД не вводятся;

7) результатом вычисления, определяющим, является ли значение аспекта результатом вычисления. Например, полученное в результате вычисления число + I23I в зонированном формате может быть представлено как *F1F2F3C1* , и если не указать, что оно является результатом вычислений, его можно интерпретировать как I23A, так как

код "CI" служит для изображения цифры "I" и знака "+" (если это результат вычислений) и буквы "A" (если это не указано);

8) признаком пользователя, определяющим специальный байт, использующийся для задания произвольной информации пользователя при описании аспекта и для дополнительной индентификации псевдоаспекта в повторяющейся группе.

Управление данными. Управление данными охватывает все функции СИОД, связанные с управлением форматами данных и управлением составом данных.

Управление форматами данных заключается во вводе описаний аспектов, в создании на дисках аспектного каталога и в его обновлении. Управление составом данных заключается во вводе фактических данных в банк данных и в их корректировке (изменении, удалении и т.д.).

Для работы программы управления форматом данных необходимо описать каждый аспект, входящий в БД. Описание аспектов заключается в заполнении специальных бланков (размещении одной строки бланка на одной перфокарте), в которых указываются все изложенные выше компоненты формата аспектов. Все описания аспектов накапливаются в аспектном каталоге. Так что в СИОД данные и их описатели хранятся раздельно.

С помощью программы управления форматом данных имеется возможность:ввести в БД новые аспекты, удалить аспекты из БД, изменить имена или форматы аспектов.

Структура банка данных и обращение к СИОД из прикладных программ.

Структурно банк данных состоит из двух файлов: центральных данных *ZDR* ; организационных данных *ORG* .

В файле *ZDR* хранится целевая информация, а в файле *ORG* - вторичная, к которой относится следующая информация: каталог значений порядковых аспектов, каталог значений аспектов с прямым доступом, список вербальных имен аспектов, служебные каталоги (метки МД и т.д.). Каталог порядковых аспектов связан адресами связи с записями файла *ZDR* . Каждый пользователь может получить из банка данных информацию по специальным указаниям.

Основа поиска и выборки заключается в том, что непосредственное осуществление самих операций выполняет резидентная программа, имеющая прямую связь с банком данных. Запросы на выполнение операций передаются из прикладной программы через коммуникационный модуль, входящий в каждую прикладную программу и являющийся програм-

мным интерфейсом СИОД. Обмен данными в этих программах осуществляется через участок сопряжения прикладной программы, в котором имеются следующие области: инструкции, справки, ответа и вопроса.

В области инструкции указывается, какая должна быть осуществлена операция, задается ее спецификация в виде имен аспектов, связей между ними, условий поиска.

В области справки СИОД сообщает пользователю, правильно ли прошла вызванная операция и получен ли ответ, включая сообщение об ошибках.

В области ответа СИОД поставляет данные, полученные из банка данных.

В области вопроса указываются дополнительные (нестандартные) спецификации поисковой операции.

Обращение прикладных программ к коммуникационному модулю (т.е. к резидентной программе, так как именно этот модуль осуществляет связь с резидентной программой) осуществляется с помощью средств для межпрограммных связей, имеющихся в ДОС АСВТ, ДОС ЕС, ОС ЕС.

Ниже приведены примеры обращения к СИОД с языков Ассемблера и Фортрана.

С языка Ассемблер

Имя	Операция	Операнды	Комментарии
⋮	⋮	⋮	⋮
	STM	14,1,SAVE	Запоминание регистров 14,15, 0,1
	LA	1,PARAM	Адрес OIN → R1
	L	15,=V(REZID)	Имя REZID → R15
	BALR	14,15	Выполнение REZID
	CLI	STATUS,C'1'	(STATUS) ≷ 1 ?
	BL	На обработку ответа	(STATUS) < 1, ω = 8
	BE	На конец (нет ответа)	(STATUS) = 1, ω = 4
	B	На обработку ошибок	(STATUS) > 1, ω = 2
⋮	⋮		⋮
PARAM	DC	A(OIN)	Область инструкций
	DC	A(SPR)	Область справки
	DC	A(VOP)	Область вопроса
SAVE	DS	4F	Область сохранения регистров
⋮	⋮	⋮	⋮

С языка Фортран

```
      :
CALL     REZID (OIN, SPR [, OTU [, UOP]])
      :
```

< обработка ответа >

Здесь *REZID* – имя точки входа в коммуникационный модуль; *STATUS*-поле, в котором СИОД информирует программу пользователя о результатах выполнения операции.

Необязательное указание областей *OTV* и *UOP* обусловлено тем, что адрес области *OIN* находится в *R1* ; остальные области адресованы относительно области *OIN*.

Минимальная конфигурация, необходимая для функционирования СИОД: 128 К оперативной памяти, два магнитных диска, включая системный, одна лентопротяжка, одно п/к, одно АЦПУ, один центральный процессор.

3.2. БАНК ДАННЫХ НСИ

К нормативно-справочной информации в САПР ЛА относится следующая: характеристики уже спроектированных ЛА; весь перечень характеристик конструкционных и теплозащитных материалов, используемых в проектировании ЛА; описание геометрических форм и поверхностей различных агрегатов ЛА; различные данные для статистических оценок (например, оценок весов отдельных агрегатов ЛА); стандарты и нормативы и многие другие виды информации.

Основными составными частями банка НСИ являются: язык описания входных массивов НСИ; язык запросов пользователей и структура информации, предоставляемой пользователю по запросам; алгоритмы и программы ввода, контроля и формирования массивов НСИ на внешних носителях; алгоритмы и программы корректировки информационных массивов; алгоритмы и программы информационного поиска по запросам пользователей; алгоритмы и программы сортировки.

Язык описания данных. Под одним массивом НСИ будем подразумевать таблицу справочной информации, например таблицу характеристик современных самолетов. Массив состоит из логических записей, записи состоят из элементов данных, причем элементы могут быть числовыми и текстовыми. Вводятся следующие разделители: символ ":" – между элементами данных; символ "=" – между записями; символ "Ц" – между массивами.

Если какой-либо элемент в записи отсутствует, то разделители элементов повторяются.

Для описания элементов данных вводятся описатели. Собранные в одну нестандартную запись, описатели всех элементов информационных записей образуют макет массива. Таким образом, в макете собран весь паспорт информационного массива.

Каждому атрибуту информации (в данном случае столбцу таблицы) соответствует элемент описателя (макета) с теми же разделителями, что и в информационной записи. Каждый элемент макета описывает следующие характеристики информационных элементов:

1) тип данных - текстовой (идентификатором является символ T) или числовой (без идентификатора);

2) максимальную длину элементов данных, которая указывается в двухбайтовом поле элемента описателя, так как длина элемента данных не превышает 99 байтов;

3) за указателем длины элемента данных стоит идентификатор атрибута в виде имени длиной не более трех символов;

4) для текстового элемента данных описание макета на этом завершается.

Для числового элемента данных между указателем длины и идентификатором в двухбайтовом поле указывается число знаков после десятичной точки.

	Тип самолета	Тип двигателя	Кол-во двиг.	Длина сам-та	Размах крыла	Экипаж
1-я запись	ТУ-104	ТУРБОРЕАКТ	2	18	12	7
2-я запись	ИЛ-18	ТУРБОВИНТ	4	17	18	6
	=	=	=	=	=	=

Имя массива: ХАР␣САМ=
Запись физических имен: ТИП␣САМ:ТИП␣ДУ:КОЛ␣ДУ:ДЛИНА␣САМ:РАЗМ␣КР:ЭКИПАЖ:=
Макет: Т06ТС:Т15DB:Ф10ФКОL:Ф2ФФDС:Ф2ФФРКР:Ф2ФФЕКР:=
1-я информац. запись: ТУ-104:ТУРБОРЕАКТ:2:18:12:7:=
2-я информац. запись: ИЛ-18:ТУРБОВИНТ:4:17:18:6:=

Рис. 3.1

Для нужд автоматизированного проектирования наряду с машинными именами атрибутов необходимы физические имена, собранные также в нестандартную запись с разделителями в виде символов ":" и "=". Эти имена необходимы для диалога проектировщика с САПР терминами естественного языка.

Таким образом, для одного информационного массива на внешнем носителе хранятся следующие данные: имя массива, список физических имен элементов данных, макет массива, информационные записи.

На рис. 3.1 приведен пример использования описанного языка. Для заданного массива "характеристики самолетов" приведена структура файла на внешнем носителе.

Алгоритм ввода, формирования и контроля. Формирование массивов НСИ осуществляется в виде автоматизированных процедур, среди которых основными являются следующие: составление информационных документов; нанесение их на перфокарты; ввод информации в основную память; формирование логических записей переменной длины (или приведение их к постоянной длине путем добавления к элементам пробелов или незначащих нулей до максимальной длины); возможный машинный контроль правильности текстовых или числовых элементов данных; сплошной визуальный контроль по распечаткам массивов на АЦПУ; блокирование и вывод информации на внешний носитель; вывод ошибочных элементов информации на АЦПУ; составление корректирующих файлов; корректировка ошибочной информации.

На рис. 3.2 приведена блок-схема алгоритма, реализующего формирование массивов на МЛ.

Рассмотрим два простейших синтаксических вида контроля, легко реализуемых программным путем:

контроль по длине: элемент считается ошибочным, если его длина превышает длину, указанную в макете;

контроль на наличие в числовом элементе данных текстовых символов; числовой элемент считается ошибочным, если в левом полубайте не стоит шестнадцатиричная цифра *F*.

В случае обнаружения этих видов ошибки программа заносит на внешний носитель нули (вместо числового элемента) и пробелы (вместо текстового), а на АЦПУ выводит ошибочный элемент данных и информацию о его расположении в массивах в следующем виде:

< имя массива > : < имя ключевого элемента данных > : < идентификатор элемента данных > : < физическое имя элемента данных > : < собственно элемент данных > : =

Эта информация затем используется для построения корректирующего файла.

Алгоритм корректировки информационных массивов. Перечислим виды коррекции, которые необходимо заложить в банк данных: удаление записи (признак коррекции A); замена записи (признак коррекции B); внесение новой записи (признак коррекции C); изменение одного или нескольких элементов данных в записи (признак коррекции *D*). При использовании признака коррекции B заменяется вся запись, кроме ключевого слова. При внесении новой записи массив раздвигается и перенумеровывается.

