Структуры
и алгоритмы
обработки данных

Литература

1. **Вирт Н. Алгоритмы+структуры данных = программы: Пер. с англ.-М.Мир,1985.-406 с., ил.**
2. **Вирт Н. Алгоритмы и структуры данных: Пер. с англ.-М.Мир,1989.-360 с., ил.**
3. **Кнут Д. Искусство программирования для ЭВМ. В семи томах. т.1. Основные алгоритмы. М.: Мир, 1976**
4. **Кнут Д. Искусство программирования для ЭВМ. В семи томах. т.3. Сортировка и поиск. М.: Мир, 1978**
5. **Ахо А., Хопкрофт,Д., Ульман Д. Структуры данных и алгоритмы. Вильямс, С-П, 2000**

Введение

Структура данных **– общее свойство информационного объекта, с которым взаимодействует та или иная программа. Это общее свойство характеризуется:**

- **множеством допустимых значений данной структуры;**
- **набором допустимых операций;**
- **характером организованности.**

Введение

Любая структура на абстрактном уровне может быть представлена в виде двойки <D,R>

где D – конечное множество элементов, которые могут быть типами данных, либо структурами данных,

R – множество отношений, свойства которого определяют различные типы структур данных на абстрактном уровне.

Основные виды (типы) структур данных:

- **Множество – конечная совокупность элементов, у которой R=∅.**
- **Последовательность – абстрактная структура, у которой множество R состоит из одного отношения линейного порядка (т. е. для каждого элемента, кроме первого и последнего, имеются предыдущий и последующий элементы).**

Введение

- **Матрица – структура, у которой множество R состоит из двух отношений линейного порядка.**
- **Дерево – множество R состоит из одного отношения иерархического порядка.**
- **Граф – множество R состоит из одного отношения бинарного порядка.**

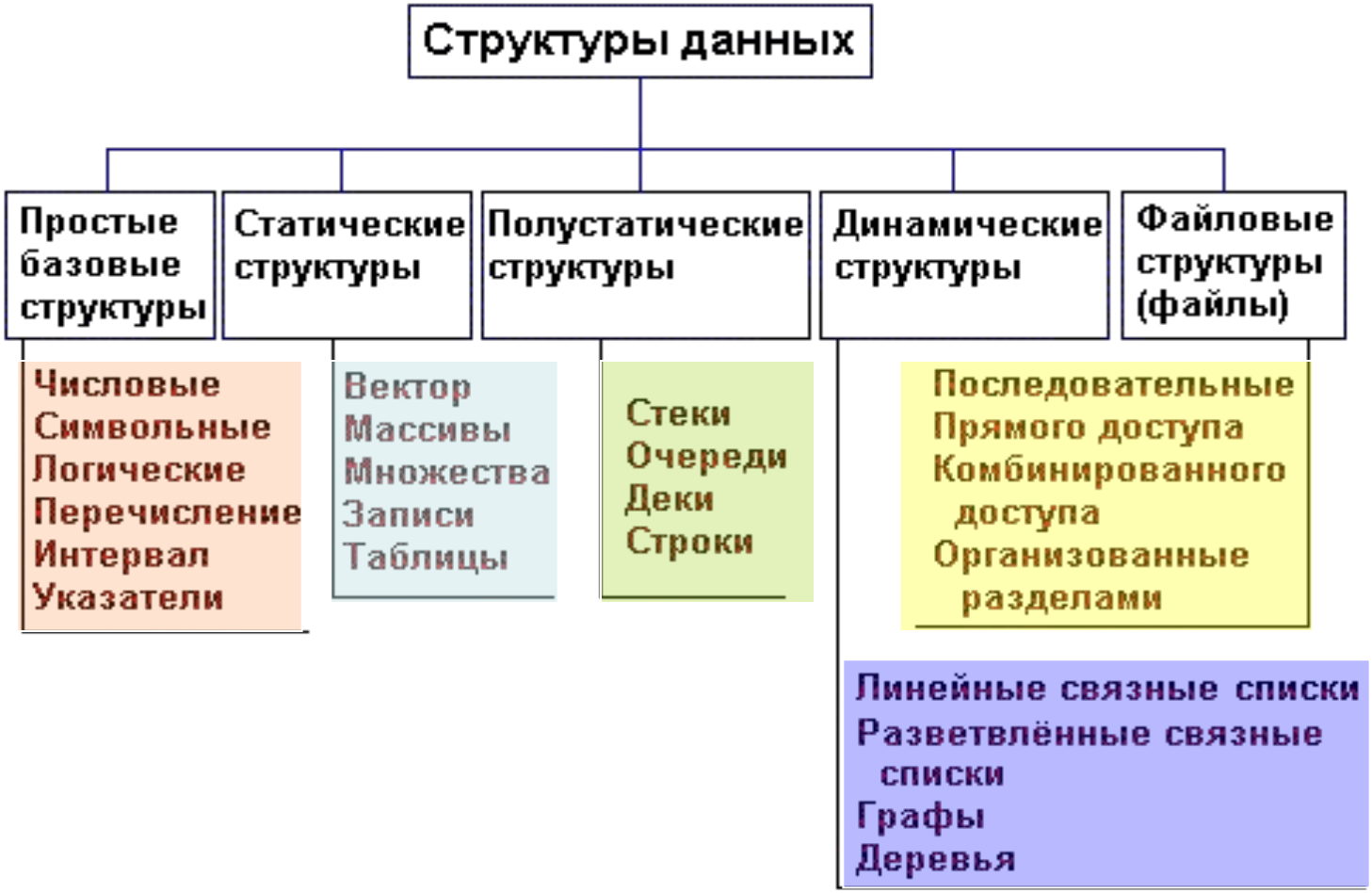
Вырожденные (простейшие) структуры данных называются также типами данных.

