

Основы алгоритмизации и программирования

Лекция 9
Массивы

Понятие массива

Введение индексированных переменных в языках программирования позволяет значительно облегчить реализацию многих сложных алгоритмов, связанных с обработкой массивов однотипных данных.

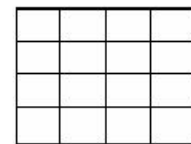
Например, использование массивов данных позволяет компактно записывать множество операций с помощью циклов.

В языке **Си** для этой цели используется сложный тип данных – **массив**, представляющий собой упорядоченную конечную совокупность элементов одного типа. Число элементов массива называют его **размером**. Каждый элемент массива определяется **идентификатором массива** и своим порядковым номером – **индексом**. **Индекс** – целое число, по которому производится доступ к элементу массива. **Индексов** может быть **несколько**. В этом случае массив называют **многомерным**, а количество индексов одного элемента массива является его **размерностью**.

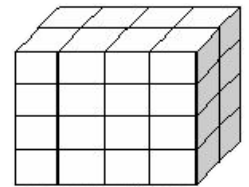
Описание массива в программе отличается от описания простой переменной наличием после имени **квадратных скобок**, в которых задается количество элементов массива. Например, ***double a [10];*** – описание массива из 10 вещественных чисел



ОДНОМЕРНЫЙ МАССИВ



ДВУМЕРНЫЙ МАССИВ



ТРЕХМЕРНЫЙ МАССИВ

Одномерные массивы

тип *ID_массива* [*размер*] = {*список начальных значений*}:

Тип – базовый тип элементов массива (целый, вещественный, символьный); **Размер** – количество элементов в массиве.

Список начальных значений используется при необходимости инициализировать данные при объявлении, он может отсутствовать.

Размер массива вместе с типом его элементов **определяет объем памяти**, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому размер массива задается только константой или константным выражением.

Нельзя задавать массив переменного размера, для этого существует отдельный механизм – **динамическое выделение памяти**.



Индексы массивов в языке Си начинаются с 0, т.е. в массиве **a** первый элемент: **a[0]**, второй – **a[1]**, ... пятый – **a[4]**.

Одномерные массивы

Обращение к элементу массива в программе на языке **Си** осуществляется в традиционном для многих других языков стиле – записи операции обращения по индексу [] (квадратные скобки)

Пример

```
a[0]=1;  
a[i]++;  
a[3]=a[i]+a[i+1];
```

Пример

Объявление массива целого типа с инициализацией начальных значений:

```
int a[5]={2, 4, 6, 8, 10};
```

Если в группе {...} список значений короче, то оставшимся элементам присваивается 0.

В языке **Си** с целью **повышения быстродействия программы** **отсутствует механизм контроля выхода за границы массивов**. При необходимости такой механизм должен быть запрограммирован явно.




Связь указателей и массивов

Идентификатор одномерного массива – это адрес памяти, начиная с которого он расположен, т.е. адрес его первого элемента. Таким образом, работа с массивами тесно взаимосвязана с применением **указателей**.

Пусть объявлены одномерный целочисленный массив a из 5 элементов и указатель p на целочисленные переменные:

```
int a[5]={1, 2, 3, 4, 5}, *p;
```

ID массива a является константным указателем на его начало, т.е. $a = \&a[0]$ – адрес начала массива. Расположение массива a в оперативной памяти, выделенной компилятором, может выглядеть следующим образом



$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	– элементы массива;
1	2	3	4	5	– значения элементов массива;
4000	4002	4004	4006	4008	– символические адреса.

Связь указателей и массивов

Указатель **a** содержит адрес начала массива. В нашем примере равен **4000** (**a = 4000**).

Если установить указатель **p** на объект **a**, т.е. присвоить переменной-указателю адрес первого элемента массива:

$$p = a;$$

что эквивалентно выражению **p = &a[0]**; то получим, что и **p = 4000**

a[0]	a[1]	a[2]	a[3]	a[4]	– элементы массива;
1	2	3	4	5	– значения элементов массива;
4000	4002	4004	4006	4008	– символические адреса.

Тогда с учетом адресной арифметики обращение к *i*-му элементу массива **a** может быть записано следующими выражениями:

$$a[i] \sim *(a+i) \sim *(p+i) \sim p[i],$$

приводящими к одинаковому результату.

