Программирование на языке Си

Лекция 6.
Массивы

МАССИВЫ

Для представления набора однотипных данных в языке Си используются массивы. Наряду с циклами массивы являются фундаментальным инструментом программирования, который позволяет избегать дублирования кода и разрабатывать компактные, понятные и красивые программы. Для наиболее полного использования возможностей массивов они должны применяться в комбинации с циклами, поэтому для успешного освоения данной темы необходимо уверенно владеть работой с циклическими конструкциями.

Чтобы продемонстрировать проблемы, которые могут возникнуть, если не использовать массивы, рассмотрим задачу вычисления среднего возраста десяти человек. Вначале объявим 10 переменных, каждая из которых хранит возраст одного человека, затем вычислим их среднее и выведем результат на экран.

ЗАЧЕМ НУЖНЫ МАССИВЫ

```
//Объявление и инициализации десяти переменных
int age_0 = 30;    int age_1 = 21;
int age_2 = 22;    int age_3 = 19;
int age_4 = 45;    int age_5 = 18;
int age_6 = 15;    int age_7 = 61;
int age_8 = 43;    int age_9 = 39;

//Вычисление и вывод среднего
int sum = age_0 + age_1 + age_2 +
          age_3 + age_4 + age_5 +
          age_6 + age_7 + age_8 +
          age_9;
float avg = (float) sum/10;
printf("Средний возраст равен %g лет\n", avg);
```

ОБЪЯВЛЕНИЕ МАССИВА

Массив – это набор элементов, которые имеют одинаковый тип и хранятся в памяти строго последовательно. Обращение к элементу массива осуществляется по имени массива и номеру элемента в этом массиве. Номер элемента в массиве называется *индексом*. В языке Си индекс первого элемента равен ***0***, а индекс последнего – ***N-1***, где ***N*** – это размер массива. При объявлении массива указывается тип его элементов, имя и размер:

```
<Тип> <Имя>[<Размер>];
```

В качестве *типа элементов массива* можно указывать любой существующий в языке Си тип. *Имя массива* следует выбирать, чтобы оно отражало суть содержимого массива. Например, абстрактный числовой массив можно назвать `numbers`, массив возрастов – `ages`, а массив весов – `weights`. *Размер массива* должен быть константой, а не переменной, поэтому существуют три способа объявления массива:

ЗАДАНИЕ РАЗМЕРА МАССИВА

```
//Способ 1.
int numbers[10];

//Способ 2.
const int n = 10;
int numbers[n];

//Способ 3.
#define SIZE 10
int numbers[SIZE];
```

ЗАДАНИЕ РАЗМЕРА МАССИВА

Поскольку размер массива не может быть переменной, то следующая запись недопустима и вызовет ошибку компиляции:

```
int size;
printf("Введите размер массива:\n");
scanf("%d", &size);

//Ошибка
int numbers[size];
```

Размер массива указывать необязательно, если при его объявлении выполняется инициализация:

```
int ages[] = {23, 45, 56, 78, 54};
```

ЭЛЕМЕНТЫ МАССИВА

Каждый из элементов массива представляет собой переменную, другими словами, именованную ячейку памяти. Отличие элемента массива от обычной переменной заключается лишь в том, что обращение к обычной переменной происходит по ее имени, а к элементу массива обращаются по имени массива и индексу элемента в этом массива. Таким образом, элементу массива можно присвоить некоторое значение так же, как и переменной:

```
numbers[0] = 99;
```

Кроме того, можно ввести значение элемента массива с клавиатуры при помощи функций ввода

```
scanf("%d", &numbers[1]);
symbols[7] = getchar();
```

или вывести на экран:

```
printf("%d", numbers[3]);
putchar(symbols[5]);
```

ЕСЛИ РАЗМЕР ЗАРАНЕЕ НЕ ИЗВЕСТЕН

Довольно часто размер массива, который планируется использовать в программе, неизвестен. Например, требуется ввести размер массива с клавиатуры, заполнить его случайными числами и вывести на экран. Очевидное решение

```
printf("Введите размер массива:\n");
scanf("%d", &size);
int numbers[size];
```