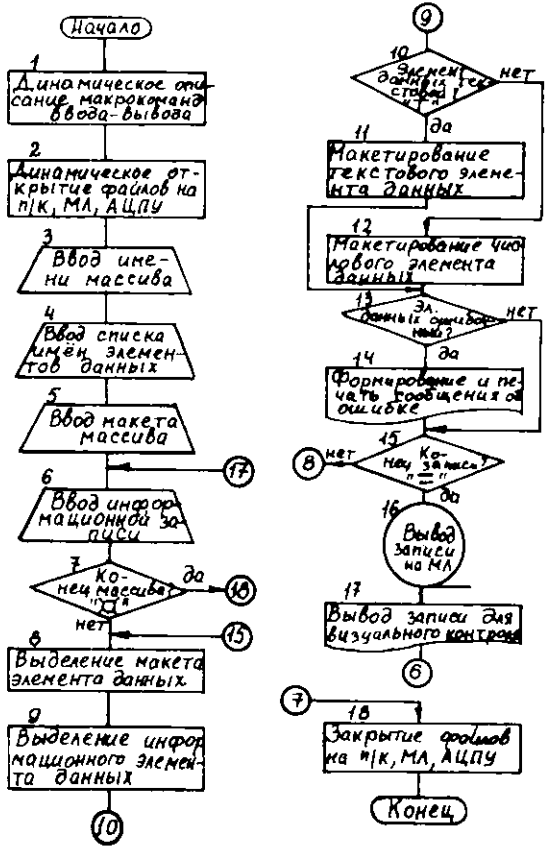

Начало
1 Динамическое описание макрокоманд ввода-вывода
2 Динамическое открытие файлов на п/к, МЛ, АЦПУ
3 Ввод имени массива
4 Ввод списка имен элементов данных
5 Ввод макета массива
6 Ввод информационной записи
7 Конец массива?
да
нет
8 Выделение макета элемента данных
9 Выделение информационного элемента данных
10
9
10 Элемент данных с таким именем уже есть?
нет
да
11 Макетирование текстового элемента данных
12 Макетирование числового элемента данных
13 Эл. данных описан верно?
нет
да
14 Формирование и печать сообщения об ошибке
15 Конец записи?
нет
да
16 Вывод записи на МЛ
17 Вывод записи для визуального контроля
6
7
18 Закрытие файлов на п/к, МЛ, АЦПУ
Конец

Рис. 3.2

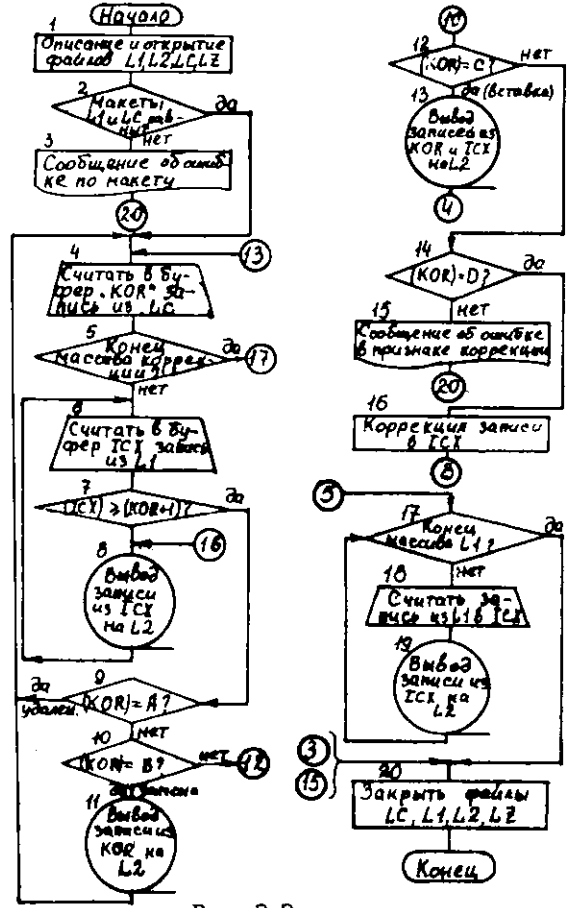

Начало
1 Описание и открытие файлов L1, L2, LC, LZ
2 Макеты L1 и LC совпадают?
да
нет
3 Сообщение об ошибке по макету
20
13
4 Считать в буфер KOR запись из LC
5 Конец массива коррекции?
да
17
нет
6 Считать в буфер TCX запись из L1
7 (TCX) ≥ (KOR+1)?
да
16
8 Вывод записи из TCX на L2
да удалить
9 (KOR) = A?
нет
10 (KOR) = B?
нет
12
да вставка
11 Вывод записи из KOR на L2
12 (KOR) = C?
нет
да (вставка)
13 Вывод записей из KOR и TCX на L2
4
14 (KOR) = D?
да
нет
15 Сообщение об ошибке в признаке коррекции
20
16 Коррекция записи в TCX
8
5
17 Конец массива L1?
да
нет
18 Считать запись из L1 в TCX
19 Вывод записи из TCX на L2
3
15
20 Закрыть файлы LC, L1, L2, LZ
Конец

Рис. 3.3

Рассмотрим алгоритм корректировки информационных массивов на МЛ (рис. 3.3). Массивы должны быть отсортированы по ключу. Для корректировки необходимо три МЛ: с исходным массивом L1; с корректирующим файлом LC; с результирующим файлом L2.

Макет файла LC начинается с символов ТФ1, так как первым символом в информационных записях этого файла стоит один из признаков коррекции A,B,C,D . Массив LC также упорядочен по ключу. Вначале сравниваются макеты массивов L1 и LC (если они не совпадают, программа прекращает свою работу с выдачей соответствующего сообщения). В основной памяти выделяются два буфера: с именем *ICX* (для записи исходного файла) и с именем *KOR* (для записи корректирующего файла). В байте *KOR* будет стоять признак коррекции. Если макеты файлов L1 и LC не совпали, то об этом выводится сообщение; если совпали, то в память *ICX* и *KOR* считываются записи из файлов L1 и LC. Если ключевое слово в *ICX* меньше ключевого слова в *KOR+1*, то запись из *ICX* коррекции не подлежит и выводится в файл L2. Если же ключевые слова равны, то отыскивается признак коррекции путем сравнения символа в памяти *KOR* с символами A,B,C,Д. При совпадении с одним из этих символов осуществляется соответствующий вид коррекции. Когда все записи файла LC исчерпаны, массив LC закрывается, а оставшаяся часть массива L1 через рабочую область *ICX* выводится в массив L2.

Корректировка записи по одному или нескольким элементам осуществляется так: корректирующая запись составляется из одних символов " ␣ ", за исключением новых элементов данных; если в полях памяти *KOR* стоят пробелы, то соответствующие байты в буфере не заменяются, если же там стоят не пробелы, то соответствующие символы в памяти *ICX* заменяются на новые из памяти *KOR*.

Алгоритм информационного поиска. Основная функциональная особенность алгоритма и программы поиска состоит в обработке запроса пользователя. Этот запрос имеет следующий вид: < символическое имя области запроса > : < символическое имя области данных > : < имя файла > : < имя ключевого элемента данных > : < физическое имя элемента данных > : =

В случае отсутствия какого-либо звена в запросе разделители ":" ставятся подряд.

Основные функции алгоритма и программы поиска следующие: динамическое формирование имен файлов, а также параметров описания файлов (таких, как длина блока, длина записи и т.д.); динами-

ческая обработка запросов пользователя; поиск информации; формирование и вывод сообщений об ошибочных запросах; предоставление данных в указанной области; связь с оператором.

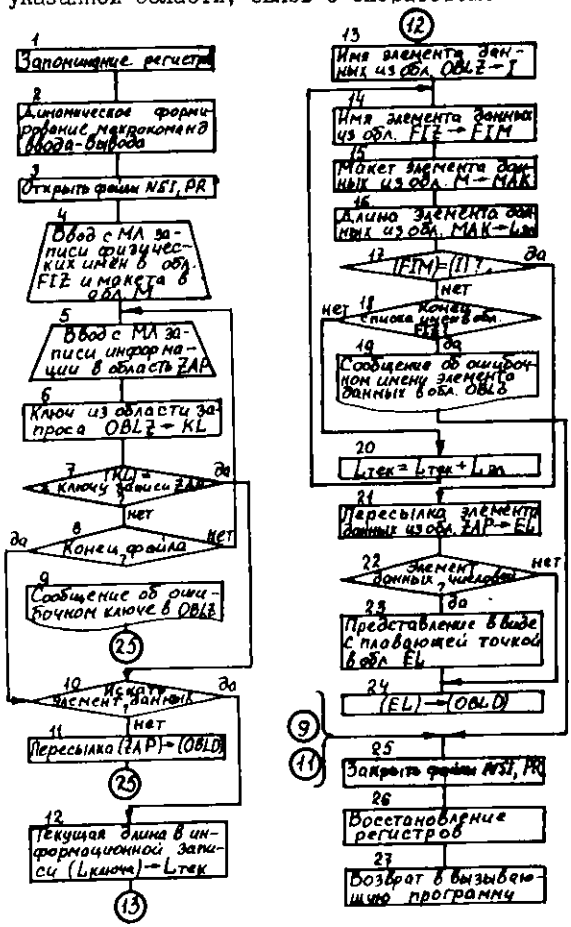


Рис. 3.4

На рис. 3.4 представлена принципиальная блок-схема программы поиска банка НСИ. Здесь символические имена имеют следующий смысл: *FIZ* – имя области в основной памяти для физических имен элементов файла; *M* – имя области макета массива; *ZAP* – имя области информационной записи; *KL* – имя области, в которой выделен ключевой элемент записи из запроса; *OBLD* – имя области данных; *FIM* – имя об-

ласти, где расположено физическое имя элемента данных, выделенное из области *FIZ* ; *MAK* – имя области, где расположен описатель информационного элемента, выделенный из макета; *L* – длина элемента в байтах; *ID* – имя области, где выделяется идентификатор элемента данных из *MAK* ; *EL* – промежуточный буфер для информационной записи.

В случае, если отсутствует запрашиваемая запись или запрашиваемый элемент данных с заданным физическим именем, формируется и выводится сообщение:

НЕТ ЗАПИСИ С ЗАДАННЫМ КЛЮЧОМ_____ или

НЕТ ЭЛЕМЕНТА ДАННЫХ С ИМЕНЕМ_____

На рис. 3.5 приведен пример поиска элемента РАЗМ␣КР в записи с ключевым элементом ТУ-104 в файле ХАРАКТЕРИСТИКИ САМОЛЕТОВ.

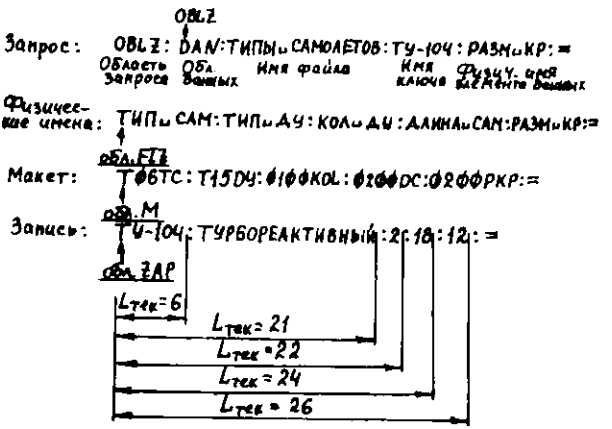


Рис. 3.5

В заключение запишем обращение к информационно-поисковой программе под именем *IPS* с языков Ассемблер и Фортран.

С языка Ассемблер

Имя	Операция	Операнды	Комментарии
⋮	⋮	⋮	⋮
	STM	*14, 1, SAVE*	Запоминание регистров I4,I5, Ø, I
	LA	*1, ZAPROS*	Адрес *OBLZ → R1*
	L	*15, =V(IPS)*	Имя *IPS → R15*

Продолжение

	BALR	*14,15*	Выполнение *IRS*
⋮	⋮	⋮	⋮
ZAPROS	*DC*	*A(OBLZ)*	Область запроса
	DC	*A(OBLD)*	Область данных
SAVE	*DS*	*4F*	Область сохранения регистров
⋮	⋮	⋮	⋮

С языка Фортран

```
      ⋮
   CALL  IPS ( OBLZ , OBLD )
      ⋮
```

В обоих случаях перед обращением должен быть сформирован запрос в области *OBLZ*.