Идентификаторы **a** и **p** – указатели, очевидно, что выражения **a[i]** и ***(a+i)** эквивалентны. Отсюда следует, что операция обращения к элементу массива по индексу применима и при его именовании переменной-указателем. Таким образом, **для любых указателей** можно использовать **две эквивалентные формы** выражений для доступа к элементам массива: **p[i]** и ***(p+i)**. Первая форма удобнее для читаемости текста, вторая – эффективнее по быстрдействию программы.

Строки как одномерные массивы данных типа *char*

В языке Си отдельного типа данных «строка символов» **нет**. Работа со строками реализована путем использования одномерных массивов типа *char*, т.е.

строка символов – это одномерный массив символов, заканчивающийся нулевым байтом.

Нулевой байт – это байт, каждый бит которого равен нулю, при этом для нулевого байта определена символьная константа `'\0'` (**признак окончания строки, или «нуль-символ»**). Поэтому если строка должна содержать *k* символов, то в описании массива размер должен быть *k+1*. По положению нуль-символа определяется фактическая длина строки.

Строку можно **инициализировать** строковой константой (строковым литералом), которая представляет собой набор символов, заключенных в двойные кавычки. Например:

```
char S[ ] = "Работа со строками";
```

Приме

`char s[7];` – означает, что ^р строка может содержать не более шести символов, а последний байт отводится под нуль-символ.

Отсутствие нуль-символа и выход указателя при просмотре строки за ее пределы – распространенная ошибка при работе со строками.



Для данной строки выделено и заполнено **19 байт** – 18 на символы и 19-й на нуль-символ.

Строки как одномерные массивы данных типа *char*

В конце строковой константы явно указывать символ `'\0'` не нужно. Компилятор добавит его автоматически.

Символ `'\0'` нужно использовать явно тогда, когда символьный массив при декларации инициализируется списком начальных значений, например, следующим образом:

```
char str[10]={'V' , 'a' , 's' , 'j' , 'a' , '\0'};
```

или когда строка формируется посимвольно в коде программы.

Операция присваивания одной строки другой в языке Си не определена (поскольку строка является массивом) и может обрабатываться при помощи оператора цикла (с использованием стандартной библиотечной функций).

Процесс копирования строки *s1* в строку *s2* имеет вид:

```
char s1[25], s2[25];  
for (int i = 0; i <= strlen(s1); i++)  
s2[i] = s1[i];
```

Длина строки определяется с помощью стандартной функции *strlen*, которая вычисляет длину, выполняя поиск нуль-символа.

Строки как одномерные массивы данных типа *char*

Пример

Не
верно!

```
char s1[51];  
s1 = "Earth";
```

константный указатель и не может использоваться в левой части операции присваивания

Большинство действий со строковыми объектами в **Си** выполняются при помощи **стандартных библиотечных функций**, так, для правильного выполнения операции присваивания в примере необходимо использовать стандартную функцию

Верно!

```
strcpy(s1, "Earth");
```

Для ввода строк, как и для других объектов программы, обычно используются две стандартные функции:

Функция *scanf* вводит значения для строковых переменных при помощи формата (спецификатора ввода) *%s* до появления первого символа "пробел" (символ «&» перед **ID строковых данных** указывать не надо);

Функция *gets* осуществляет ввод строки, которая может содержать пробелы. Завершается ввод нажатием клавиши *Enter*.

Обе функции автоматически ставят в конец строки нулевой байт.

Строки как одномерные массивы данных типа *char*

Вывод строк производится функциями *printf* или *puts* до нулевого байта. Функция *printf* не переводит курсор после вывода на начало новой строки, а *puts* автоматически переводит курсор после вывода строковой информации в начало новой строки. Например:

```
char Str[30];  
printf(" Введите строку без пробелов : \n");  
scanf("%s", Str);
```

или

```
puts(" Введите строку ");  
gets(Str);
```

Остальные операции над строками выполняются с использованием стандартных библиотечных функций, декларация прототипов которых находятся в файле *string.h*.

Часто используемые стандартные строковые функции

Функция *strlen(S)* возвращает длину строки (количество символов в строке), при этом завершающий нулевой байт не учитывается

Пример

```
char *S1 = "Гомель!\0", S2[] = "ГГУ!";  
printf(" %d , %d .", strlen(S1), strlen(S2));
```

Функция *strcpy(S1, S2)* – копирует содержимое строки *S2* в строку *S1*.

