университет имени академика С.П. Королева

ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ. СПИСКИ

Самара 1994

Государственный комитет Российской Федерации
по высшему образованию

Самарский государственный аэрокосмический
университет имени академика С.П.Королева

ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ. СПИСКИ

Методические указания к выполнению
лабораторных и самостоятельных работ

С а м а р а 1994

Составители М.А.Кораблин, Е.В.Симонова

УДК 681.142.2

Динамические структуры данных. Списки: Метод. указания к выполнению лабораторных и самостоятельных работ / Самар. госуд. аэрокосм. ун-т; Сост. М.А.Кораблин, Е.В.Симонова; Самара, 1994. 20 с.

Методические указания содержат краткие теоретические сведения по организации списковых структур и варианты заданий для выполнения лабораторных и самостоятельных работ.

Предназначены для использования при изучении курса "Основы информационной технологии" (раздел "Структурн данных") в учебном процессе специальностей "Прикладная математика","Автоматизированные системы обработки информации и управления", "Программное обеспечение вычислительных и автоматизированных систем". Разработаны на кафедре "Информационные системы и технологии".

Печатаются по решению редакционно-издательского совета Самарского государственного аэрокосмического университета имени академика С.П.Королева

Рецензент: С. В С м и р н о в

1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Ссылочный тип

Ссылочный (указательный) тип определяет особый тип объектов, каждый из которых определяет местоположение (адрес) другого объекта в памяти ЭВМ. В качестве объекта, местоположение которого определяет указатель, может выступать объект любого типа, в том числе и указатель. **Константы ссылочного типа** определяются как адреса рабочего пространства оперативной памяти. **Специальная константа** - NIL определяет особое положение указателя, который не указывает никуда (объект, на который он указывает, не существует).

Для адресации 1024K ячеек оперативной памяти любой адрес должен представляться 20 битами: 2^{20}=1024K. Для решения проблемы 20-разрядной адресации на 16-разрядной ЭВМ используются два 16-разрядных числа: сегмент и смещение. Сегментная часть адреса (сегмент) определяет номер фрагмента оперативной памяти, в которой размещается объект, идентифицируемый адресом, а смещение (относительная часть адреса) - положение объекта внутри сегмента. Размер сегмента определяется величиной 2^{16}=64K, а общее количество сегментов - величиной 1024K/64K=16=2^4.

Как структура, состоящая из двух компонент, любой адрес описывается следующим образом:

```
TYPE ADDRESS = RECORD
        OFFSET (* Смещение *): CARDINAL;
        SEGMENT (*  Сегмент *): CARDINAL
END;
```

Указатели подразделяются на **свободные** и **ограниченные**.

Свободный указатель, реализуемый машинно-зависимым типом ADDRESS, может указывать на объект, допускающий произвольную интерпретацию: TYPE ADDRESS = POINTER TO WORD.

Ограниченный указатель, реализуемый абстрактным ссылочным типом (POINTER TO...), указывает на объекты, интерпретация которых ограничена определенным типом, связанным с этим указателем.

Например,

```
TYPE
     РОкружность = POINTER TO Окружность;
       Точка=RECORD  x,y: CARDINAL   END;
       Окружность = RECORD  центр: Точка;
                            радиус: CARDINAL
       END;
```

Здесь РОкружность - абстрактный ссылочный тип, объектами которого являются ограниченные указатели, а Окружность - тип интерпретации объектов "под указателями" (т.е. объектов, на которые указывают ограниченные указатели типа РОкружность).

1.2. Действия над указателями

Над указателями возможны следующие действия: **присваивание** (передвижка, установка указателя на объект), **сравнение** указателей (=,#) **раскрытие ссылки** (открытие доступа к объекту через указатель, установленный на него).

1.2.1. Операция присваивания для указателей

Присваивание для указателей сводится к пересылке значения одного указателя другому. Совместимыми по присваиванию являются:

- два указателя одного и того же типа;
- константа NIL и свободный указатель;
- константа NIL и ограниченный указатель любого типа;
- свободный указатель и ограниченный указатель любого типа.

Ограниченному указателю одного типа можно присвоить значение ограниченного указателя другого типа с помощью функции приведения типов ADDRESS.

