ТАРТУСКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

ТРУДЫ

ВЫЧИСЛИТЕЛЬНОГО ЦЕНТРА

41

ТАРТУ
1978

ТАРТУСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

БАЗА ДАННЫХ ДЛЯ ЕС–1022

ТРУДЫ
ВЫЧИСЛИТЕЛЬНОГО
ЦЕНТРА

ВЫПУСК 41

ТАРТУ 1978

Утверждено на заседании совета математического факультета 3 ноября 1978 года.

СИСТЕМА РАМА ДЛЯ УПРАВЛЕНИЯ БАЗОЙ ДАННЫХ

Ю. Каазик, М. Томбак

Основные направления работы ВЦ ТГУ связаны с созданием различных специализированных систем обработки больших агрегатов данных (системы управления практическими работами студентов, системы статистической обработки информации, АИС для ТГУ). Такие системы можно рационально проектировать и реализовать лишь на основе единой базы данных. Общая основа такого рода позволяет значительно упростить работу по созданию конкретных систем хотя-бы потому, что отпадает необходимость повторного проектирования таких подсистем, функции которых фактически совпадают в разных системах. К тому же, по горькому опыту известно, что даже незначительное расширение конкретной системы, не основывающейся на базе данных, требует коренных переделок, вместо каких-то доделок.

Стандартное программное обеспечение ЭВМ ЕС-1022 не содержит системы управления базой данных (СУБД), созданные же в других организациях СУБД (такие, например, как ОКА, КАМА, БАНК) по разным причинам не удовлетворяют нашим требованиям.

Поэтому группа сотрудников ВЦ ТГУ предприняла попытку создания своей СУБД, в которой учитывались бы возможности и специфика проблематики ВЦ ТГУ.

К осени 1978 г. совершено проектирование этой СУБД, которая получила название РАМА, и ведутся работы по ее реализации. Статьи настоящего выпуска и содержат предварительное описание основных частей системы РАМА. Следует, конечно, учитывать возможные поправки и уточнения в ходе реализации, хотя авторы надеются, что существенных изменений в системе теперь уже не произойдет.

При проектировании системы РАМА мы исходили из следующих общих соображений:

а) создаваемая СУБД должна соответствовать возможностям имеющейся в нашем распоряжении конфигурации ЭВМ ЕС-1022;

б) в ходе создания СУБД обобщить опыт проектирования и эксплуатации систем обработки данных VILLIS (см. Труды ВЦ ТГУ № № 30, 38, 39) и STAMP (см. Труды ВЦ ТГУ № 34), реализованных на ЭВМ Минск-32;

в) дать пользователю по-возможности богатые средства определения и обработки структур оперативной памяти.

Простейшую СУБД можно построить из любого универсального языка программирования, разделив этот язык на две части: язык описания данных DDL (=Data Description Language), содержащий описательную часть исходного языка, и язык манипу-

лирования данными DML (=Data Manipulation Language), содержащий остальную часть исходного языка. Система РАМА, как и большинство СУБД, является расширением приведенной схемы.

Для описания данных система РАМА содержит два языка: язык описания записи RDL (=Record Description Language) и язык описания файла FDL (=File Description Language). Понятие записи в системе РАМА включает соответствующее понятие языка PASCAL, списковые структуры языка LISP и еще некоторые дополнительные возможности. Сравнительно простое понятие файла введено с учетом возможностей и ограничений операционной системы ОС, на которой базируются все операции по обмену информацией между накопителями. Описания языков RDL и FDL приводятся соответственно во второй и третьей статье настоящего выпуска.

Язык DML разбит в системе РАМА на три части тем путем, что из него выделены средства ввода данных с перфокарт (перфоленты) и вывода данных на АЦПУ. Таким образом получились специальные языки DIL (=Data Input Language) и DOL (=Data Output Language). Так как последний из них проходит еще допроектирование, то в четвертой статье настоящего выпуска описывается лишь язык DIL. В связи с этим языком следует подчеркнуть, что при его проектировании учитывалась предвидимая нехватка терминальных устройств в ближайшие годы.

Язык DML, описание которого приводится в пятой статье настоящего выпуска, представляет собой лишь одну возможность программирования в системе РАМА. В принципе возможно рас-

ширить систему и другими (традиционными алгоритмическими и непроцедурными проблемно-ориентированными) языками. Такое расширение планируется после реализации основных частей системы.

Ввиду того, что весь настоящий выпуск состоит из описания системы РАМА, в статьях опущены ссылки на другие статьи этого же выпуска. Предполагается, что читатель просмотрит статьи в порядке их следования.

Авторы дают себе отчет в том, что система РАМА получилась несколько эклектической. Повидимому это обусловлено тем, что РАМА является для некоторых из авторов второй системой и поэтому вступает в силу закон Брукса (см. Ф. Брукс. Как проектируются и создаются программные комплексы. Изд. "Наука", М. 1979 г.).

ЯЗЫК ОПРЕДЕЛЕНИЯ ЗАПИСИ

А.Изотамм, Ю.Каазик, М.Томбак

В этой статье рассматриваются средства системы РАМА для представления структуры данных на уровне записи. В первой главе описывается общая структура как логической, так и физической записи и вводится соответствующая терминология. Языковые средства определения записи излагаются во второй главе. В третьей главе приводятся три примера, два из которых взаимствованы из проекта "СОКРАТ" и аналитического обзора систем управления базами данных. В последней главе предоставляется формальный синтаксис языка.

1. Структура данных

1.1. Логическая структура записи. Система РАМА позволяет обработку лишь таких данных, которые разбиты на записи – основные единицы обмена информацией между накопителями. При этом каждая запись должна иметь логическую структуру корневого дерева. Примерная схема логической записи (т. е. логической структуры записи) приведена на рис. 1.

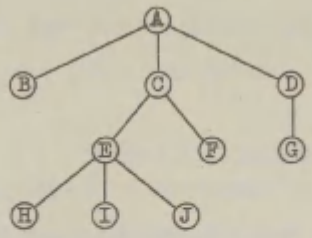


Рис. 1.

Описание логической записи в системе называется легендой. С корнем дерева легенды связывают имя легенды и определение доступа к записям такой структуры. Все остальные вершины дерева помечены двумя метками: номером уровня и именем вершины. С каждой вершиной можно также связывать определение свойств либо физической структуры, либо самих элементов данных.

По терминологии Д. Кнута [3] корни поддеревьев называются отцами (на рис. 1 вершины A, C, D и E), а их непосредственные подчиненные - сыновьями отца или же братьями между собой. Например, вершина E является отцом братьев H, I и J.

Кортеж братьев называется в рассматриваемой системе группой[1]. Элементами группы являются либо листья, либо корни поддеревьев, либо те и другие. Отец братьев будем называть корнем группы а имя корня служит именем группы. В дереве на

[1] Понятие группы совпадает с понятием блока (BLOC) системы "СОКРАТ" [2], но не совпадает с понятием группы из [1], где группой называется отец вместе со своими сыновьями.

рис. 1 выделяются следующие группы:

группа A: {B, C, D}

группа C: {E, F}

группа D: {G}

группа E: {H, I, J}

Каждая группа, кроме группы наивысшего (нулевого) уровня, может быть определена так, что ее элементы являются альтернативными. Например, если группа E объявлена альтернативной, то в каждую физическую запись включается только одна из вершин H, I, J или ничего.

Листья дерева легенды характеризуются их видами. Лист может, во-первых, быть атомом, т.е. действительным тупиком структуры. Тогда для листа определяются тип и другие свойства элементарного данного. Во-вторых, с листом можно связывать свойство повторения. В этом случае получаем т.н. множественный атом - группу значений одинакового типа. В-третьих, лист может быть корнем некоторой внешней структуры, которая не описывается в данной легенде, а определяется специальными средствами. Наконец, в-четвертых, лист может содержать ссылку на некоторую другую вершину легенды - в этом случае он по сути дела является либо атомом, либо корнем группы.

Носителями самих данных являются атомы, все другие вершины дерева легенды необходимы лишь для указания связей между этими данными. Основными типами значений атомов являются строка, число (вещественное, двоичное целое, неотрицательное двоичное целое, шестнадцатеричное или десятичное целое в упакованном виде) и дата (в полном или коротком виде). Кроме этого предвидена возможность пустого атома (в целях реализа-

ции пустой альтернативы) и атома-индекса для хранения порядковых номеров. Если сама структура дерева зависит от значения некоторого атома, то такому атому следует назначить специальный тип, запрещающий изменение этого значения после ввода данных.

Таким образом, легенда всегда состоит из корня и одной группы нулевого уровня, которая в тривиальном случае состоит только из атомов, но в общем содержит как атомы, так и корни других групп. Имена вершин должны быть уникальными лишь внутри своей группы, поэтому для ссылки на вершины приходится в общем случае пользоваться составными именами.

1.2. Структура физической записи. Физическая запись, как и легенда, имеет структуру дерева. В общем случае это дерево изображается в памяти ЭВМ в виде супермассива [7]. Если же некоторая часть записи определяется специальным образом, то соответствующие данные представляются как списочная структура [3]. Соответствие между логическими и физическими структурами устанавливается следующим образом.

Атому легенды соответствует в физической записи его значение (определенного в легенде типа) или же ссылка на это значение, если его длина превышает 8 байтов.

Корню группы в физической записи соответствует ссылка. Каждая группа (кроме группы нулевого уровня) может быть определена двояко: так, чтобы она выступила точно в одном экземпляре, или так, чтобы экземпляров группы было целое множество. В последнем случае эти экземпляры образуют по-

вторяющуюся группу[1], которая в частном случае может быть организована также и в виде массива.

Доступ к элементам массива производится по индексам. Если экземпляры группы не соединены в массив, то они сохраняются либо в порядке их поступления, упорядоченно или хэшированно. Доступ к экземпляру в первом случае организован последовательным образом, а в остальных случаях по ключам.

Если предусмотрен только один экземпляр группы, то ссылка, соответствующая корню группы, указывает непосредственно на экземпляр группы. В случае повторяющейся группы эта ссылка указывает на вектор ссылок, а каждый элемент этого вектора ссылает на один экземпляр группы. Если же повторяющейся группе прикреплена списочная структура, то ссылка указывает на первый экземпляр группы, а все экземпляры объединены при помощи ссылок в список.

Экземпляр группы изображается вектором, элементами которого являются либо атомы, либо ссылки, либо те и другие. Множественному атому соответствует ссылка на вектор значений этого атома.

Корню альтернативной группы соответствует ссылка либо на один элемент этой группы, либо на пустое слово (последнее возникает тогда, когда ни один из альтернативов не выбирается).

Если лист легенды содержит ссылку (ссылаемая вершина не является предком ссылающей вершины), то ему соответствует

---

[1] Этот термин совпадает с названием соответствующей структуры в [1]. В [2] такая структура называется сущностью (ENTITE).

такая же физическая ссылка, как и любому корню группы.

В целях иллюстрации рассмотрим простой пример. Допустим, что элементы дерева рис. 1 следующие: вершина C определяет повторяющуюся группу, D - альтернативную группу, E - неповторяющуюся группу, F - множественный атом, а остальные листья - атомы. Тогда этой легенде может соответствовать физическая запись, изображенная на рис. 2, где значения атомов обозначены строчными буквами а пустое слово - пустым ящиком.

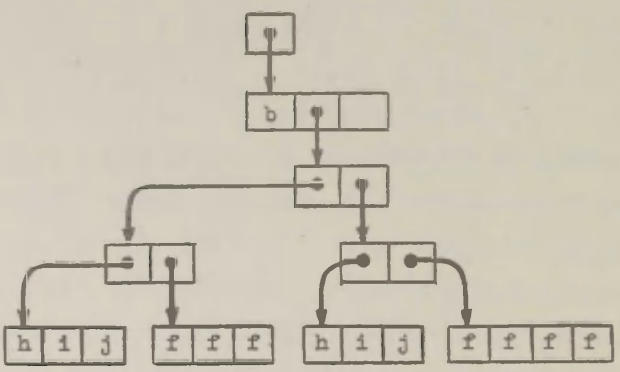


Рис. 2.

На этом рисунке указана ситуация, когда в альтернативной группе {G} реализован пустой вариант, группа {E,F} имеет два физического экземпляра - в первом из них атом F имеет три, а во втором четыре значения. Значения всех атомов имеют длину не больше восьми байтов.

1.3. Ветка и ветвь. При определении структуры данных необходимо учитывать некоторые ограничения, наложенные на движение по дереву физической записи. Из-за этих ограничений в

каждый момент обработки доступна не вся информация записи. Например, во время обработки одного экземпляра группы все другие ее экземпляры являются недоступными.

При обращении к конкретному экземпляру группы доступны не только сами элементы этого экземпляра, но и некоторые их потомки. Совокупность данных, составляющих фактическое содержание экземпляра группы, называется веткой этой группы. Ветка – это экземпляр группы вместе со своим корнем и всеми неповторяющимися потомками.

На рис. 3 приведены ветки двух групп для дерева записи, изображенного на рис. 1. Слева указаны ветки для случая, когда все группы уникальны, а справа для случая, рассмотренного при составлении рис. 2. В целях наглядности ссылки на этом рисунке указаны прописными буквами, а сами данные строчными.

Рис. 3.

Одновременно с любым экземпляром группы доступными (совместными) являются и некоторые вершины выше по дереву. Совокупность таких вершин называется ветвью данной группы. Ветвь - это корень группы вместе с ветками всех ее предков.

На рис. 4 приведены ветви двух групп для дерева с рис. 1 (слева в случае уникальных групп, а справа в случае групп, рассмотренных на рис. 2).

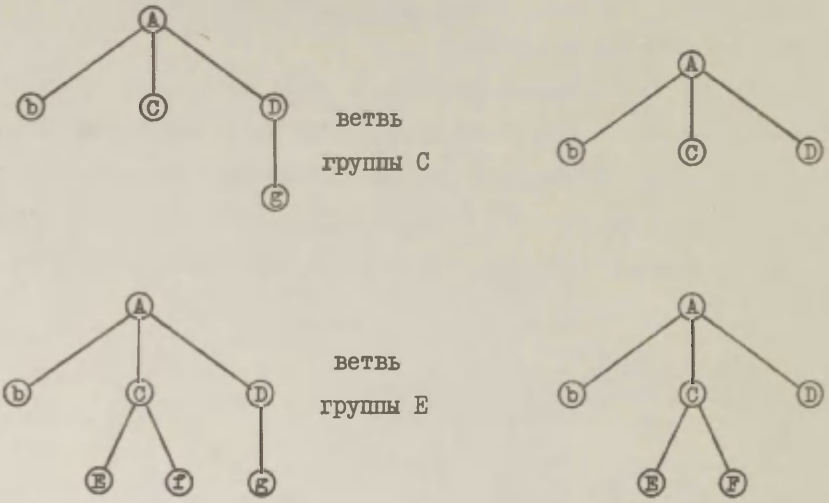


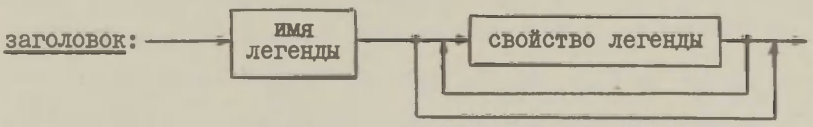
Рис. 4.

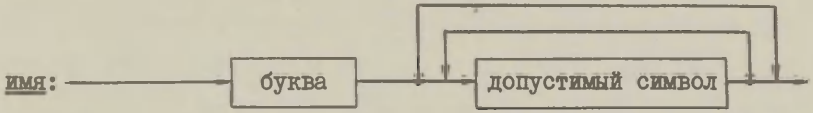
2. Определение легенды

2.1. Основные понятия. В этой главе способом Н. Вирта ([5,6]) описывается синтаксис языка RDL (=Record Description Language) для представления легенд и неформально, преимущ-

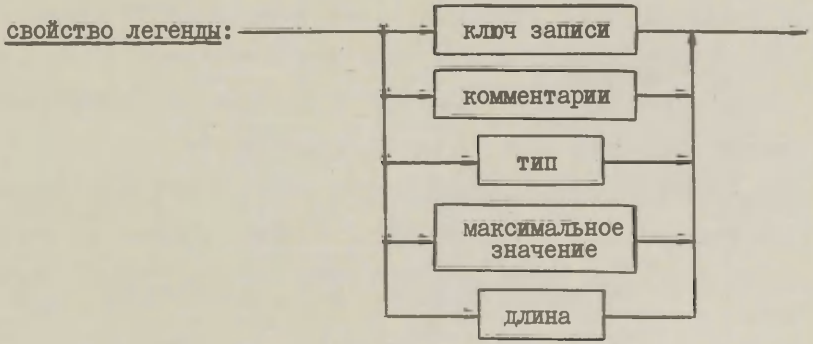
ественно на примерах, рассматривается семантика этого языка. Сама легенда определяется следующим образом:

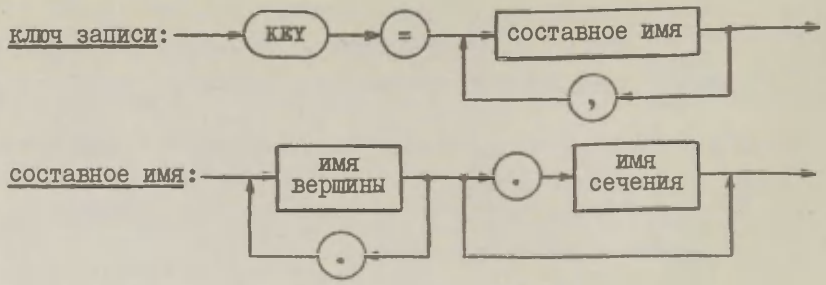
Заголовок задает свойства легенды, которые связываются с корнем дерева логической записи:





Допустимыми символами являются все буквы, цифры и знак "_". Восемь первых символов имени составляют его идентификатор. Имена с одинаковыми идентификаторами считаются совпадающими. Если количество символов меньше восьми, то имя дополняется справа пробелами.

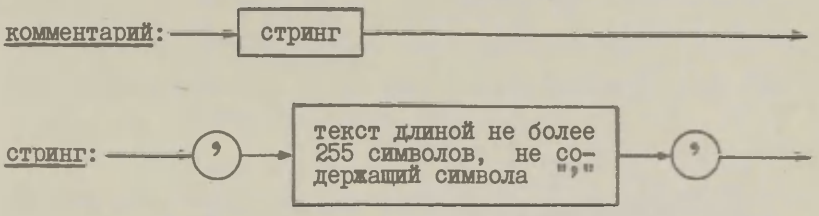




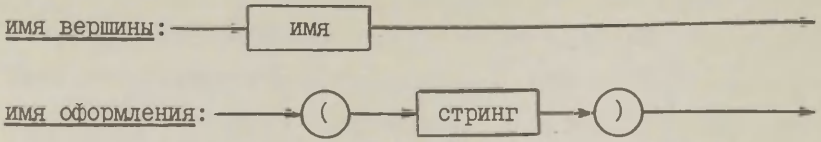
Использование составных имен обусловлено тем, что уникальность имени требуется лишь в пределах одной группы. Если в легенде имеется несколько вершин с совпадающими именами, то для указания на них следует прибавить достаточное количество имен корней групп. Однако то, что в легенде имеются синонимы, не всегда обусловливает применение составного имени. Например, если в корне группы представлен ключ, то атомы ключа должны находиться в ветке той группы, а если имена атомов ключа уникальны в пределах данной ветки, то можно ограничиваться только их именами. В некоторых случаях требуется, чтобы ссылаемая вершина находилась бы в одной ветви с ссылающей вершиной; тогда в случае уникальности имен внутри ветви можно пользоваться только именем.

В процедурах обмена информацией между внешними запоминающими устройствами и оперативной памятью ЭВМ запись является элементарной единицей обмена. Каждая физическая запись должна при этом иметь ключ, который ее идентифицирует. Если ключ задан явно, то его значением будет последовательность значений тех атомов, составные имена которых указываются после служебного слова KEY. Все они должны определяться в ветке

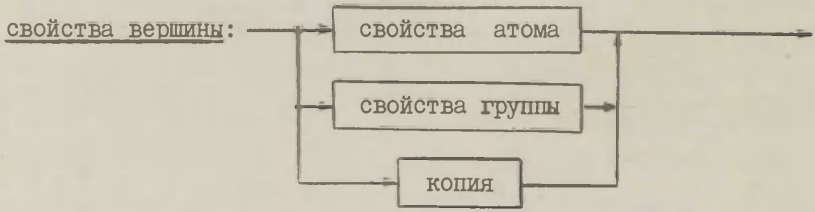
группы нулевого уровня. Если же ключ записи опущен, то возможна лишь последовательная обработка записей. Идентификатором каждой записи тогда считается ее воображаемый порядковый номер. Однако, имеется возможность сформировать ключ записи так, чтобы им был именно порядковый номер физической записи, который будет присвоен автоматически. Для этого следует ключевый атом снабжать свойством INDEX (более подробно это свойство будет рассматриваться позже).



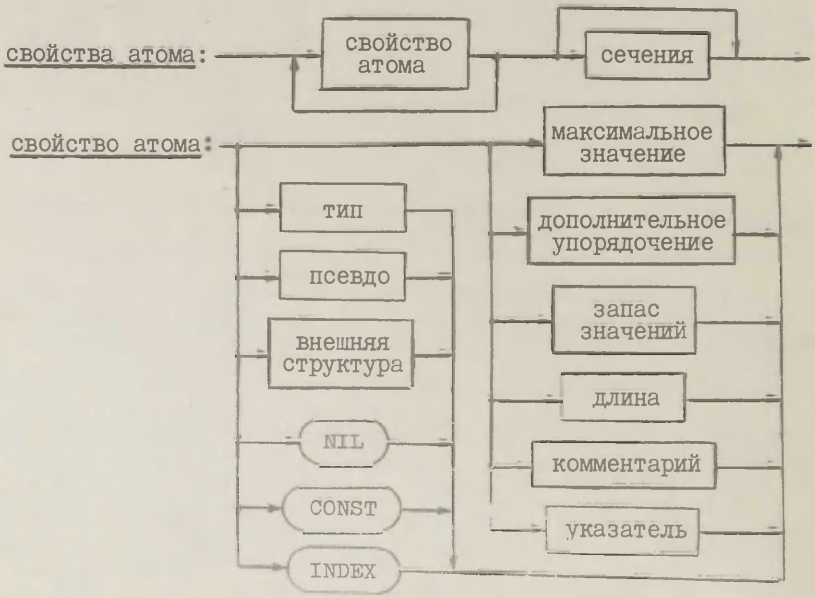
Свойства "тип", "максимальное значение" и "длина" являются свойствами атома и будут определены в следующем разделе (как и "сечение" из определения составного имени). Любое свойство из перечисленных свойств атома распространяется на все атомы легенды, если в их описаниях не сказано иначе.

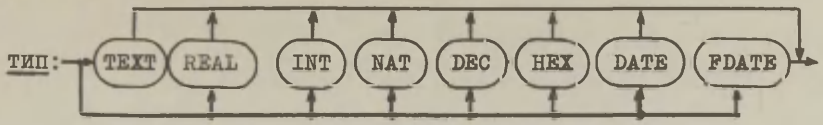


Последнее имя предусмотренно для более наглядного оформления печатываемых справок.

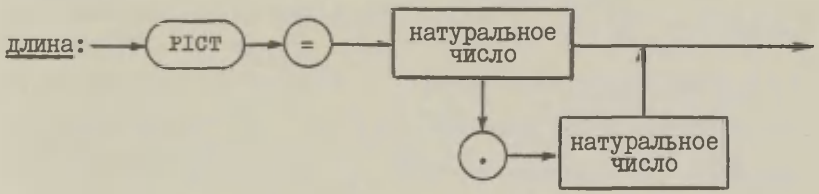


2.2. Определение атома. Если вершина является атомом, то ее свойства определяются следующим образом:





Строковой тип задается служебным словом **TEXT**, варианты же числового типа словами **REAL**, **INT**, **NAT**, **HEX** и **DEC**, которые обозначают соответственно вещественный, двоичный целый, неотрицательный двоичный целый, шестнадцатеричный и десятичный упакованный тип. Типы **DATE** и **FDATE** (=**Full DATE**) определяют, что значением атома будет короткая или полная календарная дата в виде упакованного десятичного числа. Например, число "20 марта 1978 года" имеет соответственно виды 200378 и 20031978.

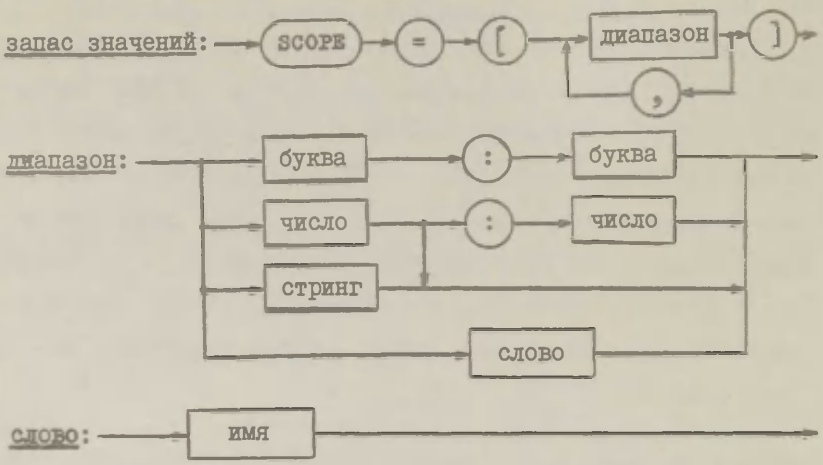


Длиной устанавливают максимально возможное количество символов печатного изображения значения атома. Два натурального числа допустимы при числовом типе: первым числом задается количество цифр перед точкой (запятой), а вторым - за ней. Таким образом устанавливается требуемый формат данных для их ввода и вывода на печать. Внутренним представлением чисел, кроме типа **REAL**, будет все же целое число.

