**Лекция 9. Указатели. Операции над указателями**

**указатели**

Оперативная память состоит из последовательный ячеек. Каждая ячейка имеет номер, называемый адресом.

В 32-битных системах относятся 232 байт (4Гб) памяти, в 64-битных - 2 64 соответственно.

Переменная (или константа), что сохраняет адрес, называется указателем.

**Для чего нужны указатели**

Указатели повышают **гибкость доступа к данным:**

1. Вместо самих данных можно хранить указатель на них. Это позволяет хранить данные в одном экземпляре и множество указателей на эти данные. Через различные указатели эти данные можно обновлять (пример - корпоративная БД).

2. Указатели можно присвоить адрес другого объекта (вместо старого появился новый телефонный справочник).

3. С помощью указателей можно создавать **сложные структуры данных**.

**типы указателей**

Указатели делятся на:

• **Типизированные** (указывают на объект некоторого типа)

• Имеют тип: ^ <тип>

• Пример. ^ Integer - указатель на integer

• **Бестиповые** (сохраняют адрес ячейки памяти неизвестного типа)

• Преимущество: могут хранить что угодно

• Имеют тип: pointer

Пример кода.

var

i: integer = 5;

r: real = 6.14;

pi: ^ integer;

pr: ^ real;

begin

pi = @i;

pr = @r;

pi = @r; // ОШИБКА компиляции

end.

**@ - Унарная операция взятия адреса**

**Операция розадресации (разыменования)**

var

i: integer = 5;

pi: ^ integer;

begin

pi = @i;

pi ^ = 8 - pi ^;

writeln (i) // 3

end.

**^ - Операция разыменования**

**pi ^** - то, на что указывает pi, то есть другое имя i или ссылки на i.

Тут надо вспомнить определение ссылки:

**Ссылки** - другое имя объекта.

**нулевой указатель**

Все глобальные неинициализированные указатели сохраняют специальное значение nil, что говорит о том, что они никуда не указывают.

Указатель, что сохраняет значение nil называется нулевым.

var

pi: ^ integer; // Указатель pi сохраняет значение nil

i: integer;

begin

pi = @i; // Pi сохраняет адрес переменной i

pi = nil; // Pi снова никуда не указывает

pi ^ = 7; // ОШИБКА времени выполнения:

// Попытка розименовуваты нулевой указатель

Попытка розименуваты нулевой указатель приводит к ошибке времени выполнения.

**Бестиповые указатели**

var

p: pointer;

i: integer;

begin

p = @i;

end.

Бестиповым указателю можно присвоить адрес переменной любого типа, то есть бестиповым указатель совместим по присваиванию с любым обычным указателем.

Попытка розименовуваты безтиповой указатель приводит к **ошибке компиляции**. То есть он может только сохранять адреса.

Оказывается, любой типизированный указатель совместим по присваиванию с бестиповым, то есть следующий код верен

var

pi: ^ integer;

i: integer;

p: pointer;

begin

p = @i;

pi = p;

pi ^ + = 2;

end.

Вопрос. Нельзя интерпретировать память, на которую указывает p, что как относится к определенному типу?

Ответ - да, можно. Вот как это сделать:

type

pinteger = ^ integer;

var

i, j: integer;

p: pointer;

begin

p = @i;

pinteger (p) ^ = 7; // Используем явное приведение типа

writeln (i) // 7

end.

запись

<Тип> (<переменная>)

показывает, что используется явное приведение типов.

Внимание! Неконтролируемая ошибка!

type

preal = ^ real;

var

i, j: integer;

p: pointer;

begin

p = @i;

preal (p) ^ = 3.14; // ОШИБКА!

end.

Область памяти, на которую указывает p трактуется как область, которая хранит вещественное число (8 байт), и поэтому константа 3.14 записывается в эти 8 байт. Однако, переменная i занимает только 4 байта, поэтому затираются еще 4 соседних байта (в этом случае они принадлежат переменной j).

**Доступ к памяти, имеет другое внутреннее представление**

type

Rec = record

b1, b2, b3, b4, b5, b6, b7, b8: byte;

end;

PRecord = ^ Rec;

var

r: real = 3.1415;

prec: ^ Rec;

begin

var temp: pointer = @r;

prec = temp;

writeln (prec ^ .b1, '', prec ^ .b2, '', {...} prec ^ .b8)

end.

Замечания. Важно, что типы real и Rec имеют один размер.

**Неявные указатели в языке Pascal**

1. procedure p (var i: integer)

2. параметра переменной при вызове на стек кладется не самая переменная, а указатель на нее.

3. var pp: procedure (i: integer)

4. Для хранения процедурной переменной используется ячейка памяти, является указателем.

5. var a: array of real;

6. Переменная типа динамический массив является указателем на данные массива, хранящихся в динамической памяти.

**динамическая память**

Особенности динамической памяти

Память, принадлежит программе, делится на:

• **Статическое** (память, занимаемая глобальными переменными и константами).

• **Автоматическая** (память, занимаемая локальными данными, то есть сток программы).

• **динамическую** (память, выделяемая программе по специальному запросу).

В дополнение к статической и автоматической памяти, которые фиксированы после запуска программы, программа может получать нефиксированную количество динамической памяти. Ограничения на объем выделяемой динамической памяти связаны лишь с настройками операционной системы и объемом оперативной памяти компьютера.

Основная проблема - явно выделенную динамическую память необходимо возвращать, иначе не хватит памяти другим программам.

Для явного выделения и освобождения динамической памяти используются процедуры:

• **New**

• **Dispose**

var

p: pinteger; // P никуда не указывает

begin

New (p) // В динамической памяти выделяется ячейка

// Размером в один integer, и

// P начинает указывать на эту ячейку

p ^: = 3;

Dispose (p) // Возвращает динамическую память,

// Контролируемую указателю p, назад ОС

end.

По окончании работы программы, вся вызвана программой динамическая память возвращается ОС.

**Но лучше освобождать динамическую память явно**! Иначе в процессе работы программы она может занимать большие объемы (еще не освобожденной) памяти, вредит общей производительности системы.

**Ошибки при работе с динамической памятью**

1.

var p: pinteger;

begin

p ^ = 5; // ОШИБКА

end.

Ошибка разыменования нулевого указателя (попытка использовать невыделенными динамическую память).

2.

var p: pinteger;

begin

New (p)

New (p) // ОШИБКА

end.

Утечка памяти (память, выделившейся в результате первого вызова New (p), принадлежит программе, но не контролируется никаким указателю.

2a.

procedure q;

var p: pinteger;

begin

New (p)

end;

begin

q; // ОШИБКА

end.

Утечка памяти в подпрограмме: конечно если динамическая память выделяется в подпрограмме, то она должна в этой же подпрограмме возвращаться. Исключение составляют т.н. "Образующие" п / п:

function CreateInteger: pinteger;

begin

New (Result)

end;

begin

var p: pinteger = CreateInteger;

p ^ = 555;

Dispose (p)

end.

Ответственность за удаление памяти, выделенной в подпрограмме, лежит на программисты, вызвавший эту подпрограмму.

3.

var p: pinteger;

begin

for var i = 1 to 1000000 do

New (p) // ОШИБКА

end.

Out of Memory (очень большие утечки памяти, в результате которых динамическая память может «иссякнуть»).

4.

var p: pinteger;

begin

New (p)

p ^ = 5;

Dispose (p)

p ^ = 7; // ОШИБКА

end.

После вызова Dispose (p), p называют висячим указателем (так как он указывает на недоступную больше область памяти).