Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №9

Массивы

Массив – это упорядоченный набор объектов, тип которых совпадает, например, набор целых чисел или символов. В массиве можно хранить множество однотипных объектов без необходимости введения отдельного имени переменной для каждого из них. Массив может быть одномерным или многомерным. Количество измерений и длина каждого из измерений определяет общую длину массива, которая ограничена лишь адресным пространством конкретного компьютера.

Массивы являются *производными типами* данных, создаваемыми из существующих типов данных.

Массив, как и любой другой объект в языке Си, должен быть объявлен перед тем, как он будет использован:

тип идентификатор[длина измерения 1][длина измерения 2]...[длина измерения n];

В объявлении массива в квадратных скобках после идентификатора (имени массива) указывается длина каждого измерения. Количество измерений массива определяется по числу квадратных скобок в его объявлении.

Индекс массива определяет используемый элемент массива и указывается в квадратных скобках после имени массива. Индекс массива — это целочисленное выражение, значение которого может быть в диапазоне от 0 до значения, равного длине измерения, уменьшенной на 1.

Примечание 1

В языке Си нумерация элементов массива всегда начинается с 0.

Пример 1

char id[8];

Это объявление отводит 8 байт для 8-элементного массива символов.

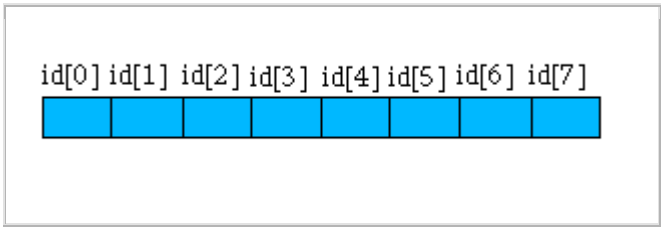

Рис. 1. Одномерный массив

Пример 2

float price[3];

Это объявление отводит место для 3 вещественных чисел.

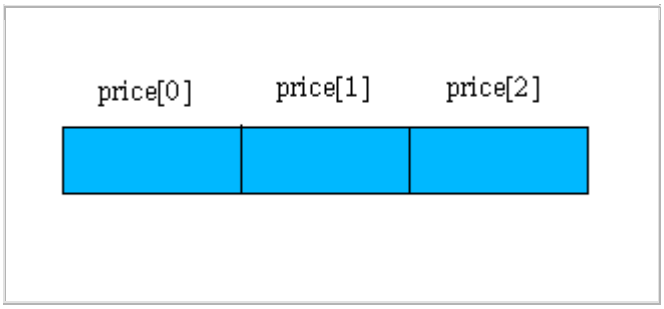

Рис. 2. Одномерный массив

Пример 3

int table[3][3];

Это объявление двумерного массива целых чисел.

Этот массив можно рассматривать как набор из трех 3-элементных массивов целых чисел.

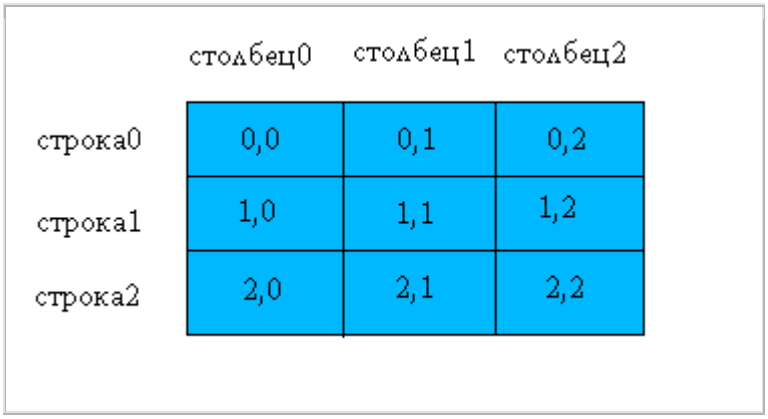

Рис. 3. Двумерный массив

Аналогично, трехмерный массив **val[5][4][3]** можно рассматривать как пять массивов размерности 4 на 3.

Обычно строкам и столбцам придается определенный смысл.

Элементы массива хранятся в памяти последовательно. Если первый элемент символьного массива **id** хранится по адресу 5000, то второй будет храниться по адресу 5001, третий – по адресу 5002 и т.д.