как уже было замечено, оказывается неправильным, поскольку **передавать в качестве размера массива переменную нельзя**. Решить проблему можно следующим образом: размер массива указать с запасом так, чтобы в него точно поместились все элементы, а фактический размер массива ввести с клавиатуры:

РАЗМЕР ЗАРАНЕЕ НЕ ИЗВЕСТЕН

```
//Массив может хранить не более 100 элементов
int numbers[100];

//Вводим размер массива с клавиатуры
int size;
printf("Введите размер массива < 100:\n");
scanf("%d", &size);

//Заполняем массив случайными числами
for (int i=0; i<size; i++) {
  numbers[i] = rand()%100;
}

//Выводим массив на экран
for (int i=0; i<size; i++) {
  printf("%d ",numbers[i]);
}
```

РЕЗУЛЬТАТ ПРОГРАММЫ

Обратите внимание на то, что память выделена под 100 элементов, но фактически работа производится элементами, количество которых определено переменной `size`.

```
Введите размер массива < 100:
12
41 67 34  0 69 24 78 58 62 64  5 45
```

ПРИСВАИВАНИЕ МАССИВОВ

Массивы, в отличие от переменных, нельзя напрямую присвоить или сравнить. Это следует делать поэлементно, и для решения этой задачи лучше всего подходят циклы. Итак, для присвоения массива ages1 массиву ages2 следует использовать такую конструкцию:

```
for (int i=0; i<size; i++) {
  ages2[i] = ages1[i];
}
```

СРАВНЕНИЕ МАССИВОВ

Сравнение массивов нужно выполнять по аналогичной схеме:

```
bool equal = true;
for (int i=0; i<size; i++) {
  if (ages1[i] != ages2[i]) {
    equal = false;
    break;
  }
}

if (equal) {
  printf("Массивы равны\n");
} else {
  printf("Массивы не равны\n");
}
```

ВЫЧИСЛЕНИЕ СУММЫ ЭЛЕМЕНТОВ МАССИВА

Сам по себе массив вряд ли представляет практический интерес. Настоящая польза заключается в обработке их элементов определенным образом. Один из вариантов обработки – это вычисление суммы элементов массива. Для этого используется рая переменная инициализируется нулем, а затем в цикле к ней прибавляются элементы массива один за другим. На каждой итерации прибавляется очередной элемент, поэтому после завершения цикла данная переменная будет хранить сумму всех элементов. Следующий фрагмент кода реализует этот алгоритм:

```
int sum = 0;
for (int i=0; i<size; i++) {
  sum += numbers[i];
}
printf("Сумма равна %d\n", sum);
```

ЭЛЕМЕНТЫ С НУЖНЫМ СВОЙСТВОМ

Другой важной задачей, связанной с массивами, является подсчет элементов с нужным свойством. Например, сколько сотрудников имеют зарплату более 100000 рублей, или, скажем, сколько отличников учится в университете. Все эти задачи сводятся к подсчету элементов массива, имеющих определенное свойство.

Начнем с задачи подсчета всех элементов массива, т.е. с ответа на вопрос, сколько всего элементов в массиве. Вопрос, конечно, искусственный, поскольку это число нам известно, но представим, что это не так. В таком случае, подсчитать это число можно так: пробежаться по всему массиву и прибавлять к некоторой переменной единицу, проходя очередной элемент. Алгоритм аналогичен алгоритму вычисления суммы элементов:

ПОДСЧЕТ КОЛИЧЕСТВА ЭЛЕМЕНТОВ С НУЖНЫМ СВОЙСТВОМ

```
int n_elem = 0;
int sum = 0;
for (int i=0; i<size; i++) {
  n_elem++;
  sum += numbers[i];
}
printf("В массиве %d элементов\n", n_elem);
printf("Сумма равна %d\n", sum);
```

Конечно, значение переменной `n_elem` должно совпадать со значением `size`, поскольку данная программа складывает `size` единиц. Усложним задачу: подсчитаем количество элементов, которые больше 100. Для этого нужно в нашей сумме заменить единицы, соответствующие элементам меньшим 100, на нули:

ПОДСЧЕТ КОЛИЧЕСТВА ЭЛЕМЕНТОВ С НУЖНЫМ СВОЙСТВОМ

```
int n_elem = 0;
for (int i=0; i<size; i++) {
  if (numbers[i]>=100) {
    n_elem+=1;
  } else {
    n_elem+=0;
  }
}
printf("В массиве %d элементов\n", n_elem);
```

Очевидно, что операция ничего не дает, поэтому окончательный вариант программы выглядит так:

```
int n_elem = 0;
for (int i=0; i<size; i++) {
  if (numbers[i]>=100) {
    n_elem++;
  }
}
printf("В массиве %d элементов\n", n_elem);
```

ПОИСК ЭЛЕМЕНТА В МАССИВЕ

Пожалуй, наиболее важной задачей, связанной с массивами, является поиск элемента в массиве. Есть ли интересующий нас товар в магазине? Включены ли вы в число студентов, получивших президентскую стипендию? Как переводится какое-то слово на русский язык? Все эти вопросы сводятся к задаче поиска элемента в массиве, который может быть перечнем товаров, списком студентов или слов в словаре.

Существует масса модификаций данной задачи, поэтому мы рассмотрим базовый алгоритм, который используется практически во всех модификациях. Суть задачи заключается в том, что по заданному элементу требуется проверить, есть ли он в массиве или нет. Существует очень распространенное ошибочное решение этой задачи, поэтому начнем с его рассмотрения.

ОШИБОЧНАЯ РЕАЛИЗАЦИЯ

```
int find = 50;
for (int i=0; i<size; i++) {
  if (find == numbers[i]) {
    printf("Такой элемент есть в массиве\n");
  } else {
    printf("Такого элемента нет в массиве\n");
  }
}
```

В данной реализации несколько ошибок. Подумайте, какие?

ПРАВИЛЬНЫЙ АЛГОРИТМ

Рассмотрим одну из них. Предположим, что элемента в массиве нет, тогда на каждой итерации цикла будет выполняться ветка `else`, и на экране появятся `size` сообщений о том, что элемента нет. Если же искомый элемент находится где-то в середине, что программа вначале выведет несколько сообщений об отсутствии, затем – одно сообщение о том, что элемент найден, а затем опять начнет сообщать, что элемента нет.

Что же делать? Нужно заметить два аспекта:

- если элемент найден, то поиск можно заканчивать;
- убедиться в отсутствии элемента можно лишь, пройдя весь массив.

Следующая программа учитывает эти аспекты:

ПРАВИЛЬНЫЙ АЛГОРИТМ

```
//Искомый элемент
int isFound = 50;
//Нашли или нет
bool found = false;
for (int i=0; i<size; i++) {
  if (find == numbers[i]) {
    //Отмечаем, что элемент найден
    isFound = true;
    //Заканчиваем чикл
    break;

//isFound эквивалентно isFound==true
if (isFound) {
  printf("Такой элемент есть в массиве\n");
} else {
  printf("Такого элемента нет в массиве\n");
}
```

ПОИСК МАКСИМУМА

Задача поиска «самого-самого» всегда актуальна. Кто из футболистов забил больше всего голов? У кого самая большая зарплата? Кто имеет максимальный рейтинг? Подобные вопросы возникают повсюду, но далеко не все представляют себе алгоритм нахождения этих «звезд». Мы изучим этот алгоритм на примере поиска максимального элемента в целочисленном массиве, а распространить его на перечисленные выше задачи не составит труда.

Для поиска максимального элемента в массиве существует стандартный алгоритм, который заключается в том, что заводится переменная (назовем ее `max`), которая инициализируется первым элементом массива, а затем в цикле переприсваивается очередным элементом массива, если он больше ее.