4. БАНК ДАННЫХ ИНЭС-2

4.1. ОБЩИЕ СВЕДЕНИЯ

Банк данных ИНЭС-2 – это совокупность программных, языковых и технических средств, предназначенных для интегрированного хранения и коллективного использования данных. Банк позволяет работать с иерархическими и сетевыми структурами данных [3]. ИНЭС-2 работает под управлением операционной системы ОС-4.1 или ОС-6.1 и предназначен для эксплуатации на ЭВМ ЕС-1033, ЕС-1040, ЕС-1050.

Обращение к банку можно производить из программ, написанных на языках Фортран, ПЛ1, Кобол, Ассемблер. В ИНЭС-2 реализованы возможности диалогового режима обработки данных. Банк позволяет осуществить поиск нужной информации, высветить ее на экране дисплея, произвести ее корректировку. Диалоговая система позволяет работать с несколькими дисплеями одновременно.

Для описания структур данных используется ДЕРЕВО ОПИСАНИЯ ДАННЫХ (ДОД), которое хранится отдельно от данных. Сами данные хранятся в ДЕРЕВЕ ДАННЫХ (ДД).

4.2. ЯЗЫК ОПИСАНИЯ ДАННЫХ

Данные, с которыми работает пользователь, можно разделить на следующие типы: простые, структурные и ссылочные (рис. 4.1).

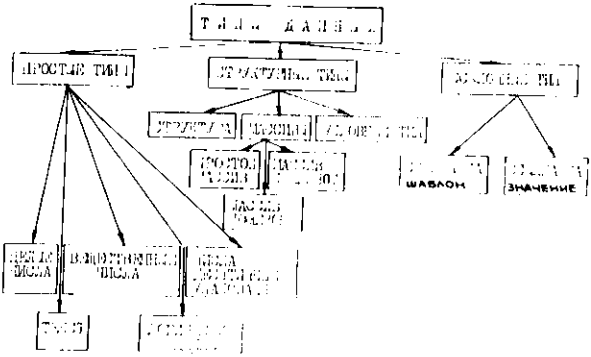


Рис. 4.I

Простые типы предназначены для хранения элементарных единиц информации. Это могут быть: целые числа (двоичные, до 4 байт); вещественные числа (двоичные, до 8 байт); десятичные числа (упакованные, до I5 байт); текст (до 256 байт); логические данные (до 256 байт).

Описание данных простых типов имеет вид

n : < описание типа > */SIZE* = число / ;

где *n* – идентификатор описываемого данного простого типа, может состоять из букв русского алфавита; *SIZE* – спецификация, в которой определяется объем памяти, занимаемый описываемым данным; число – предельное значение памяти, отводимое для хранения описываемого данного (в байтах).

Для описания простых типов введены следующие операторы:

целые числа – *INT / SIZE* = число / ;
вещественные числа – *REAL / SIZE* = число / ;
десятичные числа – *DEC /SIZE* = число / ;
текстовые данные – *TEXT / SIZE* = число / ;
логические данные – *LOG /SIZE* = число / .

Если *SIZE* не указано, то принимаются следующие значения: для *INT* – 4 байта, *REAL* – 8 байт, *DEC* – I5 байт, *TEXT* и *LOG* – по 256 байт.

Например, *A* – переменная простого типа (целая переменная) длиной 2 байта. Описание этой переменной имеет вид *A : INT /SIZE=2/;*

Структурные типы данных представляют собой объединение нескольких данных в одно.

Определены следующие структурные типы: I) структура; 2) массивы: простой, с номером, с ключом; 3) условная структура.

Структура представляет собой совокупность элементов, связанных иерархической зависимостью. Структура (рис. 4.2) описывается следующим образом:

$$\left.\begin{array}{l} n : STRUCT / v_{11}, v_{12}, \ldots, v_{1N} / ; \\ v_{11}: STRUCT / v_{111}, v_{112}, \ldots, v_{11p_1} / ; \\ v_{12}: STRUCT / v_{121}, v_{122}, \ldots, v_{12p_2} / ; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ v_{1N}: STRUCT / v_{1N1}, v_{1N2}, \ldots, v_{1Np_N} /, \end{array}\right\} \qquad (4.1)$$

где n – идентификатор нулевого уровня структуры; $v_{11}, v_{12}, \ldots, v_{1N}$ – идентификаторы первого уровня; $v_{111}, v_{117}, \ldots, v_{11p_1}$ – идентификаторы второго уровня.

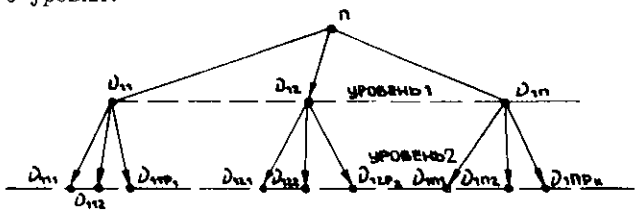


Рис. 4.2

Структура (4.I) содержит описания всех простых типов данных, входящих в структуру (терминальные данные).

Определены следующие типы массивов: простой, с номером, с ключом.

В простом массиве пользователю представляется возможность только последовательного доступа к элементам (можно указать I-й или последний элемент массива, или следующий или предыдущий по отношению к текущему элементу).

Простой массив описывается следующим образом:

$$\left.\begin{array}{l} n : ARRAY / SIZE = число / (n1) ; \\ n1: \langle описатель\ простого\ типа \rangle ; \end{array}\right\}$$

где n – идентификатор массива; $n1$ – идентификатор элементов массива; $SIZE$ = число – в этой спецификации определяет количество элементов массива.

Пример I. Элементы простого массива содержат информацию о размерах профилей материала ДЕТ, которые имеются на данном предприятии и которыми может воспользоваться конструктор. Массив имеет идентификатор ДЕТ, а элемент массива имеет идентификатор ТОЛЩИНА. Массив описывается следующим образом:

$$\left.\begin{array}{l} ДЕТ: \quad ARRAY / SIZE = 100 / \ (ТОЛЩИНА); \\ ТОЛЩИНА: \ REAL / SIZE = 4 / ; \end{array}\right\}$$

Массив с номером – это массив, к элементам которого можно обращаться по номерам элементов. В описании этого типа массива должна присутствовать спецификация *NUM YES*

n: *ARRAY* / *SIZE* = *число*, *NUM* = *YES* / (*n1*) ;
n1: < описание простого типа > ;

Здесь *n* – идентификатор массива; *n1* – идентификатор элемента массива; *SIZE* – спецификация, в которой определено количество элементов массива; *NUM* – спецификация, в которой определен тип массива.

Пример 2. Задан массив МАРКИ МАТЕРИАЛОВ, в элементах которого находятся списки авиационных материалов, используемые при проектировании заданного изделия. В массив включена следующая информация о сталях и сплавах:

легированная сталь: 30ХГСА, 25ХГСА, 30ХМА, 45ХА, 1ГХА,...
углеродистая сталь: СТ10, СТ25, СТ35, СТ45А,...
алюминиевые сплавы: Д1М, Д1Т, Д16Т, АВМ, АВТ1, Д1ВТ,...
магниевые стали: МА1, МА2, МА3, МЛ4Т4, МЛ6Т6,...

Массив МАРКИ МАТЕРИАЛОВ описывается так:

МАРКИ МАТЕРИАЛОВ: *ARRAY* / *SIZE* = 4, *NUM* = *YES* / (*материал*);
МАТЕРИАЛ: *TEXT* / *SIZE* = 100 / ;

Длина одного элемента массива равна 100 байт.

Массив с ключами является мощным средством увеличения эффективности работы с элементами массива. В этом случае элемент массива должен быть структурой, один из элементов которой объявлен ключом. Обращаться к элементам массива можно только по ключу. Этот тип массива описывается следующим образом:

n: *ARRAY* / *SIZE* = *число* / (*n1*) ;
n1: *STRUCT* / *KEY* = *имя* / (v_{11}, v_{12}, ..., v_{1m}) ;
v_{11}: < описание > ;
v_{12}: < описание > ;
. . . . . .
v_{1m}: < описание > ;

где *n* – идентификатор массива; *n1* – идентификатор элемента массива; *SIZE* – спецификация, в которой определено количество элемента массива; *KEY* – спецификация, в которой определено имя простого типа, используемое в качестве ключа; v_{11}, v_{12}, ..., v_{1m} – идентификаторы I-го уровня структуры.

Пример 3. Опишем массив с ключами БАНК САМОЛЕТОВ. Массив состоит из 100 элементов. Элемент массива имеет идентификатор САМОЛЕТ (рис. 4.3).