Число представляется, в зависимости от типа атома, как целое или вещественное число со знаком или без знака; допус-

кается и полулогарифмический вид.

Запасом значений определяются допустимые значения атома:

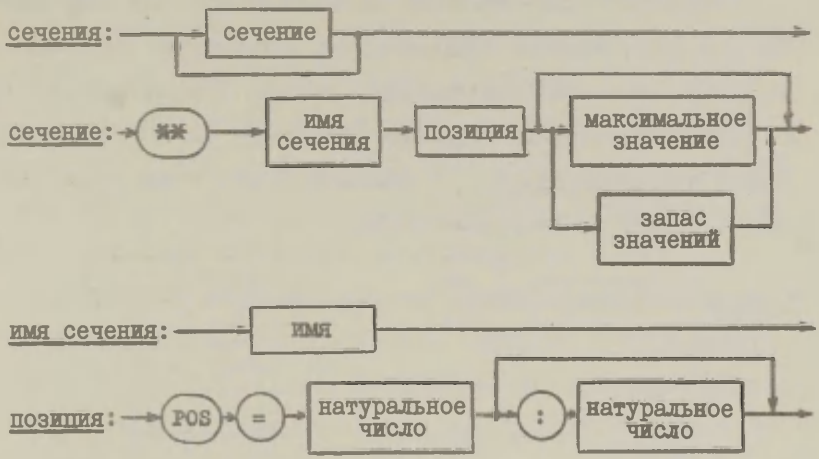


Если типом предусмотрен TEXT, то запас может в общем случае состоять только из текстовых констант (слов или стрингов). Диапазоны от константы до константы типа TEXT допустимы лишь тогда, когда константами являются одиночные буквы, например [A:F]. Следует подчеркнуть, что лексикографическое упорядочение элементов запаса не требуется.

Максимальное значение, запас значений и длина атома находятся в очевидном взаимном соответствии. Если, например, задан запас значений, то остальные свойства являются лишними. Все три свойства можно также опускать: максимальным значением атома числового типа будет тогда максимальное число заданного типа, которое можно представлять на поле памяти длиной 4 байта, а максимальной длиной атома строкового типа будет 255 байтов. Определение длины атома типа календарной даты не имеет смысла. Наконец, если сам тип атома нигде не

задан, то по умолчанию им будет MAT.

Значение типа TEXT, HEX или DEC можно расчленить на список значений его частей. Части или сечения определяются, исходя из заданной длины атома; нумерация символов начинается слева:



Для типов DATE и FDATE система по умолчанию составляет три сечения с именами DAY, MONTH и YEAR (число, месяц и год). Таким образом, описания атомов

```
    * 1   BIRTHDAY DATE
```

и

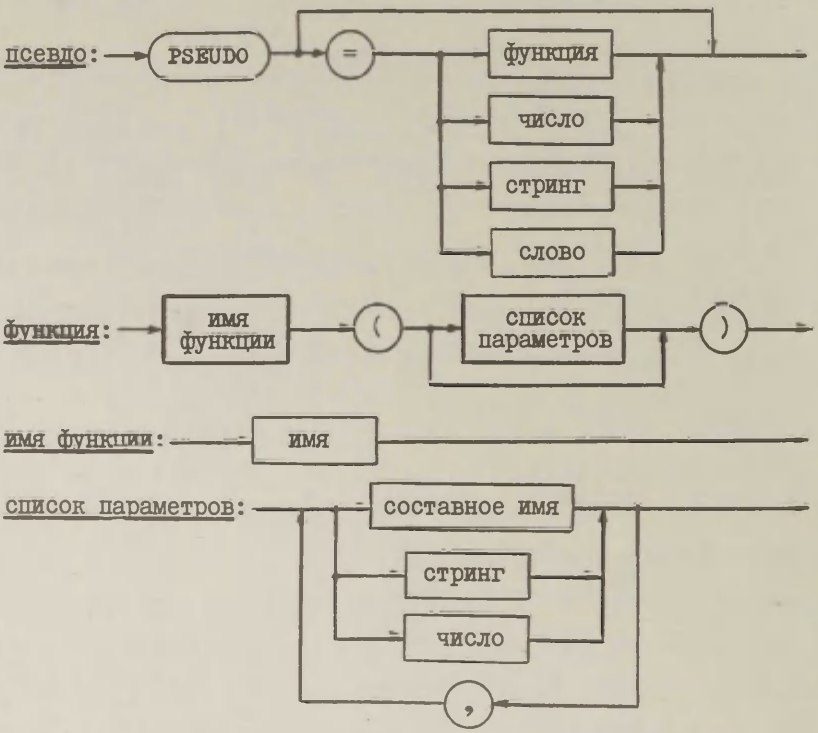
```
    * 1   TIME FDATE
```

соответственно равносильны описаниям

```
    * 1   BIRTHDAY DEC PICT=6
    * *   DAY POS=1:2 MAX=31
    * *   MONTH POS=3:4 MAX=12
    * *   YEAR POS=5:6
```

```
* 1   TIME DEC PICT=8
* *   DAY POS=1:2 MAX=31
* *   MONTH POS=3:4 MAX=12
* *   YEAR POS=5:8
```

Атомы могут получить свои значения двояко: они либо вводятся в ЭВМ с внешних носителей,либо вычисляются (присваиваются) при вводе или использовании записи. В последнем случае говорят об атоме с генерируемым значением, который в системе называется псевдоатомом. В описании данных такие атомы следует снабжать со свойством псевдо:

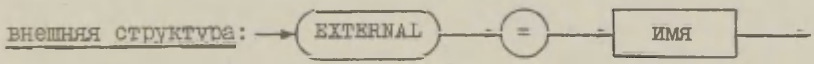


Если за словом PSEUDO имеется знак равенства и число, слово или стринг, то такая конструкция позволяет включить в запись константу, которую можно рассматривать как исходное значение атома: оно может быть изменено программами, которые используют запись, оставаясь недоступным для корректора.

Функция за знаком равенства должна указывать на библиотеку функций, доступных системе (составные имена в качестве параметров должны указать на атомы, находящиеся в одной ветви с псевдоатомом). Присвоение значения псевдоатому происходит при выводе записи во внешнюю память и это лишь в случае, когда обновлялись значения параметров.

Создаваемая библиотека функций предусмотрена открытой. В эту библиотеку входит, например, функция без параметров TODAY( ), значением которой будет текущая календарная дата. Это значение обновляется в той ветви, в которую вводят исправление любого вида.

Система допускает включение в запись атомов, структура которых описывается средствами, не входящими в язык RDL. Такая вершина (лишь условно считающаяся атомом) определяется свойством



где при помощи имени создается связь между легендой и библиотекой внешних структур.

Свойства атомов INDEX, CONST, NIL, "указатель" и "дополнительное упорядочение" будут рассматриваться позже.

2.3. Определение группы. При естественной нумерации группой считается последовательность вершин в легенде, имеющих одинаковые номера уровня и находящихся между вершинами с меньшими номерами уровня (заголовок и END условно считаются вершинами нулевого уровня). Так, например, в легенде

```
LEG A   *1 B  *1 C  *2 D  *2 E  *3 F  *3 G  *2 H
        *1 I  *2 J  *1 K  *3 L  *3 M  END
```

содержатся следующие группы:

{1 B, 1 C, 1 I, 1 K} , {2 D, 2 E, 2 H} ,

{3 F, 3 G} , {2 J} , {3 L, 3 M}

(отметим, что каждая вершина входит точно в одну группу).

Корнем группы считается вершина, непосредственно предшествующая данной группе (в этой связи и заголовок является вершиной). Корнями групп предыдущего примера являются соответственно: заголовок, 1 C, 2 E, 1 I и 1 K. Дерево этой легенды изображено на рис. 5.

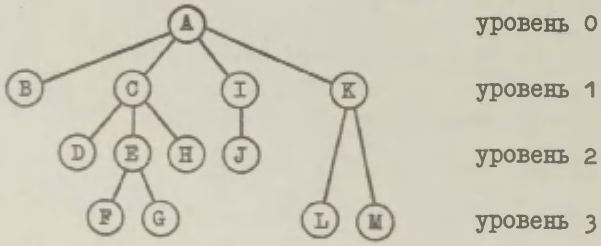


Рис. 5.

Из этого примера видно, что номер уровня группы может и больше чем на единицу превышать номер уровня корня. Допускается также неодинаковая нумерация вершин группы (хотя это не рекомендуется, так как ухудшает чтение легенды). Так, например, последнюю легенду можно писать и в виде

```
LEG A   *2 B  *1 C  *3 D  *3 E  *7 F  *7 G  *2 H
        *1 I  *4 J  *1 K  *2 L  *2 M  END
```

Вершинами одной группы могут быть как атомы, так и корни других групп. В самом простом случае вся легенда состоит из одной группы, как в следующем примере.

Пусть члены персонала какого-то учреждения определены по фамилии, имени, возрасту, семейному положению, адресу, дате приема на работу и месту работы. Соответствующая легенда может быть представлена в виде:

```
LEG  СОТРУДНИК KEY = ФАМИЛИЯ, ИМЯ 'ПЕРСОНАЛЬНЫЕ ДАННЫЕ'
     TEXT
     * 1   ФАМИЛИЯ  PICT = 25
     * 1   ИМЯ  PICT = 15
     * 1   ВОЗРАСТ  NAT  SCOPE = [18:70]
     * 1   СЕМ_ПОЛОЖ ('СЕМЕЙНОЕ ПОЛОЖЕНИЕ')
           SCOPE = [ХОЛОСТ, ЖЕНАТ, ВДОВ, РАЗВЕДЕН]
     * 1   АДРЕС 'ДО 255 СИМВОЛОВ'
     * 1   ПРИЕМ ('ПРИЕМ НА РАБОТУ')  DATE
```

```
* 1   МЕСТО ('МЕСТО РАБОТЫ') SCOPE = [ ГОРСК,
                                     ДОЛИНСК, СКЛОНОВО ]

END
```

В запись СОТРУДНИ входит единственная простая группа; все ее элементы являются по умолчанию атомами строкового типа. Значениями атомов ФАМИЛИЯ и ИМЯ будут строки символов, длины которых не превышают соответственно 25 и 15. Допустимые значения возраста (тип NAT) принадлежат в диапазон 18, 19,..., 70; максимальным значением этого атома считается 70, а длиной - 2. Атом СЕМ_ПОЛО имеет четыре возможных значения длиной в 8 символов (таково количество символов наиболее длинного значения - РАЗВЕДЕН). Поскольку длина атома АДРЕС не задана, то максимальной длиной по умолчанию будет 255. Дата приема на работу доступна в целом по имени ПРИЕМ, а ее сечения - по именам DAY, MONTH и YEAR. Длиной атома МЕСТО будет 8. Ключами экземпляра записи являются фамилия и имя сотрудника.

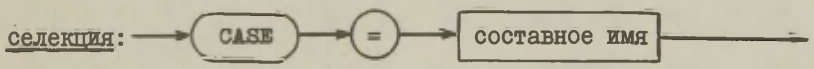
Группа может иметь ряд специфических свойств, которые определяются в вершине, являющейся корнем этой группы. Основными из этих свойств являются повторение экземпляров группы и альтернативность ее элементов. С корнем группы можно, однако, связывать также до трех свойств атома: тип, длину и максимальное значение. Эти свойства переносятся на все атомы соответствующего поддерева, для которых те же свойства не определены ниже по дереву.

Частным случаем повторяющейся группы является множественный атом, который определяется как лист легенды. Его опи-

сание может содержать все свойства атома и свойство повторения.

Альтернативность вершин группы предусмотрена в целях избежания лишних полей или поддеревьев в физической записи. Если группа при помощи свойства "селекция" объявляется альтернативной, то она в каждом экземпляре состоит лишь из одной вершины. Эта вершина выбирается из числа написанных в легенде по конкретному значению определенной вершины - ключа выбора.

Корень альтернативной группы определяется свойством



где составное имя идентифицирует ключ выбора. Значение ключа должно быть известным к моменту выбора, а это значит, что ключ выбора следует определить перед корнем альтернативной группы и он должен находиться в ветви этой группы.

Ключом выбора может быть лишь атом, для которого определен запас значений, или же атом типа NAT с указанным максимальным значением. Число вершин альтернативной группы должно равняться мощности запаса значений ключа, причем взаимно-однозначное соответствие между значениями ключа и вершинами группы устанавливается по порядку их написания в легенде (если запас значений задан диапазоном от константы до константы, то учитывается естественное упорядочение значений).

Когда запас значений ключа выбора окажется избыточным, то "лишним" значениям следует ставить в соответствие пустые вершины альтернативной группы: пустой вершиной является атом, имеющий лишь свойство NIL. Такой атом является псевдоатомом, присвоение значений которому заблокировано. Если ключ выбора имеет тип NAT, то его нулевому значению автоматически соответствует пустая вершина.

В качестве примера приведем легенду для данных о людях, где структура данных зависит от возраста человека:

```
LEG  ЧЕЛОВЕК  KEY = ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО  TEXT
   *1  ИМЕНА  PICT = 15
     *2  ИМЯ
     *2  ОТЧЕСТВО
     *2  ФАМИЛИЯ
   *1  ВОЗРАСТ  SCOPE = [МАЛЫШ, ШКОЛЬН, ВЗРОС]
   *1  АДРЕС  PICT = 40
   *1  СЕЛЕКЦИЯ  CASE = ВОЗРАСТ
```

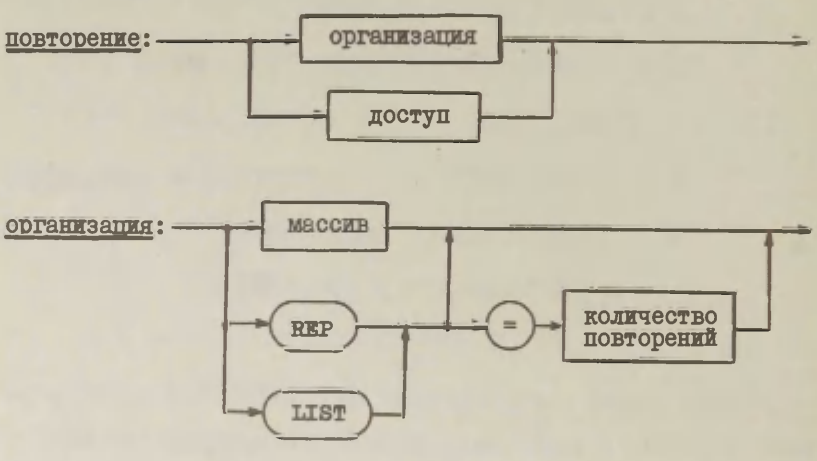
```
* 2 МАЛЫШ
  * 3 A  SCOPE = [ДОМА, ЯСЛИ, САД]
  * 3 B  CASE = A  MAT
    * 4 ДОМАШНИЙ NIL
    * 4 НОМЕР 'НОМЕР ЯСЛЕЙ'
    * 4 САД
      * 5 АДРЕС  TEXT  PICT = 40
      * 5 НОМЕР 'НОМЕР САДА'
      * 5 ГРУППА MAX = 5 'НОМЕР ГРУППЫ'
* 2 УЧЕНИК 'ДАННЫЕ О ШКОЛЕ'
  * 3 НОМЕР 'НОМЕР ШКОЛЫ'  MAX = 100  MAT
  * 3 АДРЕС  PICT = 40
  * 3 КЛАСС  PICT = 3
  * * НОМЕР  POS = 1:2
  * * П('ПАРАЛЛЕЛЬ')  SCOPE = [A:F]  POS = 3
* 2 БОЛЬШОЙ  MAT
  * 3 РАБОТА  MAX = 1  '1 = РАБОТАЕТ, 0 = ДОМАШНИЙ'
  * 3 B CASE = РАБОТА  TEXT
    * 4 ДМР 'ДОЛЖНОСТЬ И МЕСТО РАБОТЫ'
END
```

Корнем первой альтернативной группы является здесь вершина СЕЛЕКЦИЯ, а выбор происходит по значению ключа ВОЗРАСТ. Группа состоит из трех вершин (МАЛЫШ, УЧЕНИК и БОЛЬШОЙ), ко-

торые все являются корнями поддеревьев. Каждая физическая запись содержит ровно одно из этих поддеревьев, например, если значение атома ВОЗРАСТ будет ШКОЛЬН , то группа СЕЛЕКЦИЯ состоит лишь из группы, корнем которой является УЧЕНИК. Разветвление в вершине МАЛЬШ.В зависит от значения ключа МАЛЬШ.А: первому допустимому значению соответствует пустая вершина, второму атом и третьему группа САД. В группе с корнем БОЛЬШОЙ.В задана лишь одна вершина - атом ДМР, который выбирается, когда значением атома РАБОТА будет единица. В противном случае элемент БОЛЬШОЙ.В имеет пустое значение.

Свойство группы РАСК требует уплотненную запись экземпляра повторяющейся группы, если он соответствует либо простой группе, либо множественному атому. Схема упаковки составляется системой автоматически.

2.4. Повторяющаяся группа. Группа является повторяющейся, если в ее корне задано свойство "повторение":



Свойством "организация" задается, прежде всего, способ физического представления повторяющейся группы: LIST определяет списочную структуру, а REP одномерный супермассив. Массиву с n размерностями соответствует супермассив с n уровнями иерархии. Общие виды организации схематически указаны на рис. 6.

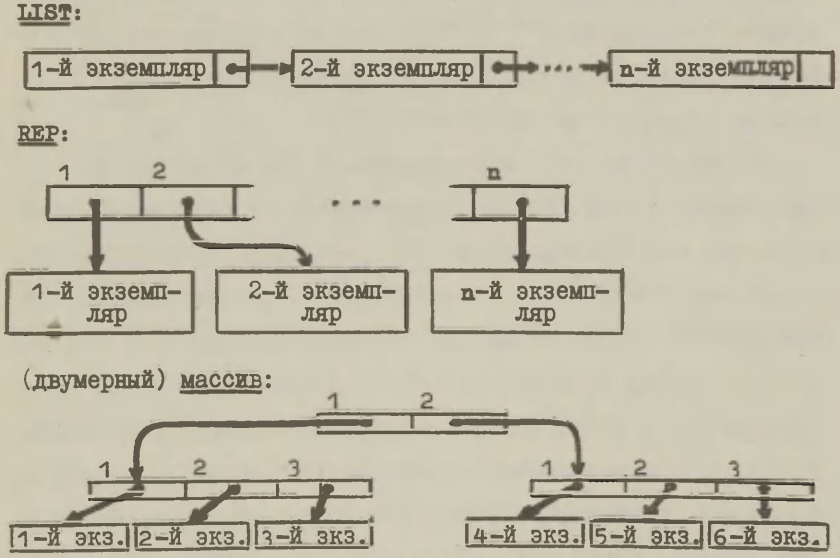


Рис. 6.

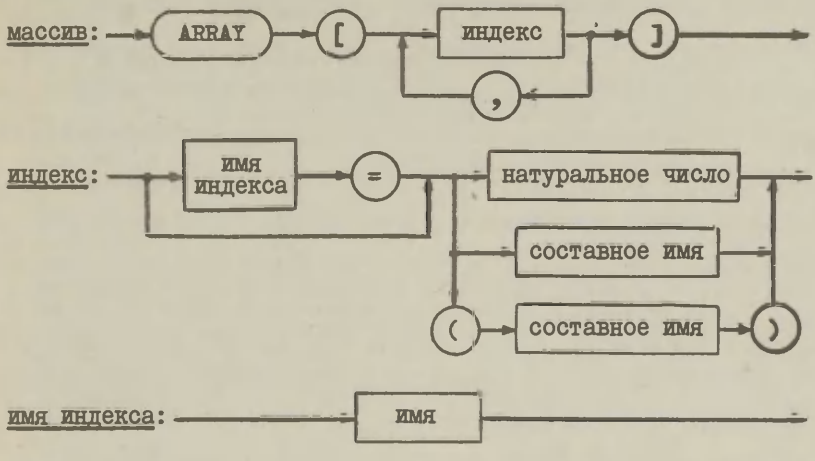
Количество экземпляров повторяющейся группы является для каждого массива всегда зафиксированным. Если вариантом организации является LIST или REP, то можно указать еще аттрибут количества повторений, указывающий на максимально возможное число экземпляров группы. Натуральным числом это количество указывается явно, а составным именем ссылаются на атом, значение которого принимается за количество повторений. В последнем случае ссылаемый атом должен находиться в ветви этой группы и иметь свойство CONST. Такое свойство дает атому тип MAT и блокирует изменение его значений во время обработки данных - менять значение можно лишь при помощи корректора или специальной процедуры.Если количество повторений не указано, то оно считается произвольным.

Замысел указания количества повторений заключается в следующем: если оно задано, то все таблицы доступа для вариантов "упорядочение" или "хэширование" сохраняются при вынесении записи во внешнюю память, а в противном случае эти таблицы образуются только в оперативной памяти.

Доступ к экземплярам повторяющейся группы зависит как от организации группы, так и от определения ее корня. Доступ к элементам массива зафиксирован и происходит по индексам, поэтому свойство "доступ" для массива не допускается. Если при REP или LIST свойство "доступ" опущен, то экземпляры повторяющейся группы сохраняются в порядке их поступления;в таком же порядке можно их обрабатывать. Если желают организовать доступ по ключу,то в общем случае следует задавать как ключ, так и вариант доступа. Ключ опускается в случае множественного атома, который сам считается по умолчанию ключем.

Ниже возможности организации повторяющейся группы рассматриваются более подробно.

Массив - это повторяющаяся группа с зафиксированным доступом:каждому экземпляру ставится в соответствие конкретный набор значений его индексов. Число индексов (размерность массива) и их области изменения определяются в корне группы конструкцией "массив":



Каждому индексу можно придать имя, чтобы упростить соответствующие ссылки в программах обработки (в физической записи этому имени ничего не соответствует).

Значения индексов, как правило, натуральные числа. Нижней границей области изменения индекса считается единица,верхняя же граница задается в виде натурального числа или ссылки на атом в ветви этой группы, имеющий свойство CONST. Если составное имя атома задано при описании массива в скобках, то индекс имеет тип этого атома и меняется по его зага-

су значений (который должен быть задан).

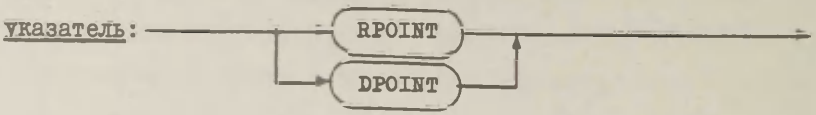
Например, в легенде

```
LEG  УЧЕНИК  KEY = ФАМ, ИМЯ  TEXT
     * 1   ИМЯ   PICT = 10
     * 1   ФАМ   PICT = 20
     * 1   ЧП ('ЧИСЛО ПРЕДМЕТОВ') CONST
     * 1   КЛАСС  PICT = 3
     * 1   У ('УСПЕВАЕМОСТЬ') ARRAY [ СЕМ = 4, ЧП ] NAT
       * 2   ОЦЕНКА  MAX = 5
       * 2   ПРОП  MAX = 100
END
```

У является двумерным массивом, где первый индекс (номер семестра) меняется от 1 до 4, а второй нидекс от 1 до заданного числа предметов. С каждым элементом массива связываются два атома: оценка и число пропущенных занятий.

Повторяющаяся группа без явного доступа определяется при помощи слов REP или LIST (с указанием количества повторений или без этого), и свойство "доступ" опускается. Фактический доступ к экземплярам такой повторяющейся группы зависит от ее организации.

Если организацией является LIST, то в состав группы следует включить один или два атома со свойством "указатель":



Указатель является атомом особого типа: его значение не вводится в ЭВМ и не вычисляется в программах обработки, а формируется только системой. Физически RPOINT ("указатель направо") содержит ссылку на следующее звено цепи, которое имеет точно такую же структуру, как группа, содержащая эту ссылку. Каждая повторяющаяся группа со списочной организацией должна содержать в своей ветке ровно один указатель направо.

Кроме такого зацепления экземпляров повторяющейся группы списочная организация представляет возможность для определения структуры с неопределенным количеством уровней иерархии: любому экземпляру может подчиняться список, каждый элемент которого имеет опять такое же описание, как сама группа. В определении группы место для ссылки на такой подсписок задается атомом с единственным свойством DPOINT ("указатель вниз"). Например, отрывок легенды

```
* 3  ЗВЕНО  LIST
  * 4  ИНФО  TEXT  PICT = 8
  * 4  ПОДЦЕПЬ  DPOINT
  * 4  СОСЕД  RPOINT
```

определяет цепь, каждое звено которой содержит три поля: восемь байтов для информации, ссылку на подчиненную цепь и ссылку на следующее звено. Схематически структура этой группы изображена на рис. 7, где используется еще термин подцепь - это набор экземпляров, связанных между собой ссылками направо.