Различают следующие уровни описания данных:

- **абстрактный (математический) уровень**
- **логический уровень**
- **физический уровень**

Классификация СД

Введение


Структуры данных
Простые базовые структуры
Статические структуры
Полустатические структуры
Динамические структуры
Файловые структуры (файлы)
Числовые
Символьные
Логические
Перечисление
Интервал
Указатели
Вектор
Массивы
Множества
Записи
Таблицы
Стеки
Очереди
Деки
Строки
Последовательные
Прямого доступа
Комбинированного доступа
Организованные разделами
Линейные связные списки
Разветвлённые связные списки
Графы
Деревья

Категории типов данных

- **Встроенные типы данных, т.е. типы, предопределенные в языке программирования или языке баз данных.**
- **уточняемые типы данных**
- **перечисляемый тип данных**
- **конструируемый тип (составной)**

Категории типов

- **Указательные типы дают возможность работы с типизированными множествами абстрактных адресов переменных, содержащих значения некоторого типа**

В современных компьютерах к таким "машинным" типам относятся

- **целые числа разного размера**

(2-8 байтов)

- **числа с плавающей точкой одинарной и двойной точности (4 и 8 байт соответственно)**

Встроенные типы данных

- **Тип CHARACTER (или CHAR) в разных языках - это**

либо набор печатных символов из алфавита, зафиксированного в описании языка (ASCII),

либо произвольная комбинация нулей и единиц, размещаемых в одном байте или 2 байтах

Уточняемые типы данных

Type T = min..max;

ПРИМЕРЫ

```
Type year =  (1900 .. 2001);
     digit = (‘0’ ..’9’);

Var y : year;
    d : digit;
```

Перечисляемые типы данных

TYPE T = (C_1,C_2,...,Cn)

где T — идентификатор нового типа,

Ci — идентификаторы новых констант

Перечисляемые типы данных

ПРИМЕРЫ:

```
Type color = (red, yellow, green);
     destination = (hell, purgatory,
                    heaven);
Var  c: color;
     d: destination;
…
C := red;
D := heaven;
```