Каждый указатель перед использованием необходимо **инициализировать**, т.е. установить на соответствующий объект. Установка указателя на объект, адрес которого неизвестен, производится с помощью встроенной функции ADR, аргументом которой является идентификатор объекта. Например,

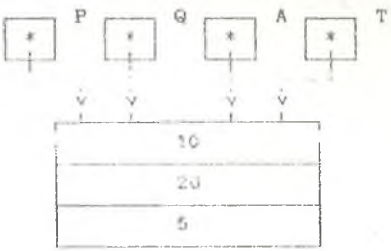
```
VAR P: POINTER TO CARDINAL; c: CARDINAL;
   BEGIN P:= ADR(c); ...
```

Установка указателя P на объект, адрес которого известен и хранится в некотором указателе B, производится с помощью присваивания P:=B. Например,

```
TYPE
 POкружность = POINTER TO Окружность;
 PTочка = POINTER TO Точка;
     Точка=RECORD  x,y: CARDINAL   END;
     Окружность=RECORD  центр: Точка;  радиус: CARDINAL  END;
```

```
VAR
 Окр: Окружность; Q,P: РОкружность; T: РТочка; A: ADDRESS;
BEGIN
 WITH Окр DO
    WITH центр DO  x:=10; y:=20; END;   радиус:=5;
 END;
 P:=ADR(Окр);  (* Установка указателя P на объект Окр *)
 Q:=P; A:=P; T:=ADDRESS(P);
```

Результат выполнения этой программы иллюстрирует рис.1.



Р и с. 1. Установка указателей

В этом примере ограниченные указатели Q и T, а также свободный указатель A указывают на тот же объект, что и указатель P. После присваивания значения одного указателя другому они указывают на один и тот же объект, объект при этом не изменяется.

1.2.2. Адресная арифметика

Процедуры адресной арифметики допустимы только для свободных указателей.

```
PROCEDURE AddAddr(A: ADDRESS; increment: CARDINAL): ADDRESS;
```

Возвращает адрес байта, отстоящего от физического адреса A на число байтов, определенных параметром increment, в сторону увеличения адресов.

PROCEDURE SubAddr(A: ADDRESS; decrement: CARDINAL): ADDRESS;

Возвращает адрес байта, отстоящего от физического адреса A на число байтов, определенных параметром decrement в сторону уменьшения адресов.

PROCEDURE IncAddr(VAR A: ADDRESS; increment: CARDINAL);

A становится адресом байта, расположенного на increment байтов после физического адреса A.

PROCEDURE DecAddr(VAR A: ADDRESS; decrement: CARDINAL);

A становится адресом байта, расположенного на decrement байтов перед физическим адресом A.

1.2.3. Доступ к объекту. Раскрытие ссылки

Доступ к объекту через ссылку связан с предварительной установкой указателя на объект и последующим раскрытием ссылки. Каждая переустановка указателя открывает доступ к новому объекту. Доступ к объекту через ограниченный указатель осуществляется с использованием его имени, за которым следует символ "^" (используемый в качестве постфикса).

Раскрытие ссылки представляет собой **двухшаговый** процесс:

- указатель используется для определения расположения объекта в оперативной памяти;

- производится доступ к атрибутам объекта.

Пример:

```
VAR Окр: Окружность; Р: РОкружность;
        (* Типы РОкружность, Окружность определены выше *)
BEGIN P:=ADR (Окр);
      P^.центр.x:=10; P^.центр.y:=20; P^.радиус:=5;
```

В подобной ситуации может быть использован оператор присоединения WITH. Заметим, что в теле оператора WITH переустановка указателя Р, присоединяющего к объекту Окр, недопустима:

```
WITH P^ DO центр.x:=10; центр.y:=20; радиус:=5 END;
```

Если два указателя одного типа указывают на разные объекты, можно присвоить объекту, на который указывает один из них, значение объекта, на который указывает другой. Подчеркнем, что после этого указатели продолжают указывать на разные объекты, но с одинаковыми значениями. Например:

```
VAR Окр1, Окр2: Окружность;  P,Q: РОкружность;
BEGIN  P:=ADR(Окр1);  Q:=ADR(Окр2);
  WITH P^ DO центр.x:=10; центр.y:=20; радиус:=5      (* 1 *)
  END;
  Q^:=P^;                                            (* 2 *)
```

Выполнение операторов (* 1 *) и (* 2 *) иллюстрирует рис. 2.

До После

Р и с. 2. Выполнение операторов (* 1 *) и (* 2 *)

Оператор (* 2 *) эквивалентен следующему оператору WITH:

```
WITH Q^ DO центр.x:=P^.центр.x; центр.y:=P^.центр.y;
           радиус:=P^.радиус
END;
```

Можно написать для этого примера аналогичный оператор, присоединяющий к другому объекту, расположенному 'под' указателем P:

```
WITH P^ DO Q^.центр.x:=центр.x; Q^.центр.y:=центр.y;
           Q^.радиус:=радиус
END;
```

Вопрос: Можно ли для этого примера использовать конструкцию вложенного WITH, присоединяющего данный фрагмент сразу к двум указателям, P и Q?