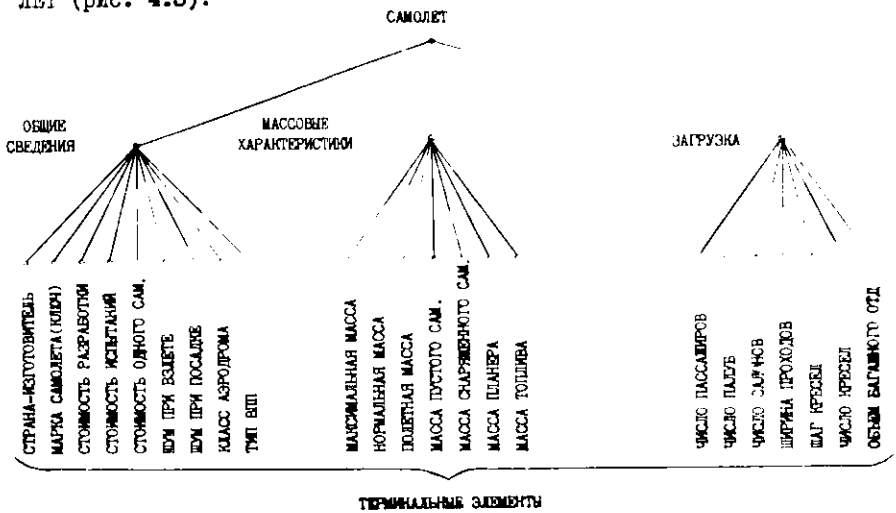


Рис. 4.3

В качестве ключа, по которому осуществляется поиск, выбрана простая переменная (терминальная вершина структуры) с именем МАРКА САМОЛЕТА.

```
БАНК САМОЛЕТОВ: ARRAY /SIZE=100/(САМОЛЕТ);
     САМОЛЕТ:   STRUCT/KEY= МАРКА САМОЛЕТА/
                        (ОБЩИЕ СВЕДЕНИЯ;
                        ВЕСОВЫЕ ХАРАКТЕРИСТИКИ;
                        ЗАГРУЗКА);
ОБЩИЕ СВЕДЕНИЯ:         (СТРАНА ИЗГОТОВИТЕЛЬ;
                        МАРКА САМОЛЕТА;
                        СТОИМОСТЬ РАЗРАБОТКИ;
                        СТОИМОСТЬ ИСПЫТАНИЙ;
                        СТОИМОСТЬ ОДНОГО САМОЛЕТА;
                        ШУМ ПРИ ВЗЛЕТЕ;
                        ШУМ ПРИ ПОСАДКЕ;
                        КЛАСС АЭРОДРОМА;
                        ТИП ВШ);
```

```
ВЕСОВЫЕ ХАРАКТЕРИСТИКИ:   (МАКС МАССА;
                          НОРМ  МАССА;
                          ПОЛЕТН  МАССА;
                          МАССА ПУСТОГО САМОЛЕТА;
                          МАССА СНАРЯЖЕННОГО САМОЛЕТА;
                          МАССА ПЛАНЕРА;
                          МАССА ТОПЛИВА);
        ЗАГРУЗКА:         (ЧИСЛО ПАССАЖИРОВ;
                          ЧИСЛО ПАЛУБ;
                          ЧИСЛО САЛОНОВ;
                          ШИРИНА ПРОХОДОВ;
                          ШАГ КРЕСЕЛ;
                          ЧИСЛО КРЕСЕЛ;
                          ОБЪЕМ БАГАЖНОГО ОТДЕЛЕНИЯ);
СТРАНА ИЗГОТОВИТЕЛЬ:        TEXT /SIZE = 30/ ;
МАРКА САМОЛЕТА:             TEXT /SIZE = 20/ ;
СТОИМОСТЬ РАЗРАБОТКИ:       REAL /SIZE = 4 / ;
СТОИМОСТЬ ИСПЫТАНИЙ:        REAL /SIZE = 4 / ;
СТОИМОСТЬ ОДНОГО САМОЛЕТА:  REAL /SIZE = 4 / ;
ШУМ ПРИ ВЗЛЕТЕ:             REAL /SIZE = 4 / ;
ШУМ ПРИ ПОСАДКЕ;            REAL /SIZE = 4 / ;
КЛАСС АЭРОДРОМА:            TEXT /SIZE = 10 / ;
ТИП ВПП:                    TEXT /SIZE = 10 / ;
МАКСИМАЛЬНАЯ МАССА:         REAL /SIZE = 4 / ;
НОРМАЛЬНАЯ МАССА:           REAL /SIZE = 4 / ;
ПОЛЕТНАЯ МАССА:             REAL /SIZE = 4 / ;
МАССА ПУСТОГО САМОЛЕТА:     REAL /SIZE = 4 / ;
МАССА СНАРЯЖЕННОГО САМОЛЕТА: REAL /SIZE = 4 / ;
МАССА ПЛАНЕРА:              REAL /SIZE = 4 / ;
МАССА ТОПЛИВА:              REAL /SIZE = 4 / ;
ЧИСЛО ПАССАЖИРОВ:           INT  /SIZE = 2 / ;
ЧИСЛО ПАЛУБ:                INT  /SIZE = 2 / ;
ЧИСЛО САЛОНОВ:              INT  /SIZE = 2 / ;
ШИРИНА ПРОХОДОВ:            REAL /SIZE = 4 / ;
ШАГ КРЕСЕЛ:                 REAL /SIZE = 4 / ;
ЧИСЛО КРЕСЕЛ:               INT  /SIZE = 2 / ;
ОБЪЕМ БАГАЖНОГО ОТДЕЛЕНИЯ:  REAL /SIZE = 4 / ;
```

Условные структуры – это структуры, полное описание которых хранится в соответствующей библиотеке загрузочных модулей. В дан-

ных хранится только часть структуры, необходимая для работы в текущий момент времени. Описание такой структуры имеет ключевое слово

$$\left.\begin{array}{l} n : \ COND\ /\&/\ (\nu_{11}, \nu_{12}, \ldots, \nu_{1n})\ ; \\ \nu_{11} : \ <\text{описание}>\ ; \\ \nu_{12} : \ <\text{описание}>\ ; \\ \ldots\ldots\ldots\ldots \\ \nu_{1n} : \ <\text{описание}>\ ; \end{array}\right\}$$

где n – идентификатор условной структуры; $\&$ – список спецификаций; $\nu_{11}, \ldots, \nu_{1n}$ – идентификаторы первого уровня структуры.

В ИНЭС-2М определены данные ссылочного типа: ссылки на шаблон и ссылки на значения.

Ссылки на шаблон предназначены для того, чтобы не дублировать описания одинаковых данных. Под новое данное, описываемое с помощью шаблона, отводится память. Использование шаблонов упрощает описание структур данных. Пользователь при описании структур указывает, что некоторый участок структуры имеет такое же строение, как и структура, описанная ранее. Описание по шаблону позволяет организовать в базе данных структуры теоретически бесконечной глубины. Объем такой базы ограничен только объемом памяти внешнего носителя. Данное с помощью шаблона описывается следующим образом:

< имя нового данного > : AS'< имя базового данного >$'$;

Здесь имя базового данного - имя данного, которое было описано ранее.

Пример 4. При проектировании заданной инженерной конструкции разработчик может воспользоваться набором профилей из альтернативных материалов: алюминия, магния, титана. Каждый тип профиля характеризуется восемью параметрами:

B	H	S	$S1$	R	F	$Y0$	JX
Ширина нижней полки профиля	Высота профиля	Регулярная толщина стенки	Толщина полки	Радиус закругления	Площадь сечения	Координата центра тяжести	Момент инерции

Базу данных назовем ПРОФИЛИ. Структура этой базы представлена на рис. 4.4.

Алюминиевые, магниевые и титановые профили разбиты на четыре типа: T1, T2, T3, T4. Каждый тип делится на четыре подтипа: T21, T22, T23, T24. Каждый подтип описывается восемью параметрами: B, H, S, $S1$, R, F, $Y0$, JX, инженерные наименования которых приве-

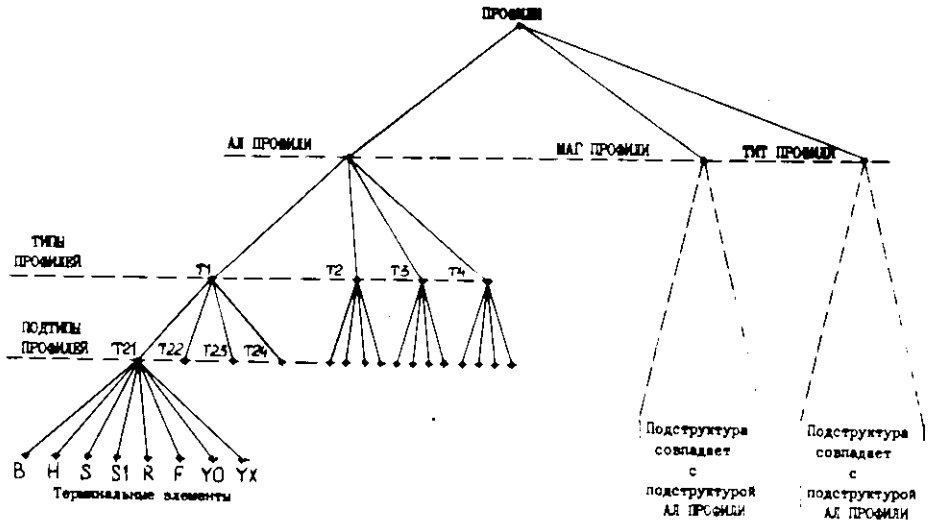

ПРОФИЛИ
АЛ ПРОФИЛИ
МАГ ПРОФИЛИ
ТИТ ПРОФИЛИ
ТИПЫ ПРОФИЛЕЙ
T1
T2
T3
T4
ПОДТИПЫ ПРОФИЛЕЙ
T21
T22
T25
T24
B H S S1 R F YO YX
Терминальные элементы
Подструктура совпадает с подструктурой АЛ ПРОФИЛИ
Подструктура совпадает с подструктурой АЛ ПРОФИЛИ

Рис. 4.4

дены в таблице. Особенность данной базы заключается в том, что три подструктуры АЛ ПРОФИЛИ, МАГ ПРОФИЛИ, ТИТ ПРОФИЛИ имеют одинаковые строения. Поэтому полностью опишем подструктуру АЛ ПРОФИЛИ, а подструктуры МАГ ПРОФИЛИ, ТИТ ПРОФИЛИ опишем с помощью шаблонов.

```
ПРОФИЛИ:    STRUCT (АЛ_ПРОФИЛИ; AS'ПР';
                    МАГ_ПРОФИЛИ; AS'ПР';
                    ТИТ_ПРОФИЛИ; AS'ПР');

ПР: STRUCT( T1:  AS'T';
            T2:  AS'T';
            T3:  AS'T';
            T4:  AS'T');

T: STRUCT( T2I:  AS'TT';
           T22:  AS'TT';
           T23:  AS'TT';
           T24:  AS'TT');

TT: STRUCT( B:  REAL /SIZE = 4/;
            H:  REAL ;
            S:  REAL ;
            S1: REAL ;
            R:  REAL ;
            F:  REAL ;
            DX: REAL ;
```

Ссылки на значения предназначены для того, чтобы не дублировать данные, которые уже созданы ранее. Использование ссылки на значение не приводит к выделению внешней памяти. Формат описания:

< имя нового данного > : REF' < имя базового данного > ';

Здесь имя базового данного - имя данного, на которое делается ссылка.

4.3. ЯЗЫК МАНИПУЛИРОВАНИЯ ДАННЫМИ

Язык манипулирования данными (язык запросов) служит для формирования вопроса пользователем и организации поиска нужной информации в базах данных.

В языке разрешается употреблять числовые и текстовые константы, рабочие ячейки, рабочие массивы. При формировании запроса можно использовать:

знаки арифметических операций:

$\pm$, / , $\uparrow$

знаки операций сравнения:

= , $\neg$ = , > , < , > = , < =

знаки логических операций:

AND, OR, $\neg$

условные выражения ПЛ1 типа

IF < e_1 > THEN ... ;

IF < e_2 > THEN ... ELSE ... ;

кванторы:

EVERY (каждый)

FXIST (существует).

Организация циклов

При организации циклического просмотра базы данных используется следующая конструкция:

ALL ␣ | ␣ < условия >

ALL указывает, что требуется просмотреть все вершины, непосредственно подчиненные текущей вершине; < условия > - условия поиска данных, в которых могут использоваться логические операции, операции сравнения, текстовые константы.

Пример. Составить запросы к базе данных ПРОФИЛИ, описание которой составлено в примере 4. Сформулируем следующие типы запросов:

Тип I. Из всех профилей, информация о которых имеется в базе, выбрать только те, которые удовлетворяют следующим ограничениям:

$$\left.\begin{array}{lll} BH \leqslant B \leqslant BB \\ HH \leqslant H \leqslant HB \\ SH \leqslant S \leqslant SB \\ SIH \leqslant SI \leqslant SIB \\ RH \leqslant R \leqslant RB \\ FH \leqslant F \leqslant FB \\ YOH \leqslant YO \leqslant YOB \\ JXH \leqslant JX \leqslant JXB \end{array}\right\} \qquad (4.2)$$

Запрос будет иметь вид

ПРОФИЛИ . *ALL* ␣ | ␣ (< условие >)

(< условие >) = (*B* < *BB*) & (*B* > *BH*) & (*H* < *HB*) ...

