В.Л. Тарасов

Лекции по программированию на C++

Лекция 4

Массивы и вектора

Массив представляет собой набор последовательно пронумерованных элементов одного типа. Все элементы массива имеют одно имя, но каждый элемент имеет свой номер, с помощью которого отличается от других элементов.

Нумерация элементов массива начинается с *нуля*.

В памяти элементы массива располагаются подряд, один за другим.

4.1. Одномерные массивы

При объявлении массива после его имени в квадратных скобках указывается *количество элементов*, например,

```
double x[100];
```

Здесь x – это имя массива из 100 элементов типа double.

Массивы могут состоять из элементов любых типов.

При обращение к элементу массива его номер указывается в квадратных скобках.

Так как нумерация элементов массива начинается с нуля, то x[0] есть первый элемент массива, x[99] – последний. Не существует элемента x[100], в том смысле, что под него не выделена память, хотя после 99-го элемента память не заканчивается и к ней можно обратиться посредством выражения x[100]. Поскольку этот участок памяти не числится за массивом x, он может использоваться для других целей или программой, или операционной системой, поэтому обращение к нему, особенно запись в него какого-то значения, может привести к ошибочному, непредсказуемому поведению программы. В других языках, например, в Паскале, Java контролируется выход индекса за границы массива. В C++ для встроенных массивов, о которых сейчас идет речь, контроля невыхода за границы массива нет, сам программист должен следить за этим. Поскольку выход за границы массива ведет к непредсказуемому поведению программы, в C++ есть возможность создавать вместо массивов *вектора*, у которых контролируется выход индекса элемента за установленные границы.

Массивы в языке C++ всегда имеют *конкретный* размер, нельзя определять массивы переменной длины. Размер массива задается константой или константным выражением, но не переменной. Константные выражения вычисляются при компиляции. Например,

```
int N;                  // N - Переменная
N = 100;
char s[N];              // Ошибка, N – переменная
const int K = 200;      // K - константа
int d[K];               // Можно, K – константа
double h[2 * K + 50];   // Можно, 2 * K + 50 – константное выражение
```

Массивы можно *инициализировать* при определении, указывая в фигурных скобках список значений элементов. Например, массив с количеством дней в каждом месяце можно инициализировать так:

```
int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Если размер массива не указан, его длину вычисляет компилятор по списку инициализации. Для приведенного примера это 12.

Если количество инициализаторов меньше заданного размера массива, оставшиеся элементы будут нулевыми. Пусть, например, определен массив:

```
int y[6] = {1, 2, 3};
```

Элементы массива y будут иметь следующие значения: `y[0] = 1, y[1] = 2, y[2] = 3, y[3] = 0, y[4] = 0, y[5] = 0`.

Задание слишком большого числа инициализаторов является ошибкой, например,

```
int z[3] = {1, 2, 3, 4, 5};  // Ошибка, слишком много инициализаторов
```

Оператор `sizeof`, примененный к массиву, дает размер массива в байтах. Например, для массива `days` выражение `sizeof(days) / sizeof(int)` равно 12, так как массив состоит из 12 элементов типа `int`. Здесь `sizeof(days)` – размер памяти в байтах, выделенной под массив, `sizeof(int)` – размер одного элемента массива типа `int` в байтах.

Над массивами в целом нельзя выполнять какие-либо действия (присваивать, складывать и т.п.), можно работать только с отдельными элементами массива.

Программа 4.1. Проверка упорядоченности массива

В программе создается массив `x` целых чисел путем инициализации. Размер массива `n` устанавливается автоматически при компиляции с помощью выражения `sizeof(x) / sizeof(int)`.

Массив называется упорядоченным по возрастанию, если каждый следующий элемент больше предыдущего или равен ему. Программа проверяет эту упорядоченность. Сначала делается предположение, что упорядоченность имеет место. Это фиксируется установкой переменной логической переменной flag в true. Затем в цикле сравниваются соседние элементы. Если окажется, что какой-то элемент больше следующего, предположение об упорядоченности отвергается, что фиксируется установкой flag в false.

