



15.11.2018 г
тема урока :

Динамические структуры данных.

Подготовила
учитель информатики:
Радова А.Ф.
Теоретический молдо-турецкий
лицей им.С.Демиреля
С.Конгаз

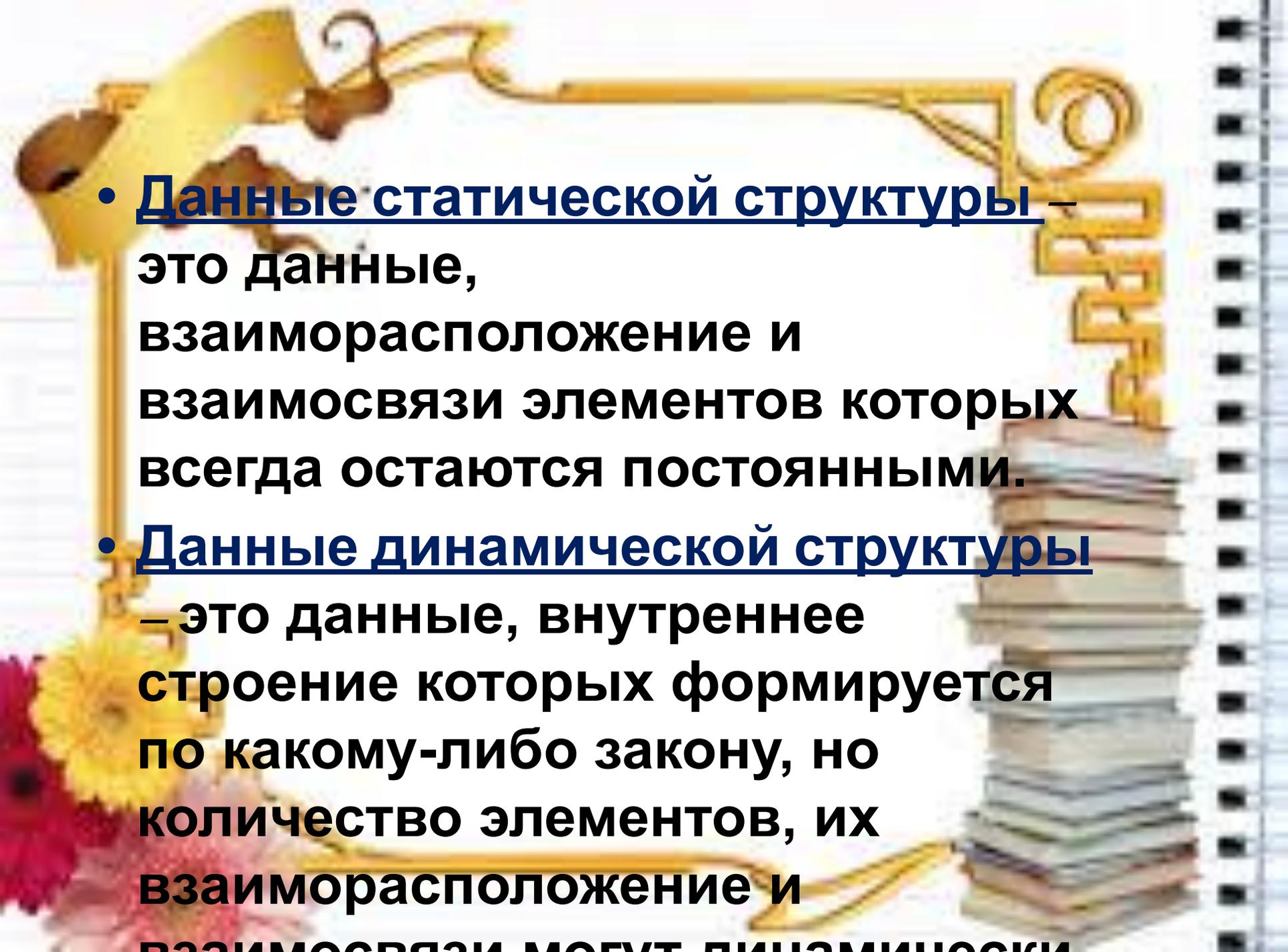


Объект данных обладает динамической структурой, если его размер изменяется в процессе выполнения программы или он потенциально бесконечен.