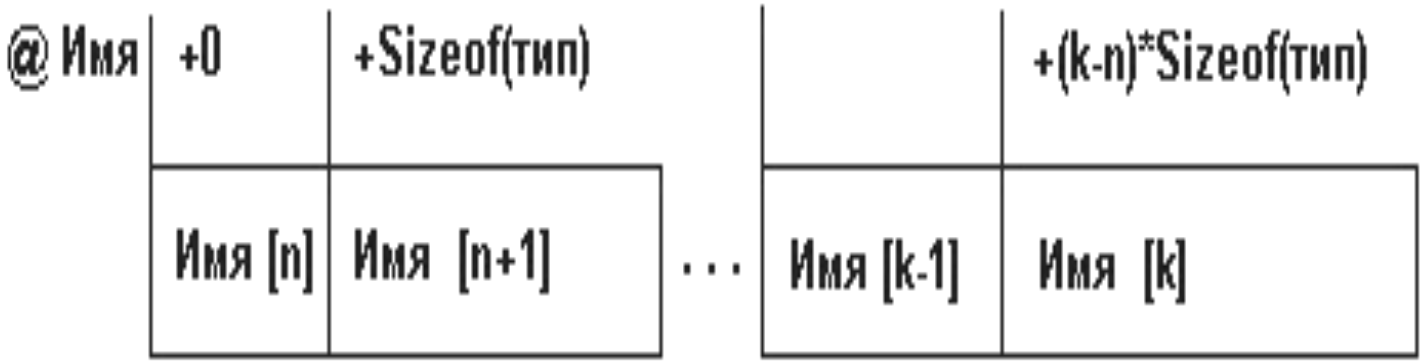
Массивы

```
Type t  = array [Ti] of To ;
Type row = array [1.. 5] of real;
Type card = array [1.. 80] of char;
Type  alfa = array [0.. 15] of char;
…
Var x: row;
```

C:

```
int x[4]
int x[] = { 0, 2, 8, 22}
```

<Имя>: array [n..k] of < тип >;



var m1:array[-2..2] of real;

Смещение элемента относит. адреса m1 (байт)	+0	+6	+12	+18	+24
Значения элем. массива	m1[-2]	m1[-1]	m1[0]	m1[1]	m1[2]

Количество байтов непрерывной области памяти, занятых одновременно вектором, определяется по формуле:

ByteSise = (k - n + 1) * Sizeof (тип)

Обращение к i-тому элементу вектора выполняется по адресу вектора плюс смещение к данному элементу.

Смещение i-ого элемента вектора определяется по формуле:

ByteNumer = (i- n) * Sizeof (тип),

а адрес его:

@ ByteNumber = @ имя + ByteNumber.

где @ имя - адрес первого элемента вектора.

