Алгоритмы и структуры данных

Лк -32, Лб-32, КР-52, экзамен, диф.зачет

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 3 томах. т.1. Основные алгоритмы. М., С-П, : Вильямс, 2000**
4. **Кнут Д. Искусство программирования для ЭВМ. В 3 томах. т.3. Сортировка и поиск. М., С-П, : Вильямс, 2000**
5. **Ахо А., Хопкрофт,Д., Ульман Д. Структуры данных и алгоритмы. Вильямс, С-П, 2000**

Литература

6. Седжвик Роберт

Фундаментальные алгоритмы на C++.

Анализ/Структуры данных/Сортировка/ : Пер. с англ./Роберт Седжвик. - К.: Издательство «ДиаСофт», 2001.- 688 с.

ofofano@tpu.ru

ftp://ftp.tpu.ru

login: osu; password: stud

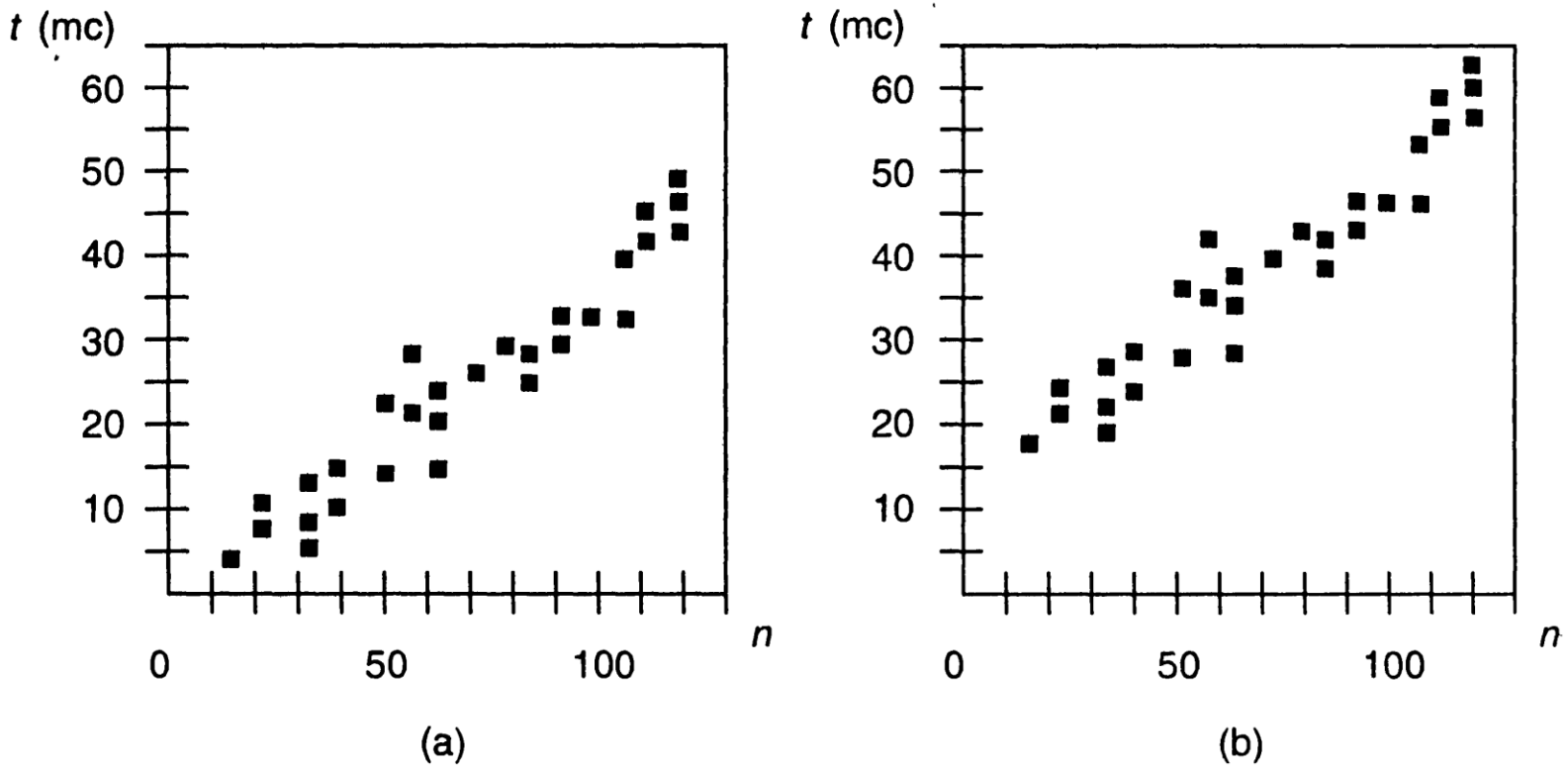
Алгоритм

Алгоритм (algorithm) — это люб
определенная вычислительная п
на вход (input) которой подает
величина или набор величин,
и результатом выполнения кото
выходная (output) величина
значений.
Таким образом, алгоритм предст
последовательность вычислительных шагов, преобразующих входные величины в выходные.

Алгоритм:свойства

1. **Конечность**
2. **Определенность**
3. **Ввод**
4. **Вывод**
5. **Эффективность**

1. **Время выполнения**
2. **Требуемая память**



Анализ алгоритмов

Экспериментальные исследования имеют три основных ограничения:

1. **эксперименты могут проводиться лишь с использованием ограниченного набора исходных данных; результаты, полученные с использованием другого набора, не учитываются;**
2. **для сравнения эффективности двух алгоритмов необходимо, чтобы эксперименты по определению времени их выполнения проводились на одинаковом аппаратном и программном обеспечении;**
3. **для экспериментального изучения времени выполнения алгоритма необходимо провести его реализацию и выполнение.**

Анализ алгоритмов

Общая методология анализа времени выполнения алгоритмов:

1. **Описание алгоритма на псевдокоде;**
2. **Определение числа операций на основе анализа структурных конструкций псевдокода;**
3. **Определение общего числа операций алгоритма**

Анализ алгоритмов

Общая методология анализа времени выполнения алгоритмов:

1. **учитывает различные типы входных данных;**
2. **позволяет производить оценку относительной эффективности любых двух алгоритмов независимо от аппаратного и программного обеспечения;**