- **Классификация структур данных**

Используемые в программировании данные можно разделить на две большие группы:





• Данные статической структуры – это данные, взаиморасположение и взаимосвязи элементов которых всегда остаются постоянными.

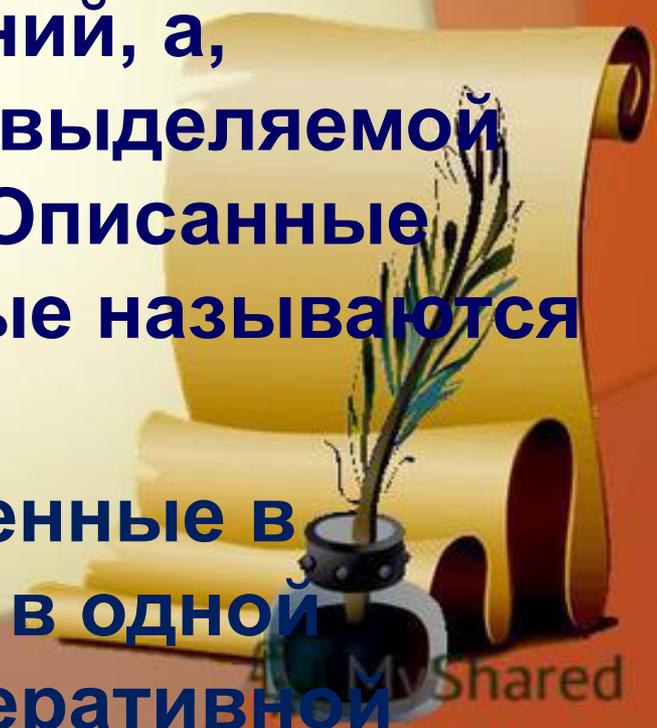
• Данные динамической структуры – это данные, внутреннее строение которых формируется по какому-либо закону, но количество элементов, их взаиморасположение и взаимосвязи могут динамически

Данные динамической структуры

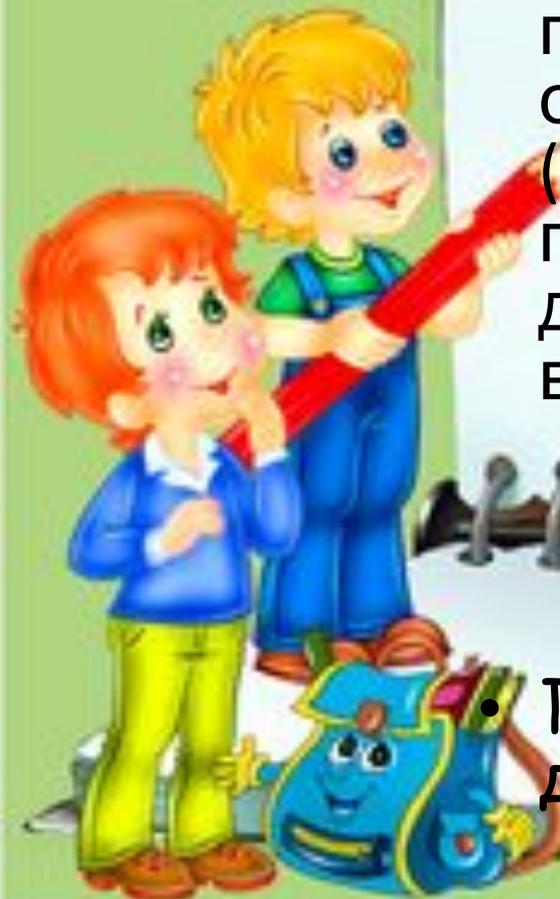


Статические и динамические переменные в Паскале

- В Паскале одной из задач описания типов является то, чтобы зафиксировать на время выполнения программы размер значений, а, следовательно, и размер выделяемой области памяти для них. Описанные таким образом переменные называются статическими.
- Все переменные, объявленные в программе, размещаются в одной непрерывной области оперативной