1.2.4. Сравнение указателей

Указатели можно сравнивать на равенство и неравенство. Два указателя равны, если они указывают на один и тот же объект или оба они никуда не указывают (оба равны NIL). Неравные указатели указывают на разные объекты (или один из них никуда не указы-

вает). Указатели можно сравнивать с константой NIL, чтобы установить, ссылается ли данный указатель на что-либо или нет. Порядок вычисления булевских выражений в условных операторах, использующих доступ к объектам через указатели, особенно важен. Например, оператор:

```
IF (P#NIL) & (P^.радиус=5) THEN ...
```

определен корректно и будет работать в любом случае, даже если значение P=NIL. Но определение:

```
IF (P^.радиус=5) & (P#NIL) THEN ...
```

некорректно, т.к. если P=NIL, то выражение P^.радиус лишено смысла, поскольку P ни на что не указывает.

1.3. Последовательная и связанная организация памяти

Существует два принципа организации памяти и соответственно два метода доступа к объектам, размещенным в оперативной памяти:

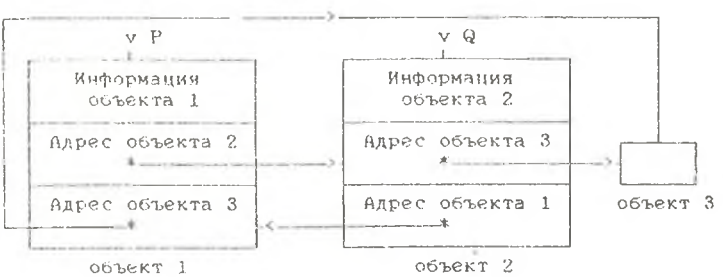
- метод вычисляемого адреса;
- метод хранимого адреса.

Согласно методу **вычисляемого адреса** на этапе трансляции исходного текста программы по имени объекта вычисляется его адрес, который закрепляется за объектом на все время работы программы. Доступ через имя всегда открывает один и тот же объект. С использованием метода вычисляемого адреса связана последовательная организация памяти.

При такой организации элементы хранения объектов размещаются в смежных последовательно расположенных ячейках памяти, что характерно, например, для хранения одномерных массивов. Более сложные методы последовательной организации связаны с методами **индексации** и соответственно вычислением адресов объектов через

индексы (например, многомерные массивы).

Согласно методу **хранимого адреса** адрес объекта не вычисляется, а хранится в некотором указателе на этот объект. Для доступа к объекту сначала необходимо получить ссылку на него (т.е. значение указателя), а затем выполнить операцию раскрытия ссылки. Каждый объект имеет возможность хранить в виде ссылок связи с другими объектами, с которыми он взаимодействует в программе. Для реализации этой возможности в структуру объекта необходимо ввести специальные **ссылочные поля** (атрибуты) для хранения таких связей. Подобная организация памяти называется **связанной**. Графическая иллюстрация структур связанной организации памяти использует фигуры прямоугольников для изображения объектов и стрелок для изображения связей (ссылок) между объектами. На рис.3 приведен пример такой иллюстрации для связанной организации структуры из трех объектов. Здесь P и Q - указатели, установленные на объекты 1 и 2 соответственно, "Адрес объекта" - ссылочные поля для хранения связей. Заметим, что на этой иллюстрации P открывает доступ к объекту 1, Q - доступ к объекту 2, а непосредственный доступ к объекту 3 возможен только через объект 2 или объект 1 с использованием соответствующих полей связей.



Р и с. 3. Связанная организация памяти

1.4. Классы памяти

Статическая память выделяется на фазе трансляции и находится в распоряжении пользователя в течение всей работы программы. Статическая память имеет последовательную организацию.

Автоматическая память выделяется в области системного стека и имеет последовательную организацию. При каждом вызове процедура автоматически получает память для размещения своей локальной среды (локальных переменных и формальных параметров). Параметры размещаются в стеке в том порядке, в котором они появляются в объявлении процедуры. Параметры-значения передаются за счет размещения фактического значения параметра в стеке. Параметры-переменные передаются путем расположения в стеке их адресов. После завершения работы процедуры память, занятая под размещение локальной среды, освобождается и может быть выделена другим процедурам. Таким образом, локальная среда процедур не сохраняется между вызовами.