Тип П. Из всех профилей заданного материала выбрать те, которые удовлетворяют двухсторонним ограничениям типа (4.2). Такая конкретизация материала ускорит поиск, так как нужно просматривать часть базы данных, а не всю базу, как в запросе типа I. В этом случае запрос будет следующим:

ПРОФИЛИ. < тип и материал >. *ALL* ␣ | ␣ (< условие >) ;

Например, при поиске нужного алюминиевого профиля, удовлетворяющего требованиям (4.2), необходимо написать запрос:

ПРОФИЛИ. *АЛ* ␣ ПРОФИЛИ. *ALL* ␣ | ␣ (< условие >) ;

Тип Ш. Из всех профилей заданного типа и заданного материала выбрать профили, удовлетворяющие условиям (4.2):

ПРОФИЛИ. < материал > . <тип профиля>. *ALL* | (< условия >);

Например:

ПРОФИЛИ. ТИТ ␣ ПРОФИЛИ . *T2. ALL* | (< условия >) ;

Циклы с условными выражениями

ALL ␣ | (< условия с условными выражениями >)

Например:

A. ALL ␣ | ␣ (*IF* ␣ *B* = *C* ␣ *OR* ␣ *B* < *D*+5) *THEN X, Y*
ELSE W, Z

Если значения в подчиненных вершинах *B* и *C* совпадают или значение в вершине *B* меньше, чем в вершине *D* плюс 5, то перейти к анализу содержимого в вершинах *X* и *Y* , в противном случае перейти к анализу содержимого в вершинах *W* и *Z* .

5. ИНФОРМАЦИОННЫЕ СИСТЕМЫ НА БАЗЕ АЛГОРИТМИЧЕСКОГО ЯЗЫКА ПЛЮП

В главе рассматриваются средства ПЛЮС и ПЛЮП* для формирования и обработки произвольных списковых структур: указатели, базированные переменные, областные переменные, структуры с динамическими изменяемыми границами массивов и строк. Рассматривается пример создания информационной системы на базе средств ПЛЮП, которая может использоваться в локальных базах данных.

*ПЛЮС - базовый ПЛ, ПЛЮП - оптимизирующий ПЛ.

5.I. ОБЩИЕ СВЕДЕНИЯ

Универсальный алгоритмический язык ПЛ1 предназначен для программирования научных и инженерных расчетов, обработки экономической информации, символьной информации, произвольных списковых структур, разработки систем математического обеспечения. ПЛ1 был разработан после появления алгоритмических языков Алгол-60, Фортран-4, Кобол. ПЛ1 объединяет возможности этих языков и предоставляет программистам ряд дополнительных возможностей [4, 9, 11, 33]:

- большой спектр данных: арифметические данные с фиксированной и плавающей запятой, с двоичным и десятичным основанием, символьные данные, битовые данные, позволяющие работать с малоразрядными переменными, данные управления программой (рис. 5.1);

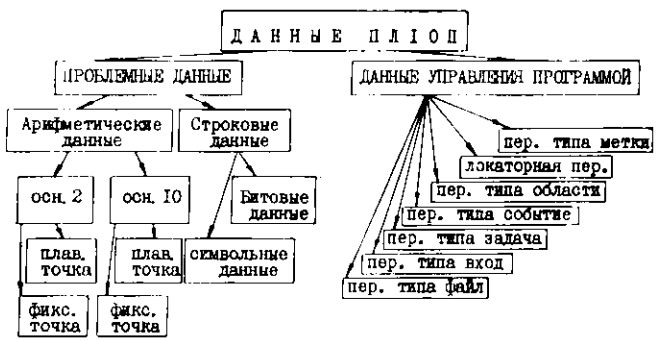


Рис. 5.1

- в языке определены операции над массивами и структурами;
- имеется большой набор встроенных функций для обработки массивов, структур, символьных данных;
- имеется развитый механизм распределения оперативной памяти: при использовании управляемого распределения памяти программист сам распределяет память под данные и освобождает эту память;
- в ПЛ1 введены средства формирования и обработки произвольных списковых структур, которые являются основными программными элементами информационно-справочных систем и трансляторов;
- пользователю предоставлены средства для параллельной обработки программы (мультиобработки), при которой отдельные ее участки, использующие различные ресурсы вычислительной машины, могут обрабатываться одновременно. Этот режим в некоторых случаях позволяет ускорить выполнение программы;

– препроцессорные средства языка позволяют модифицировать текст исходных программ, содержащих препроцессорные переменные, операторы, процедуры. После этапа препроцессирования генерируется нужный вариант программы на ПЛ1;

– в языке определены средства для работы с последовательными, региональными и индексно-последовательными наборами данных.

Оптимизирующий транслятор с ПЛ1 (ПЛ1ОП), реализованный в ОС 6.1, предоставляет пользователю ряд дополнительных возможностей [15]:

1. Сокращает время выполнения программы в среднем на 50% по сравнению с обычным транслятором ПЛ1, а во многих случаях и уменьшает объем требуемой памяти.

2. В ПЛ1ОП введены средства, которые дают возможность вызывать из программы на ПЛ1ОП программы, написанные на Коболе, Фортране или Ассемблере, а из программ на Коболе, Фортране или Ассемблере вызывать программы, написанные на ПЛ1ОП. В атрибуте *OPTIONS* указывается ключевое слово *FORTRAN*, *COBOL*, или *ASSEMBLER*. При этом производится преобразование передаваемых аргументов в соответствии с требованиями вызывающей программы [15, с.81–106].

3. В самоопределяемых структурах допускается любое количество массивов с переменными границами.

4. В ПЛ1ОП введены более мощные средства отладки, расширены возможности обработки прерываний, улучшена диагностика как во время трансляции, так и во время выполнения.

5. Расширена концепция умолчания – введен оператор *DEFAULT*, в котором для указанных имен устанавливаются атрибуты, присваиваемые по уполчанию. Формат оператора *DEFAULT*:

DEFAULT ␣ *RANGE* (*I*, *Q*, *K*) ␣ *DEC* ␣ *FLOAT* (16),
RANGE (*K*, *L*, *M*) ␣ *CHAR* (10);

В этом примере всем идентификаторам, начинающимся с символов *I* . *Q*, *K*, присваиваются атрибуты *DEC FLOAT*(16), а все идентификаторы, начинающиеся с символов *K*, *L*, *M*, объявляются символьными переменными длиной в 10 байт.

6. В ПЛ1ОП есть средства для динамической загрузки процедур в оперативную память во время выполнения программ [14]. Динамическая загрузка процедур, активизация процедур и освобождение памяти осуществляются с помощью операторов *FETCH*, *CALL*, *RELEASE*.

Поиск и динамическая загрузка процедуры, находящейся на внешнем носителе, в оперативную память осуществляются оператором

FETCH ␣ <имя процедуры>;

Активизация процедуры осуществляется оператором *CALL* после окончания загрузки процедуры в оперативную память:

CALL ␣ <имя ␣ процедуры>[(список факт. пар.)] .

Процедура может находится в оперативной памяти до конца выполнения программы. Если этого не требуется, то область памяти, занимаемая процедурой, может быть освобождена с помощью оператора

RELEASE ␣ <имя процедуры>;

Динамические загружаемые процедуры должны удовлетворять требованиям, перечисленным в работе [I4].

7. Введены переменные типа вход.

8. Введены переменные типа файл. Эти переменные служат для установки соответствия между описаниями файлов и файлами. Переменную типа файл особенно удобно использовать при программировании операций ввода-вывода с банками данных.

Приведем интересное высказывание известного американского специалиста Г. Майера [20, с. I46]: "Самое большое достоинство языков высокого уровня заключено в их способности работать со сложными структурами данных. Опытный программист согласится, что программирование состоит в работе с данными, а не с тонкостями машины...Следует понимать, что эффективность достигается разумным выбором алгоритмов и структур данных, а не благодаря микроэффективности, обеспечиваемой использованием машинного языка. Более того, современные оптимизирующие компиляторы генерируют замечательные по эффективности объектные программы. Подтверждением сказанного является большая операционная система *Multics* , реализованная на ПЛ1 в совместном проекте лабораторий Белл, Дженерал электрик и МТИ. Только 5% системы написано на машинном языке..."

В работе [7] описывается возможность создания в рамках языка ПЛ1 абстрактных типов данных, т.е. возможность использования кластероподобного стиля программирования. Кластером называется языковая конструкция для описания абстрактных типов данных. Абстракция заключается в том, что за пределами кластера доступны только операции над данными. Этим достигается независимость программ от представления данных. В работе [6] рассматриваются различные аспекты обработки списковых структур средствами языка ПЛ1. Работа [5] посвящена интерактивным средствам программирования на базе средств ПЛ1.

5.2. СРЕДСТВА ПЛ1ОС И ПЛ1ОП ДЛЯ СОЗДАНИЯ И ОБРАБОТКИ СПИСКОВЫХ СТРУКТУР

Указатели

Указатель – это переменная, содержащая адрес ячейки оперативной памяти, начиная с которой располагается какая-либо переменная.

Указатель объявляется в операторе *DCL* с описателем PTR:

DCL ␣ имя ␣ PTR ;

Указатель принадлежит к классу данных управления программой. Допустимыми операциями являются две операции сравнения =, ¬ = . С помощью встроенной функции *NULL* указателю можно присвоить пустое значение, т.е. адрес, который не является адресом ячейки оперативной памяти. Функция *NULL* не имеет аргументов. Присваивание указателю значения функции *NULL* осуществляется в операторе присваивания *P1 = NULL* ; функцию *NULL* можно использовать для проверки состояния указателя:

IF ␣ P1 ¬ = NULL ␣ THEN ␣ DO ; ... END ;

Базированные переменные

Базированными переменными называются переменные, расположение которых в памяти определяется указателем.

Базированная переменная объявляется в операторе *DCL* с описателем *BASED* :

DCL ␣ ИМЯ *␣ BASED (P) ;*

За описателем *BASED* в круглых скобках указан идентификатор собственного указателя базированной переменной. Базированная переменная размещается в оперативной памяти с помощью оператора *ALLOCATE* :

ALLOCATE ␣ ИМЯ *␣ SET* (собственный указатель базированной переменной);

Оператор *ALLOCATE* выделяет базированной переменной требуемую область оперативной памяти и присваивает указателю, заданному в конструкции *SET* , значение адреса этой области. Память, занятая базированной переменной, освобождается оператором *FREE* .

Например, рассмотрим фрагмент программы, в котором размещается базированная структура:

```
DCL ␣ 1 ␣ STR ␣ BASED (PØ),
      2 ␣ A (1Ø,1Ø) ␣ DEC ␣ FLOAT ;
      ⋮
ALLOCATE ␣ STR ␣ SET (PØ) ;
      ⋮
FREE ␣ STR ;
```

Здесь базированная структура *STR* имеет собственный указатель *PØ* . На втором уровне структуры располагается массив A(IO,IO). Память под эту структуру будет выделена оператором *ALLOCATE*.

Самоопределяющиеся структуры

Самоопределяющимися структурами называются структуры, содержащие массивы или строки, границы которых устанавливаются перед размещением. Информация о протяженности массива или строки хранится в переменной, которая объявлена в самой структуре.