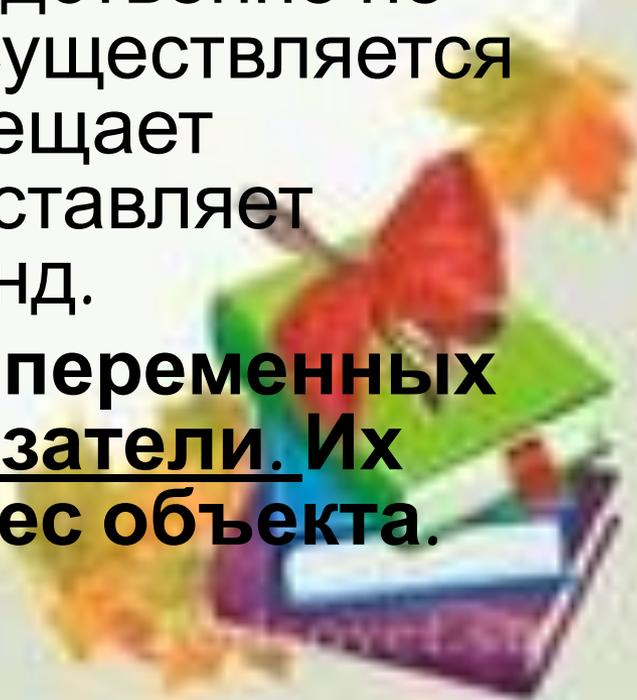


- 
- **Динамическая память (ДП)** – это оперативная память ПК, предоставляемая программе при ее работе, за вычетом сегмента данных (64 Кб), стека (16 Кб) и собственно тела программы. Размер динамической памяти можно варьировать.

- 
- По умолчанию ДП - вся доступная память ПК.

ДП – это фактически единственная возможность обработки массивов данных большой размерности. Многие практические задачи трудно или невозможно решить без использования ДП.

- И статические и динамические переменные вызываются по их адресам. Без адреса не получить доступа к нужной ячейке памяти, но при использовании статических переменных, адрес непосредственно не указывается. Обращение осуществляется по имени. Компилятор размещает переменные в памяти и подставляет нужные адреса в коды команд.
- **Адресация динамических переменных осуществляется через указатели. Их значения определяют адрес объекта.**



Для работы с динамическими переменными в программе должны быть выполнены

следующие действия:

- *Выделение памяти под динамическую переменную;*
- *Инициализация указателя;*
- *Освобождение памяти после использования динамической переменной.*
- Программист должен сам резервировать место, определять значение указателей, освободить ДП.
- Вместо любой статической переменной можно использовать динамическую, но без реальной необходимости этого делать не стоит.

Указатели

- Для работы с динамическими программными объектами в Паскале предусмотрен ссылочный тип или тип указателей. В переменной ссылочного типа хранится ссылка на программный объект (адрес объекта).

Указатель – это переменная, которая в качестве своего значения содержит адрес байта памяти.



Объявление указателей

- Указатель, связанный с некоторым определенным типом данных, называют **типизированным указателем**. Его описание имеет вид:

Имя_переменной: ^ базовый-тип;

- Например:

Пример фрагмента программы объявления указателя:

```
Type A= array [1..100] of integer;
```

```
TA= ^ A ; {тип указатель на массив}
```

```
Var
```

```
P1: ^ integer; {переменная типа указатель на  
целое число}
```

```
P2: ^ real; {переменная типа указатель на  
вещественное число}
```

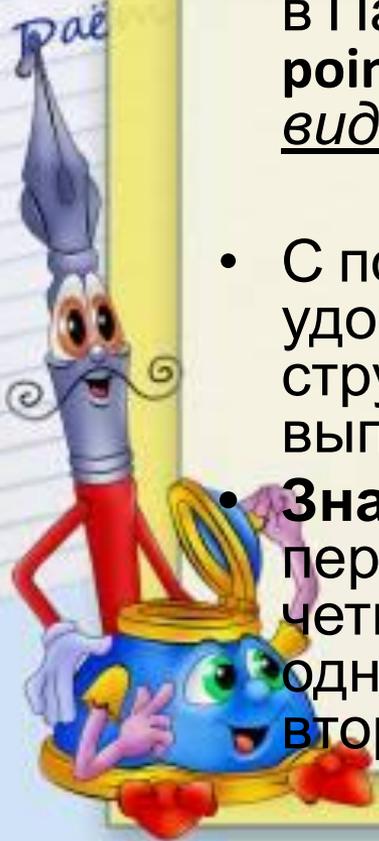


Указатель, не связанный с каким-либо конкретным типом данных, называется нетипизированным указателем.