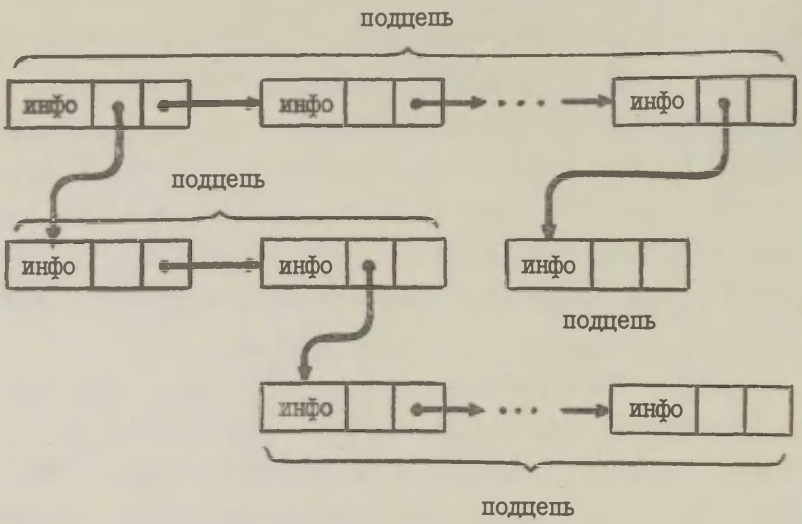


Рис. 7.

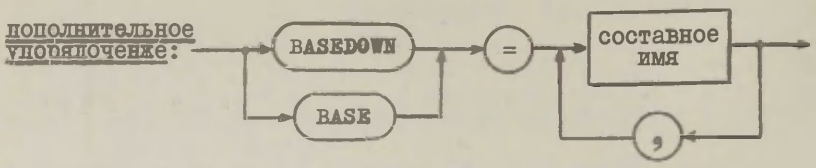
Доступ к экземплярам повторяющейся LIST-группы без явного ключа осуществляется движением по ссылкам, начиная от первого экземпляра.

Если организацией повторяющейся группы является REP, а доступ не указан, то фактический доступ к экземплярам такой группы осуществляется при помощи их порядковых номеров, присваиваемых системой по мере поступления экземпляров. В физическом экземпляре этому номеру не соответствует ни одного поля. Если существование такого поля все-таки желательно, то в группу следует включить атом со свойством INDEX. По этому свойству атом автоматически получает тип NAT и является псевдоатомом, присвоение значений которому доступно только системе. Рассмотрим, например, легенду

```
LEG  ШКОЛЫ  KEY = ШКОЛА  TEXT
  * 1  ШКОЛА  PICT = 50
  * 1  КОЛИЧЕСТВО CONST 'КОЛИЧЕСТВО КЛАССОВ'
  * 1  КЛАСС  REP = КОЛИЧЕСТ
    * 2  НОМЕР  PICT = 3
    * 2  КОЛИЧЕСТВО CONST MAX = 40 'КОЛИЧЕСТВО УЧЕНИКОВ'
    * 2  УЧЕНИК  REP = КЛАСС.КОЛИЧЕСТ  PICT = 20
      * 3  ИМЯ
      * 3  ФАМИЛИЯ
      * 3  ПОЛ  SCOPE = [ М, Д ]
      * 3  НОМЕР  INDEX
    * 2  ПРЕДМЕТ  REP  PICT = 8
END
```

Здесь каждая запись содержит данные о некоторой школе. Каждому классу соответствует по одному экземпляру группы КЛАСС, а экземпляр каждого класса содержит столько экземпляров группы УЧЕНИК, каково количество учеников в классе. Значение атома УЧЕНИК.НОМЕР указывает на порядковый номер данного ученика по порядку чтения данных в момент создания записи. Вершина ПРЕДМЕТ представляет собой множественный атом.

Если экземплярам группы необходимо присвоить порядковые номера не по их физическому расположению, а по другим признакам, то в атоме со свойством INDEX следует задать еще свойство "дополнительное упорядочение":



Список составных имен задает здесь набор атомов, по значениям которых происходит упорядочение (словами BASE или BASEDOWN устанавливается соответственно возрастающая или убывающая упорядоченность). Индекс получает свои значения по этой упорядоченности, но физически экземпляры не перемещаются. В случае совпадения значений указанных атомов нумерация происходит по физическому расположению.

Например, если группу УЧЕНИК предыдущего примера заменить следующей:

```
* 2  УЧЕНИК  REP = КЛАСС.КОЛИЧЕСТВО  PICT = 20
  * 3  ИМЯ
  * 3  ФАМИЛИЯ
  * 3  СР_ОЦЕНКА  REAL  PICT = 1.2
  * 3  ПРЫЖОК  REAL  MAX = 8.91 'В ДЛИНУ'
  * 3  A  INDEX  BASE = ФАМИЛИЯ, ИМЯ
  * 3  B  INDEX  BASEDOWN = СР_ОЦЕНК
  * 3  C  INDEX  BASEDOWN = ПРЫЖОК
  * 3  D  INDEX
```

то значения атома D те же, что и атома УЧЕНИК.НОМЕР в предыдущем примере, а остальные три индекса получают следующие

значения. Индекс A показывает на номер ученика по альфавитному порядку следования, индекс B – порядковый номер по средней оценке ученика, а индекс C – по результату прыжка в длину. Если в классе имеются тезки, то значение индекса A меньше у того из них, у кого меньше значение индекса B.

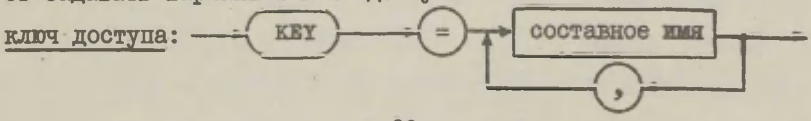
В следующем примере свойство INDEX имеют элементы массива:

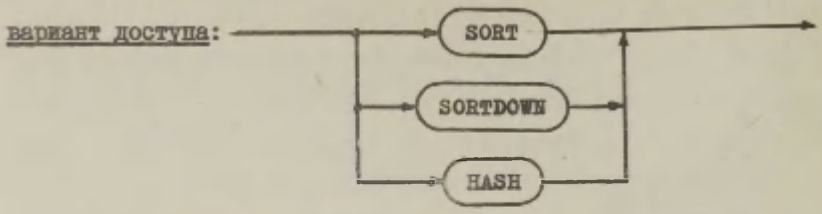
```
* 2  МАСТЬ TEXT SCOPE = [ ПИКИ, ЧЕРВИ, БУБНЫ, ТРЕФЫ ]
* 2  КАРТА TEXT SCOPE = [ ТУЗ, КОРОЛЬ, ДАМА, ВАЛЕТ,
       10, 9, 8, 7, 6, 5, 4, 3, 2 ]
* 2  КОЛОДА  ARRAY [ (МАСТЬ), (КАРТА) ]  INDEX
```

КОЛОДА является двумерным массивом с 52 элементами, которые получают значения от 1 до 52. Индексами этого массива служат заданные значения атомов МАСТЬ и КАРТА. Таким образом, элемент КОЛОДА [ПИКИ, 2] получает значение 13, КОЛОДА [ТРЕФЫ, ТУЗ] значение 40 и т.д.

Если в составе группы с LIST-организацией имеются INDEX-атомы, то их значения локализируются в пределах каждой подцепи.

2.5. Доступ к группе. Экземпляры массива доступны только по наборам значений индексов. В случае же группы с организацией REP или LIST можно построить и доступ к экземплярам этой группы по явному ключу. Для этого в корне группы следует задавать вариант и ключ доступа:

ключ доступа: 



Составные имена ссылают здесь на атомы, по значениям которых можно отыскать необходимый экземпляр. Требуется, чтобы ключи были определены как атомы той ветки, которой соответствуют экземпляры этой группы.

Семантика вариантов доступа SORT и SORTDOWN такова: экземпляры REP-группы следует упорядочить (соответственно в возрастающем и убывающем порядке) по значениям атомов ключа; в LIST-группе с указателями DPOINT все подцепи сортируются в отдельности. HASH значит, что экземпляры сохраняют свое физическое расположение, но с группой связывается хэш-таблица, содержащая ссылки на экземпляры группы.

Если задан вариант доступа, то ключ доступа можно опускать лишь в случае множественного атома: этот атом автоматически считается ключом доступа. Если в корне группы задан ключ доступа, но опущен вариант доступа, то вариантом доступа считается последовательный просмотр экземпляров группы; при том требуется уникальность значений атомов ключа в пределах данной группы.

Если атом ключа доступа имеет тип TEXT, то упорядочение происходит лексикографически по коду ДКОИ. Если атом ключа имеет запас значений, то за основу упорядочения принимается заданная в запасе порядок - слева направо, если вариантом доступа служит SORT, в обратном порядке при SORTDOWN.

Сортирование экземпляров группы естественно использовать в том случае, если последующая обработка информации происходит в определенном порядке. Например, приведенную в предыдущем пункте легенду ШКОЛЫ в целях ускорения обработки целесообразнее представить в виде:

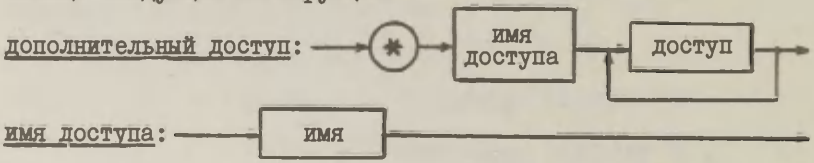
```
LEG  ШКОЛЫ  KEY = ШКОЛА  TEXT
  * 1  ШКОЛА  PICT = 50
  * 1  КОЛИЧЕСТВО  CONST  'К-ВО КЛАССОВ'
  * 1  КЛАСС  REP = КОЛИЧЕСТ  KEY = НОМЕР  SORT
    * 2  НОМЕР  PICT = 3
    * 2  КОЛИЧЕСТВО  CONST  MAX = 40  'К-ВО УЧЕНИКОВ'
    * 2  УЧЕНИК  REP = КЛАСС.КОЛИЧЕСТ  PICT = 20  SORT
         KEY = ФАМИЛИЯ, ИМЯ
      * 3  ИМЯ
      * 3  ФАМИЛИЯ
      * 3  ПОЛ  SCOPE = [М, Д]
      * 3  НОМЕР  INDEX
    * 2  ПРЕДМЕТ  SORT  PICT = 8
END
```

Классы упорядочиваются в порядке возрастания их номеров (1 А, 1 Б,..., 11 В), предметы каждого класса в альфавитном порядке их названий, а ученики в альфавитном порядке их фамилий и имен. Тезки не допускаются, так как ключ должен однозначно идентифицировать экземпляр группы. Значения атома

УЧЕНИК.НОМЕР будут теперь указывать на порядковый номер ученика в альфавитном порядке.

При варианте доступа HASH хэширование экземпляров происходит по набору значений атомов ключа доступа (соответствующая функция выбирается системой). Такой доступ естественно выбрать в том случае, когда порядок обработки заранее не известен. Как и при сортировании, ключ должен однозначно идентифицировать экземпляр группы. Если в такой группе имеется атом типа INDEX (без дополнительного упорядочения), то его значениями будут порядковые номера поступления экземпляров (как и при REP).

Иногда во время составления легенды уже известно несколько необходимых порядков просмотра экземпляров группы при обработке. В таком случае один из этих порядков выбирается в качестве доступа к группе, а использование других можно упростить путем создания дополнительных доступов к группе при помощи следующей конструкции:



В дополнительном доступе должны быть объязательно указаны как ключ так и вариант доступа (в любом порядке).

Любой дополнительный доступ реализуется путем создания ссылок на соответствующие экземпляры повторяющейся группы. Если вариантом доступа служит сортировка, то создается цепь ссылок, а звенья цепи упорядочены по значениям ключа. Дополнительное хэширование осуществляется при помощи дополнитель-

ной хэш-таблицы, элементами которой являются ссылки на физические экземпляры. Уникальность ключей теперь не требуется - при их совпадении учитывается индекс по основному доступу.

Рассмотрим, например, легенду:

```
LEG  300  KEY = ЗООПАРК  TEXT
  * 1  ЗООПАРК
  * 1  ЖИВОТНОЕ  SORT  KEY = ВИД, КЛИЧКА  PICT = 15  REP
     *   ЖКЛИЧ  HASH  KEY = КЛИЧКА
     *   ЖХАР  KEY = ХАРАКТЕР  HASH
     *   ЖМАСТЬ  SORT  KEY = МАСТЬ
       * 2  ВИД
       * 2  МАСТЬ  SCOPE = [ БЕЛАЯ, ЖЕЛТАЯ,  ТЕМНАЯ,
            ПЕСТРАЯ ]
       * 2  ПОЛ  SCOPE = [ САМЕЦ, САМКА ]
       * 2  КЛИЧКА
       * 2  ХАРАКТЕР  SCOPE = [ ХОРОШИЙ, НЕИЗВЕСТНЫЙ,
            ПЛОХОЙ ]
END
```

Экземпляры группы ЖИВОТНОЕ сортируются здесь по видам и кличкам. Для упорядоченного доступа по мастям создается цепь, структура которой показана на рис. 8. Для доступа к экземплярам группы по ключам КЛИЧКА и ХАРАКТЕР создаются еще две хэш-таблицы с аналогичной структурой.

В программах обработки выбор между доступами осуществля-

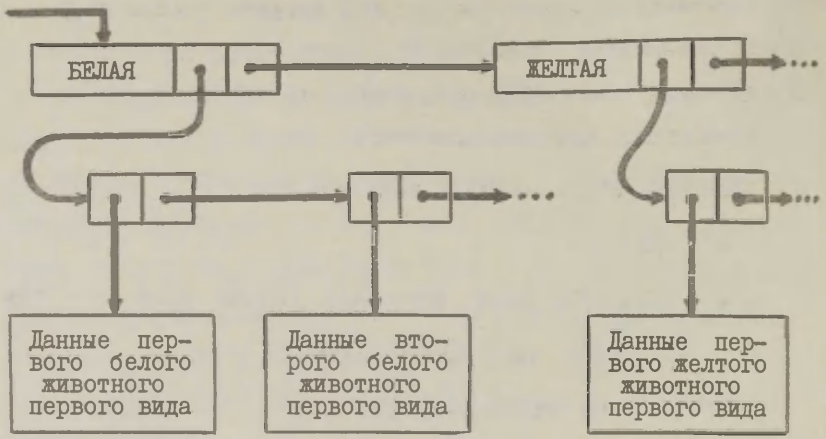


Рис. 8.

ется по имени. Имя корня группы (ЖИВОТНОЕ) связано с основным доступом, для использования дополнительного доступа вместо этого имени следует указать соответствующее имя доступа (ЖКЛИЧ, ЖХАР или ЖМАСТЬ).

Дополнительные доступы допустимы для любой организации повторяющейся группы. Например, в легенде УЧЕНИК из предыдущего раздела можно добавить дополнительный доступ к элементам массива У таким образом:

```
* 1  У ('УСПЕВАЕМОСТЬ')  ARRAY [СКМ = 4,ЧП ]  NAT
*    ПРОПУЛ  KEY = ПРОП  SORTDOWN
* 2  ОЦЕНКА  MAX = 5
* 2  ПРОП  MAX = 100
```

Если вывести данные на печать по цепи ПРОПУЛ, то легко показать ученику, как пропуск занятий влияет на его оценки.

Следующий отрывок легенды определяет племя:

```
* 3  ЧЕЛОВЕК  PICT = 20  HASH  TEXT  LIST  KEY =
     ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО
  *  ХАДРЕС   HASH KEY = АДРЕС
    * 4  ИМЯ
    * 4  ОТЧЕСТВО
    * 4  ФАМИЛИЯ
    * 4  ДАТА_РОЖД  DATE
    * 4  ДЕТИ  DPOINT  'ССЫЛКА НА ПЕРВЫЙ РЕБЕНОК'
    * 4  БРАТ  RPOINT  'ССЫЛКА НА СЛЕДУЮЩИЙ БРАТ'
    * 4  АДРЕС
```

Наконец, повторяющаяся группа может быть определена без явного доступа, но к ее экземплярам можно задавать дополнительные доступы. Например, в легенде

```
LEG  МАТЕРИАЛЫ  KEY = СКЛАД
  * 1  СКЛАД  TEXT  PICT = 20
  * 2  МАТЕРИАЛ  REP
    *   СМ  SORT KEY = НОМЕР
    *   ХЦ  KEY = ЦВЕТ  HASH
      * 3  НОМЕР  DEC  PICT = 9
      * 3  ЦВЕТ  TEXT  PICT = 10
      * 3  ЦЕНА  REAL
END
```

основной доступ к экземплярам группы МАТЕРИАЛ производится по неявному ключу (порядковому номеру), но дополнительные доступы организованы по "номерам" и цветам материалов.

2.6. Копия. Копия является свойством вершины, которое допустимо только в листе легенды. Это свойство задается конструкцией

где составное имя указывает на некоторую (копируемую) вершину, свойства и подчиненные которой должны быть перенесены в копирующую вершину. При этом копируемая вершина не может быть предком копирующей вершины.

Копия используется для сокращения текста легенды. Такое сокращение возможно тогда, когда несколько атомов имеют совпадающие свойства или же несколько независимых вершин должны быть корнями одинаковых поддеревьев. Из указанных в копируемой вершине свойств переносятся все те, которые явно не заданы в копирующей вершине. Если копируемая вершина является корнем группы, то и копирующая вершина превращается в корень точно такого же поддерева. При этом имена и свойства всех элементов поддерева сохраняются полностью, лишь номера уровней будут в случае необходимости увеличены.

Например, в отрывке легенды

```
* 4  УЧИТЕЛЬ REP HASH TEXT PICT = 15 KEY = ФАМ, ИМЯ
  * 5  ИМЯ
  * 5  ФАМ
```

```
        * 5  ПОЛ   SCOPE = [ Ж, М ]
        * 5  ВОЗРАСТ   DATE
      * 4  КЛАСС   SORT  KEY = НОМЕР   REP
        * 5  УЧЕНИК   SORT   LIKE = УЧИТЕЛЬ
        * 5  НОМЕР   DEC   PICT = 3
```

вершина УЧЕНИК представляет собой сокращенную форму следующего описания:

```
      * 5  УЧЕНИК   REP SORT TEXT   PICT=15   KEY = ФАМ, ИМЯ
        * 6  ИМЯ
        * 6  ФАМ
        * 6  ПОЛ  SCOPE  = [ Ж, М ]
        * 6  ВОЗРАСТ   DATE
```

3. Примеры

3.1. Пример проекта "СОКРАТ". В [2] приводится следующая структура данных о курортных городах:

```
ENTITE  (1000)  VILLE
   DEBUT
      NOM  MOT  (30)
      DEPARTEMENT  DE  1  A  95
      POPULATION  DE  100  A  3000000
      ALTITUDE  DE  100  A  2000
      TYPE  (THERMALE  SKI)
```

```
    SI  EXISTE  TYPE
      ALORS
        SI  TYPE = 'THERMALE'
          ALORS
            DEB-SAISON  DATE
            FIN-SAISON  DATE
          SINON
            ALT-MIN  DE  1000  A  2000
            ALT-MAX  DE  1200  A  3000
        FIN
    FIN
ENTITE  (100)  HOTEL
 DEBUT
    NOM  MOT  (30)
    TYPE  (GDLUXE  LUXE  CONFORT  SIMPLE  MODERNE)
    ADRESSE
      DEBUT
        NO-RUE  TEXTE  (1)
        TELEPHONE  MOT  (7)
      FIN
    CHAMBRES
      DEBUT
```

```
        NBRE  DE  10  A  300
        CONFORT  (BAIN  DOUCHE  BAIN-WC  DOUCHE-WC)
        PRIX-MIN  DE  20  A  70
        PRIX-MAX  DE  20  A  300
      FIN
  FIN
FIN
```

При представлении этой структуры на языке RDL принимаем за запись данные об одном городе, так что заданная в оригинале повторяющаяся группа городов будет у нас набором записей. Данные одного города можно представить в виде следующей легенды, где при помощи комментариев даны переводы идентификаторов и текстовых констант:

```
LEG  VILLE  KEY = NOM_VILLE
  * 1  NOM_VILLE  TEXT  PICT = 30  'НАЗВАНИЕ ГОРОДА'
  * 1  DEPARTEMENT  NAT  MAX = 95  'ОБЛАСТЬ'
  * 1  POPULATION  SCOPE = [100 : 3000000] 'НАСЕЛЕНИЕ'
  * 1  ALTITUDE  SCOPE = [100 : 2000] 'ВЫСОТА'
  * 1  TYPE  SCOPE = [THERMALE, SKI]  TEXT 'ТИП КУРОР-
       ТА : ЦЕЛЕБНЫЕ ВОДЫ ИЛИ ЛЫЖНЫЙ'
  * 1  SAISON_ALT  CASE = TYPE
    * 2  SAISON  DATE
      * 3  DEBUT  'НАЧАЛО СЕЗОНА'
```

```
    * 3  FIN  'КОНЕЦ СЕЗОНА'
  * 2  ALT  NAT  'МИН И МАКС ВЫСОТА ГОР'
    * 3  ALT_MIN  SCOPE = [1000 : 2000]
    * 3  ALT_MAX  SCOPE = [1200 : 3000]
* 1  HOTEL  REP  TEXT
  * 2  NOM  PICT = 30  'НАЗВАНИЕ ОТЕЛЯ'
  * 2  TYPE  SCOPE = [ GDLUXE, LUXE, CONFORT, SIMPLE,
       MODERNE ] 'КАТЕГОРИЯ : ФЕШЕНЕБЕЛЬНЫЙ, РОСКОШНЫЙ,
       КОМФОРТАБЕЛЬНЫЙ, ПРОСТОЙ, СОВРЕМЕННЫЙ'
  * 2  ADRESSE
    * 3  NO_RUE  PICT = 128  'АДРЕС ОТЕЛЯ'
    * 3  TELEPHONE  PICT = 7
  * 2  CHAMBRES
    * 3  NBRE  NAT  SCOPE = [ 10 : 300 ] 'ЧИСЛО КОМНАТ В
         ОТЕЛЕ'
    * 3  CONFORT  SCOPE = [ BAIN, DOUCHE, BAIN_WC,
         DOUCHE_WC ] 'УДОБСТВА : ВАННА, ДУШ, САНУЗЕЛ-
         ВАННА, САНУЗЕЛ-ДУШ'
    * 3  PRIX_MIN  NAT  SCOPE = [ 20 : 70 ]
    * 3  PRIX_MAX  NAT  SCOPE = [ 20 : 300 ] 'МИН И МАКС
         СТОИМОСТЬ КОМНАТЫ'
END
```

3.2. Пример РГБД. В [1] приводится структура "оргданных" на языке определения данных Рабочей группы по базам данных (РГБД) Комитета по языкам программирования CODASYL:

```
SCHEMA NAME IS ORGDATA

AREA NAME IS ORGPART

RECORD NAME IS ORG

PRIVACY LOCK IS SESAME

01 ORGCODE PICTURE IS << 9 (4) >>

01 ORGNAME TYPE IS CHARACTER 25

01 REPORTO PICTURE IS << 9999 >>

01 BUDGET TYPE DECIMAL FLOAT; IS ACTUAL RESULT
   OF SALSUM ON MEMBERS OF PERSONS

01 NOSUBORG TYPE BINARY

01 SUBORG OCCURS NOSUBORG TIMES
  02 SUBCODE PICTURE << 9999 >>

RECORD NAME IS JOB

01 JOBCODE PICTURE << 9999 >>

01 AUTHQUAN PICTURE << 99 >>

01 AUTHSAL TYPE FLOAT

RECORD NAME IS PERSON

01 EMPNO PICTURE << 9 (5) >>

01 EMPNAME TYPE CHARACTER 20

01 SEX PICTURE << A >>
```

```
01  EMPJCODE  PICTURE << 9999 >>
01  LEVEL  PICTURE << X (4) >>
01  SALARY  PICTURE << 9 (5) V 99>>; PRIVACY LOCK FOR
    GET  IS  PROCEDURE  AUTHENT
01  BIRTH
  02  MONTH  PICTURE << 99 >>
  02  DAY  PICTURE << 99 >>
  02  YEAR  PICTURE << 99 >>
01  NOSKILLS  TYPE  BINARY
01  SKILLS  OCCURS  NOSKILLS  TIMES
  02  SKILCODE  PICTURE << 9999 >>
  02  SKLYRS  PICTURE << 99 >>
SET  NAME  IS  JOBS;  ORDER  IS  SORTED
  OWNER  IS  ORG
  MEMBER  IS  JOB  OPTIONAL  AUTOMATIC;  ASCENDING  KEY
    IS  JOBCODE  DUPLICATES  NOT  ALLOWED
SET  NAME  IS  PERSONS,  ORDER  IS  SORTED
  OWNER  IS  ORG
  MEMBER  IS  PERSON  OPTIONAL  AUTOMATIC;  ASCENDING
    KEY  IS  EMPJCODE,  EMPNO  DUPLICATES  NOT  ALLOWED
```