3. **может проводиться по описанию алгоритма без его непосредственной реализации или экспериментов.**

Анализ алгоритмов

Алгоритм arrayMax(A, n):

Input:массив A, содержащий n целых чисел (n> 1).

Output: элемент с максимальным значением в массиве A.

currentMax ← A [0]	выполняется 1 раз
for i ← 1 to n - 1 do	выполняется n-1 раз
if currentMax <A[i] then	выполняется n-1 раз
currentMax ← A [i]	выполняется от 0 до n-1раз
return currentMax	выполняется 1 раз

Анализ алгоритмов

```
/**
* Тестирует программу алгоритма поиска максимального
      элемента массива.
*/
public class ArrayMaxProgram {
/** находит элемент с максимальным значением в массиве A,
* содержащем n целых чисел.
*/
static int arrayMax (int[ ] A, int n) {
  int currentMax = A[0]; // выполняется один раз
   for (int i=1, i < n, i++) // выполняется от 1 до n,
                                  // n-1 раз соответственно.
     if (currentMax < A[i]) // выполняется n-1 раз
     currentMax = A[i]; // выполняется максимально n-1 раз
   return currentMax; // выполняется один раз
}
```

Анализ алгоритмов

```
/** Тестирующий метод
*/
public static void main(String args[ ]) {
  int[ ] num = { 10, 15, 3, 5, 56, 107, 22, 16, 85 };
  int n = num.length;
  System.out.print("Array:");
  for (int j=0; i < n; i++)
     System.out.print("" + num[i]);
  System.out.println(".");
  System.out.println("The maximum element is" +
      arrayMax(num,n) + ".");
}
}
```

Описание алгоритмов (псевдокод)

• **Выражения. Для написания числовых и логических выражений используются стандартные математические символы. Знак стрелки ← применяется в качестве оператора присваивания в командах присваивания (равнозначен оператору «=» языка Java). Знак «=» используется для передачи отношения равенства в логических выражениях (что соответствует оператору «= =» языка Java).**

• **Объявление метода. Имя алгоритма (paraml, param2, ...) объявляет «имя» нового метода и его параметры.**

• **Структуры принятия решений. Условие if, then — действия, если условие верно [else — если условие не верно]. Отступы используются для обозначения выполняемых в том или другом случае действий.**

• **Цикл while, while — условие, do - действия. Отступ обозначает действия, выполняемые внутри цикла.**