Динамическая память под объекты выделяется по явному запросу пользователя в момент, когда эти объекты "начинают существовать" в процессе выполнения программы.

Участок памяти для размещения динамического объекта размера Size байт выделяется с помощью вызова процедуры

```
ALLOCATE( VAR A:ADDRESS; Size: CARDINAL );
```

При этом в A записывается адрес динамически размещенного объекта. (Переменная A ссылочного типа может храниться в любой разновидности памяти).

Процедура DEALLOCATE(VAR A:ADDRESS; Size: CARDINAL); освобождает память размера Size байт, занятую динамическим объектом, расположенным по адресу A. После выполнения этой процедуры A по-

лучает значение NIL. Процедуры ALLOCATE и DEALLOCATE требуют явного задания размера динамического объекта.

Если это представляется неудобным можно воспользоваться операторами NEW(VAR A: ADDRESS) и DISPOSE (VAR A: ADDRESS). Фактическим параметром при вызове этих процедур должен быть *ограниченный* указатель, по имени которого автоматически определяется, с каким типом объектов он связан и каков размер выделяемой (NEW) или освобождаемой (DISPOSE) памяти.

Алгоритмы динамического управления памятью позволяют резервировать и освобождать различные по размеру участки памяти, что делает возможным создание сложных динамических структур с различным взаимным расположением (порядком) объектов.

1.5. Динамические структуры списков

Динамической структурой называется упорядоченное множество объектов, состав и взаимное расположение которых может динамически изменяться. В зависимости от отношения порядка на множестве различают линейные и нелинейные структуры данных.

Линейной структурой данных (списком) называется множество элементов $S=\{s_i\}$ $i=1,\ldots,n$, на котором определены отношения предшествования (следования), причем для любого s_i ($i=2,\ldots,n-1$) существует только один "предшественник" s_{i-1} и один "последователь" s_{i+1}. s_1 не имеет предшественника и является "головой" списка, s_n не имеет последователя и является "хвостом" (концом) списка. Ситуация ($n=0$) определяет особое состояние: "список пуст".

Реализация динамической структуры линейного списка на связанной памяти требует включения в структуру каждого его элемента

полей для связи с соседними элементами. В зависимости от организации ссылочных связей между элементами списков различают односвязные, двусвязные, кольцевые, многосвязные и т.п. линейные списки.

1.5.1. Односвязные линейные списки

Элемент хранения односвязного списка состоит из двух частей: одного или нескольких информационных полей и *одного* поля связи.

```
TYPE
 PNode= POINTER TO NODE;          (* Указатель на узел списка *)
 Node = RECORD                        (* Описание узла списка *)
   Info: CARDINAL;                (* Информационное поле узла *)
    Link: PNode                              (* Поле связи узла *)
 END;
VAR First: PNode;          (* Указатель на первый узел списка *)
```

На рис.4 представлен пример связанной организации односвязного линейного списка, ссылка на первый элемент которого присвоена переменной FIRST. Доступ к списку возможен для рис.4 только через First. Если список пуст, то First = NIL.

Р и с. 4. Односвязный линейный список

Доступ к полям любого узла списка связан с установкой указателя на этот узел и последующим раскрытием ссылки. Возможен дистанционный доступ к произвольному узлу списка с использованием

цепочки указателей:

```
VAR
  First: PNode;      P: PNode; (* Вспомогательный указатель *)
  x: CARDINAL;
BEGIN
 x:=First^.Info;        (* Доступ к первому узлу списка, x=1 *)
 x:=First^.Link^.Info;  (* Дистанционный доступ ко второму
                                              узлу списка, x=2 *)
```

Дистанционный доступ ко второму узлу списка эквивалентен следующей последовательности операторов:

```
P:=First^.Link;     (* Установка вспомогательного указателя
                                          на второй узел списка *)
x:=P^.Info;                                          (* x=2 *)
```

Весьма распространенной является операция поиска в списке узла по некоторому заданному признаку (например, по значению информационного поля). Операция поиска связана с *проходом* по списку, т.е. с последовательной установкой вспомогательного указателя на узлы списка, начиная с первого. Поиск заканчивается либо при обнаружении элемента, либо при достижении конца списка.

```
VAR
 First: PNode; P: PNode; x: CARDINAL;
BEGIN
 P:=First;           (* Установка вспомогательного указателя
                                          на первый элемент списка *)
```

```
WHILE ( P<>NIL ) & ( P^.Info<>x ) DO                    (* Поиск *)
 P:=P^.Link              (* Перестановка указателя на соседний
END;                                                  элемент списка *)
```

Отношение P=NIL предполагает, что P^ не существует, и следовательно, выражение P^.Info не определено. Поэтому порядок проверки условий в цикле WHILE в этом примере не может быть изменен.