- Для описания нетипизированного указателя в Паскале существует стандартный тип **pointer**. Описание такого указателя имеет вид:

Имя-переменной: pointer;

- С помощью нетипизированных указателей удобно динамически размещать данные, структура и тип которых меняются в ходе выполнения программы.
- **Значения указателей** – это адреса переменных в памяти. Адрес занимает четыре байта и хранится в виде двух слов, одно из которых определяет сегмент, второе – смещение.

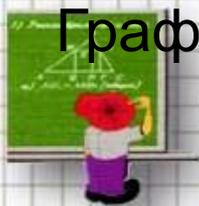


- **Выделение памяти под динамическую переменную осуществляется процедурой:**

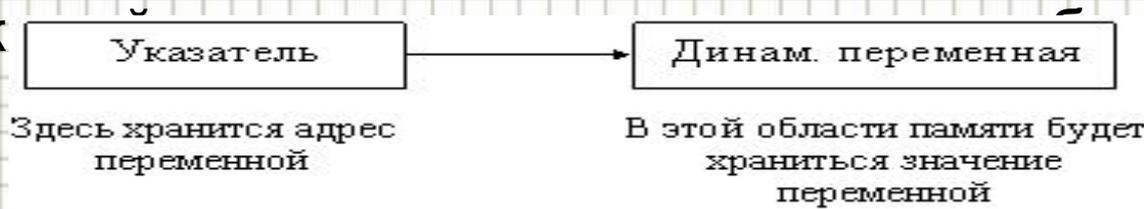
New (переменная_типа_указатель)

- В результате обращения к этой процедуре указатель получает значение, соответствующее адресу в динамической памяти, начиная с которого можно разместить данные.
- *Пример фрагмента программы объявления указателя различных типов:*

```
Var i, j: ^integer;  
    r: ^real;  
begin  
    new( i);  
    .....  
    new( r);
```



Графическ



зитель так:

- Освобождение динамической памяти осуществляется процедурой:

Dispose

(переменная_типа_указатель);

- Пример фрагмента программы процедуры *Dispose*:

Dispose (i);

Dispose (r);

- Следует помнить, что повторное применение процедуры *dispose* к свободному указателю может привести к ошибке.

- Процедура *dispose* освобождает память, занятую динамической переменной. При этом значение указателя становится

неопределенным.



Операции с указателями

Для указателей определены **только операции присваивания и проверки на равенство и неравенство**. В Паскале запрещаются любые арифметические операции с указателями, их ввод-вывод и сравнение на больше-меньше.

Правила присваивания указателей:

- любому указателю можно присвоить стандартную константу nil, которая означает, что указатель не ссылается на какую-либо конкретную ячейку памяти;
- указатели стандартного типа pointer совместимы с указателями любого типа;
- указателю на конкретный тип данных можно присвоить только значение указателя того же или стандартного типа данных.

Указатели можно сравнивать на равенство и неравенство, например:

If p1=p2 then

If p1<>p2 then



В Паскале определены стандартные функции для работы

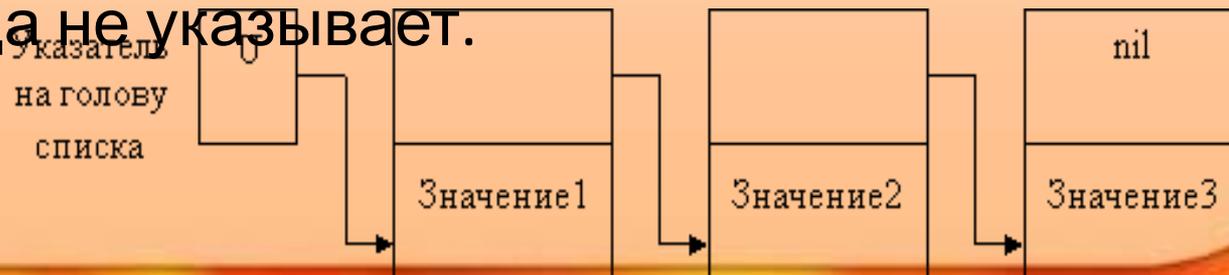
с указателями:

- **addr(x)** – тип результата `pointer`, возвращает адрес `x` (аналогично операции `@`), где `x` – имя переменной или подпрограммы;
- **seg(x)** – тип результата `word`, возвращает адрес сегмента для `x`;
- **ofs(x)** – тип результата `word`, возвращает смещение для `x`;
- **ptr(seg, ofs)** – тип результата `pointer`, по заданному сегменту и смещению формирует адрес типа `pointer`.



Динамические структуры

- Списком называется структура данных, каждый элемент которой посредством указателя связывается со следующим элементом.
- Каждый элемент связанного списка, во-первых, хранит какую-либо информацию, во-вторых, указывает на следующий за ним элемент. Так как элемент списка хранит разнотипные части (храняемая информация и указатель), то его естественно представить записью, в которой в одном поле располагается объект, а в другом – указатель на следующую запись такого же типа. Такая запись называется звеном, а структура из таких записей называется списком или цепочкой.
- Лишь на самый первый элемент списка (голову) имеется отдельный указатель. Последний элемент списка никуда не указывает.



Описание списка:

```
Type ukazat= ^ S;
```

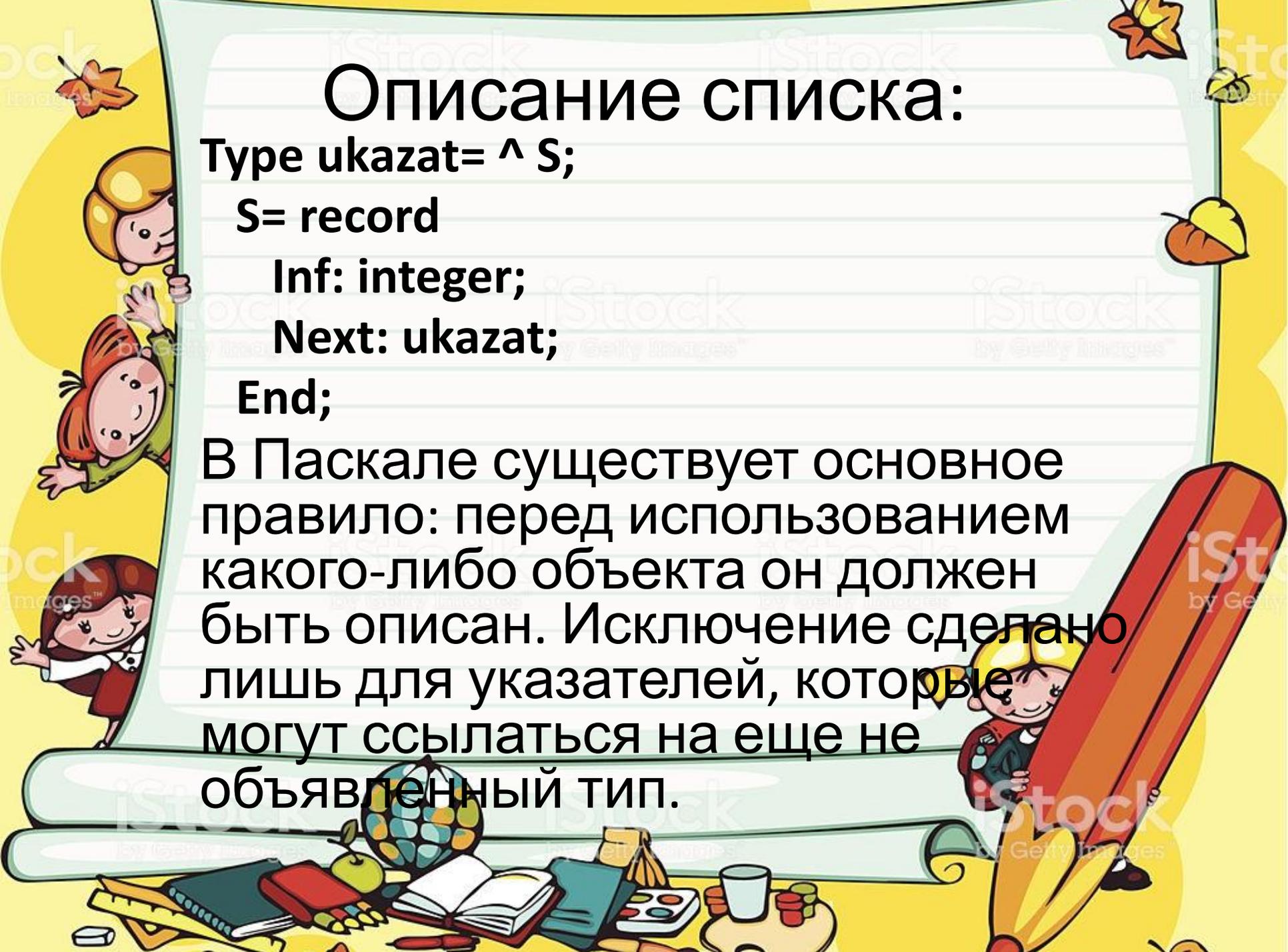
```
S= record
```

```
  Inf: integer;
```

```
  Next: ukazat;
```

```
End;
```