ПОИСК МАКСИМУМА

```
int max = numbers[0];
for (int i=1; i<size; i++) {
  if (max < numbers[i]) {
    max = numbers[i];
  }
}
```

Заметим, что часто переменную max инициализируют не первым элементом, а некоторым «очень маленьким числом», а цикл начинают не с 1, а с 0. Подобный алгоритм выглядит так:

НЕВЕРНАЯ РЕАЛИЗАЦИЯ

```
int max = -1000000;
for (int i=0; i<size; i++) {
  if (max < numbers[i]) {
    max = numbers[i];
  }
}
```

Однако данный подход нежелателен, поскольку нет гарантии, что число меньше «очень маленького числа» не встретится, и, кроме того, выполняется одно лишнее сравнение. Полный пример поиска максимума выглядит так:

ПОИСК МАКСИМАЛЬНОГО ЭЛЕМЕНТА

```
#define MAX_SIZE 100
int main() {
  int numbers[MAX_SIZE];
  printf("Введите размер массива: ");
  int size;
  scanf("%d", &size);
  for (int i=0; i<size; i++) {
    numbers[i] = rand()%100;
  }
  printf("Задан следующий массив:\n");
  for (int i=0; i<size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  int max = numbers[0];
  for (int i=1; i<size; i++) {
    if (max < numbers[i]) {
      max=numbers[i];
    }
  }
  printf("Максимальный элемент = %d\n", max);
  return 0;
}
```

```
Введите размер массива:
12
Задан следующий массив:
41 67 34 0 69 24 78 58 62 64 5 45
Максимальный элемент = 78
```

ИНДЕКС МАКСИМАЛЬНОГО ЭЛЕМЕНТА

Если требуется найти не максимальный элемент, а его индекс, то программа несколько изменится:

```
int ind_max = 0;
for (int i=1; i<size; i++) {
  if (numbers[ind_max] < numbers[i]) {
    ind_max = i;
  }
}
```

ЗАДАЧА СОРТИРОВКИ

Многие знакомы с задачей сортировки, и, вроде бы, всем понятно, что сортировать массивы нужно, однако на вопрос «Зачем это делать?» даст ответ не каждый. Подумайте, зачем сортировать массивы или списки. Представьте себе, если бы слова в словаре располагались бы не в алфавитном порядке, а в разброс; или товары в магазине не были бы разложены по типам, и разные марки колбасы находились бы по всему магазину.

Отсутствие порядка существенно затрудняет поиск. А как можно достичь порядка? Сортировкой. Именно **облегчение последующего поиска является главной причиной, по которой производится сортировка**. Существует масса методов, сортирующих массивы. Мы рассмотрим три наиболее простых для понимания и реализации метода: метод прямого выбора, метод пузырьковой сортировки и сортировку вставками. Метод прямого выбора – это наиболее очевидный для понимания способ сортировки массива, а пузырьковая сортировка – наиболее проста в реализации. Сортировка вставками довольно часто используется нами в жизни.

МЕТОД ПРЯМОГО ВЫБОРА

Сортировка *методом прямого выбора* – это наиболее очевидный способ сортировки массива. Для определенности положим, что сортировка производится по возрастанию. При сортировке методом прямого выбора вначале ищется минимальный элемент массива и меняется местами с тем, который стоит на первом месте. Затем ищется минимальный элемент среди остальных (все, кроме первого) и меняется местами с элементом, стоящем на втором месте. После этого ищется минимальный элемент среди тех, которые находятся на позициях с третьей по последнюю, и меняется местами с третьим. Данная процедура продолжается до предпоследнего элемента. Программный код сортировки методом прямого выбора выглядит так:

МЕТОД ПРЯМОГО ВЫБОРА

```
for (int i=0; i<size; i++) {

  //Поиск минимального элемента
  //среди элементов с индексами
  //i, i+1, i+2, ... , size
  int j_min = i;
  for (int j=i+1; j<size; j++) {
    if (numbers[j_min] > numbers[j]) {
      j_min = j;
    }
  }

  //Перестановка элементов
  //с индексами i и j_max
  int temp = numbers[i];
  numbers[i] = numbers[j_min];
  numbers[j_min] = temp;
}
```

