Глава 10. УКАЗАТЕЛИ И ДИНАМИЧЕСКАЯ ПАМЯТЬ

- Динамическая память
- Типизированные и нетипизированные указатели
- Процедуры работы с указателями
- Использование динамической памяти для размещения данных большого объема
- Динамические структуры
- Создание и работа со списком

ДИНАМИЧЕСКАЯ ПАМЯТЬ, УКАЗАТЕЛИ

В Delphi Pascal есть средства, которые позволяют использовать **динамическую память** (heap-область, куча) под размещение данных большой размерности, при организации динамических структур и для других целей. При этом происходит **динамическое размещение** данных, что означает использование области динамической памяти непосредственно во время работы (при динамическом размещении заранее не известен ни тип, ни количество размещаемых данных). Средство управления динамической памятью – **указатели**.

Указатель – это переменная, которая в качестве своего значения содержит адрес байта памяти. Размер указателя равен 4 байтам.

Типизированный указатель (ссылка) связывается с некоторым типом данных. Для объявления типизированного указателя используется значок ^, который помещается перед соответствующим типом:

```
Var   PInt : ^Integer;  PReal : ^Real;
```

Нетипизированные указатели (указатели) хранят просто адреса, которые не связаны с данными конкретных типов:

```
Var   P : Pointer;
```

Nil – «нулевой» адрес.

РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Динамическое распределение памяти происходит следующим образом:

- прикладная программа запрашивает у системы фрагмент памяти определенного размера;
- если в вычислительной машине есть свободная память требуемого объема, то система выделяет этот фрагмент;
- система передает прикладной программе указатель на эту непрерывную область в памяти;
- прикладная программа получает указатель;
- прикладная программа размещает данные в выделенной области, начиная с указанного адреса;
- по окончании работы с этими данными прикладная программа обязана сообщить системе о том, что выделенная область памяти может быть снова получена системой, иначе говоря, освободить выделенный фрагмент.

РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Память под динамически размещаемую переменную во время работы программы выделяется процедурой **NEW**. Процедура

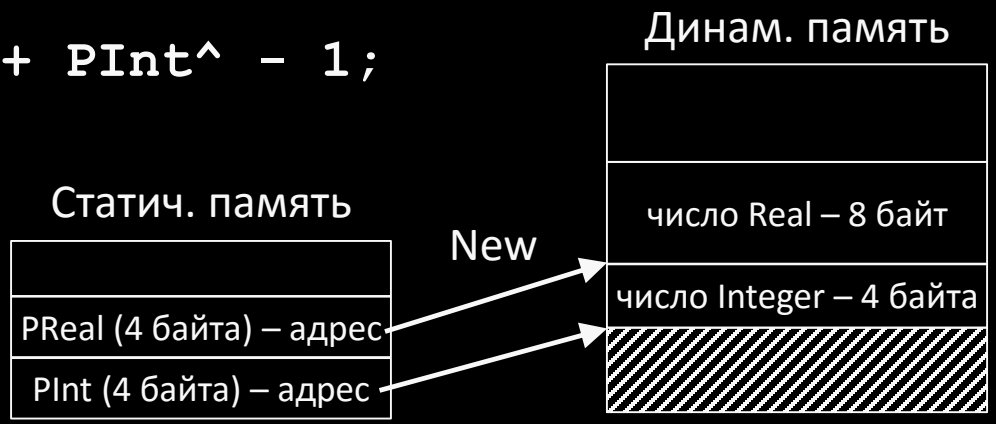
```
New(Var <типизированный указатель>);
```

возвращает адрес выделенного участка памяти через параметр-переменную. Размер участка памяти определяется базовым типом указателя.

```
Var
    PInt : ^Integer;
    PReal : ^Real;
Begin
    New(PInt);
    New(PReal);
    PInt^ := 2;
    PReal^ := 2*pi;
    PReal^ := Sqr(PReal^) + PInt^ - 1;
End.
```

```
{New – функция}
 Type TyPInt : ^Integer;
 Var  PInt : TyPInt;…
 PInt := New(TyPInt);…
```

После процедуры New в куче можно разместить значение соответствующего типа. Используется значок ^ – **разыменование указателя**.



РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Передавать значения между указателями можно в том случае, если они связаны с одним и тем же типом данных (либо один из них – нетипизированный указатель или "нулевой адрес" Nil) .

```
Type
      TType = (red, green, blue);
      PType = ^TType;
Var
      P1,P2 : PType;
      PInt : ^Integer; P : Pointer;
Begin
      New(P1); New(P2);
      P1^ := red; P2^ := green;
      P1 := P2;
      ...
      PInt := P; P := P1; P2 := Nil;
      Writeln(Cardinal(P1));
End.
```

При выполнении операции присваивания теряется участок памяти, на который ссылался указатель, стоящий слева от оператора присваивания. Это типичный случай создания "мусора" (garbage) в динамической памяти.

РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Процедура

```
Dispose(<типизированный указатель>);
```

освобождает память по адресу, хранящемуся в указателе. При этом не изменяется значение указателя – текущей границы незанятой динамической памяти. Поэтому чередование обращений к процедурам New и Dispose обычно приводит к **фрагментации** памяти – память разбивается на небольшие фрагменты с чередованием свободных и занятых участков.

При работе с нетипизированными указателями, в основном, используются другие процедуры:

```
GetMem(Var P : Pointer, Size : Word)
                        {резервирование памяти}
FreeMem(P : Pointer, Size : Word)
                        {освобождение памяти}
```

где P – нетипизированный указатель, Size – размер в байтах требуемой или освобождаемой части кучи.

Использование процедур GetMem/FreeMem (как и вообще вся работа с динамической памятью) требует особой осторожности и соблюдения правила: освобождать нужно ровно столько памяти, сколько ее было зарезервировано, и именно с того адреса, с которого она была зарезервирована.

РАЗМЕЩЕНИЕ ДАННЫХ БОЛЬШОГО ОБЪЕМА В ДИНАМИЧЕСКОЙ ПАМЯТИ

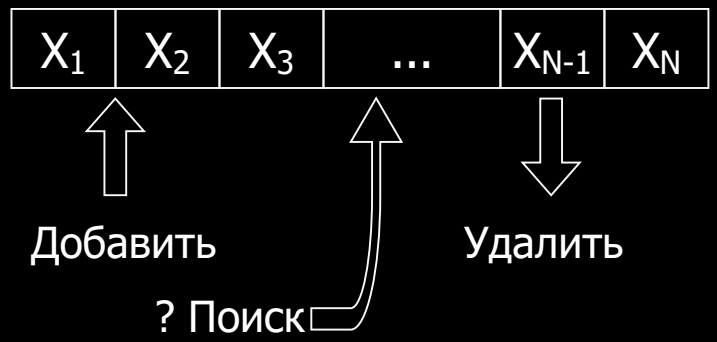
```
Type  Dim100x200 = array [1..100,1..200] of Real;
```

Целесообразно «большие» массивы размещать в динамической памяти.

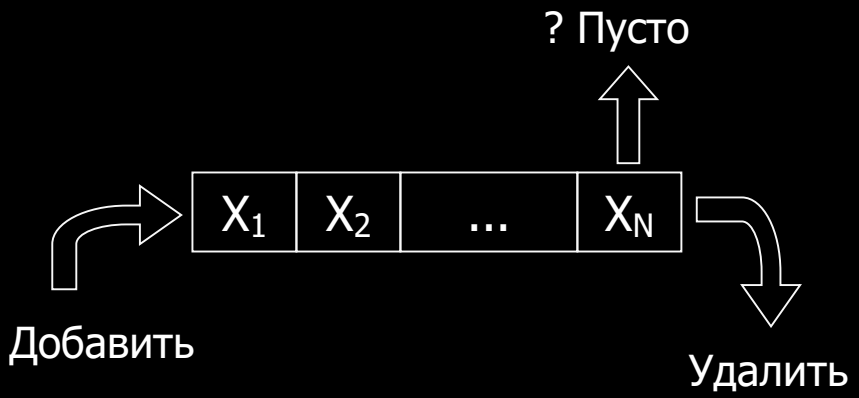
```
Type
   Dim100 = array [1..100] of Real;
            {строка-массив размером 800 байт}
   Dim100Ptr = ^Dim100;
            {указатель на строку размером 4 байта}
   Dim100x200 = array [1..200] of Dim100Ptr
            {массив указателей размером 800 байт}
Var
   D : Dim100x200;    I,J : Byte;
Begin
   for I := 1 to 200 do New(D[i]);
…  D[I]^[J] := 0.5;     {I = 1..200, J = 1..100}
End.
```