В Паскале существует основное правило: перед использованием какого-либо объекта он должен быть описан. Исключение сделано лишь для указателей, которые могут ссылаться на еще не объявленный тип.



Формирование списка:

- Чтобы список существовал, надо определить указатель на его начало.

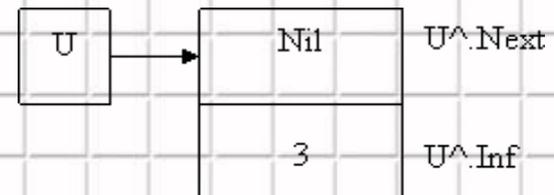
```
Type ukazat= ^S;
```

```
S= record
```

```
  Inf: integer;
```

```
  Next: ukazat;
```

```
End;
```



Создадим первый элемент списка:

```
New (u); {выделяем место в памяти}
```

```
u^. Next:= nil; {указатель пуст}
```

```
u^. Inf:=3;
```



Продолжим формирование списка. Для этого нужно добавить элемент либо в конец списка, либо в голову.

А) Добавим элемент в голову списка. Для этого необходимо выполнить последовательность действий:

- получить память для нового элемента;
- поместить туда информацию;
- присоединить элемент к голове списка.

New(x);

Readln(x^.Inf);

x^. Next:= u;

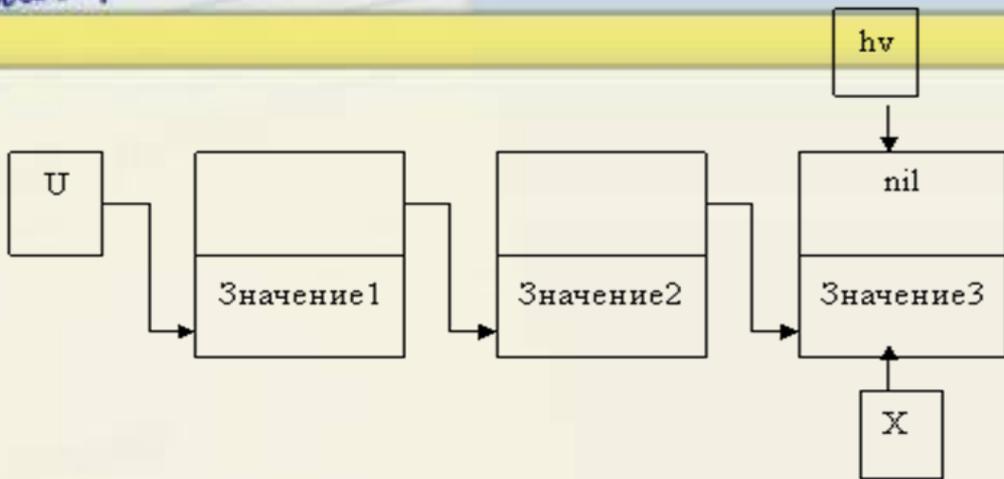
u:= x;

