Лабораторная работа №8

Одномерные массивы

Цель работы: Изучение теоретических сведений и получение практических навыков по работе с одномерными массивами.

Теоретические сведения

Массив – это структура данных, которая позволяет хранить последовательный набор однотипных элементов фиксированного размера. Массивы используются для хранения наборов данных, однако представление массива в качестве набора однотипных переменных является оправданным.

Вместо того, чтобы создавать 100 переменных типа `int`, перечисляя их при объявлении `number0, number1, …, number99`, достаточно просто объявить один массив элементов

```
int number[100];
```

Для обращения к какому-либо элементу массива `number` в программе достаточно будет указать индекс элемента, например, `number[0]`. Таким образом, доступ к элементу массива осуществляется через его индекс. **Обратите особое внимание**, что при индексации массива `number` первый элемент массива всегда будет иметь индекс `0`, а последний `99`.

В памяти компьютера элементы массива хранятся в смежных ячейках памяти (см. рис. 1). Наименьший адрес ячейки памяти будет соответствовать первому по порядку элементу массива, а наибольший – последнему элементу массива.

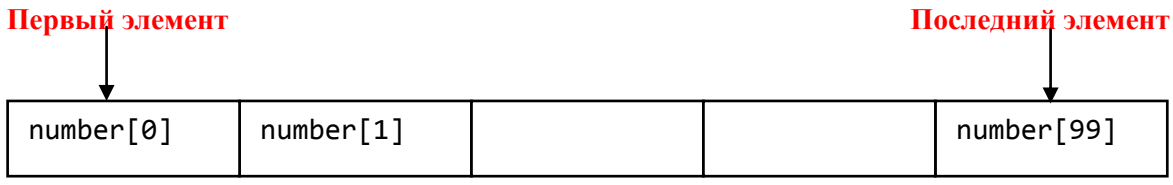

Рисунок 1 – Расположение элементов массива в памяти компьютера.

Объявление и инициализация массива

Как уже было показано в предыдущем пункте, для объявления массива в языке Си программист указывает тип элементов массива (`type`), имя массива (`ArrayName`) и количество элементов (`ArraySize`):

```
type ArrayName [ArraySize];
```

Массивы, заданные подобным образом, называются одномерными массивами. Здесь `ArraySize` представляет собой целочисленную константу, значение которой больше нуля. В качестве типа данных `type` может быть любой тип данных, зарезервированный в языке Си: `int`, `float`, `double`, `long` и т.д.

Например, создадим массив из `5` элементов типа `double`, имя которого будет `elements`:

```
double elements[5];
```

Элементами данного массива будут `elements[0]`, `elements[1]`, `elements[2]`, `elements[3]`, `elements[4]`. Обращение в программе к элементу с именем `elements[5]` будет приводить несанкционированному доступу к ячейке памяти, следующей за последним элементом массива[1].

Для инициализации[2] элементов массива при объявлении массива будет выглядеть следующим образом:

```
double elements[5]={1000, 2, 8, 68, 403};
```

При такой инициализации массива число элементов, указанных в фигурных скобках, и размер массива (указан в квадратных скобках [ ]) должны совпадать! Отметим, что размер массива при инициализации можно опускать:

```
double elements[ ]={1000, 2, 8, 68, 403};
```

На рисунке 2 представлено распределение элементов массива `elements` в памяти компьютера.

индекс	0	1	2	3	4
`elements`	1000	2	8	68	403

Рисунок 2 – Расположение элементов массива `elements` в памяти компьютера.

Еще одной формой инициализации элементов массива является задание значений для каждого элемента массива:

```
elements[0] = 1000;
```

В данном случае `0`-му элементу массива `elements` будет присвоено значение `1000`.

При работе с массивами, в том числе при инициализации, наиболее удобным инструментом является цикл, позволяющий перебрать по очереди все элементы массива.

---

[1] Как правило, компилятор в этом случае выдает сообщение: Segmentation fault. Однако некоторые компиляторы могут игнорировать подобные ошибки, что неизбежно приводит к серьезным проблемам.

[2] Инициализация обозначает задание значений переменной.

Следующий пример иллюстрирует инициализацию массива набором случайных числе (random numbers), генерируемых генератором случайных чисел.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int i, Arr[20];
        printf("\n Array containing 20 Random Numbers\n");
        for(i=0;i<20;i++)
        {
                Arr[i]=rand();
                printf("%d\n",Arr[i]);
        }
        return 0;
}
```

В данном примере функция `rand()` возвращает значение случайного числа. Для корректной работы программы требуется подключение файла библиотеки `stdlib.h`.

Тестовые примеры

Рассмотрим пример выполнения следующего задания: *Написать программу, определяющую количество положительных элементов, которые располагаются между максимальным и минимальным элементами целочисленного массива.*

Представим алгоритм выполнения данного задания в виде словесного описания.

1. Необходимо определить максимальный и минимальный элементы массива и запомнить их индексы. Поскольку расположение элементов заранее неизвестно, то после определения индексов необходимо определить который из них больше (меньше). Также необходимо учесть ситуацию, если индексы совпали.
2. Проверить являются ли положительными элементы, располагающиеся между максимальным и минимальным элементами, и посчитать их количество.