Описание алгоритмов (псевдокод)

- **Цикл repeat, repeat — действия, которые выполняются, пока выполняется условие until. Отступ обозначает действия, выполняемые внутри цикла.**
- **Цикл for, for — описание переменной и инкремента, do — действия. Отступ обозначает действия, выполняемые внутри цикла.**
- **Индексирование массива. A[i] обозначает i-ую ячейку массива A. Ячейки массива A с количеством ячеек n индексируются от A[0] до A[n - 1] (как в Java и C).**
- **Обращения к методам, object.method (args) (часть object необязательна, если она очевидна).**
- **Возвращаемое методом значение. Значение return. Данный оператор возвращает значение, указанное в методе, вызывающим данный метод.**

Анализ алгоритмов:аналитический подход

1. **Записать алгоритм в виде кода одного из развитых языков программирования (например, Java или C++).**
2. **Перевести программу в последовательность машинных команд (например, байт-коды, используемые в виртуальной машине Java).**
3. **Определить для каждой машинной команды i время t_i, необходимое для ее выполнения.**
4. **Определить для каждой машинной команды i количество повторений n_i команды i за время выполнения алгоритма.**
5. **Определить произведение $t_i * n_i$ всех машинных команд, что и будет составлять время выполнения алгоритма.**

Анализ алгоритмов

Простейшие операций высокого уровня, не зависящие от используемого языка программирования и использующиеся в псевдокоде:

- **присваивание переменной значения**
- **вызов метода**
- **выполнение арифметической операции**
- **сравнение двух чисел**
- **индексация массива**
- **переход по ссылке на объект**
- **возвращение из метода**

Анализ алгоритмов

Число T(n) простейших операций, выполняемых внутри алгоритма, пропорционально действительному времени выполнения данного алгоритма

число простейших операций T(n),

выполняемых алгоритмом arrayMax,

минимально равно

2 + 1 + n + 4(n -1) + 1 = 5n

а максимально

2 + 1 + n + 6(n -1) + 1 = 7n – 2

Алгоритм arrayMax(A, n):

```
currentMax ← A [0]
for i ← 1 to n - 1 do
    if currentMax <A[i] then
        currentMax ← A [i]
return currentMax
```

те упрощенный анализ даст следующий результат: алгоритм выполняет

от $5n$ до $7n - 2$ шагов

при размере исходных данных n.

Временная и пространственная сложности алгоритмов

Для многих программ время выполнения является функцией входных данных

Например, некая программа имеет время выполнения

$$T(n) = cn^2,$$

где c — константа.

Единица измерения *T(n)* точно не определена, обычно понимается под *T(n)* количество инструкций, выполняемых на идеализированном компьютере.

Временная и пространственная сложности алгоритмов

Для описания скорости роста функций используется

O-символика. (Paul Bachman 1894г.)

Когда мы говорим, что время выполнения *T(n)* некоторой программы,например, имеет порядок $O(n^2)$, то подразумевается, что существуют положительные константы *c* и n_0 такие, что для всех $n \geq n_0$, выполняется неравенство $T(n) < c\, n^2$.

Когда мы говорим, что *T(n)* имеет степень роста *O(f(n)),*

то подразумевается, что f(n) является верхней границей скорости роста *T(n)*.

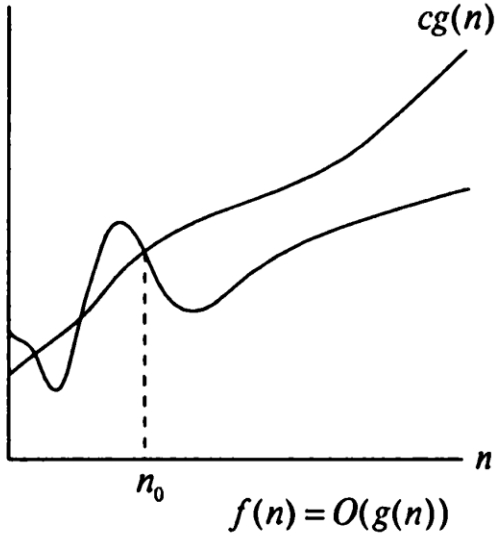
Чтобы указать нижнюю границу скорости роста *T(n),* используется обозначение: *Ω(g(n))* это подразумевает существование такой константы *c*, что бесконечно часто (для бесконечного числа значений n) выполняется неравенство *T(n) > cg(n).*