Самоопределяющаяся структура объявляется следующим образом:

```
DCL ␣ 1 ␣ W ␣ BASED (P),
          2 ␣ A ␣ BIN ␣ FIXED,
          2 ␣ B ␣ BIN ␣ FIXED,
          2 ␣ C (X ␣ REFER (A)) ;
```

В состав базированной структуры *W* входят :две скалярные переменные *A* и *B* и массив *C* с изменяемой границей. В конструкции *(X ␣ REFER(A)* переменная *X* определяет число элементов массива . При размещении структуры *W* в оперативной памяти с помощью оператора *ALLOCATE* ее элементу *A* присваивается значение *X* и для элементов массива выделяется требуемая память. Например, при размещении

```
X = 1Ø ; ␣ ALLOCATE ␣ W ;
```

массив *C* будет состоять из IO элементов, а при размещении

```
X = 1ØØ ; ␣ ALLOCATE ␣ W ;
```

массив *C* будет состоять из IOO элементов.

Самоопределяющиеся структуры ПЛIOС должны удовлетворять следующим ограничениям:

I. *REFER* – объекты задаются в структуре только один раз при объявлении верхней границы первого измерения массива или строки.

2. *REFER* – объекты объявляются как элементы структуры только на втором уровне.

Самоопределяющиеся структуры ПЛIOП имеют следующие дополнительные возможности по сравнению с самоопределяющимися структурами ПЛIOС:

I. Структура может иметь на вторых уровнях несколько *REFER* объектов.

2. В качестве элемента структуры может выступать область, размер которой может изменяться динамически, если эта область объявлена с режимом *REFER*.

Пример I. Разместить в оперативной памяти самоопределяющуюся структуру ПЛIОП,в состав которой входят два массива с переменными границами:

```
DCL ␣ 1 ␣ STR ␣ BASED(P),
      2 ␣ A1 ␣ BIN ␣ FIXED,
      2 ␣ A2 ␣ BIN ␣ FIXED,
      2 ␣ R1 (X1 ␣ REFER (A1), 10) ␣ DEC ␣ FLOAT,
      2 ␣ R2 (X2 ␣ REFER (A2), 20) ␣ DEC ␣ FLOAT,
      (X1, X2) ␣ BIN ␣ FIXED ;
      X1, X2 = 10 ; ␣ ALLOCATE ␣ STR ␣ SET (P) ;
      X1 = 20 ; X2 = 30 ; ␣ ALLOCATE ␣ STR ␣ SET(P) ;
```

При первом размещении структуры *STR* массивы R1 и R2 имеют по I0 элементов, при втором размещении массив R1 имеет 20 элементов, а массив R2 - 30 элементов.

Областные переменные

Область или переменная типа области представляет собой область оперативной памяти, в которой размещаются базируемые переменные.

Область объявляется в операторе *DCL* с описателем *AREA*:

DCL ␣ имя ␣ *AREA* [(РАЗМЕР)] ;

Размер области в байтах задается десятичной константой, выражением или звездочкой. По умолчанию размер области - I000 байт. Максимальный размер области для ПЛIОС - 32767 байт и для ПЛIОП - I67772I5 байт [I4]. Область, кроме объема, указанного при объявлении, содержит I6 байт служебной информации, описывающей состояние области и передаваемой вместе с областью.

Размещение области в оперативной памяти осуществляется оператором *ALLOCATE*, а освобождение памяти - оператором *FREE*. Например:

```
DCL ␣ Q1 ␣ AREA (5000) ;
    ALLOCATE ␣ Q1 ;
    FREE ␣ Q1 ;
```

Области могут быть введены и выведены с помощью записеориентированного ввода-вывода.

После размещения в оперативной памяти область имеет состояние *EMPTY* (пусто). Это означает, что в области нет размещений базированных переменных. Если есть необходимость устранить все размещения, сделанные ранее в области, то можно использовать встроенную функцию *EMPTY* . Например:

```
DCL ␣ A ␣ AREA (6ØØØ), RQ ␣ BASED(P);
        ⋮
        ALLOCATE ␣ A ;
        ALLOCATE ␣ RQ ␣ IN(A) ;
        ⋮
        A = EMPTY ;
```

Второй оператор *ALLOCATE* размещает в области *A* базированную переменную *RQ* . После обработки в области *A* все размещения переменной *RQ* уничтожаются с помощью функции *EMPTY* в операторе присваивания *A = EMPTY* ;

Смещения

Смещение представляет собой адрес объекта, размещенного в базированной области, относительно ее начала. Смещение объявляется в операторе *DCL* с атрибутом *OFFSET* , за которым следует имя области

DCL ␣ ИМЯ ␣ *OFFSET* (имя области);

Смещения имеют определенные преимущества перед указателями, так как не зависят от положения области внутри оперативной памяти. Это преимущество является решающим при вводе областей с внешних носителей в различные места оперативной памяти, а также при присваивании значений одних областей другим.

Над смещениями определены две операции: = и ¬= .

Значение смещения может быть присвоено указателю и наоборот. Если смещение присваивается указателю, то его значение увеличивается на величину указателя, базирующего область. Если смещению присваивается значение указателя, то значение последнего уменьшается на величину указателя, базирующего область.

Смещению можно присвоить пустое значение с помощью встроенной функции *NULL* в операторе присваивания.

Пример. Разместить в оперативной памяти самоопределяющуюся структуру, в состав которой входят области с переменными размерами.

```
2    1   0  | DCL 1 R BASED(PR),
            |       2 (N1,N2,N3) BIN FIXED,
            |       2 T1 AREA(E1 REFER(N1)),
            |       2 T2 AREA(E2 REFER(N2)),
            |       2 T3 AREA(E3 REFER(N3)),
            |       2 SMT1 OFFSET(T1),
            |       2 SMT2 OFFSET(T2),
            |       2 SMT3 OFFSET(T3),
            |       (E1,E2,E3) BIN FIXED;
3    1   0  | DCL PR PTR,(EMPTY,NULL) BUILTIN;
            |
            |
4    1   0  |             E1,E2,E3=5000; ALLOCATE R SET(PR);
6    1   0  |             T1,T2,T3=EMPTY;
7    1   0  |       SMT1,SMT2,SMT3=NULL;
            |
```

В структуре *R* объявлены три области T1, T2, T3 с переменными размерами и три смещения *T1* , *T2* , *T3* относительно начала соответствующих областей. В операторе 4 задаются размеры областей T1, T2, T3. После размещения структуры *R* в оперативной памяти области займут объемы по 5000 байт. Затем всем трем областям в операторе 6 присваивается значение "пусто", а трем смещениям присваивается значение встроенной функции *NULL* .

Списки

Однонаправленные списки. В этом типе списка каждый элемент состоит из содержательной части *S* , объем и структура которой определяются пользователем, и указателя, в котором находится адрес последующего элемента (рис. 5.2). После просмотра содержательной части очередного элемента считывается адрес последующего элемента и осуществляется переход на этот элемент. Для фиксации начала и конца списка можно ввести головной и концевой указатели. На рис. 5.2 головной указатель - *PSTART*, концевой указатель - *PEIN* .

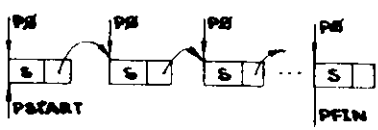


Рис. 5.2

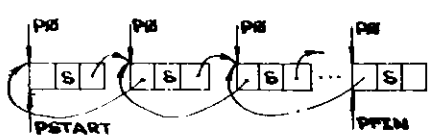


Рис. 5.3

Двухнаправленные списки. В этом типе списка каждый элемент состоит из содержательной части *S* , объем и структура которой определяются пользователем, и двух указателей. Один из указателей содержит адрес последующего элемента списка, другой - адрес предыдущего элемента списка (рис. 5.3). После просмотра содержательной части очередного элемента списка осуществляется переход по одному из адресов. Этот тип списка можно обрабатывать в двух направлениях.

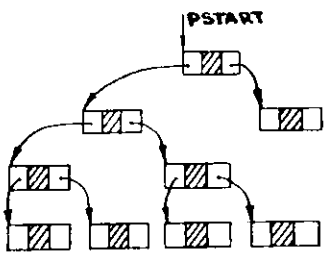


Рис. 5.4

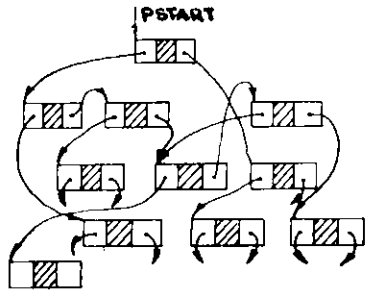


Рис. 5.5

Бинарные списки. Структура этого типа списка показана на рис. 5.4. Каждый элемент списка состоит из содержательной части S, объем и структура которой определяются пользователем, и двух указателей. После анализа содержательной части элемента осуществляется переход по одному из адресов.

Сети. Этот тип списка показан на рис. 5.5. В сети связи организуются между любыми элементами.

5.3. ОРГАНИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Рассмотрим информационную систему, содержащую сведения по авиационным двигателям. Система реализована на ПЛ1ОП в виде однонаправленного списка, размещенного в заданной области. Каждый элемент списка содержит следующую информацию:

- аннотацию (*ANNOT*), в которой содержится информация (реклама) по данному двигателю; аннотация представляет массив, состоящий из 80-байтовых записей, количество записей определяется пользователем;
- массив субподрядных фирм (*SUBE*) содержит названия фирм, участвующих в производстве данного двигателя; массив состоит из 20-байтовых записей, количество которых определяется пользователем;
- название двигателя;
- 10 параметров двигателя.

Размещение списка в области

Размещение списка в области осуществляется процедурой *PAZM* , текст которой приведен на рис. 5.6, а структура области после размещения списка показана на рис. 5.7. Рассмотрим пооператорно эту процедуру. В операторах 2 – 5 описываются:

```
PL/I OPTIMIZING COMPILER          PAZM:PROC OPTIONS(MAIN);

                     SOURCE LISTING

    STMT LEV NT

      1       0  | PAZM:PROC OPTIONS(MAIN);
      2    1  0  | DCL 1 STR BASED(PO),
                 |        2 OBL AREA(13[illegible]),
                 |        2 SMSTART OFFSET(OBL),
                 |          BUFER CHAR(14016),STROKA CHAR(14016) BASED(PO);
                 |
      3    1  0  | DCL 1 DVIG BASED(PD),
                 |        2 NAMED CHAR(20),      /*НАЗВАНИЕ ДВИГАТЕЛЯ*/
                 |        2 TIP CHAR(20),        /*ТИП ДВИГАТЕЛЯ     */
                 |        2 SHEMA CHAR(20),      /*СХЕМА ДВИГАТЕЛЯ   */
                 |        2 FIRMA CHAR(20),      /*ФИРМА-ИЗГОТОВИТЕЛЬ*/
                 |        2 VES DEC FLOAT,       /*ВЕС ДВИГАТЕЛЯ     */
                 |        2 LDV DEC FLOAT,       /*ДЛИНА ДВИГАТЕЛЯ   */
                 |        2 DDV DEC FLOAT,       /*ДИАМЕТР ДВИГАТЕЛЯ */
                 |        2 PS  DEC FLOAT,       /*СТАРТОВАЯ ТЯГА    */
                 |        2 PF  DEC FLOAT,       /*ТЯГА С ФОРСАЖОМ   */
                 |        2 P   DEC FLOAT,       /*ТЯГА БЕЗ ФОРСАЖА  */
                 |        2 NVZ DEC FLOAT,       /*ВЗЛЕТНАЯ МОЩНОСТЬ */
                 |        2 CS BIN FIXED,
                 |        2 SUBF (NS REFER(DVIG.CS)) CHAR(80), /*СУБФИРМЫ*/
                 |        2 C BIN FIXED,
                 |        2 ANNOT (NT REFER(DVIG.C)) CHAR(80),
                 |        2 SM OFFSET(OBL);
                 |
      4    1  0  | DCL (NT,NS) BIN FIXED,(PZ1,PO,PD) PTR,NULL BUILTIN;
      5    1  0  | DCL ML FILE SEQUENTIAL UNBUF;
                 |
      6    1  0  |           ON ENDFILE(SYSIN) GOTO FIN;
      7    1  0  |           ALLOCATE STR SET(PO); PZ1=NULL;
                 |
      9    1  0  | MX1:      GET LIST(NT,NS);
     10    1  0  |           ALLOCATE DVIG IN(OBL) SET(PD);
     11    1  0  |           GET LIST(DVIG);
     12    1  0  |           IF PZ1=NULL THEN DO; SMSTART=PD;
     14    1  1  |                            GOTO MX2; END;
                 |
     16    1  0  |           PZ1->DVIG.SM=PD;
     17    1  0  | MX2:      PD->DVIG.SM=NULL;
     18    1  0  |           PZ1=PD; GOTO MX1;
                 |
     20    1  0  | FIN:      OPEN FILE(ML) OUTPUT;
     21    1  0  |           BUFER=STROKA;
     22    1  0  |           WRITE FILE(ML) FROM(BUFER);
     23    1  0  |           CLOSE FILE(ML);
     24    1  0  |           END PAZM;
```

