Массивы

Артамонов Ю.Н.

Международный университет
природы, общества и человека "Дубна"
филиал Котельники

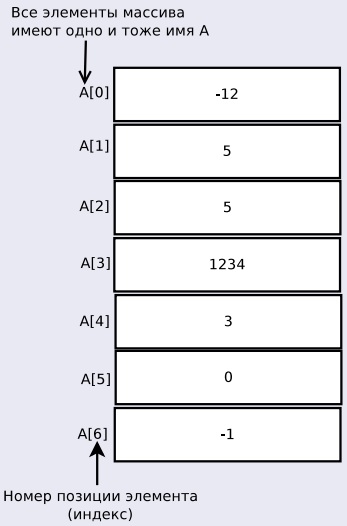
21 февраля 2018 г.

Содержание

Понятие массива

Массив - это структура данных, состоящая из логически связанных элементов *одного и того же типа.*

Логическая связь состоит в том, что все элементы имеют одно и тоже имя и один и тот же тип данных. Для обращения к конкретной области (элементу массива), необходимо указать имя массива и *номер позиции* этого элемента в массиве (индекс). На рисунке показан созданный целочисленный массив и указаны ссылки на его элементы. Массив содержит 7 элементов. **Первый элемент** массива имеет **нулевой порядковый номер**. Таким образом, i-й элемент массива можно получить как $A[i-1]$.



Способы задания массивов

Массивы занимают определенное место в памяти. Программистом устанавливается тип каждого элемента и число элементов, требуемых для каждого массива, чтобы компьютер мог зарезервировать соответствующий объем памяти. Например, чтобы зарезервировать семь целочисленных элементов в виде массива A, необходимо в программе сделать объявление:

```
int A[7];
```

Кроме того, с помощью одного объявления можно резервировать память для нескольких массивов. Например, с помощью объявления:

```
int A[7], B[100];
```

кроме массива A из 7 элементов, также создается массив B из 100 элементов.

```
int A[7], B[100]; //Объявление целочисленных массивов
char C[300]; //Объявление массива символов
float D[450]; //Объявление массива вещественных чисел
```

Примеры использования массивов

```
#include <stdio.h>
main()
{
  int A[6], i;
  //Выводим элементы не инициализированного массива
  for (i = 1; i<=6; i++)
    {
      if (i % 3 != 0)
        printf("A[%d] = %-10d\t", i, A[i-1]);
      else
        printf("A[%d] = %-10d\n",i, A[i-1]);
    }
  return 0;
}
//Вот так это работает
A[1] = 4195728    A[2] = 0          A[3] = 4195344
A[4] = 0          A[5] = -2056494144 A[6] = 32765
```

Неинициализированный массив может быть заполнен произвольными целыми числами

Способы инициализации массива

В программе элементы массива можно инициализировать нулями:

```
#include <stdio.h>
main()
{
  int A[6], i;
  for (i = 1; i<=6; i++) A[i-1] = 0;
  for (i = 1; i<=6; i++)
    {
      if (i % 3 != 0)
    printf("A[%d] =%-10d\t", i, A[i-1]);
      else
    printf("A[%d] =%-10d\n",i, A[i-1]);
    }
  return 0;
}
```

Однако существуют и другие способы.

Способы инициализации массива

Элементы массива можно инициализировать при объявлении массива путем помещения вслед за его объявлением знака равенства и списка (заключенного в фигурные скобки) *инициализирующих значений*, разделенных запятыми. Вот пример:

```
int A[6] = {1,3,5,7,9,11};
```

Если инициализирующих значекний меньше, чем элементов массива, остальные элементы автоматически инициализируются нулями. Например, массив можно инициализировать нулями следующим образом:

```
int A[6] = {0};
```

Следует помнить, что такая инициализация происходит на этапе компиляции. Поэтому, если инициализировать массив требуется многократно, то этот способ не подходит.