Б) Добавление элемента в конец списка. Для этого введем вспомогательную переменную, которая будет хранить адрес последнего элемента. Пусть это будет указатель с именем hv (хвост).

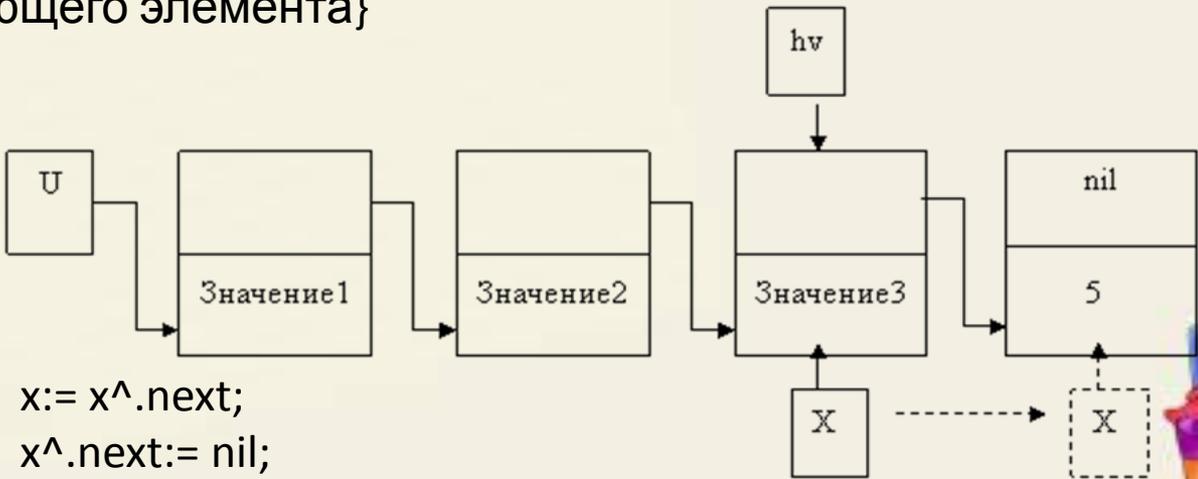
x:= hv;