Рис. 5.6

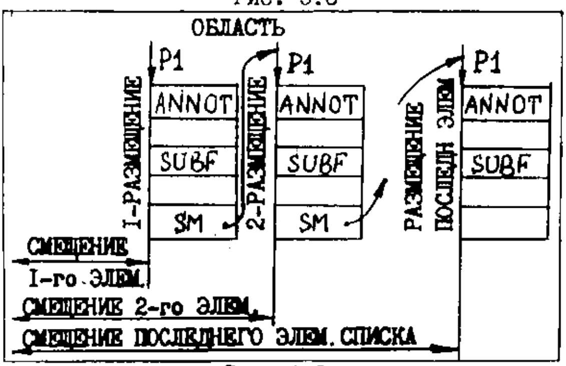


Рис. 5.7

– базированная структура *STR* с собственным указателем *PO* , в состав которой входит область *OBL* объемом I5996 байт и смещение *SMSTART* , фиксирующее смещение первого элемента списка в области *OBL* .

Кроме структуры *STR* в операторе 2 описаны две вспомогательные символьные переменные: *BUFER* длиной I40I6 байт и базированная *STROKA* длиной I40I6 байт, собственный указатель которой совпадает с собственным указателем структуры *STR*;

– базированная самоопределяющаяся структура *DVIG* с собственным указателем *PD* , в состав которой входят два символьных массива *ANNOT* и *SUBF* с переменными границами первых измерений, название двигателя, I0 параметров, характеризующих двигатель, две вспомогательные переменные *C* и *CS* , являющиеся элементами памяти для массивов с переменными границами, и смещение следующего элемента относительно начала области *OBL* ;

– вспомогательный указатель *PZ1* , который используется при организации списка, указатели *PO* , *PD* и встроенная функция *NULL* ;

– файл *ML* с последовательной организацией, в который записывается область с размещенным в ней списком.

Структура *STR* , содержащая область *OBL* , размещается в оперативной памяти с помощью оператора *ALLOCATE* 7. В операторах 9–I8 в области *OBL* размещается список. Перед размещением элемента в области с перфокарт считываются протяженности двух массивов с переменными границами (*NT* , *NS*) 9. Далее элемент списка физически размещается в оперативной памяти I0. В операторе II в элемент списка вводится исходная информация (числовая и текстовая).

При организации однонаправленного списка необходимо запомнить смещение первого элемента списка относительно начала области (*SMSTART*). До входа в цикл 9 – I8 вспомогательному указателю *PZ1* присваивается значение встроенной функции *NULL* . В операторе I2 фиксируется первый элемент списка, и смещению *SMSTART* присваивается значение собственного указателя *PD* первого элемента. После этого управление передается на метку *MX2* :

IF PZ1=NULL ␣ THEN ␣ DO; SMSTART=PD; GOTO ␣ MX2 ; END ;

В операторе I7 смещению первого элемента присваивается значение *NULL* , так как при первом размещении первый элемент списка является одновременно и последним. Вспомогательному указателю *PZ1* присваивается значение собственного указателя первого элемента *PD* , и управление передается на метку M.I.

При размещении второго и всех последующих элементов, вспомогательный указатель *PZ1≠NULL*. Поэтому после размещения и заполнения очередного элемента списка управление передается на оператор I6. В этом операторе смещению предпоследнего элемента присваивается значение абсолютного указателя *PD* последнего (вновь размещенного) элемента. Тем самым устанавливается связь между предпоследним и последним элементами списка. В операторе I7 смещению последнего элемента присваивается значение функции *NULL*. Далее указателю *PZ1* присваивается значение собственного указателя *PD* последнего элемента (указатель *PZ1* сдвигается на один элемент), и управление передается на метку MXI. Процесс размещения элементов списка в области будет продолжаться до тех пор, пока не закончатся исходные данные. После этого сработает исключительное состояние 6, и управление будет передано на метку *FIN*.

В операторах 20 - 22 структура *STR* вместе с областью *OBL*, в которой размещен однонаправленный список, записывается в последовательный файл *ML*.

5.4. МОДИФИКАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Рассмотрим два типа модификации: удаление элемента из списка и включение элемента в список.

Модификации структуры списка осуществляются процедурой MOD3, текст которой приведен на рис. 5.8. Рассмотрим пооператорно эту процедуру. В процедуре описываются (см. разд. 5.3):

- базированная структура *STR*;
- базированная структура *DVIG*;
- файл *ML*;
- указатели *PZ1*, *PO*, *PD*, встроенная функция *NULL*.

Кроме перечисленных переменных вводятся три дополнительных указателя *TECUS*, *PREV*, *TEST* и символьная переменная *FUN*.

Структура *STR*, содержащая область *OBL*, размещается в оперативной памяти 6. Содержимое записи файла *ML* считывается в структуру *STR* 8. Указателям *TEST*, *PREV*, *TECUS* присваиваются значения смещения первого элемента относительно начала области I0. При этом к значению смещения *SMSTART* автоматически прибавляется значение собственного указателя PO структуры *STR*. Далее анализируется значение символьной переменной *FUN*, и управление передается на соответствующую метку. Если из списка удаляется элемент, то *FUN='CAN'*; если в список добавляется элемент, то *FUN='ADD'*;

```
OPTIMIZING COMPILER          MOD3: PROC(FUN,PS0);

                SOURCE LISTING

STMT LEV NT

   1     0    | MOD3: PROC(FUN,PS0);
   2   1 0    | DCL 1 STR BASED(P0),
              |       2 OBL AREA(13996),
              |       2 SMSTART OFFSET(OBL),
              |         BUFER CHAR(14016),STROKA CHAR(14016) BASED(P0);
   3   1 0    | DCL 1 DVIG BASED(PD),
              |       2 NAMED CHAR(20),    /*НАЗВАНИЕ АВИГАТЕЛЯ*/
              |       2 TIP CHAR(20),     /*ТИП АВИГАТЕЛЯ    */
              |       2 SHEMA CHAR(20),   /*СХЕМА АВИГАТЕЛЯ  */
              |       2 FIRMA CHAR(20),   /*ФИРМА-ИЗГОТОВИТЕЛЬ*/
              |       2 VES DEC FLOAT,    /*ВЕС АВИГАТЕЛЯ    */
              |       2 LDV DEC FLOAT,    /*ДЛИНА АВИГАТЕЛЯ  */
              |       2 DDV DEC FLOAT,    /*ДИАМЕТР АВИГАТЕЛЯ*/
              |       2 PS  DEC FLOAT,    /*СТАРТОВАЯ ТЯГА   */
              |       2 PF  DEC FLOAT,    /*ТЯГА С ФОРСАЖОМ  */
              |       2 P   DEC FLOAT,    /*ТЯГА БЕЗ ФОРСАЖА */
              |       2 NVZ DEC FLOAT,    /*ВЗЛЕТНАЯ МОЩНОСТЬ*/
              |       2 CS BIN FIXED,
              |       2 SUBF (NS REFER(DVIG.CS)) CHAR(20), /*СУБФИРМЫ*/
              |       2 C BIN FIXED,
              |       2 ANNOT (NT REFER(DVIG.C)) CHAR(80),
              |       2 SM OFFSET(OBL);
   4   1 0    | DCL (NT,NS) BIN FIXED,(PF1,PD,PS,TECUS,TEST,PREV) PTR,
              |     FUN CHAR(3),NULL BUILTIN;
   5   1 0    | DCL ML FILE SEQUENTIAL UNBUF;
   6   1 0    |         ALLOCATE STR SET(P0);
   7   1 0    |         OPEN FILE(ML) INPUT;
   8   1 0    |         READ FILE(ML) INTO(STR);
   9   1 0    |         CLOSE FILE(ML);
  10   1 0    |         PREV,TEST,TECUS=SMSTART;
  11   1 0    |         IF FUN='CAN' THEN GOTO C1;
  12   1 0    |         IF FUN='ADD' THEN GOTO A1;
  13   1 0    |         IF FUN='RED' THEN GOTO R1;
              |                   /*ИСКЛЮЧЕНИЕ ЭЛЕМЕНТА*/
  14   1 0    | C1:     IF TEST->DVIG.PS=PS0 THEN DO;
  15   1 1    |         PREV->DVIG.SM=TEST->DVIG.SM;
  16   1 1    |         FREE TEST->DVIG;
  17   1 1    |         PUT SKIP(2) LIST('ЭЛЕМЕНТ ИСКЛЮЧЕН');
  18   1 1    |         GOTO FIN1; END;
  20   1 0    |         PREV=TEST; TEST=TEST->DVIG.SM;
  22   1 0    |         IF PREV->DVIG.SM^=NULL THEN GOTO C1; ELSE GOTO FIN1;
              |                   /*ВКЛЮЧЕНИЕ ЭЛЕМЕНТА*/
  24   1 0    | A1:     IF TEST->DVIG.PS > PS0 THEN DO;
  25   1 1    |         GET LIST(NT,NS);
  26   1 1    |         ALLOCATE DVIG IN(OBL) SET(PD);
  27   1 1    |         GET LIST(DVIG);     TECUS=PD;
  29   1 1    |         PREV->DVIG.SM=TECUS;
  30   1 1    |         TECUS->DVIG.SM=TEST;
  31   1 1    |         PUT SKIP(2) LIST('ЭЛЕМЕНТ ВКЛЮЧЕН');
  32   1 1    |         GOTO FIN1; END;
  34   1 0    |         PREV=TEST; TEST=TEST->DVIG.SM;
  36   1 0    |         IF PREV->DVIG.SM^=NULL THEN GOTO A1; ELSE GOTO FIN1;
              |                   /*ПОИСК ЭЛЕМЕНТА*/
  38   1 0    | R1:     IF TEST->DVIG.PS=PS0 THEN
  39   1 0    |         PUT SKIP(2) LIST(TEST->DVIG);
  41   1 0    |         PREV=TEST; TEST=TEST->DVIG.SM;
              |         IF PREV->DVIG.SM^=NULL THEN GOTO R1; ELSE GOTO FIN1;
  43   1 0    | FIN1:
              |         OPEN FILE(ML) OUTPUT;
  44   1 0    |         BUFER=STROKA;
  45   1 0    |         WRITE FILE(ML) FROM(BUFER);
  46   1 0    |         CLOSE FILE(ML);
  47   1 0    |         END MOD3;
OPTIMIZING COMPILER          MOD3: PROC(FUN,PS0);
```

Рис. 5.8

если требуется найти элементы списка, удовлетворяющие определенным требованиям, то *FUN='RED'*. Рассмотрим три фрагмента процедуры, выполняющие перечисленные функции.

