Массивы

Массивом называется последовательность переменных одного типа, использующая одно имя; для ссылки на конкретное значение применяется индекс.

С помощью массивов можно решить проблему работы с последовательностями. В приведенном ниже фрагменте объявляется массив A, в котором можно хранить до 128 целых значений. Затем он заполняется числами с клавиатуры:

```
int x; /*В x будут сохраняться вводимые с клавиатуры числа*/
int A [128]; // Объявление массива
// Определение индекса для доступа к элементам массива:
for (int i = 0; i < 128; i++)
{
   cin >> x;
   if (x < 0) break;
   A [i] = x;
}
```

Примечание. Необязательно пользователь будет вводить все 128 элементов массива.

```
int A [128];
```

В этой строке кода объявлен массив A. Первым в объявлении указывается тип элементов массива (int). За ним следует имя массива (A). Последним элементом является максимальное число элементов массива в []. В этом примере массив может содержать не более 128 целочисленных значений.

Доступ к элементам массива обеспечивается с помощью имени массива и индекса, указанного в []. Считывается введенное с клавиатуры число и сохраняется в следующем элементе массива. Первый элемент массива обозначается как A [0], второй – как A [1] и т.д.

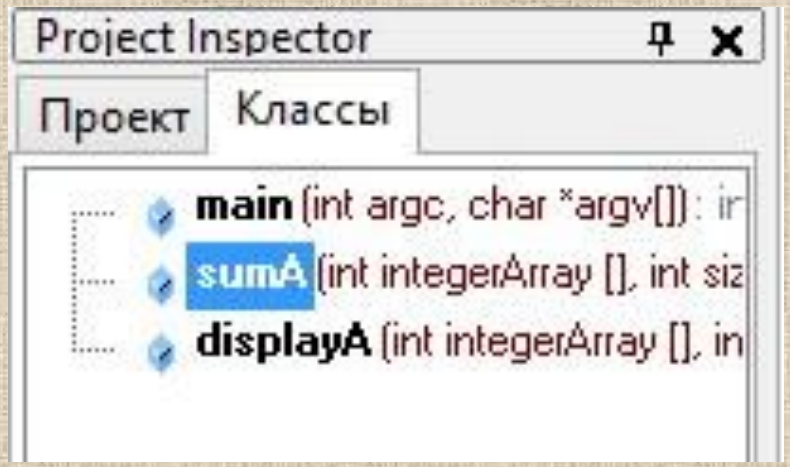
Запись A [i] представляет собой i-й элемент массива. Индексная переменная i должна быть перечислимой, т.е. ее типом может быть char, int или long. Если A – массив целых

```
// Программа massiv считывает последовательность целых чисел и отображает их по порядку
#include <cstdlib>
#include <iostream>
#include <locale>

using namespace std;
```

```
// Объявления прототипов функций
int sumA (int integerArray[], int sizeA);
void displayA (int integerArray[], int sizeA);

int main(int argc, char *argv[])
{
    setlocale(LC_ALL,"rus");
    // Описываем счетчик цикла
    int nAccumulator = 0;
    cout << "Эта программа суммирует числа, введенные пользователем\n";
    cout << "Цикл прерывается, когда юзер вводит отрицательное число\n";

    // Сохраняем числа в массиве
    int A [128];
    int j = 0;
    for (j = 0; j < 128; j++)
    {
        // Ввод очередного числа
        int x;
        cout << "Введите следующее число: ";
        cin >> x;

        // Если оно отрицательное...
        if (x < 0)
        {
            // ...тогда выходим из цикла
            break;
        }
        // ...иначе сохраняем число в массиве
        A [j] = x;
    }
    // Теперь выводим значения и их сумму
    displayA (A, j);
    cout << "Сумма введенных чисел равна " << sumA (A, j) << endl;

    system ("pause");
    return 0;
}
```