Временная и пространственная сложности алгоритмов

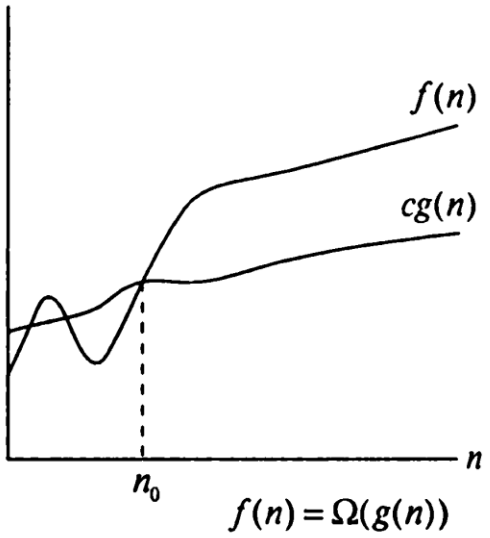
Временная сложность алгоритма в худшем случае — функция размера входных данных, которая показывает максимальное количество элементарных операций, которые могут быть затрачены алгоритмом для решения экземпляра задачи указанного размера-O(f(n)).

Аналогично определяется понятие временная сложность алгоритма в наилучшем случае- *Ω(g(n)).*

Временная и пространственная сложности алгоритмов



б)



в)

$$O(g(n)) = \left\{ \begin{array}{l} f(n): \text{ существуют положительные константы } c \text{ и } n_0 \\ \text{такие что } 0 \leqslant f(n) \leqslant cg(n) \text{ для всех } n \geqslant n_0 \end{array} \right\}$$

$$\Omega(g(n)) = \left\{ \begin{array}{l} f(n): \text{ существуют положительные константы } c \text{ и } n_0 \\ \text{такие что } 0 \leqslant cg(n) \leqslant f(n) \text{ для всех } n \geqslant n_0 \end{array} \right\}$$

Временная и пространственная сложности алгоритмов

Используя O-нотацию можно записать:

$$1^2 + 2^2 + \cdots + n^2 = O(n^4),$$
$$1^2 + 2^2 + \cdot\ \cdot + n^2 = O(n^3),$$
$$1^2 + 2^2 + \cdots + n^2 = \tfrac{1}{3}n^3 + O(n^2).$$

Можно показать $P(n) = O(n^m)$, где $P(n)$ многочлен степени <= m

$$P(n) = a_0 + a_1 n + \cdots + a_m n^m$$

Временная и пространственная сложности алгоритмов

можно записать

$$1/2n^2 + n = O(n^2) \qquad (1)$$

Но ни в коем **е !!!**

несимметричное отношение
«если функция такова, как написано слева от знака равенства,
то она и такова, как записано справа»:

Временная и пространственная сложности алгоритмов

Некоторые операции с символом O

$$f(n) = O\big(f(n)\big),$$

$$c \cdot O\big(f(n)\big) = O\big(f(n)\big), \qquad \text{если } c \text{ — константа},$$

$$O\big(f(n)\big) + O\big(f(n)\big) = O\big(f(n)\big),$$

$$O\big(O(f(n))\big) = O\big(f(n)\big),$$

$$O\big(f(n)\big)O\big(g(n)\big) = O\big(f(n)g(n)\big),$$

$$O\big(f(n)g(n)\big) = f(n)O\big(g(n)\big).$$

Временная и пространственная сложности алгоритмов

По аналогии с временной сложностью, определяют ***пространственную сложность алгоритма***, только здесь говорят не о количестве элементарных операций, а о количестве затраченной памяти

На основе О-символики можно классифицировать сложность алгоритмов в зависимости от размера входных данных (n)

- **Постоянные- сложность не зависит от n: O(1)**
- **Линейные – сложность O (n)**
- **Полиномиальные- сложность $O(n^m)$**
- **Логарифмическая сложность O (log n)**
- **Экспоненциальные- сложность $O(t^{f(n)})$, t >1**
- **Суперполиномиальные- сложность $O(c^{f(n)})$, c- константа, f(n)- функция возрастающая быстрее постоянной, но медленнее линейной**