Способы инициализации массива

Если в массиве меньше элементов, чем инициализирующих значений, это приводит к ошибке:

```
int A[3] = {1,3,5,7,9,11}; //Возникнет сообщение об ошибке
```

Если размер массива не указан, то его размер становится равен количеству инициализирующих элементов:

```
int A[ ] = {1,3,5,7,9,11};
```

Для задания размера массива часто используют **символическую константу**. Для этого используется директива препроцессора #define

```
#include <stdio.h>
#define Size 6 //Использование символических констант делает программу
  масштабируемой
main(){
  int A[Size] = {1,3,5,7,9,11}; return 0;}
```

Обратите внимание, что точка с запятой после #define Size 6 не ставится! Иначе константе Size будет присвоено значение «10;».

Примеры использования массивов

Заполнить массив из 10 элементов случайными целыми числами.
Найти:

- минимальный, максимальный элементы массива;
- количество четных чисел;
- среднее значение: $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$ (в нашем случае $n = 10$);
- среднее квадратическое отклонение:$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$ (в нашем случае $n = 10$).

Решение задачи

```
#include <stdio.h>
#include<math.h>
#define SIZE 10
main()
{
   int A[SIZE] = {0}; //Инициализируем массив нулями
   /*Устанавливаем псевдослучайную последовательность по системному времени */
   srand(time(NULL));
   //Заполняем массив случайными числами от 0 до 99
   int i;
   for (i = 1; i <= SIZE; i++) A[i-1] = rand() % 100;
   //Выводим числа на экран
   for (i = 1; i <= SIZE; i++)
        printf("A[%d] = %-5d\n",i-1,A[i-1]);
```

Решение задачи

```
  int max  = A[0], min = A[0], chet = 0;
  for (i = 0; i<SIZE; i++)
    {
      if (min>A[i]) min = A[i];
      if (max<A[i]) max = A[i];
      if (A[i] % 2 == 0) chet++;
    }
  //Выводим на экран min, max, chet
  printf("Максимальный элемент массива равен: %d\n", max);
  printf("Минимальный элемент массива равен: %d\n", min);
  printf("Количество четных элементов массива равно: %d\n", chet);
  //Подсчет среднего значения
  float sr = 0;
  for (i = 0; i<SIZE; i++) sr +=A[i];
  sr /= SIZE;
  printf("Среднее значение элементов массива равно: %.3f\n", sr);
}
```

Решение задачи

```
    //Подсчет среднего квадратического отклонения
    float sko = 0;
    for (i = 0; i<SIZE; i++) sko +=pow(A[i]-sr,2);
    sko /=SIZE;
    sko = sqrt(sko);
    printf("Среднее квадратическое отклонение массива равно: %.3f\n", sko);
    return 0;
}
```

Задача

Задать координаты двух векторов (используя два массива): $\bar{A}(x_a, y_a), \bar{B}(x_b, y_b)$. Найти угол (в градусах) между этими векторами:

$$\bar{A} \cdot \bar{B} = |A| \cdot |B| \cdot \cos(\varphi) \Rightarrow$$

$$\varphi = \arccos\left(\frac{\bar{A} \cdot \bar{B}}{|A| \cdot |B|}\right)$$

$$|A| = \sqrt{x_a^2 + y_a^2}, |B| = \sqrt{x_b^2 + y_b^2}$$

$$\bar{A} \cdot \bar{B} = x_a \cdot x_b + y_a \cdot y_b$$

Решение задачи

```
#include <stdio.h>
#include<math.h>
#define SIZE 2
main()
{
   float A[SIZE], B[SIZE];
   printf("x_a="); scanf("%f", &A[0]);
   printf("x_b="); scanf("%f", &B[0]);
   printf("y_a="); scanf("%f", &A[1]);
   printf("y_b="); scanf("%f", &B[1]);
   float fi;
   fi = 180*acos((A[0]*B[0]+A[1]*B[1])/(sqrt(A[0]*A[0]+A[1]*A[1])*
  sqrt(B[0]*B[0]+B[1]*B[1])))/M_PI;
   printf("Угол между векторами составляет %f градусов \n", fi);
   return 0;
}
```

Задача

Заполнить массив из 10 элементов случайными числами и отсортировать его по возрастанию методом выбора.
Метод выбора: из всех элементов с первого по последний выбирается минимальный и он меняется местами с первым элементом массива, затем из элементов со второго по последний выбирается минимальный и он меняется местами со вторым элементом массива и т.д.