Массивы

Информация, содержащаяся в дескрипторе вектора, должна позволять,

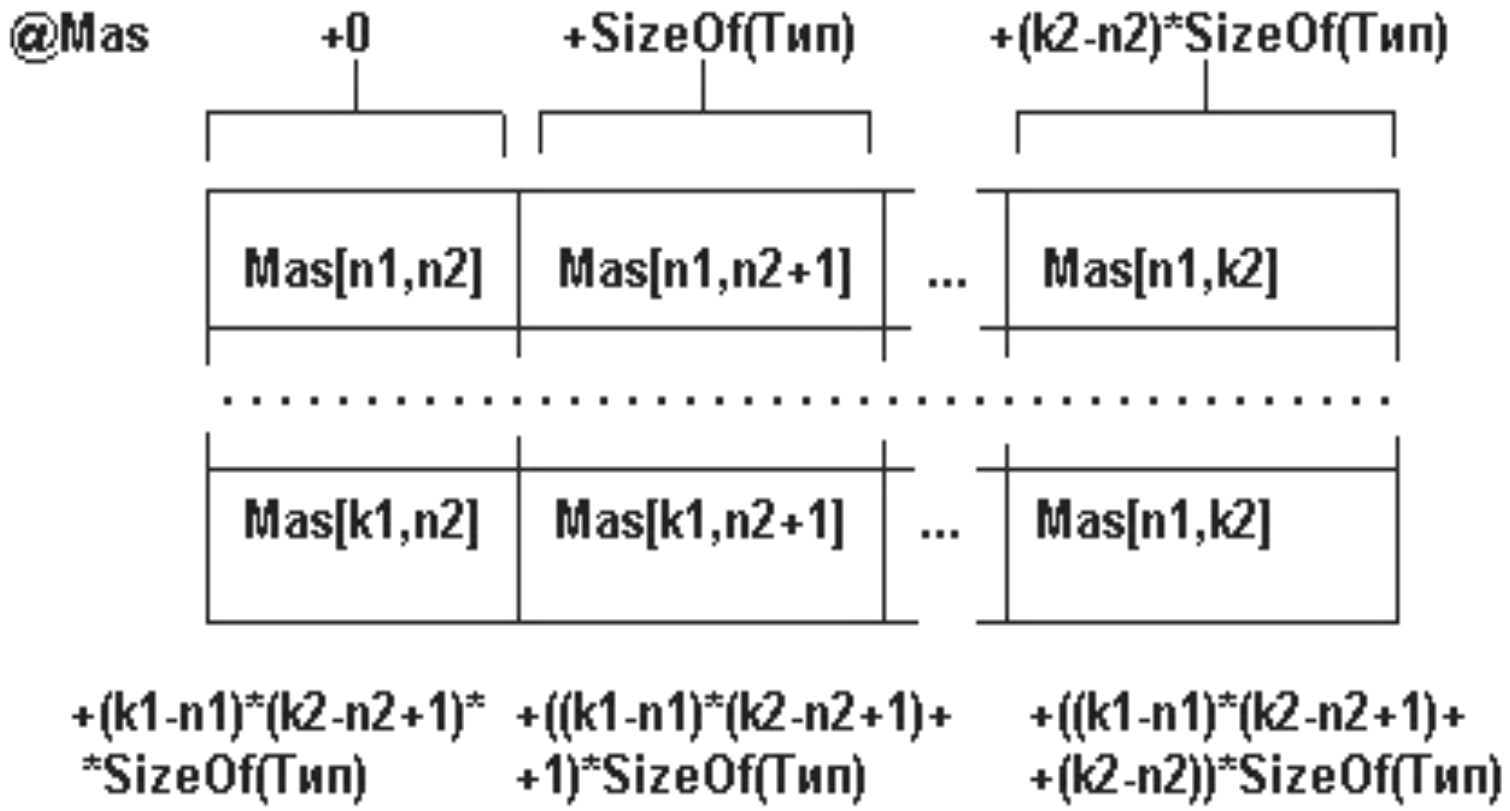
сократить время доступа,

(A0 = @Имя - n*Sizeof(тип))

обеспечивает проверку правильности обращения (верхняя и нижняя границы массива).

Массивы (многомерные)

Var Mas2D : Array [n1..k1 , n2..k2] of < Тип >



Количество байтов памяти, занятых двумерным массивом, определяется по формуле :

ByteSize = (k1-n1+1)*(k2-n2+1) *SizeOf(Тип)

Адресом массива является адрес первого байта начального компонента массива.

Смещение к элементу массива

Mas[i1,i2] определяется по формуле:

ByteNumber = [(i1-n1)*(k2-n2+1)+(i2-n2)]
***SizeOf(Тип)**

его адрес :

@ByteNumber = @mas + ByteNumber

Например:

```
var Mas : Array [3..5] [7..8] of Word;
```

Этот массив будет занимать в памяти:

(5-3+1)*(8-7+1)*2=12 байт;

а адрес элемента Mas[4,8]:

@Mas+((4-3)*(8-7+1)+(8-7)*2 = @Mas+6

для двумерного массива с границами изменения индексов:

[B(1)..E(1)][B(2)..E(2)], размещенного в памяти по строкам, адрес элемента с индексами [I(1),I(2)] может быть вычислен как:

Addr[I(1),I(2)] = Addr[B(1),B(2)] +

{ [I(1)-B(1)] * [E(2)-B(2)+1] + [I(2)-B(2)] } *SizeOf(Тип)

для массива произвольной размерности:
Mas[B(1)..E(2)][B(2)..E(2)]...[B(n)..E(n)]
получим:

$$Addr[I(1),I(2),...,I(n)] = Addr[B(1),B(2),...B(n)] + Sizeof(Тип)*\sum [B(m)*D(m)] + Sizeof(Тип)*\sum [I(m)*D(m)]$$

(m=1..n)

где Dm зависит от способа размещения массива.
При размещении по строкам:

$$D(m)=[E(m+1)-B(m+1)+1]*D(m+1),$$

где m = n-1,...,1 и D(n)=1

Массивы

При вычислении адреса элемента наиболее сложным является вычисление третьей составляющей формулы, т.к. первые две не зависят от индексов и могут быть вычислены заранее.