Одна из самых простых операций по модификации списка - включение нового узла в его начало (рис.5): элемент типа Node размещается в памяти и ссылка на него присваивается некоторой вспомогательной переменной P; затем устанавливается связь между вставляемым узлом и списком, указатель на первый элемент списка получает новое значение. Программируется это следующим образом:

```
VAR  First,P: PNode;
BEGIN
 ALLOCATE( P,SIZE( Node ) );
 WITH P^ DO Info:=100; Link:=First END;
 First:=P;
```

Р и с. 5. Вставка узла в начало списка

Операция включения узла в начало списка объясняет, как можно

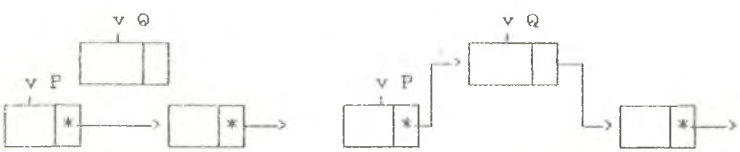
формировать список: начать с пустого списка и последовательно добавлять узлы в начало. Процесс формирования списка из n элементов может быть представлен следующим образом:

```
VAR First,P: PNode;
    n: CARDINAL;          (* Количество элементов в списке *)
BEGIN   n:= ...                                    (* n > 0 *)
 First:=NIL;                          (* В начале список пуст *)
 WHILE n>0 DO
  ALLOCATE (P, SIZE(Node));
  WITH P^ DO Info:=n; Link:=First END; First:=P; DEC(n)
 END;
```

Это один из самых простых способов построения списка.

В некоторых случаях явное использование вспомогательных ссылок намного упрощает операции над списковыми структурами. Пусть, например, необходимо включить в список элемент, на котором установлен указатель Q, после элемента, на который предварительно установлен указатель P. Это выполняется следующей последовательностью операций, которую иллюстрирует рис. 6:

```
VAR P,Q: PNode;      (* Вспомогательные указатели *)
BEGIN  NEW(Q); Q^.Link:=P^.Link; P^.Link:=Q;
```

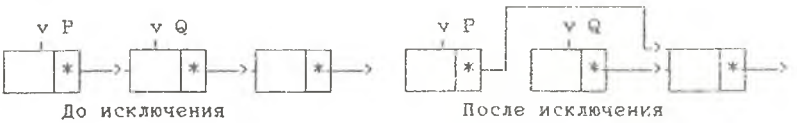


Р и с. 6. Включение в список после элемента P^

Операция исключения из списка элемента за тем элементом, на

который указывает предварительно установленный указатель P, выполняется аналогично и иллюстрируется на рис. 7:

```
VAR P,Q: PNode;
BEGIN  Q:=P^.Link; P^.Link:=Q^.Link;
```

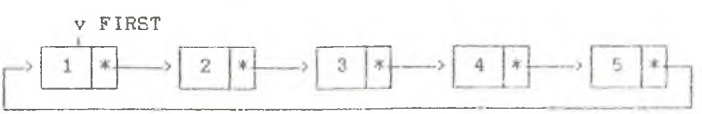


Р и с. 7. Исключение из списка элемента после элемента P^

Если требуется включение/исключение перед указанным элементом или исключение самого указанного элемента, то необходима предварительная установка ссылки на элемент, предшествующий указанному, т.к. в этом случае с целью сохранения структуры списка у узла-предшественника должна быть изменена связь. Для установки указателя на узел, предшествующий данному, необходим проход от начала списка до узла-предшественника, т.к. в элементе односвязного линейного списка нет ссылки на предыдущий узел.

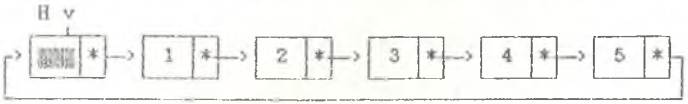
1.5.2. Односвязные циклические списки

Циклически связанный список (сокращенно - циклический список) обладает той особенностью, что связь его последнего узла не равна NIL, а указывает на первый узел списка. Структура односвязного циклического списка показана на рис. 8.