Решение задачи

```
#include <stdio.h>
#include<math.h>
#define SIZE 10
main()
{
  int A[SIZE];
  srand(time(NULL));
  int i;
  for (i = 1; i<=SIZE; i++)
    {
      A[i-1] = rand() % 1000; printf("%d; ",A[i-1]);
    }
}
printf("\n");
```

Решение задачи

```
  int j, min, index_min;
  for (i  = 0; i<SIZE; i++)
    {
      min = A[i];
      index_min = i;
      for (j = i; j<SIZE; j++)
if (min > A[j])
  {
    min = A[j]; index_min = j;
  }
      A[index_min] =A[i];
      A[i] = min;
    }
  for (i = 1; i<=SIZE; i++) printf("%d; ",A[i-1]);
  printf("\n");
  return 0;
}
```

Задача 6.1

Заполнить массив из 10 элементов случайными числами и отсортировать его по возрастанию методом пузырька.
Метод пузырька: первый элемент сравнивается со вторым, если второй меньше, они меняются местами. Затем второй сравнивается с третьим, если третий меньше они меняются местами и т.д. до последнего элемента. После одного такого прохода самый минимальный элемент станет первым. Далее алгоритм повторяется со второго элемента.

Задача 6.2

Определить, симметричны ли элементы массива?

Задача 6.3

Требуется переставить элементы массива в обратном порядке.

Задача 6.4

Требуется циклически сдвинуть элементы массива на одну позицию вправо.

Задача 6.5

Требуется циклически сдвинуть элементы массива на k позиций вправо.

Задача 6.6

Каждый элемент массива (кроме первого и последнего) заменить на полусумму соседних элементов.

Задача 6.7

Требуется сгруппировать положительные элементы массива в его начале, а отрицательные - в конце с сохранением их порядка.

Задача 6.8

Введите с клавиатуры число x и определите индекс и значение элемента массива, ближайшего к числу x.

Задача 6.9

Введите с клавиатуры число x и определите, к какому значению ближе всего x: к минимальному в массиве, максимальному или среднему арифметическому.

Задача 6.10

Отсортировать массив, заполненный случайными числами. Проверить, содержится ли элемент x, введенный с клавиатуры, в данном массиве методом половинного деления.

Задача 6.11

Даны два неубывающих массива, требуется построить из них третий неубывающий массив, который является объединением двух первых.

Задача 6.12

Реализовать алгоритм решета Эратосфена для поиска всех простых чисел от 1 до n.

Передача массивов в функции

Для передачи массива в качестве параметра функции следует указать просто его имя без скобок. Например, пусть массив A объявлен как:

```
int A[100];
```

тогда оператор вызова функции будет выглядеть так:

```
my_func(A);
```

Часто в функцию передают и размерность массива:

```
my_func(A,100);
```

В функцию массив **передается по ссылке**. Это означает, что в функции мы имеем непосредственный доступ ко всем элементам массива. На самом деле в функцию передается просто физический адрес первого элемента массива. Чтобы получить физический адрес элемента массива, используется конструкция: **&A[0]** - физический адрес первого элемента массива.

Передача массивов в функции (продолжение)

Следующая программа демонстрирует вывод адресов элементов массива и адрес самого массива.

```
  #include <stdio.h>
main()
{
   int A[4],i;
   printf("%p\n", &A);
   for (i = 1; i<=4; i++)
      printf("%p\n", &A[i-1]);
   return 0;
}
0x7ffedf25ee00
0x7ffedf25ee00
0x7ffedf25ee04
0x7ffedf25ee08
0x7ffedf25ee0c
```

Обратите внимание, что массив и первый элемент имеют один и тот же адрес!

Передача массивов в функции (продолжение)

Для печати адреса используется специальный спецификатор %p (он выводит адреса в виде шестнадцатиричного числа). Следует иметь ввиду, что отдельные элементы массива **передаются в функцию по значения** (то есть, в функцию передается копия элемента, и все его изменения в функции не отражается на его внешних изменениях).

Для того, чтобы функция при обращении к ней смогла принять массив, в списке параметров этой функции должно быть указано, что ожидается передача массива:

```
void my_func(int Ar[], int size)
```

В таком объявлении видно, что функция my_func ожидает в параметре Ar массива целых чисел, а в параметре size - его размера. Размер массива в квадратных скобках указывать не нужно (он все равно будет проигнорирован компилятором).