Временная и пространственная сложности алгоритмов

n	$\log n$	$\sqrt{n}$	n	$n \log n$	n^2	n^3	2^n
2	1	1,4	2	2	4	8	4
4	2	2	4	8	16	64	16
8	3	2,8	8	24	64	512	256
16	4	4	16	64	256	4 096	65 536
32	5	5,7	32	160	1 024	32 768	4 294 967 296
64	6	8	64	384	4 096	262 144	$1{,}83 \times 10^{19}$
128	7	11	128	896	16 384	2 097 152	$3{,}40 \times 10^{38}$
256	8	16	256	2 048	65 536	16 777 216	$1{,}15 \times 10^{77}$
512	9	23	512	4 608	262 144	134 217 728	$1{,}34 \times 10^{154}$
1 024	10	32	1 024	10 240	1 048 576	1 073 741 824	$1{,}79 \times 10^{308}$

Рост некоторых функций

Временная и пространственная сложности алгоритмов

Время выполнения алгоритма aггayMax, есть функция O(n).

- **время выполнения алгоритма aггayMax для размера исходных данных n равно максимально a(7n - 2).**
- **Используя нотацию большого O для c = 7a, а n0 = 1, можно сделать вывод, что**

$$a(7n-2) < cn$$

т.е. временная сложность алгоритма aггayMax - O(n).

Временная и пространственная сложности алгоритмов

Примеры:

Пример : $20n^3 + 10n\log n + 5$ есть $O(n^3)$.

Доказательство: $20n^3 + 10n\log n + 5 \le 35n^3$ для $n \ge 1$.

В сущности, любой многочлен (полином) $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0$ является $O(n^k)$.

Временная и пространственная сложности алгоритмов

Внимание!

Нужно осторожно и корректно использовать асимптотические нотации.

нотация большого О и аналогичные ей могут привести к ошибочным результатам,

если «скрываемые» ими постоянные множители достаточно велики.

(например, $c=10^{100}$)

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).**

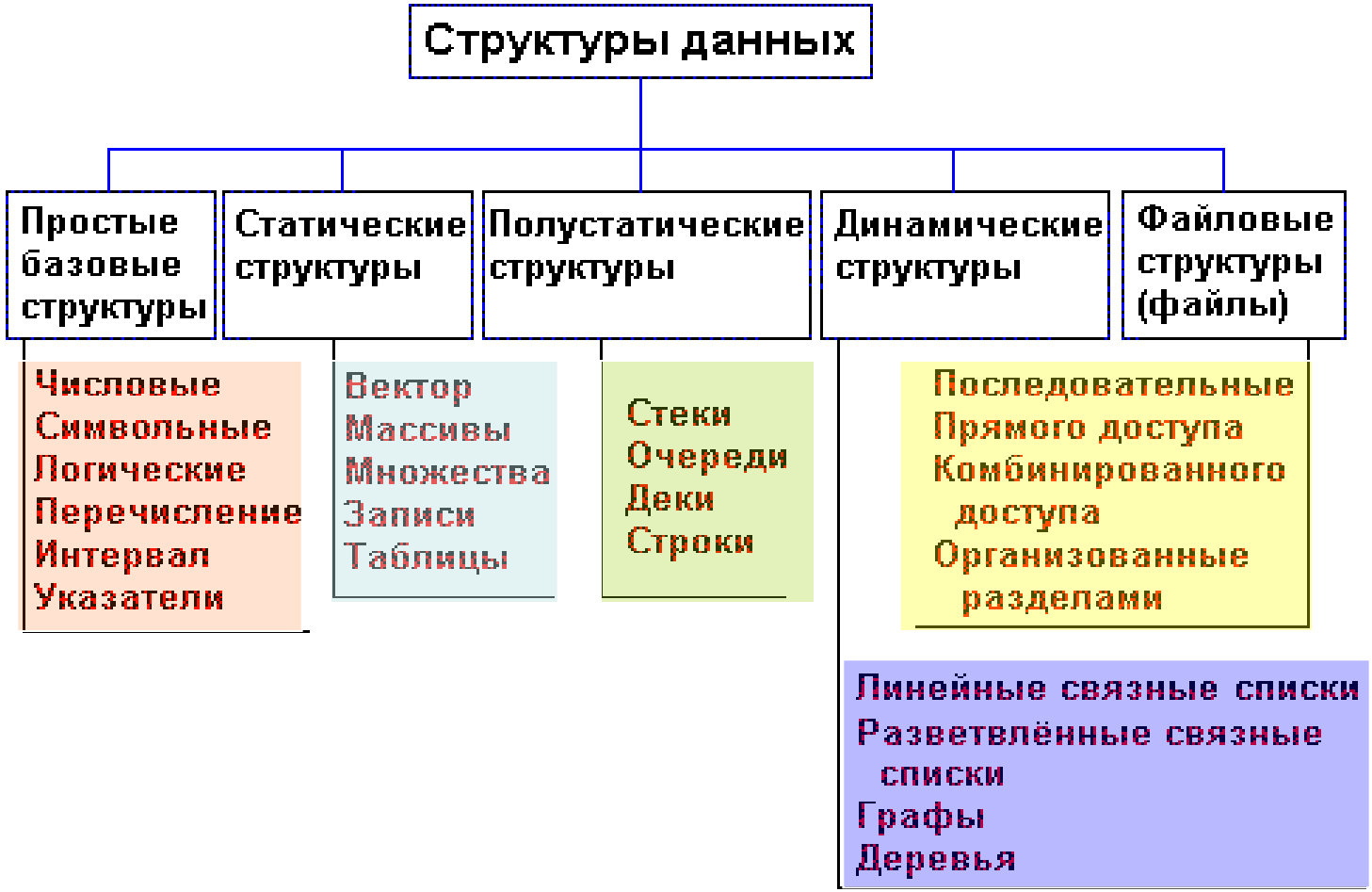
- **Матрица – структура, у которой множество R состоит из двух отношений линейного порядка.**
- **Дерево – множество R состоит из одного отношения иерархического порядка.**
- **Граф – множество R состоит из одного отношения бинарного порядка.**