Для ускорения вычислений множители D(m) также могут быть вычислены заранее и сохраняться в дескрипторе массива

Массивы

Дескриптор массива, таким образом, содержит:

начальный адрес массива - Addr[I(1),I(2),...,I(n)];

число измерений в массиве - n;

постоянную составляющую формулы линеаризации (первые две составляющие формулы);

для каждого из n измерений массива:

значения граничных индексов - B(i), E(i);

множитель формулы линеаризации - D(i).

Массивы

Представление массивов с помощью векторов Айлиффа

Для массива любой мерности формируется набор дескрипторов: основного и несколько уровней вспомогательных дескрипторов, называемых векторами Айлиффа

Каждый вектор Айлиффа определенного уровня содержит указатель на нулевые компоненты векторов Айлиффа следующего, более низкого уровня, а векторы Айлиффа самого нижнего уровня содержат указатели групп элементов отображаемого массива.

Массивы

В Java имеется большое количество классов и интерфейсов для массивов и массивоподобных структур:

Arrays, Vector, Collection, Map, Hashtable, LinkedList, ArrayList

Массивы

В JDK 5.0 ArrayList был преобразован в универсальный класс с параметром типа.

Массив, предназначенный для хранения объектов Employee.

```
ArrayList<Employee> staff = new
    ArrayList<Employee>();
```

Записи

Запись - конечное упорядоченное множество полей, характеризующихся различным типом данных.

```
type complex = record re: real;
                      im: real
               end ;
var x: complex;
```

C:

```
struct complex { float re;
                 float im;
               }
struct complex x;
```

Записи

```
struct { float re;
         float im;
       } x, y;
x=y; …

struct { float r;
         float i;
       } z;
```

Записи

```
var rec:record
        num :byte; {  номер  студента }
        name :string[20]; { Ф.И.О. }
        fac, group:string[7];
        math,comp,lang:byte;{оценки}
      end;
```

Представление в виде последовательности полей, занимающих непрерывную область памяти

@rec	+0	+1	+21	+29	+37	+38	+39
	24	Иванов В.И.	АП	АВТФ	8В50	5	5

Записи

в виде связного списка с указателями на значения полей записи

Множества

Множество - такая структура, которая представляет собой набор неповторяющихся данных одного и того же типа.

```
type T =set of To
```

Примеры

```
type bitset = set of (0..15);
type tapestatus = set of exception;

var B : bitset;
t : array [1.. 6] of tapestatus;
```

Множества

Множество в памяти хранится как массив битов, в котором каждый бит указывает является ли элемент принадлежащим объявленному множеству или нет.

Множества

где @S - адрес данного типа множество.

Множества

Число байтов, выделяемых для данных типа множество, вычисляется по формуле:

ByteSize = (max div 8)-(min div 8) + 1,

где max и min - верхняя и нижняя границы базового типа данного множества.

Номер байта для конкретного элемента E вычисляется по формуле:

ByteNumber = (E div 8)-(min div 8),

номер бита внутри этого байта по формуле: BitNumber = E mod 8

Множества

Например, S : set of byte;

S:=[15,19];

Содержимое памяти при этом будет следующим:

@S+0 - 00000000

@S+1 - 10000000

@S+2 - 00001000

. . . . . .

@S+31 – 0000000

Указатели

Понятие указателя в языках программирования является абстракцией понятия машинного адреса.