Прототип для функции будет выглядеть так:

```
void my_func(int [], int );
```

Передача массивов в функции (продолжение)

В программах могут возникать ситуации, когда функция не должна иметь возможность изменить массив, а может его только читать. Для этого в языке C предусмотрен специальный **модификатор типа const**. Когда параметру - массиву предшествует **const**, элементы массива становятся константами и любая попытка изменить массив приводит к ошибке.

Пример работы с const

```
#include <stdio.h>
#include <stdlib.h>
#define size 100
void test(const int [ ]);
main()
{
  int A[size], i;
  srand(time(NULL));
  for (i = 0; i<size; i++)
    {
      A[i] = rand()%500;
       printf("%d  ", A[i]);
    }
  printf("\n"); return 0;
}
void test (const int A []){
  A[0] = 666; //test.c:25:8: error: assignment of read-only location
   '*'A A[0] = 666;
}
```

Еще одна особенность работы с массивами в C

Важно! В языке C отсутствует проверка на выход за пределы массива, предотвращающая ссылку на несуществующий элемент. Ответственность за это возлагается на программиста. Например, следующая программа будет успешно скомпилирована:

```
#include <stdio.h>
#include <stdlib.h>
#define size 100
main()
{
  int A[size/2], i;
  srand(time(NULL));
  for (i = 0; i<size; i++)
    {
      A[i] = rand()%500;
    }
  printf("\n");
}
```

Еще один пример работы с массивами

Написать программу, которая моделирует подбрасывания шестигранного кубика N раз, подсчитывает выпадение каждой грани и выводит соответствующую гистограмму в виде символов *.

```
#include <stdio.h>
#include <stdlib.h>
#define N 20
main()
{
  int Arr[6] = {0};
  srand(time(NULL));
  int i, event;
  for (i = 0; i<N; i++)
    {
      event = 1+ rand()%6;
      Arr[event-1]++;
    }
```

Еще один пример работы с массивами (продолжение)

```
  printf("Гистограмма частот\n");
  int j;
  for (i = 0; i<6; i++)
    {
      printf("%d: ", i+1);
      for (j = 1; j<=Arr[i]; j++) printf("*");
      printf("\n");
    }
  return 0;}
```

```
1: **
2: *****
3: ***
4: **
5: ****
6: ****
```

Особенности работы с массивами из символов

В языке C строка типа «Hello» на самом деле является массивом отдельных символов. Символьные массивы имеют несколько особенностей. Символьный массив может быть инициализирован строкой. Например, код

```
char my_str[] = "first string";
```

инициализирует элементы массива my_str отдельными символами строки «first string». Размер массива my_str определяется компилятором исходя из длины строки. Строка "first string"содержит 12 символов плюс специальный символ окончания строки, называемый *нулевым символов*. Для представления нулевого символа используется специальная символьная константа '\0'. Таким образом, предыдущее объявление эквивалентно следующему:

```
char my_str[] = {'f', 'i', 'r', 's', 't', ' ', 's', 't', 'r', 'i', 'n', 'g', '
  \0'};
```

Особенности работы с массивами из символов

Поскольку строка является символьным массивом, используя индексы, можно обращаться к элементам строки - символам:

```
#include <stdio.h>

main()
{
  char ar1[] = "First string";
  int i;

  for (i = 0; i<12; i++) printf("%c\n",ar1[i]);

  return 0;
}
```

Особенности работы с массивами из символов

Также можно вводить строку в символьный массив непосредственно с клавиатуры, используя спецификатор %s. Например,

```
char ar2[10];
scanf("%s", ar2);
```

Вначале определен массив символов из 10 символов. После этого с использованием scanf осуществляется считывание строки и запись ее в массив ar2.Важно отметить, что контроль считывания не более 20 символов при таком способе нигде не контролируется. Поэтому, если пользователь введет более 20 символов, они все будут записаны в память, что может привести к потере данных и другим ошибкам. Вся ответственность за это возлагается на программиста.