Двумерный массив располагается в памяти построчно: сначала нулевая строка, потом 1 и т.д. Можно сказать, что "последний индекс меняется быстрее других".

Элементы **table[3][3]** хранятся в следующем порядке:

[0][0], [0][1], [0][2], [1][0], [1][1], [1][2], [2][0], [2][1], [2][2]

Доступ к элементам массива

Операция доступа к элементу массива — операция "квадратные скобки", которая используется следующим образом:

имя_массива[индекс]

Эта операция относится к первичным и находится в самой высокой строке таблицы приоритетов.

Имя массива — идентификатор, которому ставится в соответствие начальный адрес массива (адрес первого элемента).

Для доступа к элементу массива (его изменения, печати, присваивания значения другой переменной и т.д.) надо использовать операцию "квадратные скобки".

В скобках должно стоять целочисленное выражение, значение которого должно находиться в диапазоне от 0 до (размер массива – 1).

Примечание 1

В языке Си не выполняется проверка выхода за границу массива. За этим должен следить программист.

Пример 1

```
char id[8];
id[0] ='j';
```

Элементу номер 0 массива **id** присваивается код символа **'j'**.

Результат операции доступа к элементу массива — значение, хранящееся по адресу необходимого элемента массива. Этот адрес определяется следующим образом:

начальный адрес массива + смещение

Смещение — это разность между адресом данного элемента массива и начальным адресом массива.

Для одномерного массива смещение определяется по формуле:

смещение = индекс элемента · длина элемента в байтах

Пример 2

```
char id[8];
id[3] = id[0];
```

Байт из элемента **id[0]** копируется в элемент **id[3].**

Адрес элемента id[3]:

начальный адрес массива id + смещение = начальный адрес массива id + 3·1

Пример 3

```
float price[3];
price[num] = price[num-1]*2;
```

Если значение **num** равно 2, то **price[2] = price[1]*2;**

Адрес элемента price[2]:

начальный адрес массива price + смещение = начальный адрес массива price + 2.4

Для массива произвольной размерности смещение определяется по формуле:

смещение = количество предыдущих элементов · длина элемента в байтах

Пример 4

```
int table[4][3];                    /*table[длина 2-го измерения][длина 1-го измерения]*/
printf ("%d\n", table[2][1]);       /*table[индекс2][индекс1]*/
```

Печатает целое число, хранящееся в 3-й строке 2-го столбца массива **table** .
Адрес элемента table[2][1]:
начальный адрес массива table + смещение =
начальный адрес массива table + (индекс2 . (длина 1-го измерения) +
+ индекс1) . длина элемента в байтах =
=начальный адрес массива table + (2.3 + 1).4

Пример 5

```
/* Запоминает рост вклада по годам в массиве и печатает его */
#include <stdio.h>
int main()
{
const float investment = 6000, interest = .085;
float  rate;
double value[5];
int    year;

rate = 1 + interest;                    /* 108.5% от предыдущей суммы */
value[0] = investment;
year = 1;
while (year < 5)
  {
   value[year] = value[year-1] * rate;
   year += 1;                           /* year = year+1 */
  }
printf("начальный вклад: %.2f\n", value[0]);
year = 1;
while (year < 5)
   {
    printf("год %.d\n", year);
    printf("%.2f\n", value[year]);
    year += 1;                          /* year = year+1 */
  }
}
```

Массивы символов

Символьная строка — это последовательность символов из некоторого набора, например из таблицы кодирования ASCII. В языке Си для хранения символьных строк используются символьные массивы. Например, для того, чтобы зарезервировать место для буфера объемом в 1кбайт, можно воспользоваться следующим объявлением:

char buf[1024];

Строки в языке Си должны заканчиваться *признаком конца строки* — "нулевым" символом `'\0'`, т.е. байтом, все биты которого равны 0. Библиотечные функции, которые работают со строками, "ищут" нулевой символ, так как это единственный способ определить конец строки. Таким образом, при объявлении символьного массива, в котором будет храниться строка, необходимо предусмотреть дополнительный байт для признака конца строки.

Пример 1

```
/* запрашивает у пользователя имя и выводит его*/
#include <stdio.h>
int main()
 { int   i;
   char first[11];                    /* длина имени  до 10 символов */

   printf ("Пожалуйста, введите имя \n");
   i = 0;
   while (i<10 && (first[i] = getchar ()) != '\n')
       i += 1;
   first[i] = '\0';
   printf (" Введено имя: %s\n", first);
 }
```

Обработка массивов

Выход за границу массива

Выход за границу массива — обращение к элементу массива, индексы которого выходят за указанные в объявлении массива пределы.