Вырожденные (простейшие) структуры данных называются также типами данных (примитивные типы).

Различают следующие уровни описания данных:

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

Классификация СД



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**

Встроенные (примитивные) типы данных

В современных компьютерах к таким "машинным" типам относятся

- **целые числа разного размера**

(1-8 байтов)

Встроенные типы данных

- **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 символов из алфавита, зафиксированного в описании языка (ASCII),

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

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

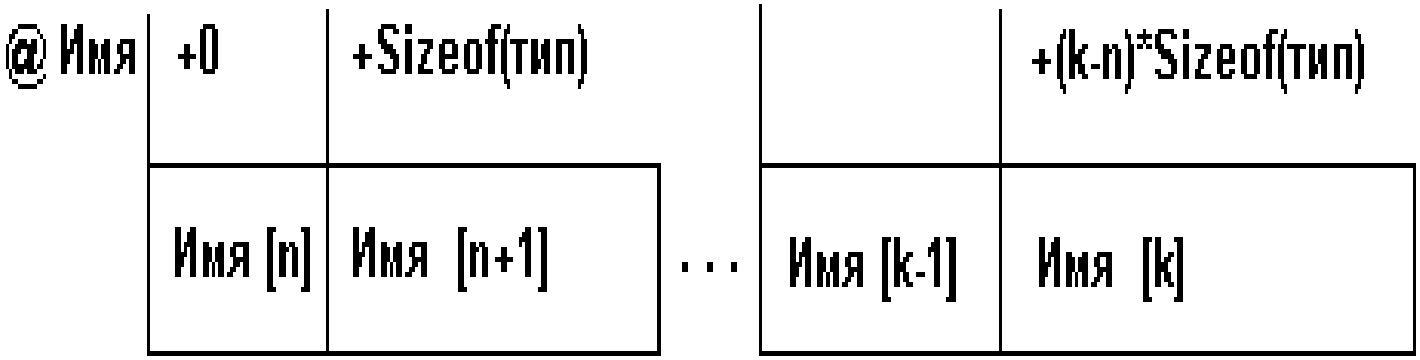
```
Type t  = array [Ti] of To ;
Type row = array [1.. 5] of real;
Type card = array [1.. 80] of char;
Type  alfa = array [0.. 15] of char;
…
Var x: row;
```

C:

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

Массивы

<Имя>: array [n..k] of < тип >;