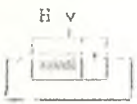


Р и с. 8. Структура односвязного циклического списка

Циклические списки можно использовать не только для представления структур, которым свойственна цикличность, но также для моделирования линейных структур. Иногда для такого моделирования в структуру циклического списка включают специальный узел (элемент) с особым содержанием информационного поля. На рис.9 это поле заштриховано (▒). Пустой циклический список с таким узлом представляется структурой элементарного кольца (рис.10).



Р и с. 9. Структура односвязного циклического списка с особым элементом



Р и с. 10. Элементарное кольцо

Ниже приведена процедура создания односвязного циклического списка из n узлов.

```
TYPE PNode= POINTER TO NODE;
     Node = RECORD  Info: CARDINAL; Link: PNode END;
VAR  Head: Node;                              (* Специальный узел *)
     H: PNode;                      (* Указатель на голову списка *)
     P: PNode;                       (* Вспомогательный указатель *)
     n: CARDINAL;                    (* Количество узлов в списке *)
BEGIN  n:=...;                                          (* n > 0 *)
 H:=ADR( Head ); H^.Link:=H;
                                 (* Создание элементарного кольца *)
```

```
WHILE n>0 DO   (* Создание циклического списка из n узлов *)
 ALLOCATE( P,SIZE( Node ) );
 WITH P^ DO Info:=n; Link:=H^.Link; H^.Link:=P END;  DEC(n)
END;
```

1.5.3. Двусвязные циклические списки

Для достижения большей гибкости в работе с линейными списками, можно включить в каждый узел две связи, указывающие на "левого" и "правого" соседей данного узла:

```
TYPE
PNode= POINTER TO NODE;        (* Указатель на узел списка *)
 Node = RECORD                    (* Описание узла списка *)
  Info: CARDINAL;               (* Информационное поле узла *)
  LLink: PNode;               (* Поле связи с левым соседом *)
  RLink: PNode               (* Поле связи с правым соседом *)
 END;
```

Если VAR P: PNode, то справедливо выражение:

```
P^.LLink^.RLink = P^.RLink^.LLink=P;
```

Для элементарного двусвязного кольца справедливо:

```
P^.LLink = P^.RLink
```

Списки с двумя связями занимают больше памяти, чем односвязные, однако в процессе поиска они дают возможность продвигаться как "вперед", так и "назад" по списку, что может существенно повысить лаконичность программ.

Например, исключение из такого списка узла, заданного указа-

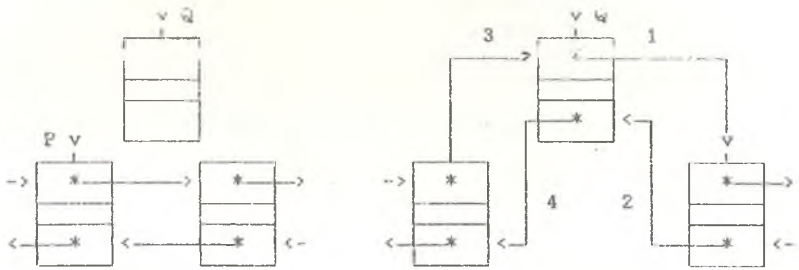
телем P, осуществляется всего двумя операторами:

```
VAR P : PNode;
BEGIN P^.LLink^.RLink:=P^.RLink; P^.RLink^.LLink:=P^.LLink;
```

и не связано с каким-либо "проходом" по списку.

Совершенно аналогично реализуется включение в список нового узла, такое включение связано всего с четырьмя операциями установления связей и иллюстрируется на рис. 11:

```
VAR P,Q: PNode; (* Включение узла Q "справа" от узла P *)
BEGIN  NEW(Q);
 Q^.RLink:=P^.RLink;  (* 1 *)  P^.RLink^.LLink:=Q; (* 2 *)
        P^.RLink:=Q;  (* 3 *)  Q^.LLink:=P;   (* 4 *)
```