С использованием спецификатора %s можно также выводить массив символов:

```
printf("%s", ar2);
```

Особенности работы с массивами из символов

Символы строки выводятся до тех пор, пока не будет встречен завершающий нулевой символ. Сами функции printf, scanf не заботятся о размерах символьного массива.

```
#include <stdio.h>
#define size 5
main()
{
  char ar[size];
  int i=0;
  while ((i<size-1)&&((ar[i] = getchar()) != EOF)) i++;
  ar[size-1] = '\0';
  printf("\n");
  for (i = 0; ar[i] != '\0'; i++) printf("%c ", ar[i]);
  printf("\n");
  return 0;
}
```

Особенности работы с массивами внутри функций

Часто в функциях приходится работать с довольно длинными массивами. Обычное объявление массива внутри функции

```
int A[200000];
```

интерпретируется со спецификатором памяти auto. При выходе из функции память, занятая под массив, автоматически освобождается. Повторный вызов такой функции приведёт к повторному выделению памяти под массив, а это требует времени. При частом вызове такой функции работа программы становится медленной. В этих случаях целесообразно использовать спецификатор памяти static (при этом массив сохраняется для повторного использования). Ниже код демонстрирует различия в использовании массивов в функции с разными спецификаторами класса памяти.

Пример на использование массива static в функции

```
#include <stdio.h>
void StaticArray(void);
main()
{
  StaticArray(); printf("\n"); StaticArray();
  return 0;
}
void StaticArray(void)
{
  static int ar[5];
  int i;
  for (i = 0; i<5;i++) printf("a[%d] =%d  ",i,ar[i]);
  for (i = 0; i<5;i++) ar[i]+=5; printf("\n");
  for (i = 0; i<5;i++) printf("a[%d] =%d  ",i,ar[i]);
  printf("\n");
}
```

Пример на использование массива static в функции (результат)

```
a[0] = 0   a[1] = 0   a[2] = 0   a[3] = 0   a[4] = 0
a[0] = 5   a[1] = 5   a[2] = 5   a[3] = 5   a[4] = 5

a[0] = 5   a[1] = 5   a[2] = 5   a[3] = 5   a[4] = 5
a[0] = 10   a[1] = 10   a[2] = 10   a[3] = 10   a[4] = 10
```

Задача 6.13

Реализовать функцию, которая находит **медиану, среднее геометрическое, среднее гармоническое** введенных с клавиатуры значений.

Медиана - это середина ранжированной совокупности.

Среднее геометрическое: $\bar{x}^{\text{геом}} = (\prod_{i=1}^{n} x_i)^{\frac{1}{n}}$

Среднее гармоническое: $\bar{x}^{\text{гарм}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$

Определение многомерного массива

Массивы в C могут иметь несколько индексов. Например, двумерные массивы часто применяют для представления плоских таблиц из столбцов и строк. Для получения конкретного элемента такой таблицы мы должны указать номер строки и номер столбца, на пересечении которых находится данный элемент. Иногда применяют массивы размерности более двух. В стандарте ANSI C установлено, что система должна поддерживать не менее 12 индексов массива (т.е. как минимум 12-мерный массив). На практике, конечно, обычно обходятся 2, 3, 4 мерными массивами.

Каждый элемент двумерного массива индексируется как **A[i][j]**, где i - номер строки (нумерация с нуля), j - номер столбца (нумерация с нуля).

Обратите внимание, что ошибочно писать A[i,j].

Для объявления двумерного массива из целых чисел размерностью 3 строки, 2 столбца следует написать:

```
int A[3][2];
```

Определение многомерного массива

Многомерный массив, подобно одномерному, может быть инициализирован при его объявлении. Например,

```
int A[3][2] = {{1,2},{3,4},{5,6}}
```

Значения группируются в фигурных скобках по строкам. Так элемент A[0][0] = 1; A[0][1]=2; A[1][0] =3; A[1][1] =4; A[2][0]=5; A[2][1]=6.
Если для данной строки недостаточно инициализирующих значений, остальные ее элементы инициализируются нулями, например:

```
int A[3][2] = {{1},{3,4},{}}
```

Ниже приведен пример вывода значений этого массива.

