УНИВЕРСИТЕТ СИНЕРГИЯ | БОЛЬШЕ ЧЕМ ОБРАЗОВАНИЕ

Основа алгоритмизации и программирования

«Указатели и динамические структуры»

Практическое занятие
Лекция

Кафедра цифровой экономики

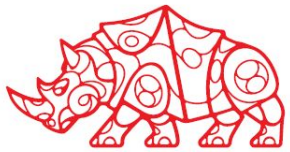
Гринева Е.С.,преподаватель

22 ноября 2022 г.

УНИВЕРСИТЕТ СИНЕРГИЯ

Цель занятия:

- **Изучить:**
- **Что такое указатели**
- **Описание указателей**
- **Массив указателей**
- **Динамические структуры данных**



В Турбо-Паскале есть два способа распределения памяти: *статический* и *динамический*.

Указатель – это элемент данных, представляющий собой ссылку на определенную ячейку оперативной памяти (т.е. адрес ячейки), начиная с которой записывается значение переменной.

Описание указателей

1)

- Type имя_типа = ^тип;
- Var имя_переменной = имя_типа;

2)

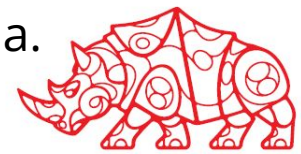
- Var имя_переменной: ^имя_типа;

3)

- Var p: Pointer;

Допустимо также использование оператора присваивания, при этом в правой части может находиться либо функция определения адреса **Addr(X)**, либо выражение **@X**, где **@** — унарная операция взятия адреса, **X** – имя переменной любого типа, в том числе процедурного.

Для динамических переменных (**A^**, **B^**, **C^**) допустимы все те же операции, что и над обычными переменными данного типа.



Стандартные процедуры и функции для работы с указателями

Процедуры:

1. Любым действиям с динамической переменной должна предшествовать процедура ее размещения в ОЗУ. Эта процедура имеет вид: **New (Var p: Pointer)**.
2. Когда в ходе вычислительного процесса переменная становится ненужной, ее следует удалить. Это осуществляется процедурой **Dispose (Var p: Pointer)**. Type Str = string[6];

- Var P = ^Str;
- Begin
- **New**(P);
- P^:= ?Hellow?;
- **Dispose**(P)
- End.

3. Процедура **GetMem (Var p: Pointer, Size: Word)**
4. Процедура **FreeMem (Var p: Pointer, Size: Word)**
5. Процедура **Mark (Var p: Pointer)**
6. Процедура **Mark (Var p: Pointer)**

Пример:

- Var
- p: pointer;
- p1, p2, p3: ^Integer;
- Begin
- **New**(p1); {Создает указатель на переменную типа Integer}
- **Mark** (p); {Запоминает состояние (все дальнейшие указатели будут размещаться за указателем p)}
- **New**(p2); {Создает указатели еще на две переменные типа Integer}
- **New**(p3);
- **Release** (p) {Память, выделенная для p2^, p3^ освобождена, но p1^ все еще можно использовать}

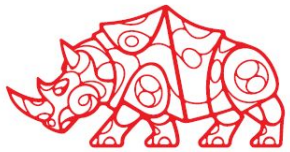
Функции:

MaxAvail: LongInt – возвращает длину в байтах самого длинного свободного участка динамической памяти.

MemAvail: LongInt – возвращает размер свободной области динамической памяти (в байтах).

Addr(X): Pointer – возвращает адрес объекта, где ***X*** — любая переменная, имя процедуры или функции.

SizeOf(X): Word возвращает объем в байтах, занимаемый ***X***, причем ***X*** может быть либо именем переменной любого типа, либо именем типа.



Массив указателей

Для хранения большего числа строк разместим в статической памяти лишь указатели на них, а сами строки перенесем в динамическую память. Заметим, что размер одного указателя – 4байта.

Указатели чаще всего используются для работы с ***динамическими массивами памяти***, которые представляют собой массивы переменной длины, память под которые может выделяться и изменяться в процессе выполнения программы, как при каждом новом запуске программы, так и в разных ее частях. Обращение к **i**-му элементу динамического массива **x** имеет вид **x^[i]**.

Рассмотрим описание динамической матрицы. Пусть есть типы данных **massiv** и указатель на него **din_massiv**:

```
type massiv=array [1..1000] of real;
```

- din_massiv=^massiv;

Динамическая матрица X будет представлять собой массив указателей:

- var X: array[1..1000] of din_massiv;

Работать с матрицей необходимо следующим образом:

определить ее размеры (пусть N – число строк, M – число столбцов);

выделить память под матрицу:

6. for i:=1 to N do
   - **getmem**(X[i], M***sizeof**(real));

Каждый элемент статического массива X[i] – указатель на динамический массив, состоящий из M элементов типа real. В статическом массиве X находится N указателей.