Программист сам должен заботиться о том, чтобы индексы элемента массива лежали в пределах, указанных при объявлении массива.

Для увеличении скорости работы программы не выполняется проверка, лежит ли индекс массива в указанных при объявлении пределах. Такая проверка должна производится во время работы программы каждый раз при обращении к массиву с использованием этого индекса. Если бы Си проверял допустимость индекса массива, то программа работала бы намного медленнее.

Но помимо увеличения скорости работы программы, отсутствие проверки индексов имеет и нежелательные последствия — индекс может выйти за границы массива. В этой ситуации возможно одно из двух:

- что-то будет прочитано или записано в память за границами массива;
- получившийся адрес будет лежать за пределами доступной области памяти, например, при попытке записи в блок памяти, доступной только для чтения (при этом выполнение программы аварийно прекратится).

Программист должен обеспечивать изменение индекса массива в диапазоне от 0 до (размер_массива-1).

Пример 1

```
/*Пример выхода за границу массива: */
int ray[20], i;
i = 0;
while (i <= 20)
   {
    ray [i] = i*10;
    i += 1;
   }
```

Копирование массивов

Массивы должны копироваться поэлементно.

Нельзя копировать массивы так, как это показано в прим. 2. В левой части операции присваивания нельзя использовать только имя массива.

Имя массива, используемое в программе без скобок, воспринимается как адрес, начиная с которого хранится массив. Этот адрес нельзя изменить, он не является именующим выражением.

Пример 2

```
int prev[20], current[20];
prev = current;              /*Неправильно: так копировать  массив нельзя */
```

Пример 3

```
int prev[20], current[20];
for (i=0; i<20; i+=1)        /*Правильно: копировать массив надо так */
  prev[i] = current [i];
```

Примечание 1

Для обработки массивов очень удобно использовать цикл **`for`**.

Ввод массивов

Ввод массивов оуществляется поэлементно:

Пример 4

```
int a[5];                    /*массив на 5 целых чисел*/
for(i=0;i<5;i++)
     scanf("%d", &a[i]);
```

Примечание 2

Использовать функцию **`scanf()`** надо очень аккуратно и грамотно, так как при ошибке ввода она не очищает буфер ввода.

Вывод массива

Вывод массива также осуществляется поэлементно:

Пример 5

```
int b[7]={-3,5,8,-45,0,-1,8};
for(j=0;j<7;i++)
   printf("%5d", a[j]);
printf("\n");                    /*переходим на следующую строку*/
```

Инициализация массива

Инициализация массива — это присваивание элементам массива начальных значений.

Внешние, статические внешние, статические внутренние и автоматические переменные составных типов данных, таких как массивы, могут быть проинициализированы при их объявлении. Любая автоматическая переменная или ее часть (например, элемент массива), которые не были инициализированы, остаются неопределенными. Любая внешняя или статическая переменная или часть ее (например, элемент массива), будучи не проинициализированными, устанавливаются в 0 или **`NULL`** (для указателей).

Пример 1

```
char alpha[10];
char beta[10] = {'a, 'b', 'c'};
char gamma[10] = "This is gamma";
char delta[5][10] ={"line 1",
                    "line 2",
                    "line 3"};
int num1[10];
int num2[10] = {2, 4, 6, 8, 10};
int num3[5][10] ={{0, 1, 2, 3, 4},
                  {2, 4, 6, 8, 10},
                  {3, 6, 8, 10, 12}};
int main ()
{
```

```
static char local[] = "local string";
char  errmsg[]     = "name", "too", "long";
int     lengths[10]   = {23, 56, 83};
...
}
```

Размеры строк **gamma** и **local** равны количеству символов в строке, увеличенному на 1 (для символа **NULL** — признака конца строки).

Автоматические переменные инициализируются при каждом входе в блок. Внешние и статические переменные инициализируются один раз, перед началом выполнения программы.

Передача в функцию адреса массива

Язык Си не допускает копирования всего массива для передачи его в функцию. Тем не менее, функции можно передавать элемент массива или начальный адрес массива.

Адрес массива — адрес первого элемента массива.