Пример вывода значений многомерного массива

```
#include <stdio.h>
main()
{
  system("clear"); //вызываем системную команду очистки экрана в Linux
  int A[3][2] = {{1},{3,4},{}};
  int i,j;
  for (i = 0; i<3; i++)
    for (j = 0; j<2; j++)
      printf("A[%d][%d] =%d\n", i,j,A[i][j]);
  return 0;
}
```

```
A[0][0] = 1
A[0][1] = 0
A[1][0] = 3
A[1][1] = 4
A[2][0] = 0
A[2][1] = 0
```

Передача многомерных массивов в функции

Важно! В определении функции параметр массив объявляется с указанием количества столбцов:

```
void printarray(int A[][2])
```

Когда функция принимает одномерный массив, в списке параметров функции скобки остаются незаполненными. Первый индекс многомерного массива также не требуется, однако все последующие индексы необходимы. Это относится и к прототипу функции:

```
void printarray(int [][2]);
```

Компилятор использует эти индексы для определения местонахождения в памяти элементов многомерного массива. Все элементы хранятся в памяти последовательно, независимо от количества индексов. Для двумерных массивов за первой строкой в памяти следует вторая строка и т.д. Таким образом, указание значений индексов при объявлении параметров функции дает возможность компилятору сообщить ей о способе отыскания элементов в массиве. Ниже приведен соответствующий пример.

Пример передачи многомерных массивов в функцию

```
#include <stdio.h>
void PrintArray(int [][2]);
main()
{
  system("clear");
  int A[3][2] = {{1},{3,4},{}};
  PrintArray(A);
  return 0;
}

void PrintArray(int Ar[][2])
{
  int i,j;
  for (i = 0; i<3; i++)
    for (j = 0; j<2; j++)
      printf("Ar[%d][%d] =%d\n", i,j,Ar[i][j]);
}
```

Задача 6.14

Двумерный массив 10X10 заполнить его случайными числами от 0 до 9, вывести на экран, транспонировать его и еще раз вывести на экран.
Решить аналогичную задачу для прямоугольного массива размерностью 10X5.

Задача 6.15

Используя двумерный массив, реализовать функцию вычисления определителя матрицы размерностью 2 X 2, а также размерностью 3 X 3.

Задача 6.16

Используя результаты задачи 6.15, реализовать алгоритм решения системы двух и трех линейных уравнений методом Крамера.
В качестве контрольного примера использовать следующие системы:

$$\begin{cases} 3x + 2y = 7 \\ 4x - 5y = 40 \end{cases}$$

(ответ: $x = 5; y = -4$)

$$\begin{cases} 2x - y + z = 2 \\ 3x + 2y + 2z = -2 \\ x - 2y + z = 1 \end{cases}$$

(ответ: $x = 2; y = -1; z = -3$)

Задача 6.17

Найти минимальный элемент каждой строки и максимальный элемент каждого столбца заданной матрицы.

Задача 6.18

Реализовать алгоритм умножения матрицы $N \cdot M$ на матрицу $M \cdot N$

Задача 6.19

Реализовать алгоритм нахождения обратной матрицы для заданной трехмерной матрицы.

Пример: Найти обратную матрицу A^{-1}, если

$$A=\begin{pmatrix}3 & 5 & -2\\ 1 & -3 & 2\\ 6 & 7 & -3\end{pmatrix}.$$

Решение:

$\det A = 10$

$A_{11}=-5 \quad A_{12}=15 \quad A_{13}=25$

$A_{21}=1 \quad A_{22}=3 \quad A_{23}=9$

$A_{31}=4 \quad A_{32}=-8 \quad A_{33}=-14$

$$A^{-1}=\frac{1}{10}\begin{pmatrix}-5 & 1 & 4\\ 15 & 3 & -8\\ 25 & 9 & -14\end{pmatrix}.$$

Проверка:

$$A^{-1}A=E=\frac{1}{10}\begin{pmatrix}-5 & 1 & 4\\ 15 & 3 & -8\\ 25 & 9 & -14\end{pmatrix}\begin{pmatrix}3 & 5 & -2\\ 1 & -3 & 2\\ 6 & 7 & -3\end{pmatrix}=\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}\blacktriangle$$