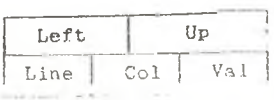


Р и с. 11. Включение узла в двусвязный циклический список

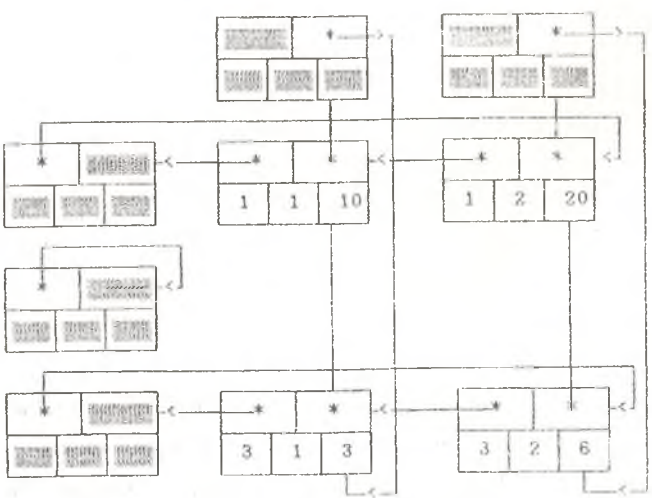
В общем случае один элемент (узел) может быть одновременно включен в несколько различных списков, это порождает множество разнообразных списковых структур. В качестве примера рассмотрим организацию **ортогональных списков**. Их удобно использовать, например, для хранения разреженных матриц большого размера. Такие матрицы содержат большое количество нулевых элементов. Ниже приводится описание типа "Элемент разреженной матрицы", графическая

иллюстрация элемента хранения разреженной матрицы (рис. 12) и графическая иллюстрация ортогональной списковой структуры для хранения матрицы: 10 20 (рис. 13).
0 0
3 6

```
POrt=POINTER TO Ort;
Ort=RECORD                          (* Элемент разреженной матрицы *)
 Val:  CARDINAL;                      (* Значение элемента матрицы *)
 Col:  CARDINAL;                        (* Индекс столбца элемента *)
 Line: CARDINAL;                          (* Индекс строки элемента *)
 Left: POrt;                  (* Указатель на предшествующий узел *)
 Up  : POrt;                          (* Указатель на верхний узел *)
END;
```



Р и с. 12. Элемент хранения разреженной матрицы



Р и с. 13. Структура ортогонального списка

В этом примере для идентификации строк и столбцов матрицы используются специальные особые элементы.

В заключение отметим, что списки, состоящие из элементов (узлов) одного типа, принято называть **однородными**, а разных типов - разнородными.

2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Лабораторная работа выполняется в четыре этапа.

1.Ознакомительный, на котором студент изучает методы представления и обработки динамических структур данных.

2. Подготовительный, на котором составляются алгоритм и программа работы с динамическими структурами данных.

3. Лабораторный, на котором производится отладка программ.

4. Составление отчета.

Отчет должен содержать:

- название лабораторной работы и текст варианта задания;

- листинг программы.

Для сдачи отчета необходимо продемонстрировать работу программы на ЭВМ. Информация, необходимая для выполнения работы и опущенная в тексте задания, должна быть определена самостоятельно.

Требования к программе:

- корректность исходных данных, вводимых с клавиатуры, должна контролироваться Вашей программой;

- работа программы должна иллюстрироваться средствами графического и текстового вывода;

- взаимодействие пользователя с Вашей программой должно осу-

ществляться через систему "меню", обеспечивающую многократное переопределение исходных данных и завершение работы программы.

3. ВАРИАНТЫ ЗАДАНИЙ

1. Найти среднее арифметическое значений информационных полей элементов связанного списка.

2. Подсчитать количество положительных и отрицательных элементов в связанном списке.

3. Построить копию связанного списка.

4. Построить копию связанного списка, изменив порядок составляющих его элементов на обратный.

5. Переставить элементы связанного списка в обратном порядке, начиная с номера N до номера K, не меняя их размещения в памяти ЭВМ.

6. Исключить из связанного списка элементы, начиная с номера N до номера K.

7. Исключить из связанного списка все элементы между двумя элементами с заданными значениями информационных полей.

8. Элементы связанного списка хранят слова, состоящие из 10 символов, включающих только русские и английские большие и малые буквы. Исключить из связанного списка слова, начинающиеся с заданной комбинации символов.

9. Выполнить задание 8, исключая слова, содержащие заданную комбинацию символов.

10. Из исходного списка получить два новых списка, не меняя размещения элементов в памяти: 1-й список должен содержать элементы, у которых значение информационного поля больше заданного

значения N; 2-й - все остальные элементы.

11. Из исходного списка получить два новых списка путем копирования: 1-й список должен содержать слова, начинающиеся с заданной комбинации символов; 2-й список - заканчивающиеся заданной комбинацией символов.

12. Вставить в список новый элемент перед каждым элементом с заданным значением информационного поля.

13. Объединить два связанных списка в один путем копирования в следующем порядке: 1-й элемент 1-го списка, 1-й элемент 2-го списка; 2-й элемент 1-го списка, 2-й элемент 2-го списка и т.д.