ДИНАМИЧЕСКИЕ СТРУКТУРЫ

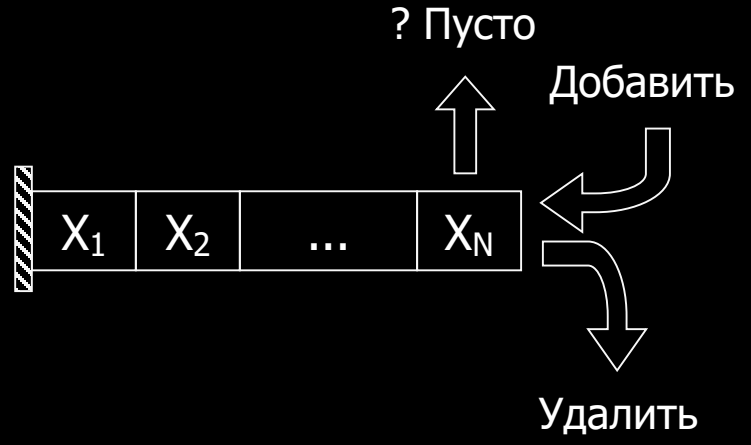
- **Список** – упорядоченный набор элементов одного типа. Размер списка может изменяться. Элемент списка (любой динамической структуры) состоит из двух частей: информационной, содержащей данные, и адресной, где хранятся указатели на соседние элементы.



- **Очередь** – список, в один конец которого элементы добавляются, а из другого изымаются. Очередь – это устройство FIFO (First In, First Out).

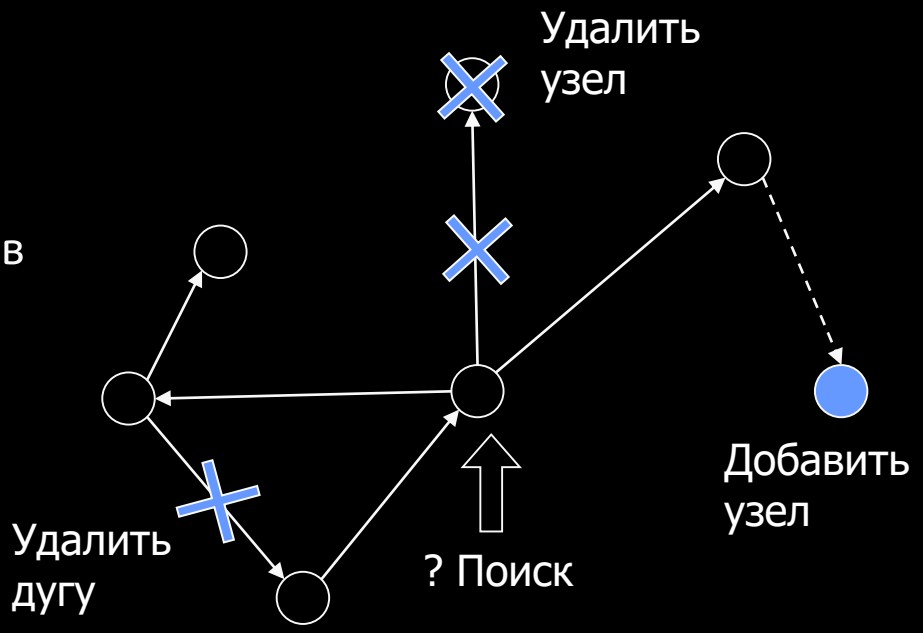


- **Стек** – список, в один конец которого элементы добавляются, и из него же изымаются. Стек – это устройство LIFO (Last In, First Out).

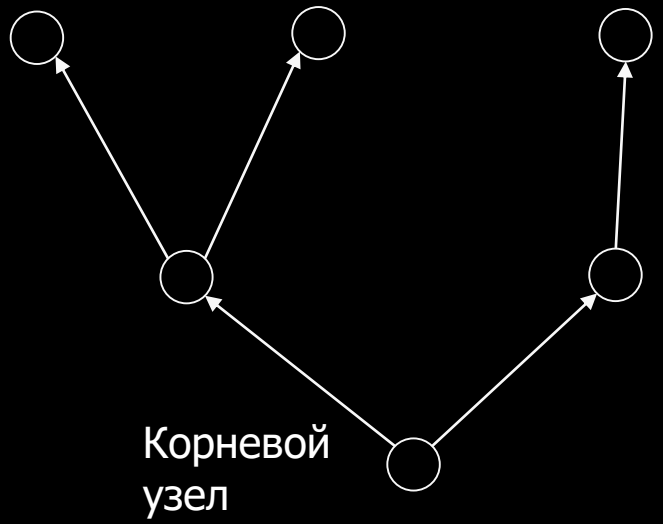


ДИНАМИЧЕСКИЕ СТРУКТУРЫ

- **Граф** – структура, состоящая из узлов и дуг, каждая дуга направлена от одного узла к другому.