Для представления этого описания на языке RDL следует сделать две оговорки. Во-первых, надо предполагать, что объемы как данных, так и оперативной памяти ЭВМ позволяют по-

местить любую ветвь из "оргданных" любой организации в оперативную память ЭВМ. Во-вторых, опускаются замки защиты данных: их место в рассматриваемой системе принадлежит структуре более высокого уровня, чем уровень записи. С этими оговорками соответствующая легенда принимает вид (допустим, что функция SUM найдет сумму зарплат всех личностей каждой организации):

```
LEG  ORGDATA  KEY = ORGCODE
  * 1  ORGCODE  DEC  MAX = 9999
  * 1  ORGNAME  TEXT  PICT = 25
  * 1  REPORTO  LIKE = ORGCODE
  * 1  BUDGET  REAL  PSEUDO = SUM (SALARY)
  * 1  NOSUBORG  CONST
  * 1  SUBCODE  DEC  PICT = 4  REP = NOSUBORG
  * 1  JOB  KEY = JOBCODE  SORT  DEC  REP
    * 2  JOBCODE  PICT = 4
    * 2  AUTHQUAN  PICT = 2
    * 2  AUTHSAL  REAL
  * 1  PERSON  SORT  KEY = EMPJCODE,  EMPNO  REP  TEXT
    * 2  EMPNO  DEC  MAX = 99999
    * 2  EMPNAME  PICT = 20
    * 2  SEX  PICT = 1
    * 2  EMPJCODE  PICT = 4  DEC
```

```
    * 2  LEVEL  PICT = 4
    * 2  SALARY  REAL  PICT = 3.2
    * 2  BIRTH  DATE
    * 2  NOSKILLS  CONST
    * 2  SKILLS  REP = NOSKILLS  DEC
      * 3  SKILCODE  PICT = 4
      * 3  SKLYRS  PICT = 2
END
```

3.3. Большой пример. В заключительном примере применяются почти все возможности определения данных. В некоторых случаях разнообразию дано предпочтение перед разумностью, но авторы надеются, что это им простят.

```
LEG  ШКОЛА  KEY = НОМЕР  'ДАННЫЕ ОБЩЕОБРАЗОВАТЕЛЬНОЙ
     ШКОЛЫ'  TEXT
  * 1  НОМЕР  DEC  MAX = 100
  * 1  ИМЯ
  * 1  АДРЕС
  * 1  ДИРЕКТОР
    * 2  ИМЯ
    * 2  ОТЧЕСТВО
    * 2  ФАМИЛИЯ
  * 1  ЗАМЕСТИТЕЛИ  HASH  LIST  KEY = ФАМИЛИЯ,  ИМЯ,
       ОТЧЕСТВО  'ЗАМ. ДИРЕКТОРА'
```

```
  * 2 ИМЕНА LIKE = ДИРЕКТОР
  * 2 ДОЛЖНОСТЬ
  * 2 СЛ RPOIBT
* 1 ДНИ SCOPE = [ПОНЕДЕЛЬНИК, ВТОРНИК, СРЕДА,
    ЧЕТВЕРГ, ПЯТНИЦА, СУББОТА] PSEUDO 'ЭТОМУ
    АТОМУ НЕ БУДУТ ПРИСВАИВАТЬ ЗНАЧЕНИЯ'
* 1 КЛАССЫ CONST 'К-ВО КОМПЛЕКТОВ'
* 1 РАСПИСАНИЕ ARRAY [СЕМЕСТР = 4, КЛАССЫ,
    УРОК = 7, ДЕНЬ = (ДНИ)] PICT = 8
  * ПРЕПОДАВАТЕЛЬ HASH KEY = УЧИТЕЛЬ
  * 2 ПРЕДМЕТ
  * 2 УЧИТЕЛЬ
  * 2 КЛАСС PICT = 5
* 1 УЧИТЕЛИ REP
  * ФУ KEY = ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО HASH
  * 2 ИМЕНА LIKE = ДИРЕКТОР
  * 2 ПРЕДМЕТЫ REP PICT = 8
  * 2 НП 'ПОРЯДКОВЫЙ НОМЕР' BASE = ФАМИЛИЯ,
      ИМЯ INDEX
* 1 КОЛИЧЕСТВО PSEUDO = SUM (КЛАСС.КОЛИЧЕСТ)
    'КОЛИЧЕСТВО УЧЕНИКОВ В ШКОЛЕ' NAT
* 1 КЛАСС REP = КЛАССЫ KEY = НОМЕР SORT
```

```
* УСПЕВАЕМОСТЬ SORTDOWN KEY = СР_ОЦ
* Х_РУК KEY = РУКОВОДИТЕЛЬ NASN
* 2 НОМЕР PICT = 5
* * КЛАСС POS = 1 : 4 SCOPE = [ I, II, III, IV,
    V, VI, VII, VIII, IX, X, XI ]
* * ПАРАЛ POS = 5 SCOPE = [ A : F ]
* 2 РУКОВОДИТЕЛЬ LIKE = ДИРЕКТОР
* 2 СР_ОЦ REAL PSEUDO = MEAN2 (УЧЕНИК, ТАБЕЛЬ)
    PICT = 1.2
* 2 КОЛИЧЕСТВО NAT PSEUDO = SNOOP (УЧЕНИК)
    'КОЛИЧЕСТВО УЧЕНИКОВ В КЛАССЕ; УСТАНАВЛИВАЕТСЯ
    ФУНКЦИЕЙ SNOOP'
* 2 КП ('КОЛИЧЕСТВО ПРЕДМЕТОВ') MAX = 20 CONST
* 2 ПРЕДМЕТЫ PICT = 8 ARRAY [ КП ]
* 2 УЧЕНИК HASH KEY = ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО
    LIST
  * 3 ИМЕНА LIKE = ДИРЕКТОР
  * 3 ДАТА ('ДАТА РОЖДЕНИЯ') DATE 'POS 5 : 6 =
      YEAR, 3 : 4 = MONTH, 1 : 2 = DAY'
  * 3 ПОЛ SCOPE = [ Д, М ] 'Д = ДЕВОЧКА,
      М = МАЛЬЧИК'
  * 3 ТАБЕЛЬ ARRAY [ 4, КП ] NAT MAX = 5 'ПЕРВЫЙ
```

```
      ИНДЕКС УСТАНАВЛИВАЕТ СЕМЕСТР, ВТОРОЙ - ПРЕД-
      МЕТ. ЭЛЕМЕНТАМИ МАССИВА ЯВЛЯЮТСЯ ОЦЕНКИ.'
   * 3 СРЕДНЯЯ PICT = 1.2 PSEUDO = MEAN1 (ТАБЕЛЬ)
      REAL 'СР.ОЦЕНКА'
   * 3 МЕСТО INDEX BASEDOWN = СРЕДНЯЯ
   * 3 ИНТЕРЕС SCOPE =[ НЕТ, ТЕХНИКА, ИСКУССТВО,
      СПОРТ, ПРОЧЕЕ ]
   * 3 ГДЕ CASE = ИНТЕРЕС
     * 4 НЕТ NIL
     * 4 КРУЖОК ('НАИМЕНОВАНИЕ ТЕХНИЧЕСКОГО
        КРУЖКА')
     * 4 ИСКУССТВО
       * 5 ВИД
       * 5 АДРЕС 'МУЗЫКАЛЬНОЙ И ТП ШКОЛЫ'
     * 4 СПОРТ
       * 5 ВИД
       * 5 ТРЕНЕР LIKE = ДИРЕКТОР
       * 5 РАЗРЯД NAT
     * 4 РАЗНОЕ EXTERNAL = E 'ЛЮБЫЕ ДАННЫЕ ЛЮБОЙ
        СТРУКТУРЫ'
   * 3 СЛЕДУЧЕНИК RPOINT
* 1 ИНВЕНТАРЬ 'ИМУЩЕСТВО ШКОЛЫ' SORT REP KEY = НН
```

```
PACK
  * X_ИМЯ HASH KEY = ИМЯ
  * 2 НН ('НОМЕНКЛАТУРНЫЙ НОМЕР') MAX = 99999 DEC
  * 2 ИМЯ ('НАИМЕНОВАНИЕ')
  * 2 К_ВО ('КОЛИЧЕСТВО') REAL PICT = 3.2
  * 2 ЦЕНА PICT = 4.2 REAL
  * 2 СТОИМОСТЬ REAL PICT = 5.2 PSEUDO = MULT
      (ЦЕНА, К_ВО) 'MULT УМНОЖАЕТ ЗНАЧЕНИЯ ПАРАМЕТ-
      РОВ'
END
```

4. Определение языка

4.1. Терминальный альфавит языка RDL состоит из следующих элементов:

слово (в дальнейшем - i): последовательность из 1 до 8 символов[1] кода ДКОИ, начинающая буквой и не содержащая разделителей;

буква (l): одна буква кода ДКОИ;

натуральное число без знака (n);

число (c): натуральное или вещественное число со знаком или без знака в обычном или полулогарифмическом виде;

---

1 Если в тексте представляется более длинное слово, то символы начиная с девятого игнорируются.

стринг (s): произвольный текст из не более чем 255 символов между символами " ' ", не содержащий символа " ' ";

разделители[1]: * ** ␣ = , . : [ ] ( )

служебные слова: **ARRAY BASE BASEDOWN CASE CONST DATE DEC DPOINT END EXTERNAL FDATE HASH HEX INDEX INT KEY LEG LIKE LIST MAX NAT NIL PACK PICT POS PSEUDO REAL REP RPOINT SCOPE SORT SORTDOWN TEXT**

4.2. Нетерминальный альфавит языка RDL состоит из следующих понятий[2]:

легенда, заголовок, свойства легенды, свойство легенды, ключ доступа, список составных имен, составное имя, описание, вершина, идентификатор, свойства вершины, свойство вершины, тип, длина, ограничение, максимальное значение, запас значений, диапазон, пара, псевдо, функция, список параметров, параметр, дополнительное упорядочение, селекция, повторение, организация, массив, границы, размер, индекс, количество повторений, доступ, вариант доступа, копия, указатель, внешняя структура, сечения, сечение, позиция, дополнительные доступы, дополнительный доступ.

Исходным элементом альфавита является "легенда".

1 Символ "␣" является общим разделителем, который следует написать на место разделителя всегда, когда не заданы другие разделители. В последовательности этих символов учитывается только первый, а остальные игнорируются.

2 Список элементов нетерминального альфавита не совпадает с системой понятий, представленной во второй главе. В интересах читаемости там приходилось пользоваться более содержательными определениями и систематизировать понятия так, чтобы облегчить объяснение их семантики.

4.3. Продукции.

⟨легенда⟩ → LEG ⟨заголовок⟩ ⟨описание⟩ END

⟨заголовок⟩ → i ⟨свойства легенды⟩

⟨свойства легенды⟩ → ⟨свойство легенды⟩

→ ⟨свойства легенды⟩ ⟨свойство легенды⟩

⟨свойство легенды⟩ → ⟨ключ доступа⟩

→ s

→ ⟨максимальное значение⟩

→ ⟨тип⟩

→ ⟨длина⟩

⟨ключ доступа⟩ → KEY = ⟨список составных имен⟩

⟨список составных имен⟩ → ⟨составное имя⟩

→ ⟨список составных имен⟩ , ⟨составное имя⟩

⟨составное имя⟩ → i

→ ⟨составное имя⟩ . i

⟨описание⟩ → ⟨вершина⟩

→ ⟨описание⟩ ⟨вершина⟩

⟨вершина⟩ → ⟨идентификатор⟩

→ ⟨идентификатор⟩ ⟨свойства вершины⟩

→ ⟨идентификатор⟩ ⟨свойства вершины⟩ ⟨сечения⟩

→ ⟨идентификатор⟩ ⟨сечения⟩

→ ⟨идентификатор⟩ ⟨свойства вершины⟩ ⟨дополнительные доступы⟩

⟨идентификатор⟩ → * n i

→ * n i (s)

⟨свойства вершины⟩ → ⟨свойство вершины⟩

→ ⟨свойства вершины⟩ ⟨свойство вершины⟩

⟨свойство вершины⟩ → s

→ ⟨тип⟩

→ ⟨длина⟩

→ ⟨ограничение⟩

→ ⟨псевдо⟩

→ ⟨дополнительное упорядочение⟩

→ ⟨селекция⟩

→ ⟨повторение⟩

→ ⟨копия⟩

→ ⟨указатель⟩

→ **INDEX**

→ **NIL**

→ **CONST**

→ ⟨внешняя структура⟩

→ **PACK**

⟨тип⟩ → **TEXT**

→ **REAL**

→ **INT**

→ **NAT**

→ **HEX**

→ **DEC**

→ **DATE**

→ **FDATE**

⟨длина⟩ → **PICT** = c

⟨ограничение⟩ → ⟨максимальное значение⟩

→ ⟨запас значений⟩

⟨максимальное значение⟩ → **MAX** = c

⟨запас значений⟩ → **SCOPE** = [⟨диапазон⟩]

⟨диапазон⟩ → ⟨пара⟩

→ ⟨диапазон⟩ , ⟨пара⟩

⟨пара⟩ → c

→ i

→ s

→ c : c

→ l : l

⟨псевдо⟩ → **PSEUDO**

→ **PSEUDO** = i

→ **PSEUDO** = c

→ **PSEUDO** = s

→ **PSEUDO** = ⟨функция⟩

⟨функция⟩ → i ( )

→ i (⟨список параметров⟩)

⟨список параметров⟩ → ⟨параметр⟩

→ ⟨список параметров⟩ , ⟨параметр⟩

⟨параметр⟩ → ⟨составное имя⟩

→ c

→ s

⟨дополнительное упорядочение⟩ → **BASE** = ⟨список составных имен⟩

→ **BASEDOWN** = ⟨список составных имен⟩

⟨селекция⟩ → **CASE** = ⟨составное имя⟩

⟨повторение⟩ → ⟨организация⟩

→ ⟨массив⟩

→ ⟨организация⟩ = ⟨количество повторений⟩

→ ⟨доступ⟩

⟨организация⟩ → **REP**

→ **LIST**

⟨массив⟩ → **ARRAY** [⟨границы⟩]

⟨границы⟩ → ⟨размер⟩

→ ⟨границы⟩ , ⟨размер⟩

⟨размер⟩ → ⟨индекс⟩

→ i = ⟨индекс⟩

⟨индекс⟩ → ⟨количество повторений⟩

→ (⟨составное имя⟩)

⟨количество повторений⟩ → n

→ ⟨составное имя⟩

⟨доступ⟩ → ⟨ключ доступа⟩

→ ⟨вариант доступа⟩

⟨вариант доступа⟩ → **HASH**

→ **SORT**

→ **SORTDOWN**

⟨копия⟩ → **LIKE** = ⟨составное имя⟩

⟨указатель⟩ → **DPOINT**

→ **RPOINT**

⟨внешняя структура⟩ → **EXTERNAL** = i

⟨сечения⟩ → ⟨сечение⟩

→ ⟨сечения⟩ ⟨сечение⟩

⟨сечение⟩ → * * i ⟨позиция⟩

→ * * i ⟨позиция⟩ ⟨ограничение⟩

→ * * i ⟨ограничение⟩ ⟨позиция⟩

⟨позиция⟩ → POS = n

→ POS = n : n

⟨дополнительные доступы⟩ → ⟨дополнительный доступ⟩

→ ⟨дополнительные доступы⟩ ⟨дополнительный доступ⟩

⟨дополнительный доступ⟩ → * i ⟨ключ доступа⟩ ⟨вариант доступа⟩

→ * i ⟨вариант доступа⟩ ⟨ключ доступа⟩

Литература

1. Информационные системы общего назначения. "Статистика", Москва, 1975.
2. Проект "СОКРАТ". Всесоюзный центр переводов научно-технической литературы и документации государственного комитета Совета Министров СССР по науке и технике и АН СССР. Перевод № Ц – 10844. Москва, 1973.
3. Кнут Д., Искусство программирования для ЭВМ. Т.1, "Мир", Москва, 1976.
4. Виллемс А.Л., Изотамм А.А., Организация данных в системе VILLIS. Труды ВЦ ТГУ, 1976, № 38, 7 – 39.
5. Wirth, N., The programming language PASCAL. Acta Informatica, 1971, 1.
6. Вирт Н., Систематическое программирование. "Мир", Москва, 1977.
7. Томбак М.О., Микли Т.И., Аллик К.К.-Э., Об отношении субординации. Труды ВЦ ТГУ, 1974, № 30, 8 – 22.

ОБРАЗОВАНИЕ ФАЙЛОВ

Ю. Каазик

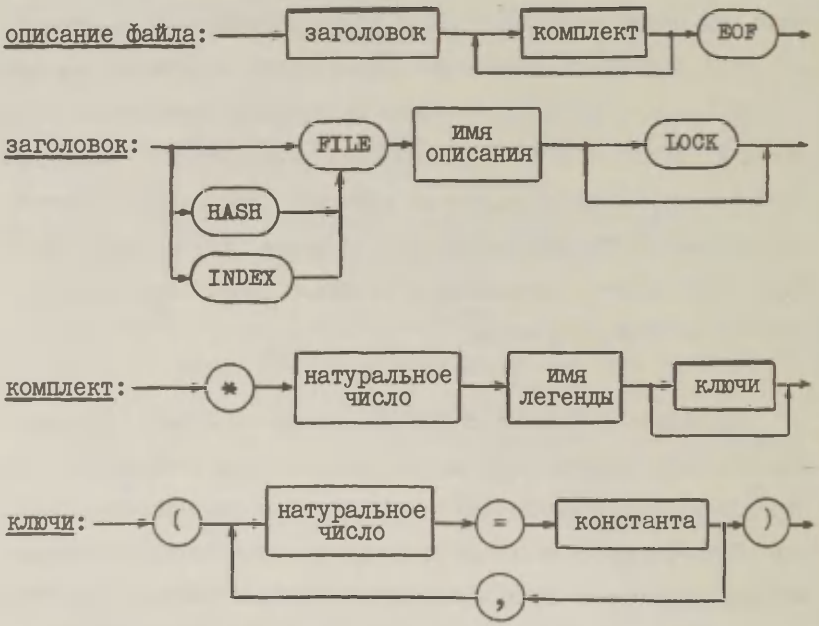
В статье рассматриваются средства системы РАМА для объединения записей в файлы и установления структуры файлов. Эти средства выбраны пока сравнительно элементарными, чтобы упростить их реализацию. Отсутствует, например, непосредственная возможность описания файла с альтернативными записями, а также возможность изменения объявленных в легенде ключей записи или их очередности (такие построения осуществимы лишь путем предварительной корректировки используемых легенд). Наиболее серьезным упрощением является отсутствие понятия повторяющейся группы файлов. Тем не менее, описываемые средства позволяют построить файлы со всемы наиболее важными в практике типами структуры.

Вся информация в банке данных храняется в виде физических записей. Однако, с точки зрения пользователей обмен информацией между накопителями происходит по логическим записям. Поэтому здесь опускаются вопросы возможного разбиения или объединения логических записей во время образования физических записей (блоков). Под термином "запись" в дальнейшем понимается логическая запись, структура которой в сис-

теме **РАМА** описывается легендой.

Хранение записей в банке данных разрешается только в виде их объединений, называемых файлами (речь идет, конечно, о логических файлах). В один файл при этом можно включить записи различной структуры, т.е. описанные разными легендами. Множество таких записей в файле, которые описаны одной легендой, будем называть комплектом (записей).

Перед включением очередного файла в банк данных должно быть составлено (и транслировано) соответствующее описание файла. Язык описания файлов FDL (=File Description Language) имеет весьма простой синтаксис:



Следует учитывать, что имя описания не является именем создаваемого файла - последнее устанавливается при открытии

конкретного файла по данному описанию. К тому же, по одному описанию можно открыть несколько файлов.

Если перед словом FILE не указано имя доступа (INDEX или HASH), то создаваемые по этому описанию файлы являются последовательными. Наличие имени доступа INDEX требуе создания индексно-последовательных файлов, а имени HASH - файлов прямого доступа. Имя доступа имеет смысл лишь тогда, когда у всех ссылаемых в комплектах легенд имеются ключи, т.е. среди их свойств встречается конструкция "ключ записи", или же в самом описании файла к ним добавлена конструкция "ключи" (если это условие не выполнено, то описание файла считается ошибочным).

Если имя доступа не указано, а у всех ссылаемых легенд имеются ключи, то в создаваемом последовательном файле записи сортируются в порядке возрастания значений их ключей (создается т.н. сортируемый последовательный файл).

Слово LOCK в заголовке описания указывает на то, что при открытии соответствующего файла следует уточнить способ защиты (полная защита или защита от изменений) и закон образования паролей.

В комплектах одного описания файла нельзя многократно использовать одну и ту же легенду. Точнее, запрещается повторение имени легенды, во всем остальном легенды могут совпадать (таким образом, чтобы использовать одну легенду дважды, придется ее дублировать с другим именем).

Натуральными числами в начале комплектов указываются номера уровней, которые порождают иерархию комплектов точно таким же образом, как аналогичные номера в легенде порождают

иерархию вершин.Основываясь на этой иерархии для комплектов, строятся т.н. ключи доступа, по которым происходит фактический доступ к записям, а также их упорядочение, если имя доступа не указано.Содержание понятию иерархии комплектов дается, по существу, правилами образования ключей доступа.

Так как все операции по обмену информацией между накопителями осуществляются в системе РАМА средствами операционной системы ОС,то каждая запись файла должна иметь ключ доступа. При этом все ключи доступа должны в пределах одного файла иметь одинаковую длину, но уникальное для каждой конкретной записи содержание.

Ключ доступа можно рассматривать как последовательность полей, число которых фиксировано для данного файла. Для всех записей одного комплекта устанавливается одинаковое правило заполнения этих полей. Поэтому мы и можем говорить о ключе доступа данного комплекта.

Когда описывается последовательный файл (нет имени доступа),то в целях упрощения обработки такого файла требуется, чтобы с ключами записи были или все ссылаемые легенды, или же ни одна из них. В последнем случае описывается т.н. простой последовательный файл. ключ доступа которого образуется состоящимся из двух полей. В первое из них пишется порядковый номер поступления конкретной записи, а во второе - номер уровня комплекта. В этом случае иерархия комплектов не имеет смысла и номера уровней можно выбрать произвольно, важно лишь, чтобы они были различными (иначе по ключу доступа нельзя было бы определить, к какому комплекту относится данная конкретная запись).

Такой тип файла понадобится главным образом тогда, когда предвидимое использование данных происходит путем просмотра всех записей всех комплектов,причем их порядок не имеет значения.

Далее рассмотрим случай, когда все ссылаемые в комплектах легенды имеют ключи. Приводимые примеры будут при этом опираться на следующие конкретные легенды (в которых опущены конструкции "описание"):

```
LEG  A  KEY = A1      . . .     END

LEG  B  KEY = B1, B2      . . .     END

LEG  C  KEY = C1, C2, C3      . . .     END

LEG  D  KEY = D1, D2, D3, D4      . . .     END
```

Содержанием последнего поля ключа доступа всегда является номер уровня комплекта.Остальных полей имеется по крайней мере столько, чтобы разместить значения всех ключей записи данного комплекта. Эти ключи указываются в соответствующей легенде или же в самом комплекте конструкцией "ключи". Последняя дает возможность дополнить последовательность ключей записи мнимыми ключами, имеющими во всех записях комплекта постоянные значения.

Натуральное число в конструкции "ключи" указывает порядковый номер ключа в окончательной последовательности ключей записи, а константа дает постоянное значение этого ключа. Кроме обычных констант разрешается еще звездочка,которая означает максимальное возможное значение:единицы во всех битах

соответствующего поля. Такое значение будем в целях наглядности далее обозначать через ∞.

Например, если в описании файла содержится комплект

*2 B (1 = 0, 3 = *)

(легенда B приведена выше), то у этого комплекта четыре ключа записи. При этом первый и третий ключ имеют во всех записях комплекта постоянное значение (соответственно 0 и ∞), значения же второго и четвертого ключей определяются конкретными значениями атомов B1 и B2 в записи. В этой связи отметим еще, что ∞ не может быть значением ключевого атома. Ведь такое "значение" считается в системе РАМА признаком отсутствия значения, а запись (как и экземпляр любой повторяющейся группы) не принимается,если хотя-бы один ключ не имеет значения.

В самом простом случае поля ключа доступа содержат по порядку значения ключей записи, за которыми может еще быть некоторое число полей с постоянным нулевым значением (для получения установленного числа полей). Когда же у данного комплекта имеется отец по иерархии или левый брат, то могут добавляться еще поля с постоянным значением ∞ между ключами записи или перед ними.

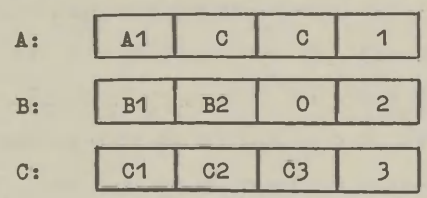
Если уже имеется правило образования ключа доступа для комплекта-отца, то это правило переносится в состав соответствующего правила для комплекта-сына. Только значения ключей записи отца заменяются значениями первых ключей записи сына. Отсюда вытекает,что число ключей записи не может у сына быть меньше чем у отца (иначе описание файла содержит ошибку!).