```
// Файл CheckOrderArray.cpp

// Проверка упорядоченности массива

#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;
int main()
{
   setlocale(LC_ALL, "Russian");
   int x[] = {77, 1, 3, 4, 7, 13, 17, 19, 23};    // Массив
   int n = sizeof(x) / sizeof(int);               // Размер массива
   cout << "Задан массив: \n";
   for(int i = 0; i < n; i++)                     // Вывод массива
      cout << x[i] << " ";
   bool flag = true;     // Считаем, что массив упорядочен по возрастанию
// Цикл сравнений соседних элементов
   for(int i = 0; i < n - 1; i++)             // Надо сделать n-1 сравнение
      if(x[i] > x[i + 1]){
         flag = false;                        // Массив не упорядочен
         break;                               // Выход из цикла
      }
   if(flag)                                   // Вывод результата
      cout << "\nМассив упорядочен \n";
   else
      cout << "\nМассив не упорядочен \n";
   system("pause");
   return 0;
}
```

Так как оператор if(выр) проверяет истинность выражения выр, можно писать if(flag) вместо if(flag == true).

Приведенная программа вывела следующее:

```
Задан массив:
77 1 3 4 7 13 17 19 23
Массив не упорядочен
```

Затем программа была изменена, из списка инициализации было удалено число 77. В этом случае программа выдала:

```
Задан массив:
```

```
1 3 4 7 13 17 19 23
Массив упорядочен
```

4.2. Двумерные массивы

Кроме одномерных массивов с одним индексом, можно создавать многомерные массивы с несколькими индексами. Многомерный массив рассматриваются как одномерный, элементами которого являются массивы с размерностью на единицу меньше.

Здесь рассмотрим двумерные массивы, которые встречаются чаще, чем массивы с большим числом измерений. Разберем работу с двумерными массивами на конкретном примере.

Пусть некий предприниматель имеет *три* магазина, по которым ведет ежедневный учет выручки от продажи *четырех* видов продуктов (например, чая, сахара, крупы, колбасы). Для учета продаж можно использовать следующий двумерный массив:

```
double r[3][4];
```

Здесь имеется две пары квадратных скобок. Первые скобки говорят, что `r` есть массив из 3 элементов. Вторые квадратные скобки говорят, что каждый из этих трех элементов есть массив из 4-х элементов типа `double`.

Память под многомерные массивы отводится *построчно*, то есть сначала размещаются первые 4 элемента, затем вторые 4 и т.д. Схема расположения элементов двумерного массива в памяти показана на рис. 4.1.

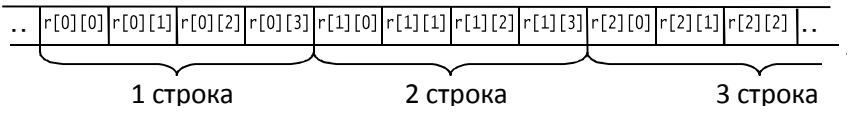

Рис. 4.1. Построчное расположение двумерного массива в памяти

Многомерные массивы можно инициализировать при их определении, записывая инициализаторы для всех массивов, начиная с первого. Для целей тестирования программы массив `r` можно было бы проинициализировать так:

```
double r[3][4] = {{500, 700, 450, 1000},
                  {600, 710, 480, 1100},
                  {800, 750, 550, 1200}
};
```

Программа 4.2. Подсчет выручки

Пусть предприниматель, о котором речь шла выше, каждый день подводит итоги: вычисляет выручку каждого магазина, выручку от продажи каждого товара и общую выручку за день. Эти расчеты можно выполнить с помощью следующей программы.