Функция *int strcmp(S1, S2)* сравнивает строки *S1* и *S2* и возвращает значение <0 , если $S1 < S2$; >0 , если $S1 > S2$; $=0$, если строки равны, т.е. содержат одно и то же число одинаковых символов

Функция *strcat(S1, S2)* – присоединяет строку *S2* к строке *S1* и помещает ее в массив, где находилась строка *S1*, при этом строка *S2* не изменяется. Нулевой байт, который завершал строку *S1*, заменяется первым символом строки *S2*.

Функции преобразования строковых объектов в числовые описаны в библиотеке *stdlib.h*

Функции преобразования строковых объектов в ЧИСЛОВЫЕ

Функции преобразования строковых объектов в числовые описаны в библиотеке *stdlib.h*

Преобразование строки *S* в число:

– целое: *int atoi(S)*;

– длинное целое: *long atol(S)*;

– действительное: *double atof(S)*;

при возникновении ошибки данные функции возвращают значение 0.

Функции преобразования числа *V* в строку *S*:

– целое: *itoa(V, S, kod)*;

– длинное целое: *ltoa(V, S, kod)*;

$2 \leq kod \leq 36$,

для десятичных чисел со знаком *kod = 10*.

Приме

^p
Участок кода программы, в котором из строки *s* удаляется символ, значение которого содержится в переменной *c* каждый раз, когда он встречается

```
char s[81], c;
```

```
...
```

```
for( i = j = 0; s[i] != '\0'; i++)
```

```
    if( s[i] != c) s[j++] = s[i];
```

```
s[j]='\0';
```

```
...
```

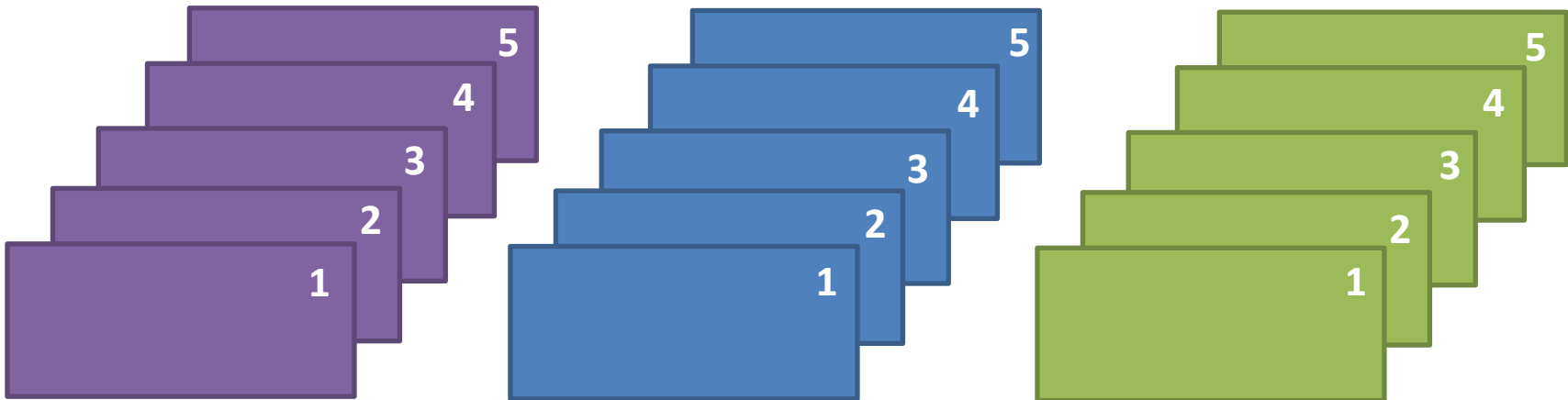
Многомерные массивы

Декларация многомерного массива имеет следующий формат:

```
тип ID[размер1][размер2]...[размерN] =  
{  
  {список начальных значений},  
  {список начальных значений},  
  ...  
};
```

Списки начальных значений – **атрибут необязательный**.