Если же у сына ключей записи больше, то остаток будем называть его собственными ключами.

В самом простом случае значения собственных ключей сына размещаются непосредственно за последним ключом отца. Так, например, по описанию файла

```
FILE  F1  * 1  A   * 2  B   * 3  C   EOF
```

для комплектов образуют ключи доступа, которые схематически можно изображать так:

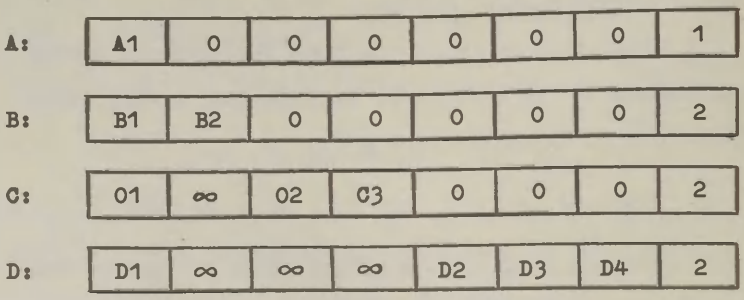


Собственными ключами комплектов A, B и C являются здесь соответственно A1, B2 и C3. Записи этого последовательного файла сортируются так,что за каждой записью комплекта A размещаются некоторые записи комплекта B(те,в которых значение атома B1 совпадает со значением атома A1 в той записи комплекта A), а за каждой из них некоторые записи комплекта C (те,в которых значения C1 и C2 совпадают со значениями B1 и B2 в их предшественнике).

Если у комплекта имеется левый брат, то все собственные ключи этого брата становятся первыми собственными ключами рассматриваемого комплекта, причем им придается постоянное значение ∞. Так, например, по описанию файла

```
FILE  F2  * 1  A   * 2  B   * 2  C   * 2  D   EOF
```

для комплектов образуют следующие ключи доступа:

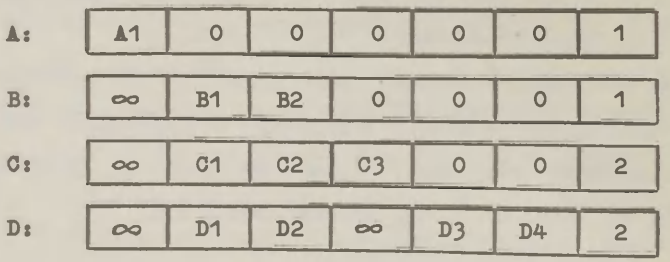


(к собственным ключам D2, D3 и D4 комплекта D присоединяются собственные ключи 02, 03 его левого брата C и присоединенный к ним собственный ключ B2 комплекта B). Записи этого файла сортируются так, что за каждой записью комплекта A размещаются некоторые записи комплекта B,за ними некоторые записи комплекта C,за ними еще некоторые записи комплекта D, и лишь тогда появляется следующая запись комплекта A.

По тем же правилам происходит образование ключей доступа и в более сложных случаях. Например, по описанию файла

FILE F3 * 1 A * 1 B * 2 C * 2 D EOF

получаем следующие ключи доступа:



Как уже отмечалось, все ключи доступа данного файла со-

держат одинаковое число полей: это число определяется как минимальное для размещения всех указанных выше значений. Чтобы эти ключи могли быть использованы операционной системой, в них и все поля на одной позиции должны быть одинаковой длины. В качестве этой длины принимается максимальная из указанных в легендах длин ключевых атомов. В последнем примере длина первого поля определяется длиной атома A1, длина второго поля максимальной из длин атомов B1, C1, D1 и т.д. Естественно ставится еще требование, чтобы все ключевые атомы, занимающие то же поле в ключах доступа, имели один и тот же тип.

Добавление мнимых ключей записи с постоянными значениями (конструкцией "ключи") используется главным образом в целях получения специальных типов упорядочения записей. Допустим, например, что приведенные выше легенды A и B описывают соответственно общие данные о школе (атом A1 содержит номер школы) и о классе (B1 содержит номер школы, B2 - номер класса). Если в будущей обработке данные о конкретной школе понадобятся всегда сразу после просмотра всех классов этой школы, то рассматриваемые данные естественно объединить в один файл (файл всех школ одного города) следующим описанием:

```
FILE  F4  * 1  B   * 2  A  (2 = *)   EOF
```

Если же данные о школе понадобятся после просмотра данных о первых четырех классах, то следует использовать описание файла

```
FILE  F5  * 1  B   * 2  A  (2 = 4)   EOF
```


10

В связи с этим примером следует еще обратить внимание на важность номера уровня в конце ключа доступа. Действительно, при отсутствии номеров уровня нельзя было бы ввести запись какой-то школы, если в файле уже имеется запись четвертого класса этой-же школы (ведь ключи доступа должны быть уникальными).

Приведенные выше примеры относятся все к сортируемым последовательным файлам, так как в этом случае главным образом и используются наиболее сложные типы иерархии. Вопросы выбора наилучшего типа иерархии имеют значение также при создании индексно-последовательных файлов, но не в случае файлов прямого доступа.

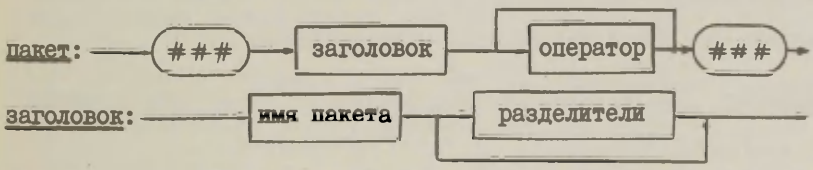
ВВОД И КОРРЕКТИРОВКА ДАННЫХ

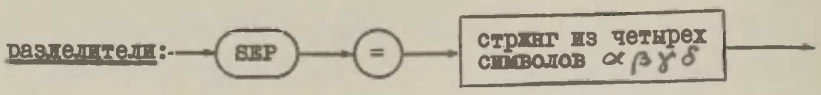
Ю. Каазик, П. Ээльма

В этой статье рассматриваются средства системы РАМА для ввода данных с перфоносителей и описывается соответствующий язык. Такой язык отделен от общих языков обработки данных главным образом потому, что возможные изменения в нем обуславливаются не столько характером решаемых задач, сколько уровнем комплектованности конкретной ЭВМ.

1. Структура пакета

Вводимая с перфоносителей в базу данных информация должна быть оформлена в виде специальной программы, написанной на языке DIL (=Data Input Language). Такая программа называется в этом языке пакетом и имеет следующую общую структуру:





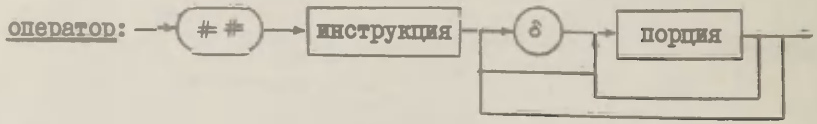
Пакет всегда относится только к одному файлу. Удобнее всего в качестве имени пакета использовать имя соответствующего файла: в таком случае отпадает необходимость указать эти два имени в заказе.

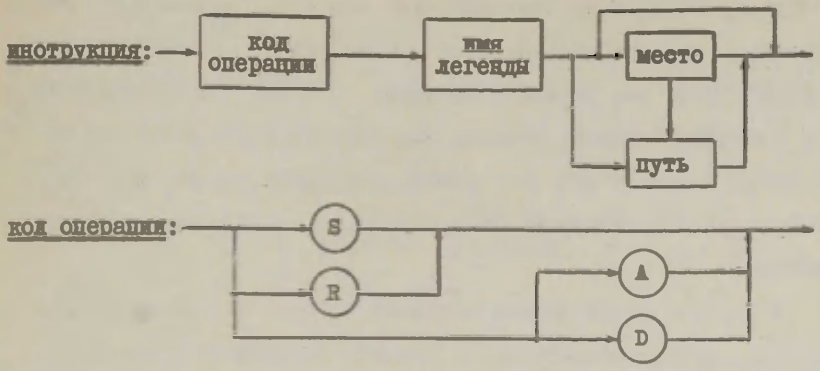
Среди всех используемых в языке DIL символов особое место занимают пять разделителей. В качестве одного из них зафиксирован символ #, но остальные четыре пользователь может в каждом пакете выбрать по своему усмотрению (только не из символов @ * # ' . +). В приводимых ниже определениях эти четыре разделителя условно обозначаются греческими буквами α, β, γ и δ. Конкретные символы для них назначаются в заголовке пакета конструкцией "разделители". Если эта конструкция опущена, то система автоматически вставит на ее место

SEP = '␣;()'

т. е. стандартным символом для α является пробел, для β точка с запятой, для γ открывающая и для δ закрывающая скобка. В приводимых ниже примерах всюду используются именно эти стандартные разделители.

Каждый оператор пакета предназначен только для однотипных операций, выполняемых в пределах записей одного комплекта данного файла. Поэтому, в операторе прежде всего зафиксируются как код операции, так и имя легенды:





Используемые коды операций являются первыми буквами соответствующих английских слов:

S (=Store) - включить;

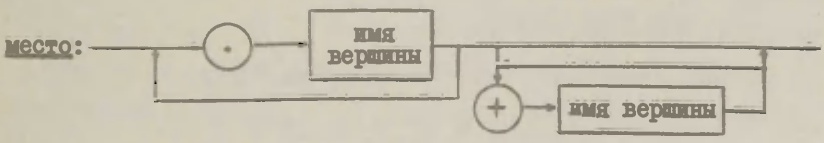
R (=Replace) - заменить;

A (=Add) - добавить;

D (=Delete) - удалить.

2. Определение обновляемого объекта

Для внесения изменений в базу данных необходимо задавать четыре компоненты: характер изменения, обновляемый объект, его экземпляр и вводимые данные (последние не требуются при удалении). Характер изменения определяется кодом операции, данные содержатся в порциях оператора, а объект и экземпляр определяются соответственно конструкциями "место" и "путь".



Место определяет составное имя вершины, подлежащего обновлению (при отсутствии этой конструкции обновляется вся запись). Если эта вершина является атомом, то обновляется его значение; если же вершина является корнем группы, то обновлению подлежат все его непосредственные подчиненные (сыновья), которые не являются указателями, индексными и псевдоатомами.

В случае, когда желают обновить не все, а лишь некоторые вершины группы, конструкцию "место" заканчивают перечислением необходимых имен вершин, отделенных друг от друга плюсами. Все перечисляемые таким образом вершины должны принадлежать одной группе, т.е. быть братьями между собой.

Пусть в файле, к которому относится составляемый пакет, содержится комплект записей, структура которых определена следующей легендой (на этой легенде базируется большинство приводимых ниже примеров):

```
LEG  КЛАСС  KEY = НОМЕР  TEXT  PICT = 15
  * 1  НОМЕР  PICT = 3
  * 1  КЛАССРУК
    * 2  ФАМИЛИЯ
    * 2  ИМЯ
    * 2  ОТЧЕСТВО
  * 1  ПРЕДМЕТЫ  REP
  * 1  КОЛ_ПРЕД  CONST
  * 1  УЧЕНИК  REP  KEY = ФАМИЛИЯ, ИМЯ  SORT
```

```
    * 2  ИМЯ
    * 2  ФАМИЛИЯ
    * 2  СР_ОЦ  PSEUDO  REAL
    * 2  ДАТА_РОЖД  DATE
    * 2  ОЦЕНКИ  ARRAY [4, КОЛ_ПРЕД ]  NAT  MAX = 5
END
```

Если в составляемом пакете содержится оператор, начало которого имеет вид

```
## S КЛАСС
```

(без указания имени вершины), то в файл включается новая запись. Если же начало оператора имеет один из следующих видов:

```
## A КЛАСС.КЛАССРУК.ИМЯ
## D КЛАСС.УЧЕНИК
## R КЛАСС.УЧЕНИК.ДАТА_РОЖД + ИМЯ
```

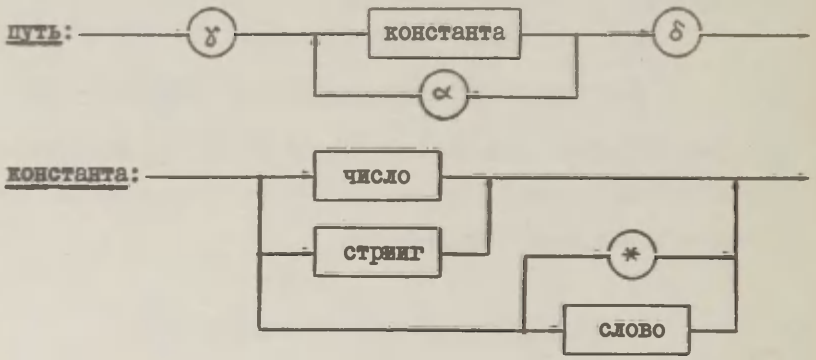
то соответственно добавляется имя классного руководителя, удаляются все данные о каком-то ученике и заменяются дата рождения и имя какого-то ученика.

3. Определение обновляемого экземпляра

Как из приведенных только-что примеров видно, место не уточняет еще ни конкретной записи, ни экземпляра повторяющейся группы: не ясно, к какому классу или ученику относятся последние операторы. Для определения обновляемого экземпляра следует задавать путь - последовательность значений ключей,

начиная от ключей записи и заканчивая ключами доступа последней группы, в которую входят определяемые местом вершины. Важно учитывать, что речь идет всюду о ключах основного доступа, так как ввод и корректировка данных по дополнительному доступу не разрешается.

Ключи доступа последней группы,до которой доходит место, в дальнейшем будем называть локальными ключами этого места (если оно определяет всю запись,то локальными являются ключи записи). Особая роль локальных ключей выражается в том, что их можно не задавать в пути. Так, например, когда обновляется полностью вся группа или запись, то значения локальных ключей в пути и не понадобятся, потому что эти значения содержатся среди самих вводимых данных (но они необходимы при удалении экземпляра группы или записи).



Значения ключей в пути следует задавать строго в том порядке, в каком эти ключи встречаются в легенде при движении от имени легенды до рассматриваемого места (если группа имеет несколько ключей доступа,то их порядок также определен в легенде). Например, если предпоследнее из приведенных выше

начал оператора переписать в виде

```
## D КЛАСС.УЧЕНИК (4А ИВАНОВ ВАНЯ)
```

то удалению подлежат данные об ученике 4-го класса "А" Ване Иванове.

Когда место относится к повторяющейся группе без явного доступа, тогда ключом считается порядковый номер экземпляра. Например, начало оператора

```
## R КЛАСС.ПРЕДМЕТЫ (7Б 3)
```

означает, что заменяется то название предмета для седьмого класса "Б", которое было записано в базу данных в качестве третьего.

В случае массива ключами считаются его индексы.Например, начало оператора

```
## S КЛАСС.ОЦЕНКИ (4А ИВАНОВА 'ВАЛЯ' 4 3)
```

означает, что собираются ввести оценку по третьему предмету в четвертом семестре для ученицы 4-го класса "А" Вали Ивановой (имя ученицы записано в виде стринга лишь для того, чтобы напомнить об этой возможности!).

Константа * на месте значения ключа означает, что соответствующая операция проводится для всех имеющихся значений этого ключа. Так, например, начало оператора

```
## S КЛАСС.ДАТА_РОЖД (4А * ВАНЯ)
```

означает,что вводится одинаковая дата рождения для всех учеников 4-го класса "А" с именем Ваня.

Константу * нельзя использовать в качестве значения ключа записи, т.е. не допускаются циклы по всем имеющимся запи-

сям данного комплекта. Так, например, при помощи оператора

```
## D КЛАСС.УЧЕНИК (4А * *)
```

можно удалить все данные о всех учениках 4-го класса "А", а в то же время оператор

```
## D КЛАСС (*)
```

считается ошибочным.

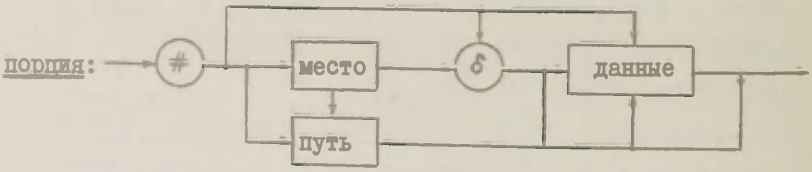
4. Уточнение объекта и экземпляра

Если после инструкции (перед всеми порциями этого оператора) ставится "лишний" разделитель δ, то этим сообщают, что определение места и пути совершено и в порциях следуют только вводимые данные. Например, началу оператора

```
## S КЛАСС.ПРЕДМЕТЫ (4А))
```

могут следовать порции, содержащие лишь последовательности вводимых названий предметов.

Если разделитель δ после инструкции опускается, то в порциях данного оператора кроме самих данных могут быть и указания на продолжение как места, так и пути (данные в порции или же порции в операторе могут отсутствовать только при операции удаления). Таким образом, порция имеет в общем случае следующую структуру:



В порции нельзя менять заданные в инструкции место или путь, но они могут быть продолжены справа. Такой способ естественно использовать в тех случаях, когда порции одного оператора относятся к разным объектам или экземплярам. Например, если желают исправить дату рождения и одну оценку для ученика 4-го класса "А" Вани Иванова, то можно включить в пакет оператор

```
## R КЛАСС.УЧЕНИК (4А ИВАНОВ ВАНЯ)
  # .ДАТА_РОЖД) 140768 # .ОЦЕНКИ (2, 7) 5
```

Отметим, что разделитель δ является признаком начала данных в порции. Он опускается в случае наличия пути в порции (как в последнем примере) и конечно тогда, когда в порциях сразу после разделителя # начинаются данные, т.е. когда разделитель δ имеется уже после инструкции.

5. Представление данных

Данные в порции образуют последовательность конкретных значений, отделенных друг от друга разделителями. Сами значения могут быть числовыми (для атомов типа NAT , INT, CONST, DEC, HEX, DATE, FDATE и REAL) или текстовыми (для атомов типа TEXT). Кроме этого разрешается еще ввести значение * в качестве признака отсутствия конкретного значения (константа * имеет, таким образом, одну семантику в составе пути и другую в составе данных). Ввод значения * недопустим для атома, который является ключом записи или (основного) доступа.

Числовые значения представляются в составе данных в обычном виде, причем для всех типов кроме HEX, DATE и FDATE разрешается и полулогарифмическая форма. Например, целое число 2500000 можно представить также в виде 25E5 или 2.5E6, а число -0.0006 в виде -6E-4.

Если для атома типа NAT, INT или DEC в легенде указана длина в виде двух натуральных чисел, то соответствующие значения вводятся с учетом указанного масштаба. Например, если в легенде имеется атом

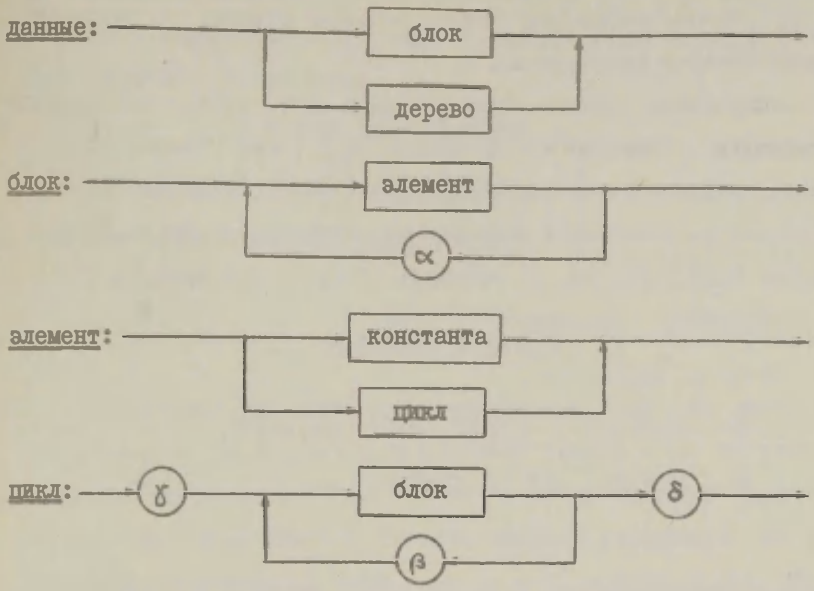
```
* 5  ЦЕНА  NAT  PICT = 5.2
```

то конкретное значение для этого атома следует в составе данных представить в сто раз уменьшенном виде, т.е. вместо цены 1500 (копеек) следует ввести 15 (рублей) или 15.00 или 1.5E1.

Каждое текстовое значение представляется в виде идентификатора (слова) или стринга (последний используется тогда, когда текст содержит разделители). Кроме этого в случае атома строкового типа с неопределенной длиной одним значением считается последовательность символов до первого разделителя # (пользователь должен позаботиться о том, чтобы такое значение было последним в порции).

6. Структура данных

Последовательность значений в одной порции должна иметь строго определенную структуру. Использование разделителей в этой последовательности регламентируется следующим образом:



Данные в виде дерева потребуются только тогда, когда определяемый конструкцией "место" объект является корнем повторяющейся группы со списочной организацией. В случае всех других объектов данные должны иметь вид блока, причем для каждого его элемента по легенде однозначно можно определить соответствующую вершину. Если эта вершина является (неповторяющимся) атомом, то элемент должен быть константой подходящего типа (константа * означает отсутствие значения любого типа).

Если элемент данных соответствует корню группы, то он может быть только циклом. В случае неповторяющейся группы цикл содержит ровно один блок. В случае же повторяющейся группы блоков может быть больше - каждый из них соответст-

вует одному экземпляру группы. Таким образом, β является разделителем экземпляров.

Все блоки одного цикла должны быть точно одинаковой структуры. В каждом из них должно быть ровно столько элементов, сколько соответствующая группа легенды содержит вершин-братьев, не являющихся указателями, индексными или псевдоатомами. Таким образом, α является разделителем братьев.

Например, оператором

```
## A КЛАСС (4A)
  # .КЛАССРУК ) ПЕТРОВ ИВАН ИВАНОВИЧ
  # .УЧЕНИК ) (ПЕТЯ СИДОРОВ * (*) ;
                ВАНЯ ИВАНОВ 140768 (*))
```

в запись 4-го класса "А" добавляются данные о классном руководителе и двух новых учениках. Группа КЛАССРУК содержит по легенде три вершины, следовательно, блок данных в первой порции должен состоять ровно из трех текстовых значений. Повторяющаяся группа УЧЕНИК содержит четыре вводимых вершины (СР_ОЦ является псевдоатомом), следовательно, каждый блок имеющегося во второй порции цикла должен состоять из четырех элементов - два текстовые значения, дата и цикл для вершины ОЦЕНКИ. В этом примере дата рождения Сидорова не известна, а оценки для новых учеников еще не вводятся. Последнее обстоятельство кратко обозначается символами (*), что означает отсутствие значения для всех экземпляров повторяющейся группы.

Если место определено в виде перечисления вершин-братьев, то соответствующий блок данных должен содержать ровно столь-

ко элементов, сколько братьев перечислено. При этом совпадение типов должно быть соблюдено в порядке этого перечисления, как, например, в операторе

```
## R КЛАСС.УЧЕНИК.ДАТА_РОЖД+ИМЯ (4A)
  # (ИВАНОВ ВАНЯ) 140768 ИВАН
  # (ИВАНОВА ВАЛЯ) 230868 ГАЛЯ
```

7. Семантика операций

Приведенные выше названия операций не дают еще полного представления об их содержании хотя-бы потому, что не уточнены условия осуществимости. Осуществимость операций зависит от наличия или отсутствия соответствующего экземпляра (т.е. экземпляра с заданными значениями ключей) в базе данных. Если обновляемым объектом является атом, то он считается отсутствующим тогда, когда нет содержащего его экземпляра или же значением данного атома является признак отсутствия значения (физически такой признак изображается единицами во всех битах соответствующего поля).

Добавление (код операции A) означает ввод такого экземпляра (значения), которого еще нет в базе данных. При наличии экземпляра в базе операция не выполняется, соответствующая часть оператора пропускается и на АЦПУ выдается сообщение об ошибке (это не ошибка по синтаксису языка DIL, а по состоянию базы данных!).

Замена (R) и удаление (D) выполнимы лишь при наличии соответствующего экземпляра (значения) в базе данных. В случае

отсутствия выдается сообщение об ошибке.

Включение (S) выполнимо как при наличии, так и отсутствии экземпляра. В случае отсутствия происходит добавление нового экземпляра, а в случае наличия замена старого экземпляра новым (и выдача сообщения об удалении старого экземпляра).

Например, если во время выполнения оператора

```
# #  A  КЛАСС.КЛАССРУК (4A)
   #  ) ПЕТРОВ ИВАН ИВАНОВИЧ
```

в записи для 4-го класса "А" уже имеются все данные о классном руководителе, то в записи ничего не меняется и выдается только сообщение об ошибке. Если же хоть один из этих трех атомов не имеет значения, то на место признака отсутствия значения пишется соответствующий элемент данных. Например, если ранее были введены значения СИДОРОВ * ПЕТРОВИЧ, то после выполнения приведенного оператора в атомах группы КЛАССРУК получаем значения СИДОРОВ ИВАН ПЕТРОВИЧ (одновременно выдается сообщение о том, что не все элементы данных удалось использовать).