Подобно тому, как зная машинный адрес можно обратиться к нужному элементу памяти, имея значение указателя, можно обратиться к соответствующей переменной.

```
var ipt : ^integer; cpt : ^char;
```

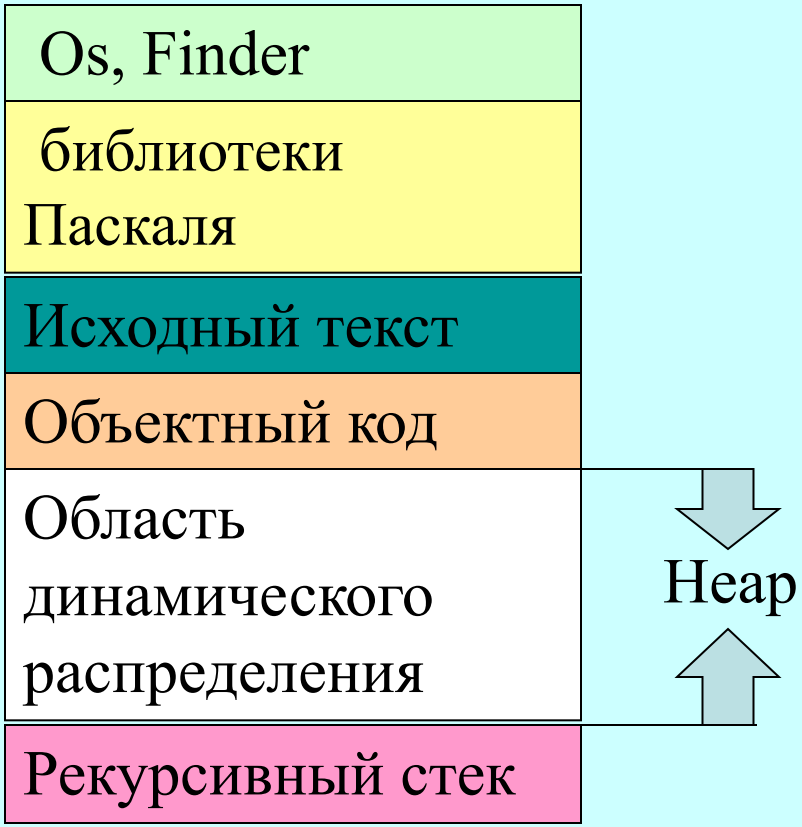
C:

```
int *ipt;  char *cpt;
```

Динамическая память (указатели)

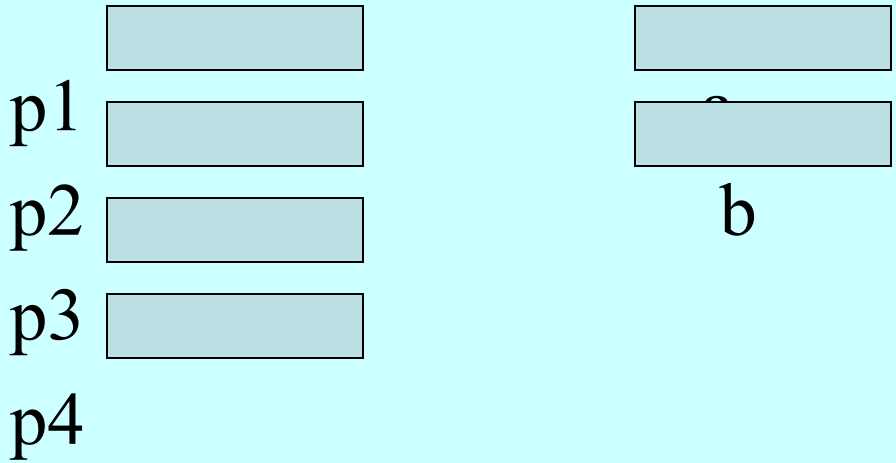
Статическое распределение памяти

Динамическое распределение памяти



Описание переменных типа указатель

```
Type c = array [1..100] of real;
Var p1 : ^integer ;
        p2 : ^real ;
        p3,p4 : ^c;
        a, b : integer ;
```