- **Дерево** – направленный граф, у которого имеется корневой узел, не имеющий дуг, входящих в него; в каждый узел входит одна дуга; в каждый узел можно попасть из корневого за несколько шагов.



СОЗДАНИЕ ЛИНЕЙНОГО ОДНОСВЯЗНОГО СПИСКА

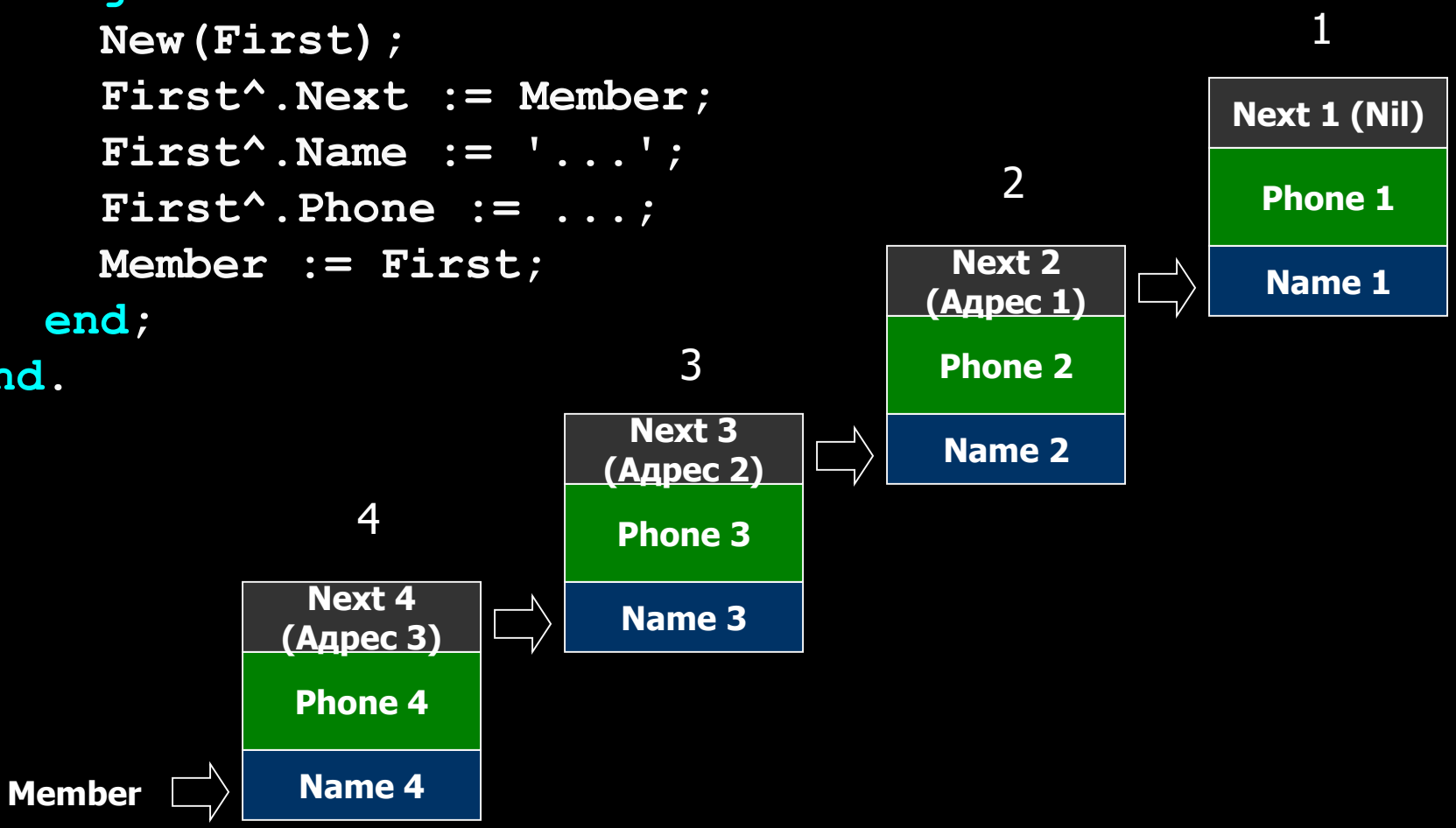
```
Type
    PMemberType = ^MemberType;
    MemberType = record
                 Name : String;
                 Phone : Word;
                 Next : PMemberType
                 end;
Var
    First, Member : PMemberType;
```

Указатель на тип MemberType описан до того, как описан сам тип MemberType. В Delphi Pascal такое предварительное использование идентификатора типа разрешено только при создании указателя на этот тип.