Если в рассмотренном операторе код операции A заменить на R, то в последнем случае атомы классного руководителя получают значения ПЕТРОВ * ИВАНОВИЧ и выдается сообщение об отсутствии имени. Если же код операции заменить на S, то независимо от прежних значений получаем ПЕТРОВ ИВАН ИВАНОВИЧ.

Если во время выполнения рассмотренного оператора (с любым кодом операции) еще нет записи для 4-го класса "А", то выдается лишь сообщение об ошибке. Это вытекает из общего

правила, по которому путь не является носителем вводимой информации: запись для 4-го класса "А" (в которой имелись бы только номер класса и данные о классном руководителе) не открывается потому, что значение ключевого атома НОМЕР указано в пути, а не среди данных.

Когда в пути заданы значения локальных ключей, то данные могут содержать только один экземпляр (блок). Это даже в том случае, когда значением локального ключа является *. Например, оператор

```
# #  S  КЛАСС.ОЦЕНКИ (4А ИВАНОВ ВАНЯ  *  3)
   #  ) 5
```

вводит для Вани Иванова оценку 5 по третьему предмету на все четыре семестра.

В случае повторяющейся группы без явного доступа (или когда единственным ключом является атом типа INDEX) семантика операции добавления зависит от наличия или отсутствия в пути значения для локального ключа (индекса). Если локальный ключ задан, то новый экземпляр добавляется за указанным. Если же ключ не задан, то новый экземпляр добавляется в качестве последнего. Например, оператор

```
# #  A  КЛАСС.ПРЕДМЕТЫ (4А)
   # (0) РУССКИЙ # ) МАТЕМ
```

вводит название предмета РУССКИЙ в качестве первого, а МАТЕМ в качестве последнего.

8. Правила упрощения

В целях сокращений, а также большей наглядности представления данных при оформлении пакетов допускаются некоторые упрощения, не указанные в синтаксисе языка DIL.

Если данные в порции представлены блоком, единственный элемент которого является циклом, то разделители γ и δ вокруг данных можно опускать. Например, часть одного из приведенных выше примеров можно переписать так:

```
## A  КЛАСС.УЧЕНИК (4A)
 #  ) ПЕТЯ СИДОРОВ  *  (*);
      ВАНЯ ИВАНОВ  140768  (*)
```

Разрешается также опускать разделители γ и δ вокруг конструкции "путь". Однако, если эти разделители опущены в порции, то перед данными нельзя опускать разделитель δ. Например, допустимым считается следующий оператор:

```
## R  КЛАСС.УЧЕНИК 1Б
 #  .ДАТА_РОЖД МЫШКИНА МАША ) 290272
 #  ) ПЕТЯ ПЕТРОВ  301071  (*)
```

Если в качестве разделителя α используется пробел, то рядом с другими разделителями его можно и не писать. В то же время вместо одного пробела можно писать несколько. Лишние пробелы можно вообще писать рядом с любым разделителем. Если α не пробел, то последовательность $\alpha\alpha$ интерпретируется как $\alpha*\alpha$ (это позволяет упрощать ввод скудных данных).

Отсутствие всех экземпляров в цикле можно вместо $\gamma*\delta$

обозначать и через $\gamma\delta$. Например, последнюю порцию можно представить в виде

```
# ) ПЕТЯ ПЕТРОВ 301071 ( )
```

В случае множественного атома (в том числе одноатомного массива) вместо разделителя экземпляров β разрешается использовать α . Если множественный атом при этом окажется единственным элементом блока, то вокруг него можно опускать разделители γ и δ , т. е. α окажется теперь разделителем экземпляров для внутреннего, а β для внешнего цикла.

Разделитель β между экземплярами можно опускать в таком случае, когда экземпляры окажутся циклами и вокруг них имеются разделители γ и δ.

Последние правила упрощения используются главным образом для ввода массивов. Многомерный массив истолковывается при этом как иерархия одномерных массивов. Например, в рассматриваемой нами легенде вершина ОЦЕНКИ интерпретируется при выполнении пакета как дерево

```
* 2 ОЦЕНКИ ARRAY [4]
  * 3 ОЦ_СЕМ ARRAY [ КОЛ_ПРЕД ] NAT MAX = 5
```

(только имя ОЦ_СЕМ недоступно).

Допустим, что количество предметов в 1-ом классе "Б" равно трем. Тогда для включения в базу данных всех оценок ученика Пети Петрова можно составить оператор:

```
## S КЛАСС.ОЦЕНКИ 1Б ПЕТРОВ ПЕТЯ)
  # 4 4 3 ; 4 5 4 ; 4 4 4 ; 4 5 3
```

Отметим, что без использования правил упрощения этот опера-

тор должен выглядеть так:

```
## S КЛАСС.ОЦЕНКИ (1Б ПЕТРОВ ПЕТЯ))
  # ((4 ; 4 ; 3);(4 ; 5 ; 4);(4 ; 4 ; 4);(4 ; 5 ; 3))
```

Следует также учитывать, что последние упрощения относятся только к одноатомному массиву. Например, если в нашей легенде вершину ОЦЕНКИ временно заменить вершинами

```
* 2 ОЦЕНКИ ARRAY [ 4, КОЛ_ПРЕД ] NAT
  * 3 ОЦЕНКА MAX = 5
  * 3 ПРОПУСК MAX = 50
```

то порцию в последнем операторе следует задавать в виде

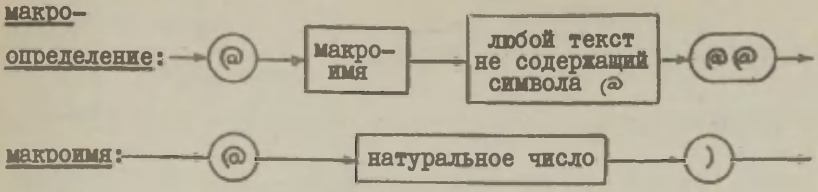
```
# (4 1 ; 4 0 ; 3 5)(4 2 ; 5 0 ; 4 0)
  (4 * ; 4 1 ; 4 0)(4 1 ; 5 0 ; 3 7)
```

Для частичного ввода данных в массив в пути задаются локальные ключи (индексы). Например, если после инструкции

```
## A КЛАСС.ОЦЕНКИ 1Б ОЛЯ ОБРАЗЦОВА
```

(вернемся опять к первоначальной легенде!) следует порция `# 1 ) 5 5 5` , то вводятся пятерки по всем предметам для первого семестра. Тот же результат дает, кстати, порция `# 1 * ) 5` , а порцией `# * * ) 5` вводятся пятерки для всех предметов и семестров.

Иногда возникает необходимость использовать некоторую последовательность символов в пакете несколько раз. В таком случае эту последовательность на месте первого появления можно оформить в виде макроопределения и заменить ее следующие вхождения соответствующим макроименем:



При выполнении пакета макроопределение заменяется его текстом, а имя и текст запоминаются. При последующих появлениях того же макроимени оно заменяется тем же текстом. Например, отрывок пакета

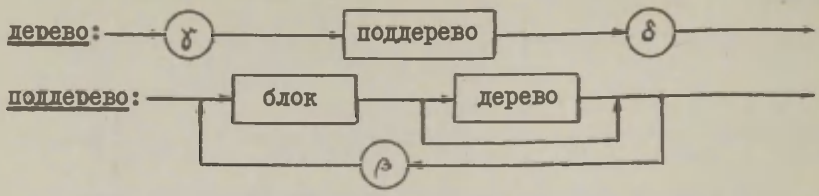
```
## R  @ @1) КЛАСС.УЧЕНИК (4А ИВАНОВ ВАНЯ) @@
   #  .ДАТА_РОЖД )  140768
## A  @1) # .ОЦЕНКИ  (1  3)  4
```

заменяет дату рождения и добавляет одну оценку для ученика 4-го класса "А" Вани Иванова.

Макроопределение остается в силе только в пределах одного пакета. Запрещается включить в текст макроопределения последовательность символов ## (признак начала оператора).

9. Списочные структуры

Ввести данные в повторяющуюся группу со списочной организацией можно только конструкцией "дерево", причем обновляемым объектом или его единственным подчиненным должен быть корень этой группы или указатель в ней (если объект определяет больше вершин, то в составе блока корню такой группы может соответствовать лишь $\gamma\delta$ или $\gamma*\delta$, как признак отсутствия всех экземпляров):



Блок в этой конструкции соответствует экземпляру группы, причем все блоки одного дерева должны быть точно одинаковой структуры. Можно интерпретировать и так, что разделитель γ обозначает движение по дереву вниз, δ – вверх, а β – направо.

В приводимых примерах будем опираться на следующую простую легенду:

```
LEG  ЛЮДИ  KEY = НОМЕР
  * 1  НОМЕР  INDEX
  * 1  ПЛЕМЯ  LIST  HASH  KEY = ИМЯ
    * 2  ИМЯ  TEXT  PICT = 8
    * 2  СЫН  DPOINT
    * 2  БРАТ  RPOINT
END
```

В случае этой легенды экземпляр группы ПЛЕМЯ состоит только из атома ИМЯ, так как указатели СЫН и БРАТ являются невводимыми. Например, для ввода в качестве третьей записи следующей структуры

требуется оператор

```
## А ЛЮДИ (2))
  # (НОЙ (СЭМ (АРАМ); ХАМ; ИАФЕТ (ИАВАН; ГОМЕР)))
```

Учитывая описанные выше правила упрощения, последнюю порцию можно представить и в виде:

```
  # НОЙ (СЭМ (АРАМ) ХАМ ИАФЕТ (ИАВАН ГОМЕР))
```

Имя корня (ПЛЕМЯ) применяется для выделения всех экземпляров или же только первого экземпляра повторяющейся группы (хотя в данном случае для всех экземпляров можно использовать и имя легенды, не имеющее больше вводимых подчиненных). Для обозначения конкретного экземпляра или подсписка используются имена указателей (СЫН и БРАТ), так как каждый экземпляр кроме первого является или сыном или братом какого-то другого экземпляра. Например, оператором

```
## А ЛЮДИ (3)
  # .СЫН (ИАФЕТ) ФИРАС
  # .БРАТ (АРАМ) ЛУД
```

прибавляется Фирас в качестве первого сына Иафета и Луд в качестве второго сына Сэма.

При обновлении экземпляра полностью обновляется и подчиненное ему поддерево. Например, оператор

```
## R ЛЮДИ.СЫН (3 НОЙ)) # СИМ
```

заменяет старшего сына Ноя на Сим, у которого уже нет сыновей. Чтобы сохранить последние, следовало заменить не экземпляр, а атом:

```
## R ЛЮДИ.СЫН (З НОЙ) # .ИМЯ ) СИМ
```

Следует учитывать, что если конструкцией "место" дается имя указателя, то данные должны быть также представлены в виде дерева. В этом смысле имя указателя эквивалентно имени корня той же группы. Например, в результате выполнения оператора

```
## R ЛЮДИ (З)
  # .БРАТ (ИАВАН) МАДАЙ; ГОМЕР (РИФАТ; АСКЕНАЗ)
```

получается следующая структура для потомков Иафета:

Если для повторяющейся группы со списочной организацией не указан доступ, то в качестве индекса служит порядковый номер экземпляра при прохождении дерева в прямом порядке.

ЯЗЫК МАНИПУЛИРОВАНИЯ ДАННЫМИ

Ю. Каазик, А. Рауп

В этой статье кратко описывается язык DML (=Data Manipulation Language) для манипулирования данными в системе РАМА. Язык DML не предназначен для программирования вычислительных алгоритмов – такого рода программы получаются на этом языке сравнительно неуклюжими. Целью являлось создание языка, в котором по возможности просто можно было бы запрограммировать наиболее типичные действия над базой данных. К таким действиям относятся, в первую очередь, поиск, просмотр и пересылка данных (из одного файла в другую), причем роль вычислительных процедур, и вообще арифметических операций, незначительна.

1. Возможности DML-программы

Программы, написанные на языке DML (или DML-программы) предназначены для выполнения функций получения справок и обновления данных. Получение справки состоит из поиска и извлечения некоторой части базы данных (обычно вместе с выдачей результатов в отпечатанном виде). Под обновлением базы

данных здесь понимается изменение конкретных значений, введение новой информации и перемещение данных из одной части базы в другую. Обновлением можно считать и создание нового файла (обновление пустого файла).

Наибольшей единицей данных для DML-программы является комплект записей (в том же смысле, что и при описании файлов). Это значит, что в программе не надо указывать, с какими файлами ведется работа, достаточно лишь сообщить, какие комплекты записей будут использованы. Описания файлов (написанные на языке FDL) будут использованы не во время трансляции DML-программы, а только при выполнении полученной программы, когда по описаниям файлов определяют правила формирования ключей доступа для необходимых комплектов записей.

Из-за такой сравнительной независимости DML-программ от описаний файлов эти программы не нуждаются в перетрансляции, когда в файлы добавляются новые комплекты или удаляются такие, которые не касаются данной программы. Одну и ту же программу можно использовать для обработки файлов разной структуры. Программа требует перетрансляции (и, как правило, корректировки) только тогда, когда изменяются легенды записей.

Несмотря на сказанное, обычно все-таки полезно при написании DML-программы учитывать описания файлов. Так, например, составляя циклы по записям, мы получим хорошую программу лишь с учетом иерархии комплектов, хотя программа пригодна (но работает медленно) и в том случае, когда менять иерархию используемых комплектов или даже поместить эти комплекты в разные файлы.

Как для обновления, так и для получения справок из комплектов приходится выделить отдельные записи, а внутри записей отдельные экземпляры повторяющихся групп. Фиксируя один экземпляр некоторой группы, получаем взаимно-однозначное соответствие между вершинами описания группы (в легенде) и вершинами физического представления этого экземпляра. Таким образом, если с какой-то вершиной экземпляра надо что-то предпринять, то можно воспользоваться именем этой вершины в легенде.

Корневую группу записи (или группу нулевого уровня) можно рассматривать как повторяющуюся группу, каждый экземпляр которой находится в своей записи, причем корнем группы является имя комплекта. Итак, выделение одной записи из комплекта означает фиксирование одного экземпляра ее корневой группы. Имея запись, можно перейти к непосредственным группам, выделяя в них также конкретные экземпляры. Имея экземпляр какой-то группы, всегда должны быть однозначно определены и экземпляры всех вышестоящих (родительских) групп, до корневой группы включительно.

Переход от вышестоящей группы к подчиненной происходит по (основному или дополнительному) доступу: для фиксирования экземпляра достаточно указать значения соответствующих ключей. Повторяющаяся группа без явного доступа ключей не имеет, но вместо них можно использовать порядковый номер экземпляра, который в дальнейшем назовем индексом экземпляра.

Следует учитывать, что индекс и ключ имеют одну существенную разницу. Значение ключа содержится в экземпляре и по нему этот экземпляр всегда можно найти, независимо от того,

какие изменения происходят с другими экземплярами данной группы. Значение же индекса характеризует только расположение экземпляра в данный момент по отношению к другим экземплярам группы. Например, если какой-то экземпляр удаляется, то значения индексов всех задних экземпляров уменьшаются. Если индекс требуемого экземпляра группы неизвестен, то данные можно найти только путем просмотра всех экземпляров группы.

Когда к группе обращаются по дополнительному доступу, то одному значению ключа этого доступа может соответствовать больше чем один экземпляр группы. Нужный экземпляр среди них можно найти только их просмотром.

Для обращения к комплектам записей они связываются с переменными особого типа, называемыми именами комплектов. Основным назначением такого имени является запоминание текущего положения, т.е. значений ключей и индексов для подлежащих обработке экземпляров повторяющихся групп. Имя комплекта можно представить как переменную, имеющую структуру дерева, содержащего на месте каждой повторяющейся группы записи неповторяющуюся группу, атомами которой являются либо ключи основного доступа, либо атом типа NAT, содержащий индекс экземпляра группы без доступа, либо несколько атомов типа NAT, соответствующих индексам массива.

Каждому имени комплекта отводится свой буфер ввода-вывода и своя рабочая область. В буфере имеется место для одной (текущей) записи, а в рабочей области - место для одного (текущего) экземпляра из каждой повторяющейся группы записи.

Обычно для одного комплекта записей требуется только одно имя, но в некоторых случаях этого недостаточно. Например, если необходимо попарно сравнить все записи одного комплекта (или экземпляры какой-то группы из одной записи), то придется все время запоминать больше одного текущего положения для комплекта записей. Имя комплекта можно также интерпретировать как синоним имени легенды. В DML-программе все имена вершин из легенд представляются при помощи составного имени, первая компонента которого - имя комплекта. Это позволяет каждый раз определить, вершина какой записи имеется в виду. Необходимость такого именования станет очевидным, если учесть, что в программе может быть использовано несколько комплектов записей одинаковой структуры или же записей разной структуры, но имеющих совпадающие имена вершин.

Для получения справки или обновления необходимо локализовать нужное место. В языке DML это осуществляется операторами цикла и обновления, которые реализуют ввод в буфер и вывод из буфера записей, а также перемещения данных между рабочей областью и буфером.

Выделение конкретного экземпляра группы означает "проложение пути" в эту группу. Для этого надо по очереди зафиксировать экземпляры всех вышестоящих групп. Отцом корневой группы является имя комплекта и локализация всегда начинается выделением конкретной записи. Эта запись вводится в буфер и ветка корневой группы перемещается в рабочую область. При продолжении локализации в рабочую область вводят ветки экземпляров групп все более низких уровней, а значения ключей и индексов запоминаются для данного имени комплекта.

Когда конкретный экземпляр группы выделен, то говорят, что проложен путь в эту группу. Путь всегда можно продолжить в группы более низкого уровня.

Возможности локализации при получении справки и обновлении несколько различны. Для получения справки обычно даются условия, которым должны удовлетворять значения ключей либо основного, либо дополнительного доступа. Осуществляется это при помощи оператора цикла, на каждом шагу которого локализуется очередной (в смысле удовлетворения условиям отбора) экземпляр группы. Выделение только одного конкретного экземпляра является частным случаем цикла, состоящим из одного шага.

При обновлении локализация может происходить только по значениям ключей основного доступа или индекса (если доступ не указан). Обновление состоит всегда в локализации только одного конкретного экземпляра, несмотря на то, имеется ли дело с добавлением, заменой или удалением. В случае удаления необходимо лишь отыскать нужный экземпляр и сообщить, что нужно его исключить.

Замена экземпляра состоит в том, что сперва его находят и перемещают в рабочую область. Там атомам ветки можно присвоить новые значения, после чего ветка помещается обратно в буфер.

Добавить новый экземпляр можно только, когда в базе данных до тех пор отсутствовал экземпляр с такими значениями ключей. Добавление в группу без явного доступа означает либо вставку нового экземпляра за каким-то конкретным экземпляром (в том числе за "нулевым"), либо включение нового экземпляра

в качестве последнего. Для добавления в рабочей области создадут "пустой" макет ветки, содержащий только значения ключей. Другим атомам ветки теперь можно присвоить значения и переместить полученный новый экземпляр из рабочей области в буфер.

Когда обрабатывается некоторая группа, ее экземпляры можно разделить на две части: новые экземпляры, включенные в группу добавлением или заменой, и старые экземпляры, не подвергавшиеся до сих пор обновлению. Старые экземпляры (вернее, ссылки на них) располагаются по порядку, определенному типом доступа. Новые экземпляры помещаются не сразу на свое место (определенное предварительной локализацией обновляемого места), а в конец набора экземпляров. При обновлении локализация происходит всегда по всем экземплярам, новым и старым. При выполнении же оператора цикла просматривают только старые экземпляры, так как просмотр должен протекать по порядку доступа, а новые экземпляры еще не установились на свои места. Это обстоятельство надо учесть при одновременном получении справок и обновлении.

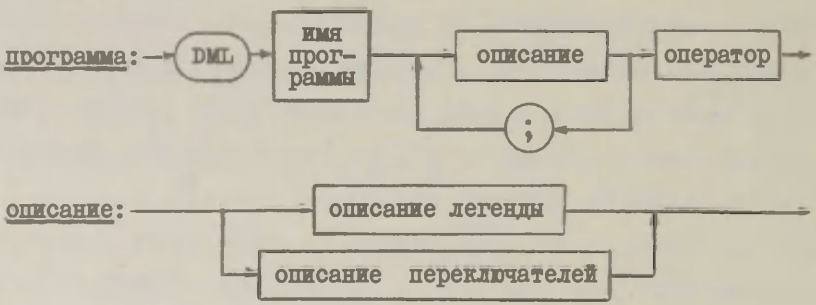
Включение нового экземпляра группы в запись вызывает упорядочение экземпляров всех подчиненных ей групп. Таким образом, набор экземпляров обрабатываемой нами группы упорядочивается только при получении приказа об обновлении экземпляра в одном из родительских групп.

Обновление записи кончается приказом обновить корневую группу. После этого в буфере полностью упорядоченная запись, выводимая из буфера и добавляемая к комплекту записей, когда в буфер надо ввести новую запись или образовать пустой макет

корневой группы. Ввиду того, что записи являются также экземплярами повторяющейся группы, новая запись помещается не на свое место, а в "конец" комплекта, точнее - в специальный системный файл новых записей всех типов, называемый "коллектором". Приказ упорядочить комплект записей дается не в DML-программе, а особым заказом.

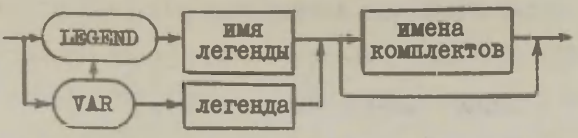
2. Описания

Каждая DML-программа состоит из двух частей. В начале программы излагаются описания, которые определяют структуру обрабатываемых данных, а за описаниями следует оператор (обычно составной оператор), содержащий указания о самом манипулировании данными:



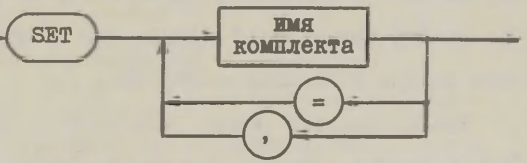
Описаниями легенд сообщаются структуры обрабатываемых записей, а переключатели - это в сущности параметры программы, которым можно присвоить значения в соответствующем заказе. Любая DML-программа должна содержать по меньшей мере одно описание легенды, но описания переключателей могут отсутствовать.

описание легенды:



Описание легенды обычно начинается с указания имени такой легенды, которой соответствуют записи в базе данных. Однако, можно описать и рабочие области программы, не связанные ни с какой структурой записей в базе данных. Это осуществляется при помощи описания, начинающегося ключевым словом VAR, за которым следует либо целая легенда на языке RDL, либо указание имени какой-то уже каталогизированной легенды. Такая рабочая область программы называется рабочей записью, чтобы не путать ее с рабочими областями имен комплектов. Рабочая запись похожа на комплект записей базы данных, но имеется она всегда только в одном экземпляре, который все время находится во внутренней памяти (можно считать, что в буфере) и в начале работы DML-программы пуст.

имена комплектов:



Для любой легенды можно назначить одно или несколько имен комплектов. Если в описании легенды имена комплектов вообще отсутствуют, то это указывает, что назначается только одно имя, совпадающее с именем легенды. Следовательно, два таких описания равнозначны:

```
LEGEND  ФИРМА

LEGEND  ФИРМА  SET  ФИРМА
```

но следующие два разные, так как первое определяет два имени, а второе - только одно:

```
LEGEND  ФИРМА  SET  Ф1, ФИРМА

LEGEND  ФИРМА  SET  Ф1
```

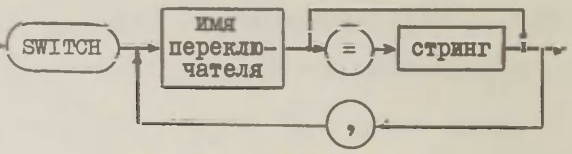
Кроме своей рабочей области каждому имени комплекта отводится и свой буфер ввода-вывода. При этом, если разные имена ссылают на одну и ту-же конкретную запись, то она находится в буферах всех этих имен. Когда эта запись обновляется по одному имени, во внутренней памяти возникают экземпляры одной записи разного содержания. Во избежание такого положения для нескольких имен комплекта можно назначить один общий буфер. Указанием этого служит разделение таких имен в описании имен комплекта не запятой, а знаком равенства. Например, описание легенды

```
LEGEND  ФИРМА  SET  Ф1 = Ф2 = Ф3, Ф4
```

определяет четыре имени для записей по легенде ФИРМА. Первые три из них имеют общий буфер, но у четвертого имени его записи всегда в отдельном буфере.

Несколько имен можно приписать и любой рабочей записи. Это означает просто создание нескольких буферов с такой структурой. Использование знака равенства указывает на намерение связать несколько имен комплектов с одной рабочей записью.