```
#include<stdio.h>
#include<math.h>
int main()
{
      const int n = 10;
      int a[n] = {1, 3, -5, 1, -2, 1, -1, 3, 8, 4};
      int i, imax, imin, ibeg, iend, counter;
      for(i = imax = imin = 0; i < n; i++)
```

```
    {
        if(a[i] > a[imax]) imax = i;
        if(a[i] < a[imin]) imin = i;
    }
    printf("\n max = %d\t min = %d\t \n", a[imax],a[imin]);
    ibeg = imax < imin ? imax : imin;
    iend = imax < imin ? imin : imax;
    printf("\n ibeg=%d\t iend=%d\n",ibeg,iend);
    for(counter = 0, i = ibeg + 1; i < iend; i++)
    if(a[i] > 0) counter++;
    printf("\n The number of positive elements = %d\n",counter);
    return 0;
}
```

Массив просматривается, начиная с элемента, следующего за максимальным (минимальным), до элемента, предшествующего минимальному (максимальному). Индексы границ просмотра хранятся в переменных `ibeg` и `iend`.

Тестовых примеров для данного задания должно быть, по крайней мере, три для следующих случав:

1. `a[imin]` расположен левее `a[imax]`;
2. `a[imin]` расположен правее `a[imax]`;
3. `a[imin]` и `a[imax]` совпадают.

В приведенной выше программе для определения их значений используется тернарная условная операция:

```
ibeg = imax < imin ? imax : imin;
```

Данная операция принимает ровно три аргумента: условие `imax < imin`, значение выражения, если условие истинно, `imax`, и значение выражения, если условие ложно, `imin`. Данная операция может быть переписана с помощью условного оператора следующим образом:

```
if(imax < imin)
    ibeg = imax;
else
    ibeg = imin;
```

Динамическое выделение памяти

Если размер массива заранее неизвестен[3], в этом случае необходимо использовать динамическое выделение памяти для получения необходимой памяти под хранение массива, а также использовать специальные функции (операторы) освобождающие память перед завершением программы. В библиотеке `stdlib.h` существуют 4 функции, позволяющие динамически работать с памятью (см. таблицу 1).

Таблица 1 – Функции для работы с динамическим выделением памяти.

Функция	Описание
`malloc()`	Выделяет запрашиваемый в байтах размер памяти и возвращает указатель[4] на первый байт выделенной памяти.
`calloc()`	Выделяет память для элементов массива, инициализирует значения элементов нулями и возвращает указатель на выделенный участок памяти.
`free()`	Освобождает память, выделенную динамически.
`realloc()`	Изменяет размер памяти выделенный динамически прежде.

Приведем примеры работы приведенных выше функций. Начнем с функции malloc().

```
int n = 10;
double *ptr = (double* ) malloc (n * sizeof (double));
```

Поскольку функция `malloc()` возвращает указатель, то для хранения результата мы используем указатель `*ptr`. Функция `malloc()` выделяет `n` ячеек памяти размером определяемым типом `double`.

Синтаксис функции `calloc()` будет выглядеть следующим образом:

```
double *ptr = (double*) calloc (n, sizeof(double));
```

Следует также отметить, что перед завершением работы программы обязательным является освобождение выделенного динамически участка памяти:

```
free(ptr);
```

Кроме того, для того чтобы динамически выделять и освобождать память, можно использовать операторы new и delete.

```
int n = 10;
int *ptr = new int[n]; // Динамическое выделение памяти
delete [] ptr;         // Освобождение памяти
```

[3] На момент компиляции программы.
[4] Указатели подробнее будут рассмотрены в следующей лабораторной работе.

Контрольные вопросы

1. Что такое массив? Приведите примеры объявления массива.
2. Какими способами можно инициализировать массив? Приведите примеры.
3. Какие действия можно проводить над массивами?
4. Какого типа могут быть элементы массива?
5. Может ли массив быть элементом массива?
6. Как организовано хранение массива в памяти компьютера?
7. Как определить размер памяти, занимаемый массивом? Приведите примеры.
8. Как нумеруются элементы массива в языке Си?
9. Динамическое выделение памяти в языке Си? Примеры.
10. Какие индексы имеют первый и последний элементы массива.

Задание на лабораторную работу

1. Модифицируйте тестовый пример для ввода значений элементов массива с клавиатуры.
2. Получите индивидуальный вариант задания у преподавателя.
3. Разработайте алгоритм, нарисуйте блок-схему и реализуйте код программы.
4. Протестируйте код программы и ответьте на контрольные вопросы.
5. Оформите отчет и отчитайте работу преподавателю.