ПУЗЫРЬКОВАЯ СОРТИРОВКА

Пузырьковый метод сортировки имеет самый короткий программный код из всех методов сортировки, хотя алгоритм ее работы не так очевиден, как алгоритм сортировки методом прямого выбора. *Пузырьковая сортировка* подразумевает просмотр массива с последнего элемента до первого, затем – с последнего до второго, с последнего до третьего и т.д. При этом сравниваются соседние элементы и меняются местами, если они расположены в неправильном порядке:

ПУЗЫРЬКОВАЯ СОРТИРОВКА

```
  for (int i=0; i<size-1; i++) {
    for (int j=size-1; j>0; j--) {

    //Сравниваем два соседних элемента
    if (numbers[j-1] > numbers[j]) {

      //Меняем их местами, если
      //они стоят не в том порядке
      int temp = numbers[j];
      numbers[j] = numbers[j-1];
      numbers[j-1] = temp;
    }
  }
}
```

МЕТОД ВСТАВОК

Сортировка методом вставок должна быть хорошо знакома людям, играющим в карты. Представьте себе, что вам раздали карты для игры. Карты следует расставить по старшинству: слева – старшие карты, а справа – младшие. Другими словами, задача заключается в том, чтобы отсортировать полученные карты в порядке убывания. Наши действия таковы:

- первую карту отодвигаем влево.
- берем вторую карту и ставим ее либо перед первой, либо после нее в зависимости от старшинства.
- берем третью карту и также в зависимости от старшинства ставим ее либо перед первыми двумя, либо между ними, либо после них.
- действия продолжаем, пока не расставим все карты.

Отсюда и название метода – «Сортировка вставками». После очередной вставки отсортированная часть массива увеличивается на 1. Таким образом, программа имеет вид:

МЕТОД ВСТАВОК

```
//Вставляем элементы, начиная со второго
for (int i=1; i<size; i++) {
  <Вставить элемент с номером i
  относительно элементов 0,…,i-1>
}
```

МЕТОД ВСТАВОК

Для того, чтобы вставить элемент нужно отодвинуть последующие один за другим и на освободившуюся позицию поставить вставляемый:

```
//Вставляем элементы, начиная со второго
for (int i=1; i<size; i++) {

  //Индекс вставляемого элемента
  int j=i;

  //Ищем позицию, куда вставить элемент
  while (numbers[j]<numbers[j-1] && j!=0) {

    //Меняем их местами, если
    //они стоят не в том порядке
    int temp = numbers[j];
    numbers[j] = numbers[j-1];
    numbers[j-1] = temp;
    j--;
  }
}
```

ДВОИЧНЫЙ ПОИСК

Рассмотрим теперь, как реализовать быстрый поиск, используя упорядоченность массива; для определенности будем считать, что массив отсортирован по возрастанию. Главное свойство упорядоченного массива, позволяющее осуществить эффективный поиск, заключается в том, что, если искомый элемент меньше некоторого элемента массива, то можно отбросить расположенную справа от этого элемента часть массива и продолжать поиск только по левой части. Отсюда получается следующий алгоритм. Выбрать элемент, расположенный в середине массива, и сравнить искомый элемент с ним. Если элемент окажется меньше, то выбрать середину левой части и осуществлять деление до тех пор, пока не будет равенства или массив будет состоять из одного элемента. Если он не равен, то такого элемента нет в массиве.

МНОГОМЕРНЫЕ МАССИВЫ

```
int array_1d[100];
float array_2d[100][200];
char array_3d[100][200][50];
double array_4d[100][200][50][10];
unsigned short array_5d[100][200][50][10][30];
```

Обращение к элементам происходит так:

```
printf("%d", array_1d[i]);
printf("%f", array_2d[i][j]);
printf("%c", array_3d[i][j][k]);
printf("%e", array_4d[i][j][k][l]);
printf("%d", array_5d[i][j][k][l][m]);
```