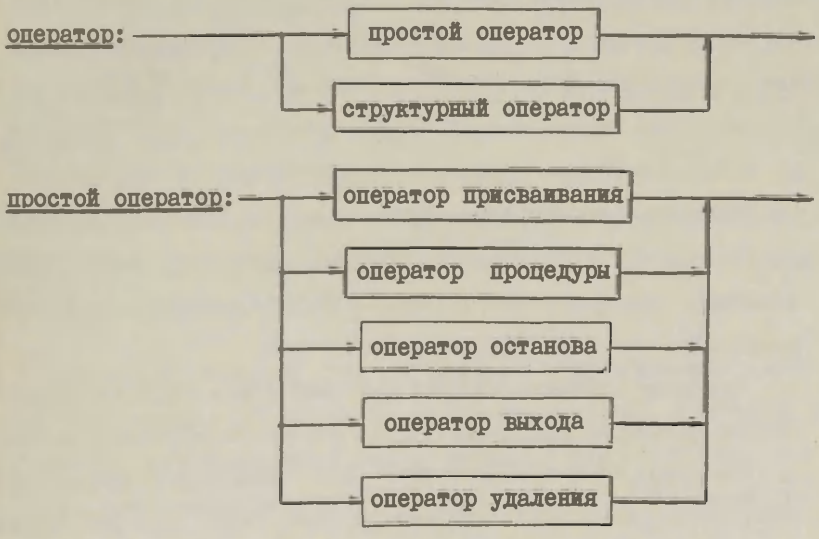
описание переключателей:

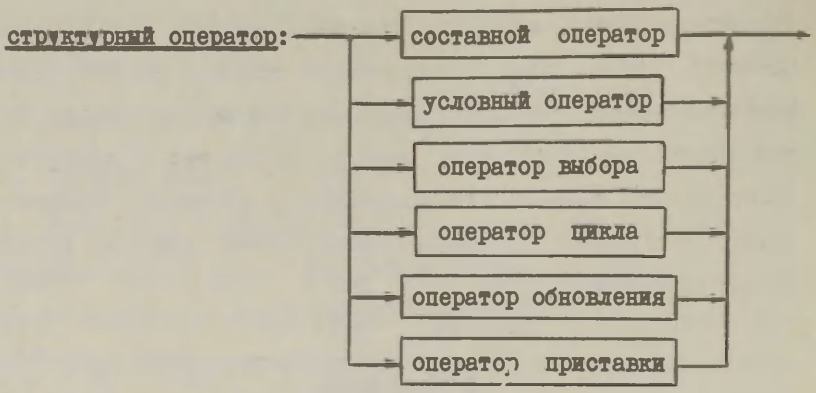


Переключатель можно рассматривать как атом типа TEXT. Значение такому атому присвоится в заказе, но в описании можно установить стандартное значение для переключателя, которое будет ему присвоено, когда в заказе это не делается. Переключатели являются необязательными параметрами DML-программы, при помощи которых в заказе можно указать режимы работы программы.

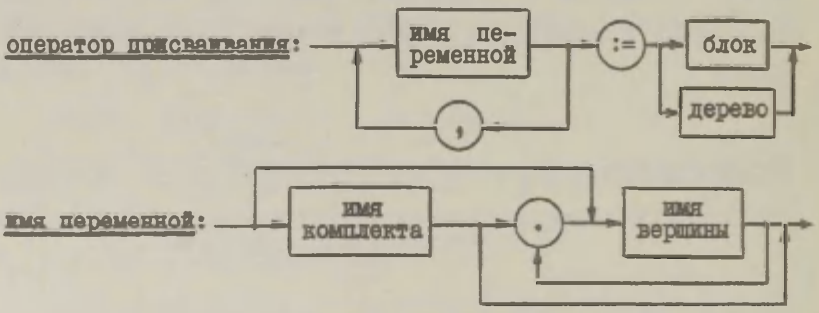
3. Операторы

Основные элементы программы - это операторы, которые условно можно разделить на две группы - простые и структурные. Простыми являются те операторы, которые не содержат других операторов, у структурных же в их составе всегда другие операторы.





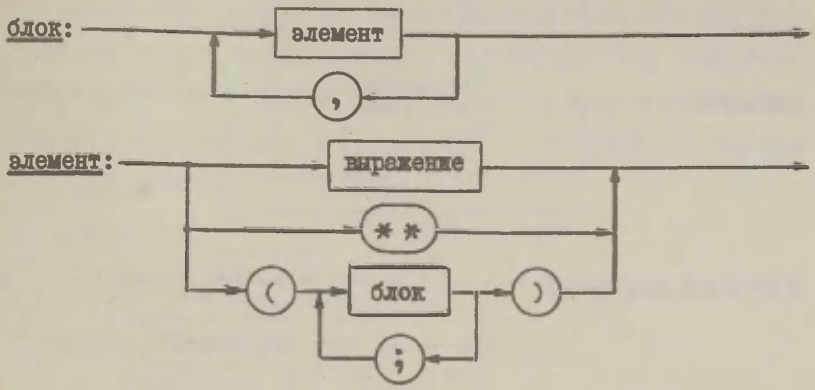
В этой главе приводятся описания всех операторов, кроме операторов выхода, удаления, цикла и обновления. Они будут рассмотрены в двух последующих главах.



Оператор присваивания служит для изменения значений вершин в рабочей области (эти вершины и называются здесь переменными). Имя комплекта в начале имени переменной можно опускать лишь в составе оператора приставки.

Оператор заменяет текущие значения всех перечисленных в начале оператора переменных новым значением, задаваемым блоком или деревом в правой части. Все переменные должны при этом иметь одинаковую структуру, т.е. быть атомами одного

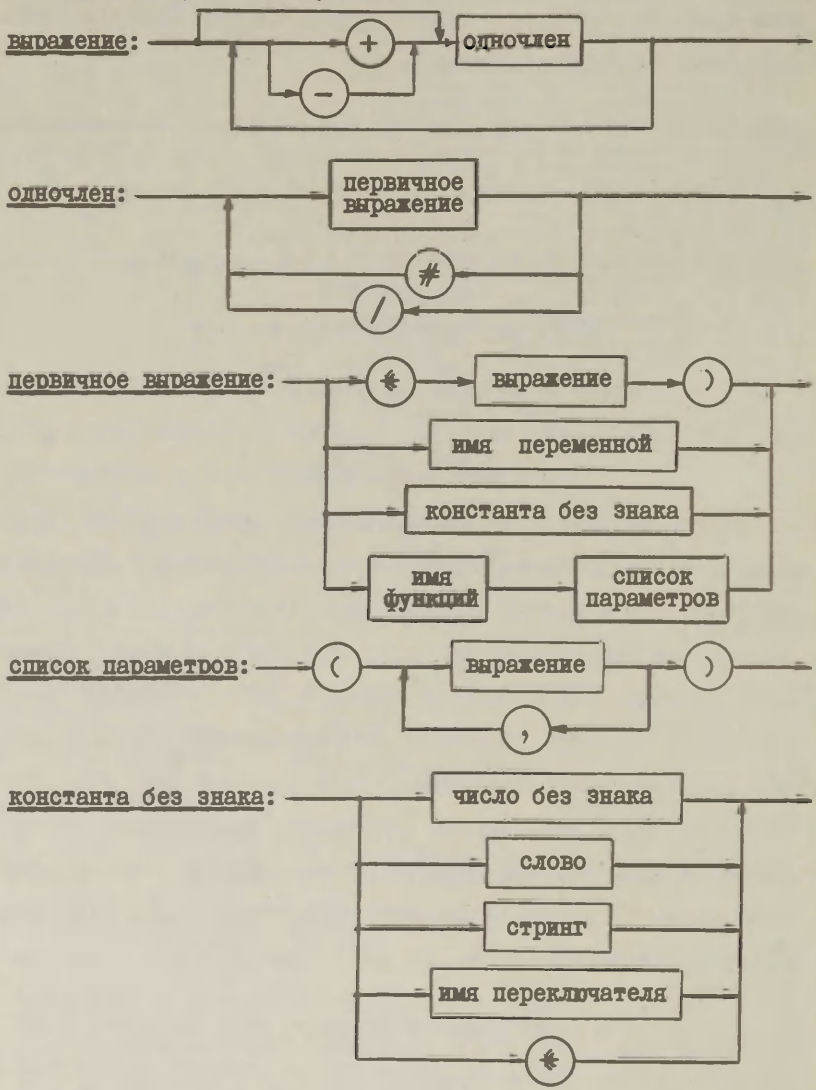
типа или корнями групп одинаковой структуры. Этой структуре должен соответствовать и блок в правой части (случай дерева рассмотрим несколько позже).



Каждый элемент блока соответствует одной вершине: выражение или две звездочки атому, а последовательность блоков в скобках - корню группы (элемент * * означает, что атому оставляют его старое значение). Отметим, что оператор присваивания применим для изменения значений всех переменных, кроме атомов типа INDEX, DPOINT и RPOINT, которые получают свои значения системными средствами. Использование имен этих вершин в качестве левой части оператора присваивания запрещается, а соответствующие им элементы в правой части должны быть опущены. Присваивание атому типа CONST допустимо лишь тогда, когда этот атом еще не имеет значения.

Перед присвоением находят значения всех выражений в блоке. Они должны согласоваться с типами атомов, описанных в легенде. Из разделителей скобки выделяют группы, точки с запятой экземпляры, а запятые - атомы. Скобки вокруг всей пра-

вой части могут быть опущены.



Символы # и / соответственно являются знаками умножения и деления. Отсутствие значения для атомов всех типов обозна-

чается звездочкой. Для указания отсутствия всех последних данных неповторяющейся группы или экземпляра повторяющейся группы достаточно одной звездочки.

Правая часть оператора состоит из одного выражения, когда присваивается значение атому. Если же вершина, соответствующая переменной, имеет подчиненные, то правая часть должна содержать значения для всех их. Пусть, например, в легенде следующий отрывок:

```
* 3 A
  * 4 Б  TEXT
  * 4 В  NAT
    * 5 Г
    * 5 Д
  * 4 Е  NAT  PSEUDO
  * 4 Ж  INDEX
  * 4 З  TEXT
```

а для нее прикреплено имя комплекта Д. Тогда оператор

```
Д.А :='А + С', (З, * *), 7 + Д.Д, *
```

выполняет ту же работу, что и последовательность операторов

```
Д.Б :='А + С';
Д.Г := З;
Д.Е := 7 + Д.Д;
Д.З := *
```

Если неизмененными хотят оставить все последние подчине-

нные некоторой вершины, тогда в конце блока две звездочки достаточно указать только раз. Например, корректным является оператор

```
Д.А :='А + С', (* *), 7 # Д.Д, * *
```

Рассмотрим теперь присвоение значения вершине, среди подчиненных которой есть повторяющаяся группа. Значения для тех атомов, которые входят в ветку рассматриваемой вершины, просто переносят из правой части оператора в рабочую область. Но ветка в рабочей области содержит также ссылку на подчиненную группу. Экземпляр этой группы можно оставить прежними (указывая это двумя звездочками), но можно и полностью заменить весь набор экземпляров подчиненной группы новым набором из правой части оператора. При выполнении оператора присваивания этот набор перемещается в буфер, а в рабочей области старая ссылка заменяется ссылкой на новый набор (если надо сохранить некоторые экземпляры подчиненной группы и добавить к ним новые, то для этого придется опускаться обработкой на уровень ниже и использовать оператор обновления).

Например, если в рассмотренный выше отрывок легенды внести изменение:

```
* 3 А
  * 4 Б  TEXT
  * 4 В  REP  NAT
    * 5 Г
    * 5 Д
  * 4 Е  NAT
  * 4 Ж  TEXT
```

то оператор

Д.А :='A + C', (* *), 3, DE7

присвоит значения только атомам ветки, а оператор

Д.А :='A + C', (3,5; 2,9; 8, *), 3, DE7

заменяет и весь набор экземпляров группы В (в рабочей области получаем ссылку на группу, состоящую из указанных трех экземпляров).

Обратим внимание еще на то, что запрещено удаление всего набора экземпляров группы при помощи одной звездочки. Некорректен, например, оператор

Д.А :='A + C', (*), 4, DEF

Правая часть оператора присваивания может содержать блок любой сложности. Например, если отрывок легенды имеет вид

```
* 3 A
  * 4 E  TEXT
  * 4 B  REP  NAT
    * 5 Г
    * 5 Д  REP
      * 6 E
      * 6 Ж
```

то можно использовать оператор присваивания

Д.А := A23, (3, (4,2; 5,3); 4, (*,5; 6,*; 7,3))

Если переменная в числе своих подчиненных имеет корень повторяющейся группы со списочной организацией, то соответс-

твующим элементом в блоке могут быть только две звездочки в скобках. Так, например, если в легенде отрывок:

```
* 3 A
  * 4 Б TEXT
  * 4 В LIST SORT KEY = Д
    * 5 Г TEXT PICT = 2
    * 5 Д NAT
    * 5 Е DPOINT
    * 5 Ж RPOINT
```

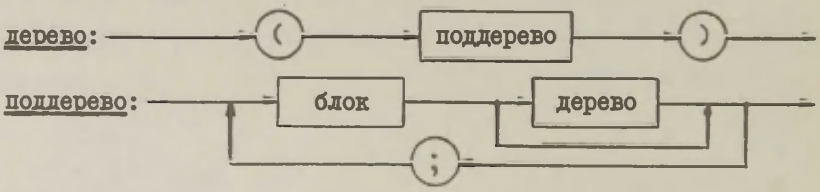
то допускается оператор

```
Д.А := А23, (* *)
```

но не оператор

```
Д.А := А23, (АА, 6)
```

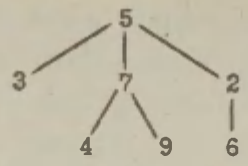
Если переменная в операторе присваивания является корнем списочной структуры, то правая часть должна быть представлена в виде дерева:



Например, корректным является оператор

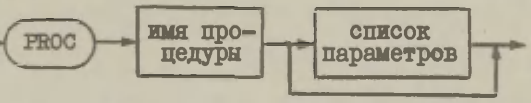
```
Д.В := (А1,5(А2,3; А3,7(А4,4; А5,9); А6,2(А7,6)))
```

который создает повторяющуюся группу из семи экземпляров со следующей структурой (указаны лишь значения ключевого атома Д):

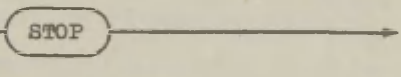


В рабочую область переносят только корневой экземпляр (в котором Д имеет значение 5), все остальные перемещаются в буфер и связываются между собой ссылками.

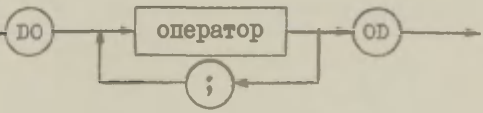
Присвоенные значения сохраняются в рабочей области до выполнения следующего оператора присваивания для тех же атомов, или же до выполнения следующего оператора цикла или обновления для этой группы.

оператор процедуры: 

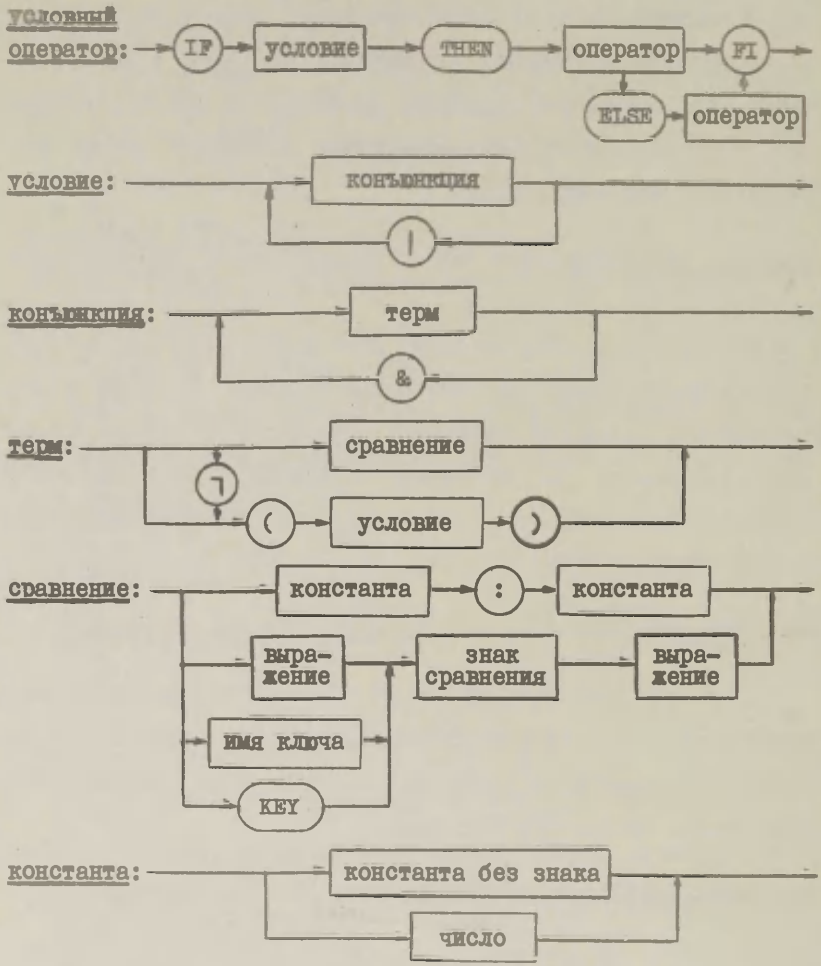
DML-программы сами не могут быть подпрограммами или содержать подпрограмм. Но имеется возможность обращения к находящимся в отдельной библиотеке процедурам (в ней же и функции, упомянутые при определении выражения). Для обращения к процедуре надо лишь сообщить ее имя и при необходимости добавить список параметров.

оператор останова: 

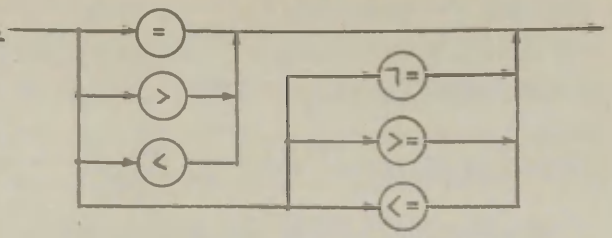
Программа окончит свою работу обычно с окончанием выполнения оператора, следующего за описаниями. Но если работу необходимо кончить где-то в середине программы, то это осуществляется оператором останова.

составной оператор: 

Составной оператор используется для объединения нескольких операторов в один. Это необходимо в таких конструкциях, где по синтаксису языка требуется наличие одного оператора, а выполнить следует несколько: например, DML-программа сама состоит по определению из одного или нескольких описаний и только одного оператора.

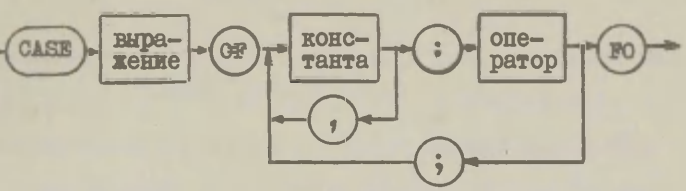


знак сравнения:



При выполнении условного оператора сперва вычисляют значение условия (логического выражения) и если оно истинно, то выполняется оператор за ключевым словом THEN. Далее из условного оператора выходят: управление передается оператору, расположенному за условным. Если значение условия ложно, то выполняют оператор за ключевым словом ELSE и выходят из условного оператора. Если у условного оператора короткая форма, т.е. отсутствуют слово ELSE и оператор за ним, то при ложном значении условия сразу выходят из условного оператора.

оператор выбора:



При выполнении оператора выбора вычисляют значение выражения и выполняют тот из перечисленных операторов, перед которым стоит константа, равная значению выражения. Если такой константы нет, то управление сразу передается следующему оператору. Например, оператор

```
CASE  A.B  OF  -1, 1: A.B := 2;
                0: A.B := 3  END
```

по содержанию равнозначен оператору

```
IF  A.B = -1 | A.B = 1  THEN  A.B := 2
        ELSE  IF  A.B = 0  THEN  A.B := 3  FI  FI
```

Во внутреннем операторе в именах переменных можно опускать то имя комплекта (вместе с точкой за ним), которое упомянуто в начале оператора приставки. Например, вместо оператора

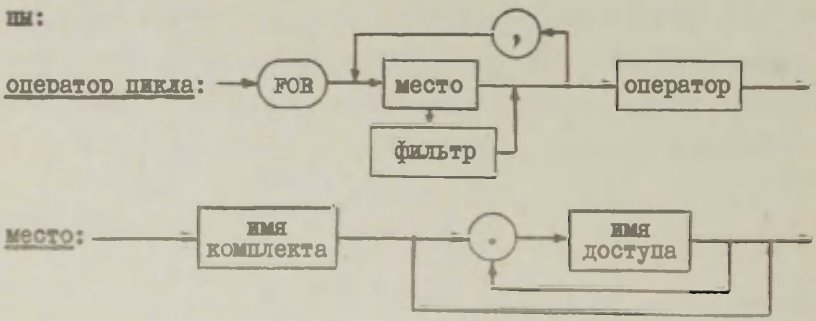
```
IF  X.A > = 3.5  THEN  X.R := X.C + X.D
                  ELSE  X.B := X.C - X.E + Y.F  FI
```

можно писать

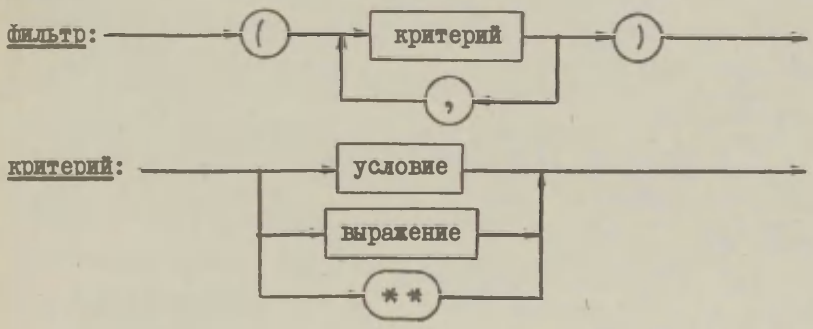
```
WITH  X  IF  A > = 3.5  THEN  B := C + D
                        ELSE  B := C - E + Y.F  FI
```

4. Извлечение данных

Как локализация, так и извлечение данных в DML-программе производится при помощи оператора цикла, предназначенного для ввода заданных экземпляров некоторой повторяющейся группы:



Желаемая группа сообщается в операторе конструкцией "место". Если требуется запись (корневая группа), то достаточно задать только имя комплекта. Для группы более низкого уровня надо указать либо имя ее корня (что означает использование основного доступа), либо имя дополнительного доступа. Если нас интересуют не все экземпляры группы, то для выделения нам нужных можно предъявить ограничения с помощью фильтра:



При первом выполнении оператора цикла из базы данных находят первый в смысле доступа экземпляр, удовлетворяющий ограничениям фильтра, вводят его в рабочую область и выполняют оператор, входящий в состав оператора цикла. При каждом последующем выполнении в рабочую область переносят следующий подходящий экземпляр, повторяют оператор и т. д., пока найдутся экземпляры группы, удовлетворяющие предъявленным в фильтре ограничениям. Если фильтр отсутствует, то подходящими считаются все имеющиеся экземпляры этой группы.

Каждый оператор цикла проложит путь к указанной в месте группе. Во время выполнения оператора, входящего в состав оператора цикла, всегда зафиксирован какой-то экземпляр

группы, последовательность значений ключей или индексов которого можно считать значением имени комплекта. Внутри оператора цикла можно (например, при помощи других операторов цикла) продлить этот путь еще дальше. Внутренние циклы всегда должны относиться только к группам более низкого уровня, и в них нельзя изменить текущее положение внешнего цикла. Только сам внешний цикл меняет указатель текущего положения своей группы. Когда кончится выполнение оператора цикла, текущее положение группы считается неопределенным в том смысле, что путь уже не ведет к этой группе, а заканчивается на уровень выше.

В приведенных ниже примерах будем часто опираться на следующую конкретную легенду (которой прикреплено имя комплекта Ф):

```
LEG  ФИРМА  KEY = ИМЯ  TEXT
     * 1  ИМЯ
     * 1  ОСНОВ  NAT  'ГОД ОСНОВАНИЯ'
     * 1  МОДЕЛЬ  SORT  KEY = ТИП  REP
       *  МОД  SORT  KEY = КЛАСС
       * 2  ТИП
       * 2  КЛАСС
       * 2  Ч  NAT  PSEUDO
       * 2  МОТОР  REP  KEY = Л, ЛС  SORT  NAT
         *  МОТ  KEY = Л  SORT
         * 3  Л  'РАБ  ОБ''ЕМ'
         * 3  ЛС  'МОЩНОСТЬ'
         * 3  КОД  TEXT
END
```

Если в программе нужно по всем фирмам просмотреть данные о моделях, то соответствующий оператор цикла начинается так:

```
FOR  Ф
    FOR  Ф.МОДЕЛЬ
```

Первый цикл сделает доступными общие сведения о фирме, а второй (который находится внутри первого на месте оператора) поочередно вводит данные о всех моделях. Для согласования текущего положения нужно, чтобы внутренний цикл по моделям закончился с исчерпыванием данных фирмы, которую определяет состояние внешнего цикла. Таких условий не надо включать в фильтр внутреннего цикла, так как при трансляции учитывается взаимное вхождение операторов цикла с одинаковым именем комплекта и внутренние циклы автоматически определяются в рамках внешних циклов. При помощи фильтров ограничения надо задать только для собственных ключей группы, так как ключи всех высших уровней определяются во внешних операторах цикла.