Массивы

```
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]

Массивы

Количество байтов непрерывной области памяти, занятых одновременно вектором (массивом array [n..k] of < тип >), определяется по формуле:

ByteSise = (k - n + 1) * Sizeof (тип)

Массивы

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).

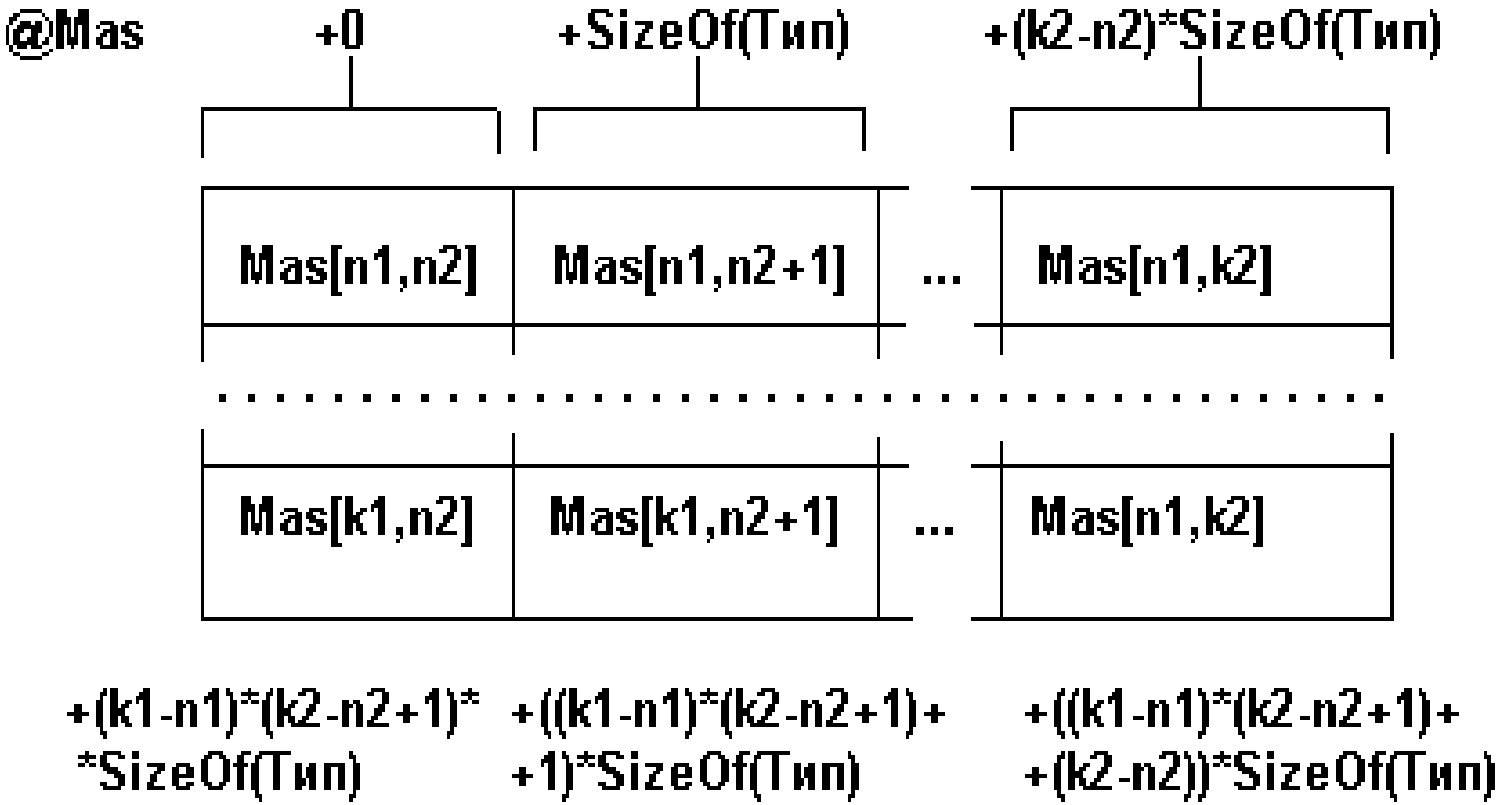
Массивы

- ***Var A: array [-5 .. 4] of Char***

дескриптор может выглядеть

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

Массивы

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)

Массивы

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 многомерного 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;
```

Записи

Представление в виде последовательности полей, занимающих непрерывную область памяти

@rec	+0	+1	+21	+29	+37	+38	+39
	24	Иванов В.И.	ИК	8В03	4	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.

Т.е. максимальное число элементов множества 256, а данные типа множество могут занимать не более 32 байт.

Множества

где @S - адрес данного типа множество.

Множества

Число байтов, выделяемых для данных типа множества, 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

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;
```