Для того чтобы функция имела доступ к массиву, ей достаточно передать адрес массива.

Пример 1

```
/* Копирует массив с использованием функции */
int main( )
{
void stringcopy (char[], char[]);
char current[30], target[30];
. . .
stringcopy(current, target);
. . .
}
/* Копирует str1 в str2 */
void stringcopy(char str1[], char str2[])
/* str1 = 600 str2 = 500 */
{
int i;
for ( i = 0; str2[i]!='\0'; i++)
  str1[i]=str2[2];                /* адрес[смещение] */
str1[i]='\0';
}
```

Выражение типа **str2[i]** интерпретируется так, что означает адрес[смещение].

Если в вызове функции в качестве фактического параметра используется имя массива без скобок, то значением этого параметра будет начальный адрес массива. Если в примере начальные адреса массивов **current** и **target** равны соответственно 600 и 500, то в момент вызова функции **stringcopy()**значения формальных

параметров **str1** и **str2** будут равны соответственно 600 и 500. Зная эти адреса, функция **stringcopy()** может обращаться к элементам массива, используя операцию доступа к элементу массива **[]**. Выражение **current[3]**, так и **str2[3]** будут интерпретированы как 500[3]. Доступ выполняется к значению, находящемуся по адресу:

$$500 + (3 \cdot (\text{количество байтов в элементе})) = 500 + 3 \cdot 1$$

В примере 2 показаны два способа объявления параметров функции при передаче адреса массива:

Пример 2

```
int main()
{
int str1[20], str2[20];
void array_cpy(char[], char[]);
void ptr_cpy(char*, char*);
...
array_cpy(str1, str2);
/*ptr_cpy(str1, str2);*/
...
}

void array_cpy(char one[], char two[])
{
int i;
for (i = 0; two[i]!='\0'; i++)
  one[i]=two[i];
one[i]='\0';
}
```

В функции **array_cpy()** формальные параметры объявляются с использованием операции доступа к элементу массива **[]**. Эта же операция применяется и в теле функции.

```
void ptr_cpy(char *one, char *two)
{
for ( ; *two!='\0'; one++, two++)
  *one=*two;
*one='\0';
}
```

В функции **ptr_cpy()** формальные параметра объявляется как указатели. В теле функции используется операция косвенного доступа *****. Такая совместимость объявлений и операций настоятельно рекомендуется по стилевым причинам, но она не является необходимой. Тела двух функций можно без ущерба поменять местами. Важно помнить, что доступ к данным, хранящимся по адресу, обеспечивают обе

операции * и [] . Операцию * можно применять даже к имени массива. В действительности, выражение **array_name[i]** преобразуется компилятором в выражение ***(array_name** + **i)**.

Передача в функцию двумерного массива

Многомерный массив — массив, имеющий более одного измерения. Число измерений массива в языке Си не ограничено.

В функцию можно передавать лишь адрес массива, а не копию всего массива. Тем не менее, в функции для доступа к элементам массива можно использовать индексы.

Пример 1

```
/* Программа для чтения текстового файла и выдачи на печать лишь
тех слов, которые начинаются с прописной буквы */


#include <stdio.h>
int main()
{
  void print_caps(int, const char [][20]);
  int amount_read = 0;
  char input[1000][20];              /* Это массив из 1000 массивов из 20 символов */

  while (scanf("%s", input[amount_read]) == 1)
     amount_read++;                  /* input[amount_read] - один из массивов из 20 символов */
  print_caps(amount_read, input);
}

/* print_caps печатает лишь те слова, которые начинаются с прописной буквы */
void print_caps(int n_words, const char words[][20])
/* words - Это массив из произвольного числа массивов из 20 символов */
{
   int i=0;
   while (i < n_words)
      {
      if (words[i][0] <= 'Z' && words[i][0] >= 'A')
        printf("%s\n", words[i]);
                                    /* words[i][0] - 0-ой символ i-го массива из 20 символов */
                                    /* words[i] - Один из массивов из 20 символов*/
      i++;
      }
 }
```

Значением операции доступа к элементу массива **words[i][j]** будет адрес, вычисленный по следующим формулам:

$$\text{адрес} = \text{начальный адрес массива} + \text{смещение}$$

$$\text{смещение} = (i \cdot (\text{число элементов в строке}) + j) \cdot (\text{длина элемента в байтах})$$