МАТРИЦЫ – ДВУМЕРНЫЕ МАССИВЫ

Среди многомерных массивов наиболее часто используемым является двумерный массив, называемый по-другому *матрицей*. Матрицы широко используются в линейной алгебре, теории графов, вычислительных методах, математической статистике, физике, механике, а также во многих других фундаментальных и прикладных науках. Рассмотрим несколько задач, связанных с матрицами. Итак, первая задача – это объявление такого массива и заполнение его некоторыми значениями. В реальных задачах значения берутся реальные, которые, например, вводят операторы. У нас на это может не быть времени, поэтому самый быстрый и простой способ заполнения двумерного (да, и не только двумерного) массива – это заполнение случайными числами.

```
//Объявление массива
int matrix[100][100]
```

ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

Задача 1. Написать программу, которая принимает с клавиатуры число ***N***, задает массив из ***N*** случайных целых чисел и выводит его на экран. Затем необходимо вывести элементы этого массива, которые не принадлежат отрезку ***[5,15]***.

```
int main() {
  int N;
  printf("Введите число N:\n");
  scanf("%d", &N);

  int numbers[100];
  for (int i=0; i<N; i++) {
    numbers[i] = rand()%50;
  }
  for (int i=0; i<N; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\nЭлементы из отрезка [5,15]:\n");
  for (int i=0; i<N; i++) {
    if (5<=numbers[i] && numbers[i]<=15) {
      printf("%d ", numbers[i]);
    }
  }
  printf("\n");
  return 0;
}
```

ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

Задача 2. Написать программу, которая принимает с клавиатуры число ***N***, задает массив из ***N*** случайных целых чисел и выводит его. Затем необходимо подсчитать количество трехзначных элементов этого массива.

```
#include <stdio.h>
#include <stdlib.h>

int main() {

  int N;
```

ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

```
  printf("Введите число N:\n");
  scanf("%d", &N);

  int numbers[100];
  for (int i=0; i<N; i++) {
    numbers[i] = rand()%2000;
  }
  for (int i=0; i<N; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");
  int digits3 = 0;
  for (int i=0; i<N; i++) {
    if (100<=numbers[i] && numbers[i]<=999) {
      digits3++;
    }
  }
  printf("Число трехзначных чисел = %d\n",
      digits3);
  return 0;
}
```

ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

Задача 4. Написать программу, которая принимает с клавиатуры число ***N***, задает массив из ***N*** случайных целых чисел и выводит его. Затем необходимо отсортировать его методом прямого выбора по убыванию.

```
#include <stdio.h>
#include <stdlib.h>

int main() {

  int N;
  printf("Введите число N:\n");
  scanf("%d", &N);

  int numbers[100];
  for (int i=0; i<N; i++) {
```

ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

```
    numbers[i] = rand()%100;
  }
  printf("\nИсходный массив:\n");
  for (int i=0; i<N; i++) {
    printf("%d ", numbers[i]);
  }

  for (int i=0; i<N-1; i++) {
    int max_index = i;
    for (int j=i+1; j<N; j++) {
      if (numbers[max_index]<numbers[j]) {
        max_index = j;
      }
    }
    int temp = numbers[i];
    numbers[i] = numbers[max_index];
    numbers[max_index] = temp;
  }

  printf("\nОтсортированный массив:\n");
  for (int i=0; i<N; i++) {
    printf("%d ", numbers[i]);
  }
  return 0;
}
```

ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

Задача 4. *След матрицы* – это сумма ее диагональных элементов. Написать программу, которая вычисляет след матрицы.

```
#include <stdio.h>
#include <stdlib.h>

int main() {

      int N;
      printf("Введите число N:\n");
      scanf("%d", &N);
```

ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

```
    int numbers[100][100];
    for (int i=0; i<N; i++) {
        for (int j=0; j<N; j++) {
            numbers[i][j] = rand()%10;
        }
    }
    printf("\nМатрица:\n");

    for (int i=0; i<N; i++) {
        for (int j=0; j<N; j++) {
            printf("%d ", numbers[i][j]);
        }
        printf("\n");
    }

    int trace = 0;
    for (int i=0; i<N; i++) {
        trace += numbers[i][i];
    }

    printf("След матрицы равен %d\n", trace);

    return 0;
}
```