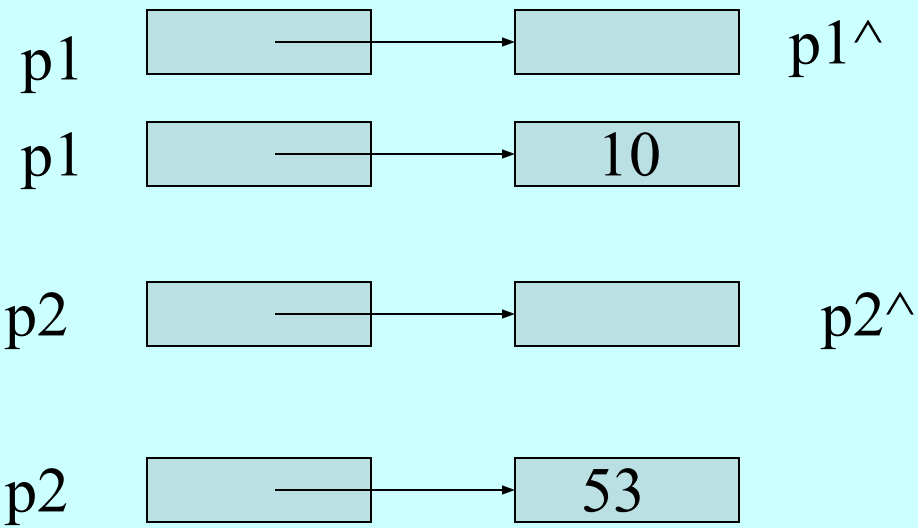
p1 := nil ;.. p4 := nil; a := 5; b := 7;

Процедуры работы с указателями:

New (p)- выделить память в **Heap** для переменной, на которую указывает p

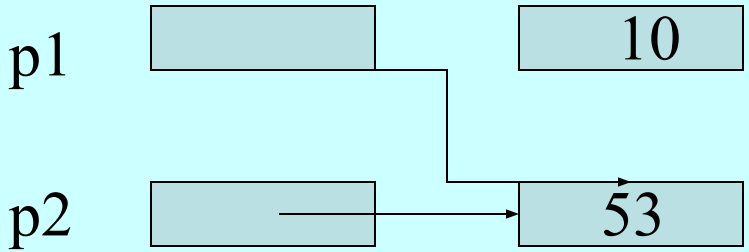
Dispose (p)-освободить память в **Heap**, выделенную для переменной, на которую указывает p

```
Var  p1, p2, p3: ^integer;
  begin
     New ( p1 );
      p1^ := 10;

     New ( p2 );

     p2^ := 53;
```

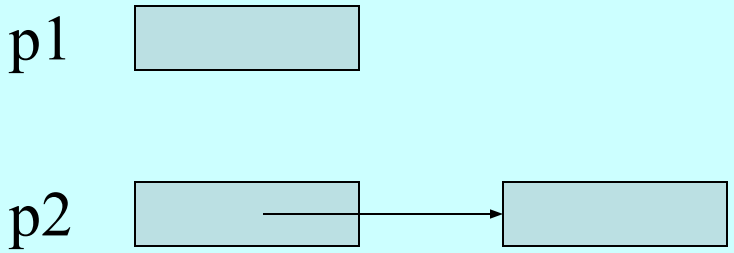


Указатели

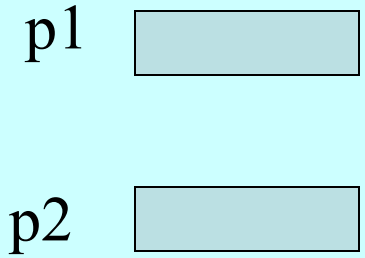
p1 := p2;



Dispose (p1);



Dispose (p2);



Обмен **данными массивов**

```
Program DemoPointer;
    Const n=1000;
    Type mas = array [ 1..n] of integer;
    Var p1,p2,p3 : ^mas;
          i : integer;
```

```
begin    New ( p1 ); New ( p2 );
   for i :=1 to n do
      begin Write (‘Введи ’,i, ‘эл-т 1-го
                        массива’ );
            Readln (p1^[i]);
            Write (‘Введи ’,i, ‘эл-т 2-го
                        массива’ );
            Readln (p2^[i])
      end;
```

```
….
{ обмен данными }
p3 := p1;
p1:=  p2;
p2 := p3;
...
Dispose ( p1 );
...
```