В запись MemberType включено поле Next, которое представляет собой указатель на запись MemberType. Это есть ссылка на соседний элемент списка.

СОЗДАНИЕ ЛИНЕЙНОГО ОДНОСВЯЗНОГО СПИСКА

```
Begin
    Member := Nil; {неопределенный указатель –
                    показывает, что память для Member не выделена}
    for I := 1 to 4 do
    begin
      New(First);
      First^.Next := Member;
      First^.Name := '...';
      First^.Phone := ...;
      Member := First;
    end;
End.
```



УНИЧТОЖЕНИЕ СПИСКА

```
while Member <> Nil do
   begin
      Element := Member^.Next;
      Dispose(Member);
      Member := Element
   end;
```

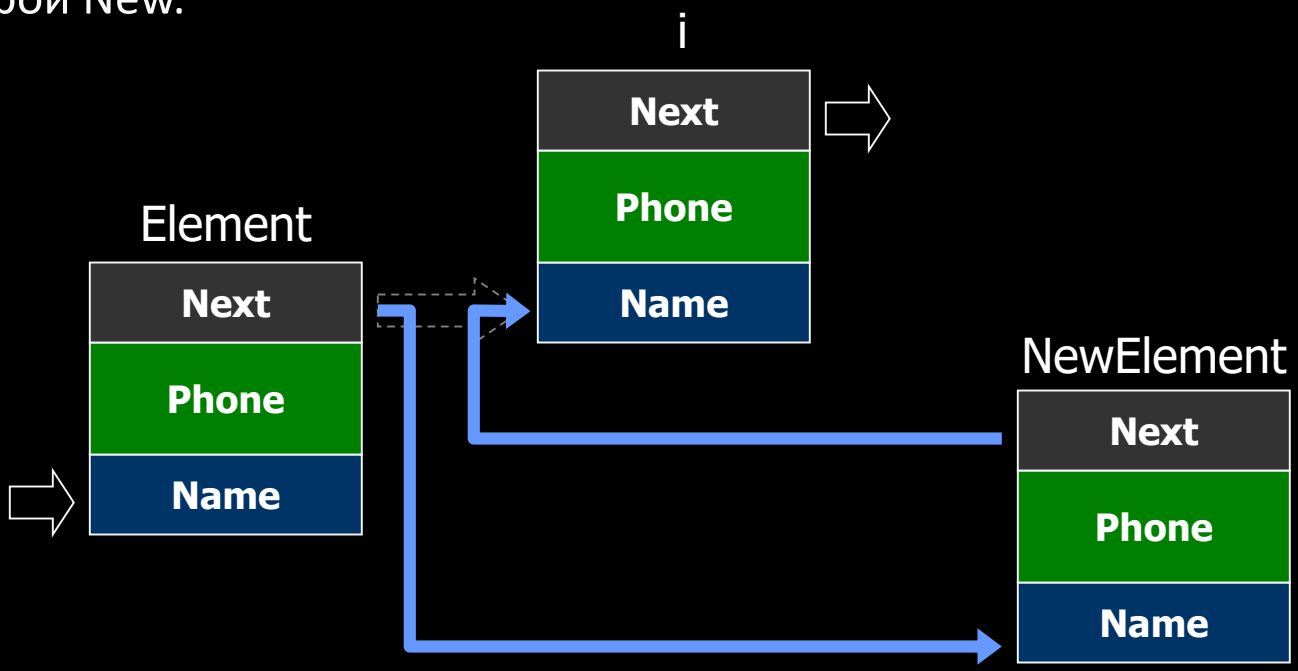
Перед удалением необходимо запомнить ссылку на следующий элемент (в буфер Element), т.к. в противном случае вся информация об оставшемся куске списка будет утеряна. Следует использовать цикл while…do, т.к. заранее не известно число повторений, и кроме того список может быть пустым.

ДОБАВЛЕНИЕ НОВОГО ЭЛЕМЕНТА В СПИСОК

после элемента Element, предварительно найденного по некоторому признаку:

```
NewElement^.Next := Element^.Next;
NewElement^.Name := '...';
NewElement^.Phone := ...;
Element^.Next := NewElement;
```

Предполагается, что NewElement имеет тип PMemberType, и был заранее создан процедурой New.



Для удобства поиска нужного элемента целесообразно формировать список в отсортированном (по некоторому полю) виде.