Наиболее **быстро** изменяется **последний** индекс элементов массива, поскольку многомерные массивы в языке **Си** размещаются в памяти компьютера **построчно друг за другом**



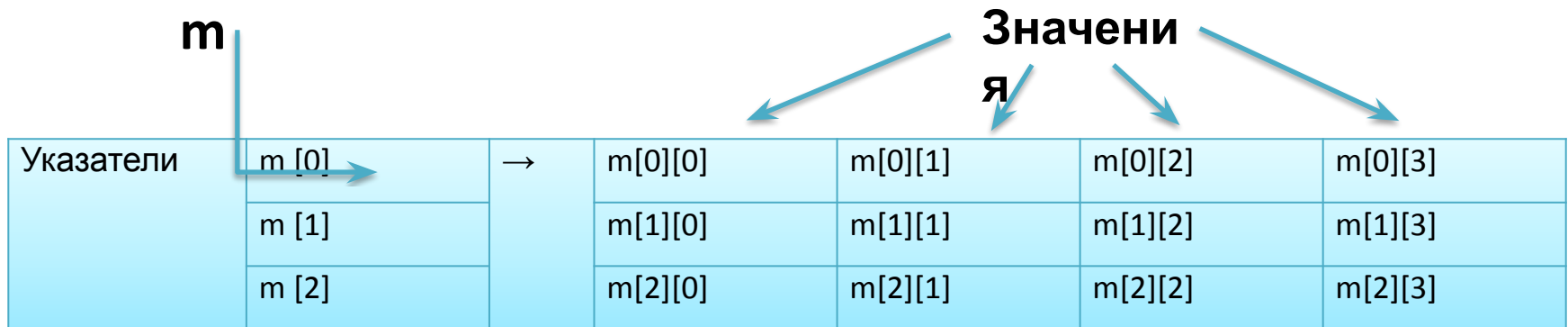
Многомерные массивы

Пусть приведена следующая декларация двумерного массива:

```
int m[3][4];
```

Идентификатор двумерного массива – это указатель на массив указателей (**переменная типа указатель на указатель: $int **m$**);).

Поэтому двумерный массив $m[3][4]$; компилятор рассматривает как массив трех указателей, каждый из которых указывает на начало массива со значениями размером по четыре элемента каждый.



в данном случае указатель $m[1]$ будет иметь адрес $m[0]+4*\text{sizeof}(\text{int})$, т.е. каждый первый элемент следующей строки располагается за последним элементом предыдущей строки.

Многомерные массивы

Пример_
1

```
#include <stdio.h>
void main()
{
int x0[4] = { 1, 2, 3,4};           // Декларация и инициализация
int x1[4] = {11,12,13,14};         // одномерных массивов
int x2[4] = {21,22,23,24};
int *m[3] = {x0, x1, x2,};        // Создание массива указателей
int i,j;
for (i=0; i<3; i++) {
    printf("\n Строка %d) ", i+1);
    for (j=0; j<4; j++)
        printf("%3d", m[ i ] [ j ]);
    }
}
```

Результаты работы программы:

Строка 1) 1 2 3 4

Строка 2) 11 12 13 14

Строка 3) 21 22 23 24

Многомерные массивы

Пример_
2

```
#include <stdio.h>
void main()
{
int i, j;
int m[3][4] = { { 1, 2, 3, 4}, {11,12,13,14}, {21,22,23,24} };
for (i=0; i<3; i++) {
printf("\n %2d", i+1);
    for (j=0; j<4; j++)
        printf(" %3d",m[ i ] [ j ]);
}
}
```

В данной программе **массив указателей** на соответствующие массивы элементов **создается** компилятором **автоматически**, т.е. данные массива располагаются в памяти последовательно по строкам, что является основанием для декларации массива *m* в виде

```
int m[3][4] = {1, 2, 3, 4, 11, 12, 13, 14, 21, 22, 23, 24};
```

Замена скобочного выражения *m[3][4]* на *m[12]* здесь не допускается, так как массив указателей не будет создан.

Многомерные массивы

Таким образом, **использование многомерных массивов** в языке Си связано с **расходами памяти** на создание массивов указателей.

Очевидна и схема размещения такого массива в памяти – последовательное (друг за другом) размещение «строк» – одномерных массивов со значениями (векторная организация памяти).

Обращению к элементам массива при помощи операции индексации $m[i][j]$ соответствует эквивалентное выражение, использующее адресную арифметику – $*(*(m+i)+j)$.

Аналогичным образом можно установить соответствие между указателями и массивами с произвольным числом измерений.