```
// Файл Receipts.cpp

// Анализ дневной выручки сети магазинов

#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;
int main()
{
   setlocale(LC_ALL, "Russian");
   const int NSHOP  = 3;                  // Число магазинов
   const int NGOODS = 4;                  // Число товаров
   double r[NSHOP][NGOODS];               // Двумерный массив для выручки
   double srshop, srg, sum;       // Выручка по магазинам, товарам, общая
   cout << "Введите выручку от чая, сахара, крупы, колбасы \n";
// Ввод данных
   for(int i = 0; i < NSHOP; i++){                    // Перебор магазинов
      cout << "Магазин " << i + 1 << ": ";
      for(int j = 0; j < NGOODS; j++)                 // Перебор товаров
         cin >> r[i][j];                  // Ввод выручки i-го магазина
   }                                      // от продажи j-го товара
   cout << "Выручка по магазинам: \n";
   for(int i = 0; i < NSHOP; i++){                    // Перебор магазинов
      srshop = 0;
      for(int j = 0; j < NGOODS; j++)                 // Перебор товаров
         srshop += r[i][j];
      cout << srshop << endl;
   }
   sum = 0;
   cout << "Выручка по товарам: \n";
   for(int j = 0; j < NGOODS; j++){                   // Перебор товаров
      srg = 0;
      for(int i = 0; i < NSHOP; i++)                  // Перебор магазинов
         srg += r[i][j];
      cout << srg << "  ";
      sum += srg;                                  // Подсчет общей выручки
   }
   cout << "\nВсего продано за день на: " << sum << endl;
   system("pause");
   return 0;
}
```

Пример работы программы:

```
Введите выручку от чая, сахара, крупы, колбасы
Магазин 1:  543.50   764.45   456.70   1003.50
```

```
Магазин 2:  604.17   710.33   550.20   1221.80
Магазин 3:  804.0     765       480.34   1100
Выручка по магазинам:
2768.15
3086.5
3149.34
Выручка по товарам:
1951.67  2239.78  1487.24  3325.3
Всего продано за день на: 9003.99
```

При работе с многомерными массивами используются вложенные циклы. Обработка двумерного массива ведется по строкам при вводе значений и подсчете выручки магазинов и по столбцам при нахождении выручки от продажи товаров.

4.3. Вектора

Проблемы встроенных массивов

Как уже говорилось, встроенные массивы потенциально опасны, так как не контролируется выход за границы массива. Это иллюстрирует следующая программа.

Программа 4.3. Выход за границы массива

В программе создается массив из трех элементов. Программа работает с памятью за пределами массива.

```
// Файл DangerousArray.cpp

// Выход за границы встроенного массива

#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");
   int x[] = {77, 1, 3};     // Массив
   int n = sizeof(x) / sizeof(int);           // Размер массива
   cout << "Задан массив: \n";
   for(int i = 0; i < n; i++)             // Вывод массива
      cout << "x[" << i << "] = " << x[i] << endl;
   cout << "За границами массива:\n";
   cout << "x[-1] = " << x[-1] << endl;       // -1 - ошибочный индекс
   cout << "x[3]  = " << x[3]  << endl;       //  3 - ошибочный индекс
   system("pause");
   return 0;
}
```

Программы работает, несмотря на то, что в выражениях `x[-1]` и `x[3]` индексы `-1` и `3` ошибочны, так ведут к обращению к памяти за пределами массива:

```
Задан массив:
x[0] = 77
x[1] = 1
x[2] = 3
За границами массива:
x[-1] = -858993460
x[3]  = -858993460
```

Содержимое памяти за пределами массива программа трактует как целые числа. Таким образом, при использовании массивов следует следить, чтобы индексы не выходили за границы массива в меньшую и большую стороны, иначе может возникнуть ошибка, которую трудно обнаружить.

Тип данных vector

Более безопасным типом данных, чем массивы являются *вектора*. Для того, чтобы использовать вектора надо включить заголовочный файл `vector`:

```
#include <vector>
```

Строка программы

```
vector<int> v(3);
```

создает вектор из 3-х элементов типа `int`.

Работа с векторами во многом похоже на работу с массивами, например, обращаться к элементам вектора можно так же как к элементам массива – с помощью индекса, заключенного в квадратные скобки.

Нумерация элементов вектора начинается с нуля, но при выходе индекса за границы возникает ошибка.

Функция `size()` возвращает текущий размер вектора, например, инструкция:

```
cout << v.size();
```

выведет 3.

Пример использования векторов приведен в программе 4.4.

Программа 4.4. Вектора

```
// Файл DemoVector.cpp
// Работа с векторами
```

```
#include <iostream>
#include <locale>
#include <cstdlib>
#include <vector>
using namespace std;

int main()
{
    setlocale(LC_ALL, "Russian");
    vector<int> v(3);                          // Вектор из 3-х элементов
    cout << "Размер вектора v = " << v.size() << endl;
    for(int i = 0; i < v.size(); i++)          // Заполнение вектора
        v[i] = 2 * i + 1;
    cout << "Вектор:\n";
    for(int i = 0; i < v.size(); i++)          // Вывод вектора
        cout << "v[" << i << "] = " << v[i] << endl;
    //cout << "v[-1] = " << v[-1] << endl;     // -1 - недопустимый индекс
    //cout << "v[3]  = " << v[3]  << endl;     //  3 - недопустимый индекс
    system("pause");
    return 0;
}
```

Программа выводит:

```
Размер вектора v = 3
Вектор:
v[0] = 1
v[1] = 3
v[2] = 5
```

Если убрать комментарий у строк:

```
cout << "v[-1] = " << v[-1] << endl;  // -1 - недопустимый индекс
cout << "v[3]  = " << v[3]  << endl;  //  3 - недопустимый индекс
```

то программа останавливается на первой из них, и выводится окно с сообщением `vector subscript out of range` (индекс вектора вне диапазона) (рис.4.2) .

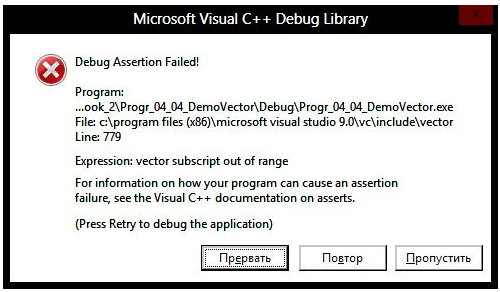

Рис. 4.2. Сообщение об ошибке при выходе индекса за границы

Таким образом, вектора позволяют делать то же, что и массивы, но безопаснее.

Возможности векторов

Вектора можно создавать из элементов любых типов. Если при создании вектора не указывать число элементов, создается пустой вектор, не имеющий элементов, например:

```
vector<double> v;    // Пустой вектор
v[0] = 3.5;          // Ошибка, вектор v не имеет элементов
```

Функция `push_back()` добавляет в конец вектора новый элемент, например:

```
v.push_back(3.5);
v.push_back(7.9);
```

Теперь вектор `v` содержит элемент `v[0]` со значение `3.5` и элемент `v[1]` со значение `7.9`.

Текущий размер вектора можно изменить функцией `resize()`, например:

```
v.resize(5); // Теперь размер v равен 5
```

Вектора можно присваивать, подобно одиночным переменным, чего нельзя делать массивами.

Возможности работы с векторами иллюстрирует программа **4**.5.

Программа 4.5. Размеры и копирование векторов

```
// Файл SizeAndCopyVect.cpp

#include <iostream>
#include <locale>
#include <cstdlib>
#include <vector>
using namespace std;
const double Max = 1000.0;
int main()
{
    srand(time(0));      // Инициализация генератора случайных чисел
    setlocale(LC_ALL, "Russian");
    vector<double> v;             // Пустой вектор
    v.push_back(3.5);             // Добавление в вектор
    v.push_back(7.9);             // элементов
    cout << "Вектор v размера " << v.size() << endl;
    for(int i = 0; i < v.size(); i++)         // Вывод
        cout << v[i] << " ";                  // вектора
    v.resize(5);                              // Теперь размер v равен 5
    cout << "\nТеперь вектор v имеет размер " << v.size() << endl;
```

```
    for(int i = 0; i < v.size(); i++)         // Вывод
        cout << v[i] << " ";                  // вектора
    vector<double>w;                          // Еще один пустой вектор
    cout << "\nРазмер вектора w " << w.size() << endl;
    w = v;                                    // Присваивание векторов
    cout << "Теперь вектор w - копия вектора v\n";
    for(int i = 0; i < w.size(); i++)
        cout << w[i] << " ";
    for(int i = 0; i < v.size(); i++)         // Заполнение вектора v
        v[i] = rand() / Max;                  // случайными числами
    cout << "\nВектор v заполнен случайными числами\n";
    for(int i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
    system("pause");
    return 0;
}
```

Программа выводит:

```
Вектор v размера 2
3.5 7.9
Теперь вектор v имеет размер 5
3.5 7.9 0 0 0
Размер вектора w 0
Теперь вектор w - копия вектора v
3.5 7.9 0 0 0
Вектор v заполнен случайными числами
3.476 0.85 1.189 29.751 14.677
```