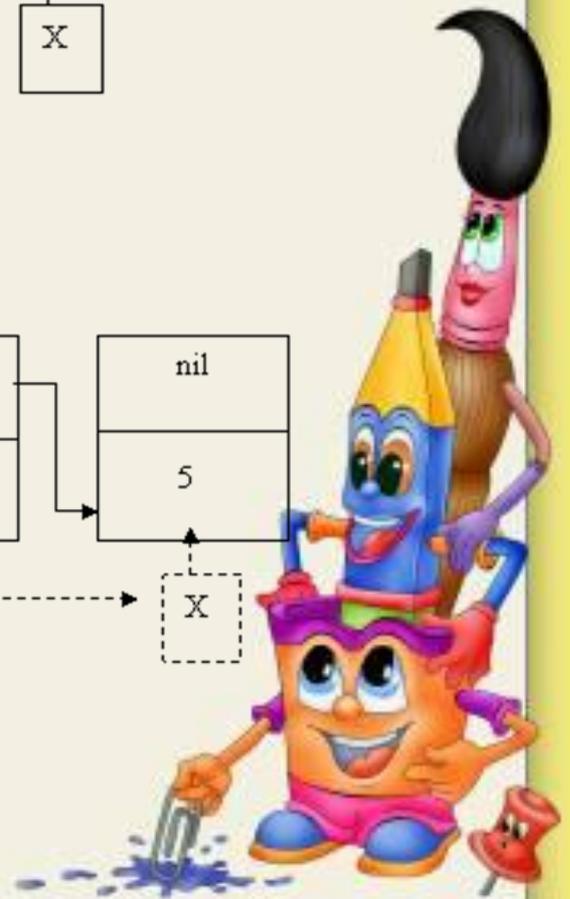
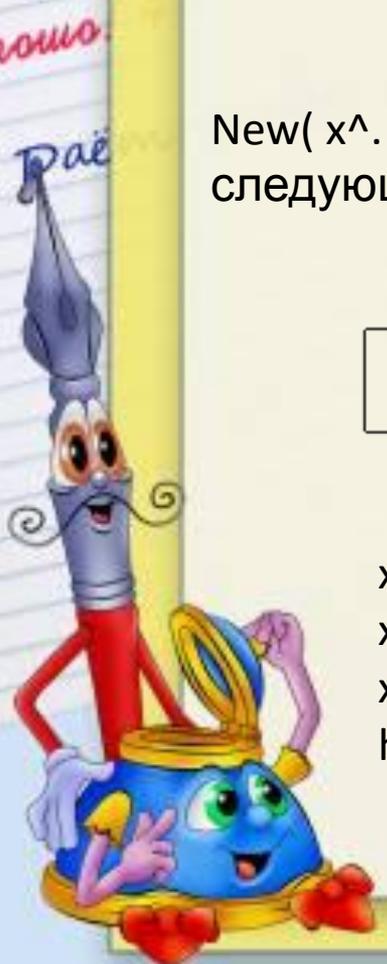
Указатель
на голову
списка



New(x^. next); {выделяем память для
следующего элемента}



```
x:= x^.next;  
x^.next:= nil;  
x^.inf:= 5;  
hv:=x;
```



The background features a warm, autumnal theme. On the left, there is an illustration of an open book with a blue bookmark. The scene is decorated with several maple leaves in shades of red, orange, and yellow. A large, glowing yellow arc frames the top and right sides of the page. The bottom edge has a red border with white snowflake-like patterns.

• Просмотр списка:

```
While u<> nil do
```

```
Begin
```

```
Writeln (u^.inf);
```

```
u:= u^.next;>
```

```
end;
```

Удаление элемента из списка

- А) Удаление первого элемента. Для этого во вспомогательном указателе запомним первый элемент, указатель на голову списка переключим на следующий элемент списка и освободим область динамической памяти, на которую указывает вспомогательный указатель.

```
x:= u;  
u:= u^.next;  
dispose(x);
```

- Б) Удаление элемента из середины списка. Для этого нужно знать адреса удаляемого элемента и элемента, стоящего перед ним. Допустим, что digit – это значение удаляемого элемента.

```
x:= u;  
while ( x<> nil) and ( x^. inf<> digit) do  
begin  
dx:= x;  
x:= x^.next;  
end;  
dx:= x^.next;  
dispose(x);
```

- В) Удаление из конца списка. Для этого нужно найти предпоследний элемент.

```
x:= u; dx:= u;  
while x^.next<> nil do  
begin  
dx:= x; x:= x^.next;  
end;  
dx^.next:= nil;  
dispose(x);
```



Прохождение списка.

Важно уметь перебирать элементы списка, выполняя над ними какую-либо операцию. Пусть необходимо найти сумму элементов списка.

```
summa:= 0;  
x:= u;  
while x<> nil do  
begin  
summa:= summa+ x^.inf;  
x:= x^.next;  
end;
```



- **Стек** – особый вид списка, обращение к которому идет только через указатель на первый элемент. Если в стек нужно добавить элемент, то он добавляется впереди первого элемента, при этом указатель на начало стека переключается на новый элемент. Алгоритм работы со стеком характеризуется правилом: «последним пришел – первым вышел».
- **Очередь** – это вид списка, имеющего два указателя на первый и последний элемент цепочки. Новые элементы записываются вслед за последним, а выборка элементов идет с первого. Этот алгоритм типа «первым пришел – первым вышел».
- Возможно организовать списки с произвольным доступом к элементам. В этом случае необходим дополнительный указатель на текущий элемент.