ИСКЛЮЧЕНИЕ ЭЛЕМЕНТА ИЗ СПИСКА производится во фрагменте I4 - 22. С помощью указателя *TEST* фиксируется очередной элемент списка. Пусть для определенности необходимо исключить элемент, у которого значение стартовой тяги *PS* совпадает с заданным значением *PSØ*. В операторе I4 производится сравнение *PS* текущего элемента с *PSØ*. Если значения *PS* и *PSØ* совпали, то найденный элемент удаляется из области и связи реорганизуются так, чтобы список можно было просматривать от начала до конца.

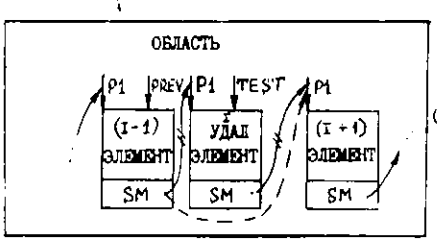


Рис. 5.9

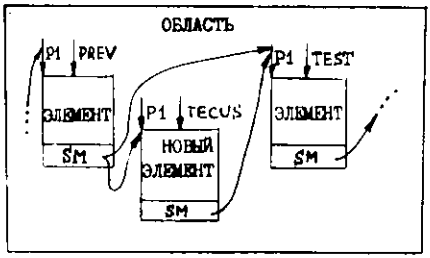


Рис. 5.I0

На рис. 5.9 представлена графическая иллюстрация реорганизации связей при удалении элемента из списка. Пусть удаляемый элемент имеет номер I. Реорганизация связей осуществляется в операторе I5:

$$PREV \text{ —>} DVIG.SM = TEST \text{ —>} DVIG.SM ;$$

В смещение (I-1)-го элемента заносится смещение (I+1)-го элемента, и тем самым связываются эти два элемента. I-й элемент физически удаляется из области. После выдачи соответствующего сообщения на АДПУ управление передается на метку *FIN1*. Если $PS \neq PS\emptyset$, то указатели *PREV* и *TEST* сдвигаются на один элемент:

$$PREV = TEST \; ; \quad TEST = TEST \text{ —>} DVIG.SM ;$$

Просмотр списка завершится тогда, когда указатель *PREV* достигнет последнего элемента, у которого смещение *SM = NULL*.

ВКЛЮЧЕНИЕ ЭЛЕМЕНТА В СПИСОК производится во фрагменте 24 - 36. Пусть для определенности требуется разместить новый элемент после первого элемента, для которого выполняется условие $PS > PS\emptyset$. С помощью указателя *TEST* фиксируется очередной элемент списка и производится сравнение значения *PS* этого элемента с заданным значением $PS\emptyset$. Если условие $PS > PS\emptyset$ выполнено, то вводится значение

протяженностей переменных массивов, новый элемент физически размещается в оперативной памяти и в него вводится информация с перфокарт (операторы 25 - 27). После размещения нового элемента необходимо реорганизовать связи так, чтобы новый элемент был доступен при просмотре списка. На рис. 5.10 представлена графическая иллюстрация реорганизации связей при добавлении элемента.

Пусть для определенности новый элемент требуется вставить между (I-1)-м и I-м элементами. Реорганизация связей осуществляется в операторах 29, 30:

29. *PREV* ——> *DVIG.SM = TECUS* ;

30. *TECUS* ——> *DVIG.SM = TEST* ;

Смещению элемента, на который указывает указатель *PREV*, присваивается значение абсолютного указателя нового элемента *TECUS* 29, тем самым организуется связь между (I-1)-м элементом и новым. Смещению нового элемента, на который указывает указатель *TECUS*, присваивается значение абсолютного указателя I-го элемента *TEST*. Тем самым организуется связь между новым элементом и I-м. На АЦПУ выдается соответствующее сообщение, и управление передается на метку *FIN1*. Если условие *PS > PSØ* не выполняется, то указатели *PREV* и *TEST* сдвигаются на один элемент. Просмотр списка завершится, когда указатель *PREV* достигнет последнего элемента.

ПОИСК ЭЛЕМЕНТОВ В СПИСКЕ производится во фрагменте 36 - 41. Это дополнительная функция процедуры *MOD3* . Требуется найти все элементы списка, у которых *PS = PSØ* . С помощью указателя *TEST* фиксируется очередной элемент списка и производится сравнение значения *PS* с *PSØ* . Если *PS = PSØ* , то на АЦПУ печатается содержимое всего элемента списка. Если *PS ≠ PSØ*, то осуществляется переход к следующему элементу. При этом осуществляется просмотр всего списка.

После завершения операции обработки списка (исключение, добавление, поиск) управление передается на метку *FIN1* . Содержимое базированной структуры *STR* с помощью символьных переменных *BUFER* и *STROKA* записывается в последовательный файл *ML* .

ЛИТЕРАТУРА

I. А к а е в А.А., М а й о р о в С.А. Когерентные статические вычислительные машины. - Ленинград: Машиностроение, 197[illegible].

2. А л ь я н а х И.Н. Внешние запоминающие устройства ЕС ЭВМ. - М.: Советское радио, 1979.

3. А р л а з а р о в В.Л., Е м е л ь я н о в Н.Б. и др. Информационная система ИНЭС. - М.: Автоматика и телемеханика, 1979, № 6, с. 109-121.

4. А у г у с т о н М.И., Б а л о д и с Р.П. и др. Программирование на ПЛ/I ОС ЕС. - М.: Статистика, 1979.

5. Б а л о д и с Р.П. Интерактивные средства программирования на базе языка ПЛ/I.-Управляющие системы и механизмы, 1977, № 3, с. 32-37.

6. Б е р е с т о в с к а я С.Н., К р и л ю к Н.И., П а ш к о в е ц Н.Д. ПОСПЛ - пакет программ, обрабатывающий списковые структуры. - Кибернетика, 1980, № 3, с. 42-46.

7. Г р и г а с Г.К. Кластероподобный стиль программирования на языке ПЛ/I. Алгоритмы и организация решения экономических задач. Вып. 14. - М.: Статистика, 1980, с. 96-102.

8. ГОСТ 23501.0...ГОСТ 23501.3-79. Системы автоматизированного проектирования. Основные положения.

9. Г р е б е н н и к о в Л.К., Л е б е д е в В.Н. Решение задач на ПЛ/I в ОС ЕС. - М.: Финансы и статистика, 1981.

10. Д е й т К. Введение в системы баз данных. - М.: Наука, 1980, с. 464.

11. Д а й и т б е к о в Д.М., Ч е р н о у с о в Б.А. Основы алгоритмизации и алгоритмические языки. - М.: Статистика, 1979.

12. Д ж е р м е й н К. Программирование на IBM / 360. - М.: Мир, 1973.

13. Д о н о в а н Дж. Системное программирование. - М.: Мир, 1975.

I4. ЕС ЭВМ. ОС.ПЛ/I. Оптимизирующий транслятор. Описание языка. ЕII.804.008 Д2, I979.

I5. ЕС ЭВМ. ОС.ПЛ/I. Оптимизирующий транслятор. Описание языка. ЕII.804.008 Д3, I980.

I6. ЕС ЭВМ. СУБД "ОКА". Вопросы конструирования баз данных. Кн. I, 2, 3. - М.: Госплан СССР, ГВЦ, I976.

I7. Калехсаев А.А. Системы интеграции и обработки данных СИОД I, СИОД2. - М.: Статистика, I977.

I8. Коссо В.П., Кузнецов И.Е. Сумарокова Т.Н. ПАРМА - сетевая СУБД на основе рекомендации КОДАСИЛ. Прикладная информатика. Вып. I / Под редакцией В.М. Савинкова. - М.: Финансы и статистика, I98I, с. I04-II7.

I9. Мартин Дж. Организация баз данных в вычислительных системах. Изд. 2-е. - М.: Мир, I980.

20. Майерс Г. Надежность программного обеспечения. Пер. с англ. под ред. В.Ш. Кауфмана. - М.: Мир, I980.

2I. Оллле Т.В. Предложения КОДАСИЛ по управлению базами данных. - М.: Финансы и статистика, I98I.

22. Пурвин Ю.В., Михайлов Ж.А. и др. Система управления базами данных СЕДАН. - М.: Финансы и статистика, I98I.

23. Поспелов Г.С. Системный анализ и искусственный интеллект. - М.: ВЦ АН СССР, I980.

24. Радиоэлектроника в I979 году. Обзор по материалам иностранной печати. I. Вычислительная техника. Математическое обеспечение ЭВМ. - М.: НИИ экономики информации по радиоэлектронике, I980.

25. Лебедь М.Я. Обеспечение виртуальной памяти в операционной системе ОС БС. - Вопросы радиоэлектроники. Сер. электронная вычислительная техника. Вып. 5, I979; вып. I, I980.

26. Стеблн Д. Логическое программирование в системе IBM/360. - М.: Мир, I978.

27. Соболь И.М. Численные методы Монте-Карло. - М.: Наука, I979.

28. Савинков В.М. и др. Использование систем управления базами данных в АСУ. Алгоритмы и организация решения экономических задач. Вып. I3. - М.: Статистика, I979, с. 29-38.

29. Фридлендер Р.Л., Гордонова В.И. Подсхема для языка ПЛ/I в системе управления базами данных сетевой структуры. В кн.: Алгоритмы и организация решения экономических задач. Вып. 3. - М.: Статистика, I979, с. I22-I50.

30. Ф о р м а л е в В.Ф. Основы построения и математическое обеспечение систем автоматизированного проектирования летательных аппаратов. - М.: МАИ, 1981.

31. Ф о р м а л е в В.Ф. Математические модели проектирующих подсистем САПР ЛА и методы их анализа. - М.: МАИ, 1981.

32. Ц а л е н к о М.Ш. Реляционные модели баз данных (обзор). Алгоритмы и организация решения экономических задач. -М.: Статистика, 1973, вып. 9, с. 18-36.

33. Язык описания данных КОДАСИЛ. - М.: Статистика, 1981.

34. *Hughes J.K. PL/1 structured programming. -2ed. 1979.*

35. Кородев Л.И. Структуры ЭВМ и их математическое обеспечение.-М.: Наука, 1978.

ОГЛАВЛЕНИЕ