3. для обращения к элементу динамической матрицы, расположенному в i-той строке и j-м столбце, следует использовать следующую конструкцию: **X[i]^[j]**;

4. после завершения работы с матрицей необходимо освободить память:

   - for i:=1 to N do
   - **freemem**(X[i], M***sizeof**(real));

Пример:

```
Const
MaxItem=2000;
Type
Pstring=^String;
TDinMas=Array[1.. MaxItem] Of Pstring;
Var
p: TDinMas;
i: Ineger;
Begin
For i:=1 To MaxItem Do New(p[i]);
p[1]^:= ?Hallo!?
End.
```

Динамические структуры данных

Структурированные типы данных, такие, как массивы, множества, записи, представляют собой ***статические структуры***, т.к. их размеры неизменны в течение всего времени выполнения программы.

Часто требуется, чтобы структуры данных меняли свои размеры в ходе решения задачи. Такие структуры называются ***динамическими***, к ним относятся *стеки, очереди, списки, деревья* и др. Описание динамических структур с помощью массивов, записей и файлов приводит к неэкономному использованию памяти ЭВМ и увеличивает время решения задач.

В поисках решения проблемы быстрой обработки больших объемов данных были предложены динамические структуры данных.
Связанный список — это такая структура данных, элементами которой служат записи одного и того же формата, связанные друг с другом с помощью указателей, расположенных в самих элементах. Элементы списка часто называются его *узлами.*

Элемент (узел) связанного списка — запись	
Содержательная часть – указующая часть	
Данные любого типа-Один или два указателя на соседний элемент	

- ***Стек*** — линейный список, в котором все добавления и удаления (и обычно всякий доступ) делаются в одном конце списка.
- ***Очередь*** — линейный список, в котором все добавления производятся на одном конце списка, а все удаления делаются на его другом конце.
- ***Дек*** (очередь с двумя концами) — линейный список, в котором все до бавления и удаления делаются на обоих концах списка.

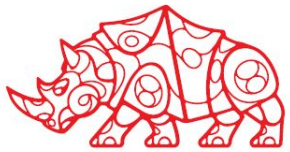
Прежде чем рассматривать действия со связанными списками, введем обо значения переменных, которыми будем пользоваться при описании соответ ствующих алгоритмов и структур данных

Элемент	Описание
item, pitem	Тип для одного элемента списка (запись) и указателя на него
data	Поле данных (информационная часть элементов списка)
next,prev	Указатели следующего, предыдущего элементов (указующая часть)
head, top	Указатели на первыйи последний элемент списка
pl,p2	Рабочие указатели

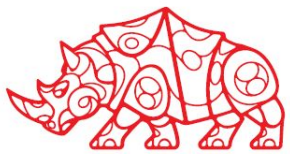
Опишем тип одного элемента односвязного списка и указателя на этот элемент:

- type pitem=^item;
- item=record
- data:... {простой или определяемый пользователем тип}
- next:pitem;{или prev:pitem;}
- end;

Обратите внимание на описание — это единственный разрешенный случай, когда тип используется до объявления (***item***). Очевидно, разработчики компилятора сделали исключение ввиду особой важности списковых структур.



```
type pitem=^item;
item=record {элемент стека}
data:integer; {значение элемента}
prev:pitem; {адрес предыдущего элемента}
end;
var top,p:pitem;
n,k:integer;
procedure add(x:integer); {добавляет элемент на вершину стека}
begin
new(p); {создаем произвольный элемент p}
p^.data:=x; p^.prev:=top;
top:=p; {}
end;
procedure deltop; {удаляет узел с вершины стека}
begin
if top<>nil then begin {если стек не пустой}
p:=top^ . prev; {запоминаем предшествующий вершине элемент}
dispose(top);
top:=p; {устанавливаем p вершиной стека}
end;
procedure writestack; {выводит стек на экран}
begin
writeln('Содержимое стека (начиная с вершины):');
p:=top;
while p<>nil do begin
write (p^.data, ' ' ); p:= p^ . prev;
end;
writeln;
end;
begin {начало программы}
top:=nil;
for k:=1 to 10 do add(k); {заполняем стек числами от 1 до 10}
writestack;
writeln('Введите значение элемента для добавления в стек:');
readln(n); add(n);
writestack;
writeln('Сколько элементов стека нужно удалить?'); readln(n);
for k:=1 to n do deltop;
writestack;
readln
```

Домашние задание (Задание на самоподготовку)

УНИВЕРСИТЕТ СИНЕРГИЯ

Ответить на вопросы:

1. Чем отличаются динамические переменные от статических?
2. Что такое указатель?
3. Если все указатели хранят адреса, зачем различать типы указателей?
4. Какие есть процедуры для работы с указателями?
5. В чем преимущество динамического выделения памяти?