В фильтре представляются критерии (условия отбора или ограничения) для значений ключей группы. Эти критерии должны располагаться в порядке собственных ключей в легенде. Если критерием является условие, то сам ключ в нем можно именовать при помощи составного имени вместе с именем комплекта, или просто именем ключа, или универсальным именем KEY для всех ключей. Например, равнозначными являются следующие три начала оператора цикла:

```
FOR  Ф.МОТ  (Ф.Л >= 1.5)
FOR  Ф.МОТ  (Л >= 1.5)
FOR  Ф.МОТ  (KEY >= 1.5)
```

Сравнение вида a : b на месте критерия есть сокращение следующего условия: KEY >= a & KEY <= b. Например, в целях выделения данных о моторах с рабочим объемом в 1 или 1,2 литра и с мощностью от 40 до 70 л.с. для всех моделей всех фирм от "Волги" до "Москвича" соответствующий оператор цикла следует начинать таким образом:

```
FOR  Ф  (ВОЛГА : МОСКВИЧ)
   FOR  Ф.МОДЕЛЬ
      FOR  Ф.МОТОР (Л = 1 | Л = 1.2, 40 : 70)
```

При написании таких операторов цикла удобно использовать упрощающее правило, по которому входящие друг в друга операторы цикла можно объединить в один оператор. Так, мы можем вместо последнего "тройного" оператора писать:

```
FOR  Ф.МОТОР (ВОЛГА : МОСКВИЧ, *, Л = 1 | Л = 1.2, 40:70)
```

Если в качестве критерия написано выражение, то это равнозначно со сравнением KEY = <выражение>. Две звездочки в качестве критерия требуются тогда, когда у группы более одного ключа и внешний цикл организуется по одним ключам, а внутренний - по другим. Например, пусть данные о моторах фирмы "Волга" рассматриваются по порядку возрастания их рабочих объемов, а при каждом значении рабочего объема надо пересчитать, сколько имеется моторов разной мощности. Такую работу проводит оператор:

```
FOR  Ф.МОТОР (ВОЛГА, *, *, *)
   DO  Ф.Ч := 0;
     FOR  Ф.МОТОР (* *, *) Ф.Ч := Ф.Ч + 1
   OD
```

Две звездочки означают здесь, что значение этого ключа уже определено внешним циклом и при выполнении данного оператора цикла его нельзя менять. Когда в фильтрах внешнего и внутреннего цикла указана звездочка для одного и того же ключа, то просмотр имеющихся значений происходит только один раз.

Если повторяющаяся группа будет обработана по дополнительному доступу, то в конструкции "место" надо указать имена комплекта и дополнительного доступа. Условия отбора для группы должны тогда относиться к ключам дополнительного доступа. Например, если предпоследний пример переписать в виде

```
FOR  Ф.МОТ  (ВОЛГА : МОСКВИЧ, *, Л = 1 | Л = 1.2)
```

то условие для мощности мотора уже не может содержаться в фильтре, так как доступ МОТ имеет только один ключ. Это условие можно учесть внутри оператора, например, следующим образом:

```
FOR  Ф.МОТ  (ВОЛГА : МОСКВИЧ, *, Л = 1 | Л = 1.2)
   IF  (Ф.ЛС>= 40 &Ф.ЛС<=70)  THEN
```

Если дополнительный доступ используется на каком-то высшем уровне, то в конструкции "место" обязательно надо сообщить полное составное имя группы, указывая для каждого уровня имя соответствующего доступа. Например, если данные о моторах рассматриваются не по типам моделей, а по их классам, последний пример начинается так:

```
FOR  Ф.МОД.МОТ  (ВОЛГА : МОСКВИЧ, *, Л = 1 | Л = 1.2)
```

Отдельно следует остановиться еще на определении критериев отбора для повторяющейся группы без явного доступа. В

таком случае критерий считается относящимся к индексу группы, а при массиве надо указать критерии для всех его индексов. Если в легенде индексам массива присвоены имена, то их можно использовать в условиях - универсальное имя KEY предназначена в основном для неименованных индексов.

Пусть у нас еще легенда ПРОД (с именем комплекта X), описывающая данные о продукции моделей разных фирм по кварталам за последние 10 лет:

```
LEG  ПРОД  KEY = ИМЯ  TEXT
  * 1  ИМЯ
  * 1  МОДЕЛЬ  REF
    *  МОД  SORT  KEY = ТИП
       * 2  ТИП
       * 2  ПР  ARRAY  [10, КВ = 4]  NAT
         * 3  КОЛИЧЕСТ
         * 3  СТОИМОСТ
END
```

Если нужно просмотреть данные о продукции всех моделей всех фирм за последние два года без данных о четвертых кварталах, то это можно осуществить оператором, который начинается так:

```
FOR  X.ПР  (*, *, KEY >=9 , КВ ¬=4)
```

Критерий для индекса простой повторяющейся группы можно составить только в случае, когда известно (например, из контрольной печати), какими по очереди являются необходимые нам экземпляры. Например, если нас интересуют данные о тех "Жи-

гулях", которые введены в качестве третьего, четвертого и пятого экземпляра в группе МОДЕЛЬ этой фирмы, то соответствующий оператор начинается так:

```
FOR  Х.МОДЕЛЬ  (ЖИГУЛИ, 3 : 5)
```

В практике порядковые номера обычно не известны и выделение нужных экземпляров происходит при помощи условного оператора. Например, данные о модели ВАЗ-2101 можем получить оператором, начало которого имеет вид:

```
FOR  Х.МОДЕЛЬ  (ЖИГУЛИ, *)
   IF  Х.ТИП = ВАЗ2101  THEN
```

Наш пример содержит, к счастью, дополнительный доступ по типам и мы можем вместо приведенного использовать более простое начало оператора:

```
FOR  Х.МОД  (ЖИГУЛИ, ВАЗ2101)
```

Во всех приведенных примерах встречались только вложенные друг в друга циклы по экземплярам повторяющихся групп. Если необходимо параллельно обработать экземпляры разных групп, то для этого в операторе цикла следует представить перечень мест, разделенных запятыми. В этом случае при каждом выполнении цикла для всех указанных мест (начиная слева) вводят соответствующие удовлетворяющие критериям экземпляры и выполнение цикла кончается тогда, когда хотя бы в одной повторяющейся группе больше нет подходящих экземпляров.

Например, пусть у нас два комплекта записей для показателей о продукции автомобилей (имена комплектов Х1 и Х2). Чтобы параллельно рассмотреть эти два комплекта (т.е. взять

первую запись из одного и первую из другого комплекта, потом вторую из одного и вторую из другого комплекта и т. д.), следует начало соответствующих операторов представить в виде:

```
FOR  X1, X2
```

Выполнение такого оператора кончается исчерпыванием одного (или сразу двух) из этих комплектов.

Если для всех фирм, данные о которых имеются в комплекте X1, надо проверить, имеются ли данные о них также в комплекте X2, то составим оператор

```
FOR  X1, X2  (X1.ИМЯ)
```

который при каждом выполнении вводит запись из комплекта X1 и затем запись с таким же ключом из комплекта X2. Выполнение этого оператора кончается либо при исчерпывании комплекта X1, либо при нахождении первой записи из X1, для которой нет соответствующей записи в комплекте X2.

Для выяснения всех таких фирм, данные о которых имеются в комплекте X1 и отсутствуют в комплекте X2, можно составить, например, следующую программу (в которой предполагается, что библиотека содержит процедуру PRINT для печатания значения атома):

```
DML  ПР1
    LEGEND  ПРОД  SET  X1, X2;
    VAR  LEG  РАБ
            * 1  СЕМАФОР  NAT
        END  SET  P
```

```
FOR  X1
   DO  P.СЕМАФОР := 0;
       FOR  X2  (X1.ИМЯ) P.СЕМАФОР := 1;
       IF  P.СЕМАФОР = 0  THEN
                 PROC  PRINT(X1.ИМЯ)  FI
   OD
```

Оператор цикла можно использовать и для обработки списочных структур, т.е. повторяющихся групп со списочной организацией. Если такая группа не имеет доступа, то ее экземпляры рассматриваются в прямом порядке прохождения дерева (ключом является порядковый номер экземпляра). Если же в легенде указан доступ, то просмотр проходит в порядке этого доступа.

Конструкция "место" должна в случае списочной структуры заканчиваться либо именем корня этой группы, либо составным именем, состоящим из имени корня и имени указателя. Например, если программа относится к легенде

```
LEG  ДЕРЕВО  KEY = К
  * 1  К  NAT
  * 1  А  LIST  KEY = ИМЯ  SORT
    * 2  ИМЯ  TEXT  PICT = 12
    * 2  СЫН  DPOINT
    * 2  БРАТ  RPOINT
END
```

(с именем комплекта Z), то в операторах цикла допустимыми являются места Z.A, Z.A.БРАТ и Z.A.СЫН . При этом оператор, начало которого имеет вид

FOR Z.A (АЛЕКСЕЙ : БОРИС),

выделяет те экземпляры группы, где атом ИМЯ имеет значение от Алексея до Бориса (в альфавитном порядке). В то же время начало оператора

FOR Z.A.СЫН (АЛЕКСЕЙ : БОРИС)

извлекает (первые) сыновья, а начало оператора

FOR Z.A.БРАТ (АЛЕКСЕЙ : БОРИС)

следующие братья упомянутых экземпляров.

Если выполнение цикла должно прекращаться раньше, чем исчерпываются данные, следует использовать оператор выхода:

В этом операторе сообщают имя повторяющейся группы (или его дополнительного доступа), зафиксированной в операторе цикла, из которого хотят выйти. Если в операторе цикла дана перечень мест, то при выходе можно указать любое из них. Натуральное число в скобках понадобится, когда оператор выхода содержится в нескольких операторах цикла (или операторах обновления), связанных с одним и тем же местом. В таком случае этим числом указывают, из скольких таких операторов следует выйти (вместо опущенного натурального числа принимается единица).

Например, в отрывке программы

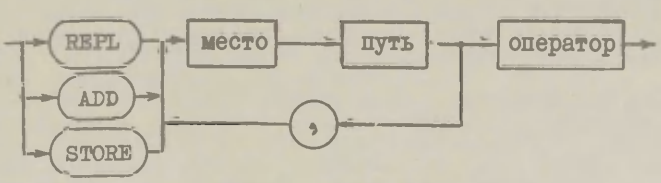
```
FOR  Ф.МОТОР
   DO  . . .
     FOR  Ф.МОТОР  (* *, *)
        DO  . . .
          LEAVE  Ф.МОТОР
             . . .
          LEAVE  Ф.МОТОР (2)
        OD
      . . .
   OD
```

первый оператор выхода дает управление оператору, следовавшему внутреннему оператору цикла, а второй оператор выхода кончает выполнение внешнего цикла.

5. Обновление данных

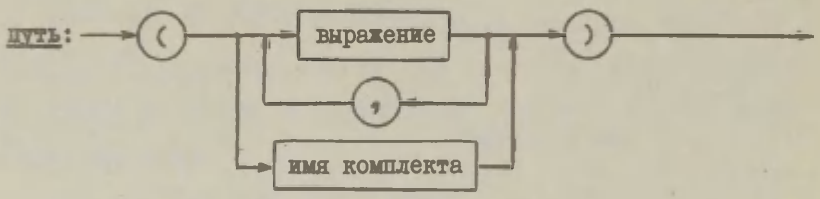
Для обновления базы данных используются два оператора – оператор обновления и оператор удаления.

оператор обновления:



В операторе обновления кроме типа операции сообщают имя обновляемой группы и значения ключей для фиксирования экземпляра. Оператор в составе оператора обновления укажет все

необходимые изменения значений в этом экземпляре. Имя группы дается таким же образом, как и в операторе цикла, но здесь запрещается использование имен дополнительных доступов.



Путь можно считать упрощенным вариантом фильтра. Он предназначен для выделения одного конкретного экземпляра группы, который и будет обновлен: значения выражений в пути являются значениями ключей. Так как оператор обновления может находиться внутри других операторов, которые тоже выделяют экземпляры групп, то в пути не обязательно надо перечислить значения всех ключей вышестоящих групп, а только те, которые продлят путь до требуемой группы.

Обновление может осуществиться путем замены (REPL), добавления (ADD) или включения (STORE). Замена вызывает перемещение требуемого экземпляра группы (определенного при помощи места и пути) в рабочую область. Затем выполняется внутренний оператор оператора обновления и после его завершения экземпляр помещается из рабочей области обратно в буфер ввода-вывода. Если в базе данных нет требуемого экземпляра, то управление сразу передается следующему оператору.

Добавление начинается с проверки, находится ли требуемый экземпляр группы уже в базе данных. Если да, то оператор обновления никаких операций больше не вызывает и управление передается следующему оператору. Когда в базе данных требуе-

мого экземпляра еще нет, то в рабочую область заносят только значения ключей группы, а остальные атомы в ветке группы получают пустые значения. Затем выполняют внутренний оператор и после этого экземпляр группы перемещают из рабочей области в буфер.

Включение является объединением замены и добавления. При наличии требуемого экземпляра выполняется операция замены, а в обратном случае – операция добавления. Следовательно, при операции включения всегда выполняется внутренний оператор и данные заносятся из рабочей области в буфер.

Внутри оператора обновления можно написать операторы обновления для подчиненных групп. В принципе возможны любые вложения друг в друга операторов цикла и обновления, если внутренний оператор продолжит путь внешнего. Однако, не всякое вложение имеет смысл, например, если оператор цикла находится внутри оператора добавления.

Значения ключей обновляемой группы в пути не меняются во время выполнения оператора обновления и когда в конце его выполнения экземпляр заносится в буфер, значения ключей в рабочей области должны быть в согласии с путем. Следовательно, нельзя менять значения ключей внутри оператора обновления. После завершения оператора обновления путь к этой группе считается определенным и сохраняется.

При перемещении данных из рабочей области в буфер создается новый экземпляр группы, помещаемый в конец набора экземпляров этой группы, несмотря на значения его ключей. На свое место он попадает только тогда, если из рабочей области в буфер перемещается экземпляр одного из вышестоящих групп.

Следовательно, для полного упорядочения записи программа должна содержать оператор обновления для корневой группы записи и этот оператор является самым внешним оператором из операторов обновления. Не обязательно иметь операторы обновления для всех уровней, но они должны быть для обновляемых мест и для всей записи.

Оператор обновления записи сообщает, что закончилось обновление и ее можно вывести в базу данных. Операторы обновления более низких уровней вызывают перемещение данных из рабочей области в буфер, но не указывают на необходимость запоминания этой записи: если они не входят в состав оператора обновления записи, то все проведенные ими обновления теряются, когда в буфер вводится новая запись.

Рассмотрим теперь несколько примеров, опираясь на приведенную в предыдущей главе легенду ФИРМА (см. стр. 120). Например, по отрывку программы:

```
FOR  X.МОД  (ФОРД, ЭСКОРТ)
   DO  STORE  X.МОТОР  (1.3, 54)
              X.КОД := 13Н ;
       ADD  X.МОТОР  (1.3, 57) ;
       REPL  X.МОТОР  (1.6, 88)
              X.КОД := 16НС
   OD
```

находят данные о конкретной модели (или одной из них, так как используется дополнительный доступ), причем обновлению подлежат данные о моторах. Данные о моторе "13N" запомина-

ются независимо от того, были ли таковые уже в базе данных или нет. Далее добавляется мотор с мощностью в 57 л.с., если такого мотора в базе раньше не было (код этого мотора остается пустым). Наконец заменяют данные о моторе в 88 л.с., если они уже существовали.

В следующем примере в комплект У добавляют все такие записи из комплекта Ф, которые до сих пор отсутствовали:

```
DML  ПР2
   LEGEND  ФИРМА  SET  Ф, У
    FOR  Ф  ADD  У(Ф.ИМЯ)  У := Ф
```

Пример обновления сразу на двух уровнях дает программа:

```
DML  ПР3
   LEGEND  ФИРМА  SET  Ф, У
    FOR  Ф
       REPL  У(Ф.ИМЯ)
           DO  У.ОСНОВ := Ф.ОСНОВ;
             FOR  Ф.МОДЕЛЬ
                ADD  У.МОДЕЛЬ (Ф.ТИП)
                     У.МОДЕЛЬ := Ф.МОДЕЛЬ
           OD
```

которая просматривает все записи комплекта Ф и находит соответствующие записи в комплекте У. В этих записях заменяют дату основания фирмы датой, взятой из данных комплекта Ф. Кроме этого в те же записи комплекта У добавляют такие данные о моделях из комплекта Ф, которые раньше отсутствовали.

раторы вложены друг в друга. Например, следующая программа похожа на предыдущую, но выполняет совсем другую работу:

```
DML  ПР4
    LEGEND  ФИРМА  SET  Ф, У
    FOR  Ф
        DO  REPL  У(Ф.ИМЯ)  У.ОСНОВ := Ф.ОСНОВ;
            FOR  Ф.МОДЕЛЬ
                ADD  У.МОДЕЛЬ (Ф.ИМЯ, Ф.ТИП)
                    У.МОДЕЛЬ := Ф.МОДЕЛЬ
        OD
```

Здесь цикл по моделям комплекта Ф не входит в оператор замены записи комплекта У. Если программа ПР3 добавляет в комплект У только такие данные о моделях из комплекта Ф, для которых оператор замены записи работает (т.е. имеются соответствующие записи в комплектах Ф и У), то программа ПР4 добавляет в комплект У все данные о моделях из комплекта Ф, которые в комплекте У отсутствуют. Поскольку оператор добавления данных о моделях не входит в состав внешнего оператора обновления, то в пути надо указать и значение ключа для вышестоящей группы.

Оператор обновления, в котором указаны значения ключей для нескольких уровней, является сокращенной записью вложенных друг в друга операторов обновления для всех этих уровней. Например, начало оператора

```
ADD  У.МОДЕЛЬ (Ф.ИМЯ, Ф.ТИП)
```

по существу эквивалентно началу оператора

```
STORE  У(Ф.ИМЯ)
       ADD  У.МОДЕЛЬ (Ф.ТИП)
```

Когда путь состоит только из имени комплекта, то оно и имя комплекта в конструкции "место" должны соответствовать одинаковой легенде, т.е. иметь одинаковую структуру. Тогда все необходимые значения ключей для определения места обновления берутся из текущих значений ключей имени комплекта, указанного в пути. Так, например, следующая программа эквивалентна программе ПР2:

```
DML  ПР5
   LEGEND  ФИРМА  SET  Ф, У
   FOR  Ф  ADD  У(Ф)  У := Ф
```

Если повторяющаяся группа не имеет доступа, то в пути надо указать индекс экземпляра. Следует, однако, считаться еще с некоторыми особенностями такой группы. Например, оператор (относящийся к легенде ПРОД со страницы 124, которой прикреплено имя комплекта Х):

```
FOR  Х (ЖИГУЛИ)
     REPL  Х.МОДЕЛЬ (3)
           Х.ТИП := ВАЗ2102
```

вызывает изменение типа на ВАЗ-2102 у той модели "Жигулей", которая введена в качестве третьей. Если в момент обновления в записи имелись данные о меньше чем трех моделях, то этот оператор никакого обновления не произведет.

При добавлении указание индекса вызывает добавление нового экземпляра за указанным экземпляром (если указать значение 0, то в начало набора). Например, оператор

```
FOR  X (ЖИГУЛИ)
   ADD  X.МОДЕЛЬ (3)
        X.ТИП := ВАЗ2102
```

помещает модель ВАЗ-2102 (без данных о продаже) в качестве четвертой, а если моделей раньше было меньше трех, то добавление не производится. Для добавления в конец набора экземпляров значение индекса должно быть опущено. Например:

```
FOR  X (ЖИГУЛИ)
   ADD  X.МОДЕЛЬ (*)
        X.ТИП := ВАЗ2102
```

(это, кстати, единственный случай, когда в пути можно указать выражение, значение которого отсутствует; при обновлении групп с доступом все выражения в пути должны всегда иметь значения).

Включение в группу без доступа означает всегда помещение нового экземпляра в конец набора экземпляров этой группы. В пути на месте индекса может быть любое выражение.

По определению оператор обновления может содержать целый перечень мест и путей. Например, начало оператора

```
ADD  Д.А (S1), Д.В (S2), Д.С (S3)
```

является, по существу, сокращением следующей записи:

```
ADD  Д.А (S1)
   ADD  Д.В (S2)
      ADD  Д.С (S3)
```

Если во время выполнения оператора обновления выяснится, что обновить все-таки не надо, то можно использовать оператор выхода, который передает управление следующему оператору без занесения данных в базу. Например, следующий оператор заменяет данные о фирме только тогда, когда в старых данных дата основания фирмы принадлежит прошлому веку:

```
FOR  Ф
   REPL  У(Ф.ИМЯ)
      DO  IF  У.ОСНОВ>=1900  THEN  LEAVE  У  FI ;
         У := Ф
      OD
```

Так как оператор обновления может обработать только один экземпляр группы, то для обновления нескольких экземпляров он должен находиться в составе оператора цикла для того же места. Например, если надо присвоить пустые значения всем датам основания фирм, следует писать оператор

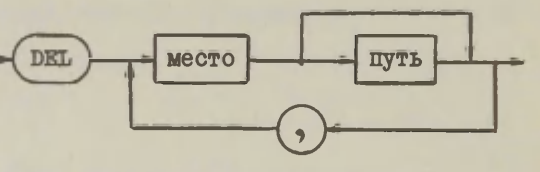
```
FOR  Ф
   REPL  Ф(Ф)  Ф.ОСНОВ := *
```

Приведем еще пример о замене значения ключа новым. Так как внутри оператора обновления нельзя менять значения ключей, для этой цели придется использовать два имени комплекта. Например, при помощи программы

```
DML ПР6
    LEGEND ФИРМА SET Ф, У
    FOR Ф (ГИГУЛИ)
        STORE У (ЖИГУЛИ) У := Ф
```

запись с ошибочным ключом (ГИГУЛИ) запоминается в качестве новой записи с другим значением ключа. При этом старая запись тоже сохраняется и для ее стирания следует применить оператор удаления.

оператор удаления:



Если в операторе удаления приводится только список мест, то предполагается, что уже определены пути, ведущие по всем этим местам. Оператор удаления вызывает удаление из базы данных всех соответствующих экземпляров групп. Так, например, оператором

```
FOR Ф (ГИГУЛИ) DEL Ф
```

удаляется запись со значением ключа "ГИГУЛИ", а оператором

```
FOR Ф (У1.ИМЯ : У2.ИМЯ) DEL Ф
```

все записи в пределах от значения атома У1.ИМЯ до значения атома У2.ИМЯ.

Оператор удаления может находиться и внутри оператора обновления, но в этом случае требуется еще оператор выхода, чтобы в конце выполнения оператора обновления данных все-таки не поместили опять в буфер. Например, следующий опера-

тор вызывает удаление записей фирм, основанных в прошлом веке, причем году основания каждой более молодой фирмы прибавляется единица:

```
STORE  Ф (У.ИМЯ)
   IF  Ф.ОСНОВ < 1900  THEN  DO  DEL  Ф;
                                 LEAVE  Ф
                             OD
       ELSE  Ф.ОСНОВ := Ф.ОСНОВ + 1  FI
```

Хотя оператор удаления несколько отличается от операторов обновления, следует и его включить в состав оператора обновления записи (если он сам не удаляет всю запись, определяя и путь). Например, для удаления данных о моторах с мощностями от 50 до 80 л.с. всех фирм и моделей следует вместо оператора

```
FOR  Ф.МОТОР  (*, *, *, 50 : 80)  DEL  Ф.МОТОР
```

писать оператор

```
FOR  Ф
   REPL  Ф(Ф)
      FOR  Ф.МОТОР  (*, *, 50 : 80)
           DEL  Ф.МОТОР
```

В случае списочной структуры конструкция "место" в операторе обновления или удаления составляется так же, как и в операторе цикла. При замене или удалении последним именем может быть как имя корня, так и имя указателя. Добавление разрешается только по имени указателя, а операция включения

неприменима для обновления списочной структуры.

Пусть по легенде ДЕРЕВО (см. стр. 27) с именем комплекта Z составлена запись, в которой атом К имеет значение 25, а экземпляры группы А образуют структуру:

Тогда после выполнения следующего отрывка программы

```
REPL  Z.A (25, ИГОРЬ)
   Z.A := (ИГОРЬ(БОРИС; МАРК));
ADD   Z.A.БРАТ (25, ПЕТР)
   Z.ИМЯ := ИВАН2;
DEL   Z.A (ГЛЕБ)
ADD   Z.A.БРАТ (25, МАРК)
   Z.A := (ОЛЕГ);
REPL  Z.A.СЫН (25, ИВАН)
   Z.A := (ПАВЕЛ(ИЛЬЯ));
```

эта структура принимает вид:

Содержание

БАЗА ДАННЫХ ДЛЯ ЕС-1022. Труды вычислительного центра. Выпуск 41. На русском языке. Тартуский государственный университет. ЭССР, г. Тарту, ул. Юликооли, 18. Ответственный редактор Ю.Таммер. Сдано в печать 23.II.1978. Бумага печатная 30х42 1/4. Печ. листов 9,0 (условных 8,37). Учетно-издат. листов 4,83. Тираж 500. МВ 07392. Типография ТГУ, ЭССР, г. Тарту, ул. Пялсони, 14. Зак. № 1484. Цена 70 коп.

70 коп.