14. Объединить два связанных списка в один (без использования копирования) в следующем порядке: 1-й элемент 1-го списка, 1-й элемент 2-го списка; 2-й элемент 1-го списка, 2-й элемент 2-го списка и т.д.

15. Реализовать представление многочлена

$$P(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1} + a_nx^n$$

с произвольными целыми коэффициентами в виде связанного списка. При этом, если $a_i=0$, то соответствующий элемент-слагаемое должен отсутствовать в списке.

Написать процедуру, проверяющую два многочлена на равенство, и процедуру вычисления значения многочлена.

16. Объединить два списка, упорядоченных отношением "<", в один путем изменения связей между элементами.

17. Реализовать представление многочлена, описанное в пункте 15, и написать процедуру сложения двух многочленов. Результатом такого сложения должен быть новый многочлен-сумма.

18. Реализовать представление многочлена, описанное в пункте 15, и написать процедуру умножения двух многочленов. Результатом

такого умножения должен быть новый многочлен-произведение.

19. Два связанных списка заданы указателями на их первые элементы F1 и F2. Определить, не содержится ли список F1 в F2, F2 в F1, и подсчитать число вхождений первого списка во второй и второго в первый.

20. Три связанных списка заданы указателями на их первые элементы F, F1 и F2. Написать процедуру, которая заменяет в списке F каждое вхождение списка F1 (если такое есть) на список F2.

21. Реализуйте моделирование очереди элементов ограниченной длины с дисциплиной "первым пришел - первым вышел" на структуре циклического списка. Моделирование связано с постановкой новых элементов в очередь, выводом из очереди и идентификацией ситуаций "Очередь полна" и "Очередь пуста".

22. Реализуйте на ортогональных списках процесс составления кроссвордов.

23. Реализуйте представление бесскобочного алгебраического выражения вида x + y * z + ... + m = f списковой структурой, в которой элементами являются переменные (x,y, и т.д.), операции (+,-,*,/) и отношение (=). На основе этого представления разработайте алгоритм разрешения исходного выражения относительно любой входящей в него переменной. Результатом такого разрешения должна быть трансформированная списковая структура, например, вида (разрешение относительно x): x = - y * z - ... - m.

Разработайте процедуру вычисления значения разрешенной переменной.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое ссылочный тип? Какие виды указателей и действия над указателями вам известны?

2. В чем отличия принципа вычисляемого адреса от принципа хранимого адреса?

3. Определите понятие линейной динамической структуры данных.

4. Сравните последовательное и связанное представление структур данных.

5. Какие преимущества дает циклическая организация линейных списков?

6. Чем отличается организация и обработка двусвязных линейных списков от односвязных?

7. Каковы особенности использования оператора присоединения к указателю?

8. Каковы особенности организации "проходов" по спискам?

9. Каковы особенности реорганизации списков с использованием процедур управления динамической памятью и без такого использования?

10. Входит ли значение константы NIL в рабочее пространство адресов оперативной памяти?

11. К какому виду объектов относится переменная типа ADDRESS: к структурному или указательному?

12. Какова мощность множества констант адресного типа, если переменная этого типа хранится в одном слове размером в 16 бит?

13. Чем отличается адресная арифметика от арифметики кардинальных чисел?

14. С какой дополнительной проверкой связан доступ к объекту

через указатель?

15. В чем преимущества двусвязной реализации списка перед односвязной? В чем недостатки?

16. В каком списке элемент является предшественником самого себя? В каком - последователем самого себя?

17. Можно ли реализовать участие одного элемента в двух разных списках с помощью одной ссылки? В каких случаях?

18. VAR P: ADDRESS; К какой структуре откроет доступ оператор присоединения WITH P DO ...END ? К какой WITH P^ DO ...END ?

Динамические структуры данных. Списки

Составители: Кораблин Михаил Александрович

Симонова Елена Витальевна

Редактор Л.Я.Чегодаева

Техн. редактор Г.А.Усачева

Подписано в печать 29.06.94. Формат 60 x 84¹/16.

Бумага офсетная. Печать офсетная.

Усл.печ.л. 1.6. Усл.кр.-отт. 1.7. Уч.-изд.л. 1.5.

Тираж 150 экз. Заказ 256. . Арт.С - 105 / 94.

Самарский государственный аэрокосмический

университет имени академика С.П.Королева.

443086 Самара Московское шоссе, 34.

ИПО Самарского государственного аэрокосмического

университета. 443001 Самара, ул.Ульяновская, 18.