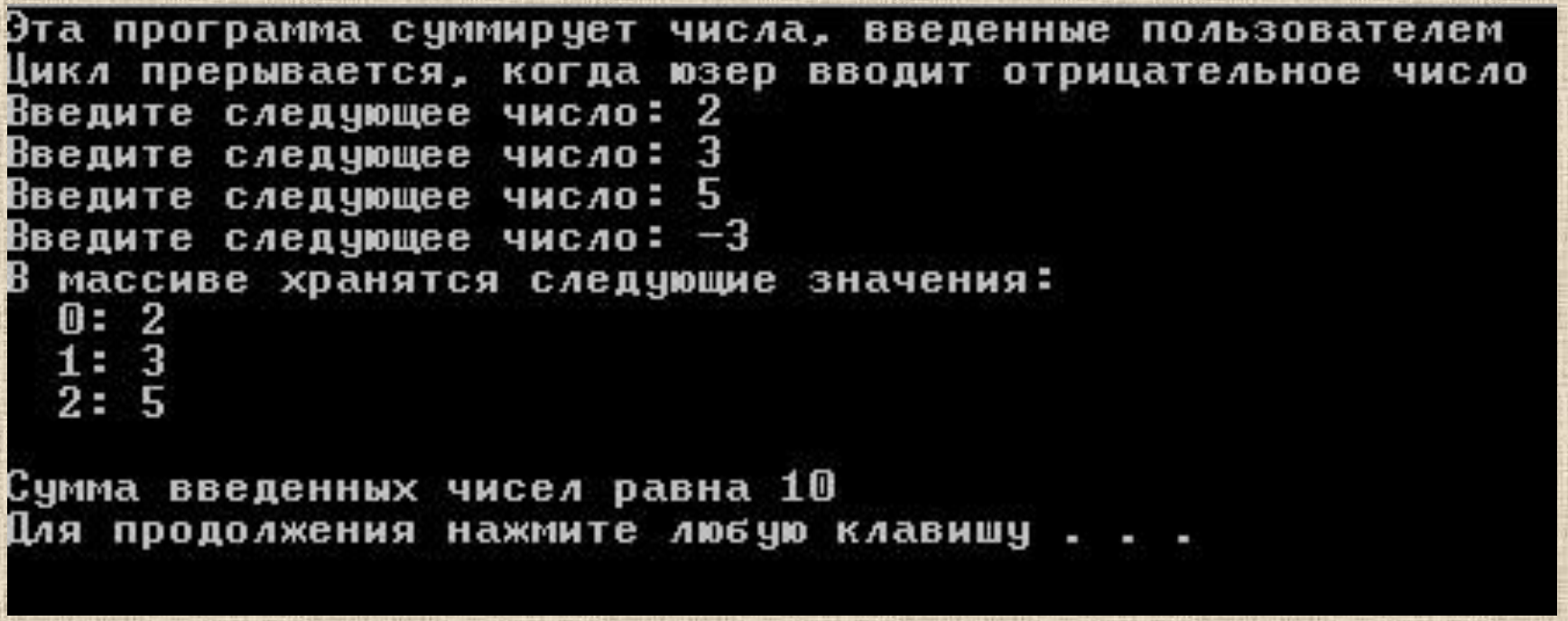
```
// displayArray - отображает элементы массива integerArray длиной sizeA
void displayA (int integerArray [], int sizeA)
{
    cout << "В массиве хранятся следующие значения:\n";
    for (int i = 0; i < sizeA; i++)
    {
        cout.width(3);
        cout << i << ": " << integerArray [i] << endl;
    }
    cout << endl;
}
```

```
//sumArray - возвращает сумму элементов целочисленного массива
int sumA (int integerArray [], int sizeA)
{
    int accumulator = 0;
    for (int i = 0; i < sizeA; i++)
    {
        accumulator += integerArray [i];
    }
    return accumulator;
}
```



Во вкладке Классы перечисляются все функции в исходном файле. Если в исходном тексте много функций, то ЛКМ на имени функции поможет быстро перейти на строку с выбранной функцией в окне редактирования.

Результат работы программы



Программа massiv начинается с объявления прототипов функций sumA и DisplayA, которые понадобятся позже. Главная часть программы содержит цикл ввода значений. Вводимые значения сохраняются в массиве A.

Если введенное значение отрицательно, цикл прерывается при помощи инструкции break, если же нет – оно копируется в массив: A [j] = x;

Целочисленная переменная j используется в качестве индекса массива. for (j = 0; j < 128; j++) Она инициализирована нулем в начале цикла for. При каждой итерации индекс увеличивается.

В условии выполнения цикла for осуществляется контроль за тем, чтобы количество введенных чисел не превышало 128, т.е. размера массива. После введения более 128 чисел программа может перейти к выводу элементов массива на экран независимо от того, ввел пользователь отрицательное число или нет, может аварийно завершить работу или работать некорректно.

Функция main заканчивается выводом на экран

Функция displayA () содержит обычный цикл for, который используется для прохождения по массиву.

Каждый очередной элемент массива добавляется к переменной accumulator. Передаваемый функции параметр sizeA включает количество значений, содержащихся в массиве.

Индекс массива в C++ отсчитывается от 0, а не от 1. Цикл for прерывается в тот момент, когда значение i становится равным sizeA. Ни один элемент массива, индекс которого больше или равен числу sizeA не будет корректно учитываться. Поэтому необходимо оставлять больше места для хранения данных, чтобы операции с массивом не приводили к выводу за его пределы.

Выход за границы массива

Самая распространенная ошибка – неправильное обращение к последнему элементу массива, например, по адресу A [15] массива, состоящего из 15 элементов. Хотя это всего лишь следующий за концом массива элемент, записывать или считывать его не менее опасно, чем любой другой некорректный адрес.

Математики перечисляют содержимое массивов, начиная с элемента номер 1. Первым элементом математического массива A является A[1]. Во многих языках программирования также начинают перечисление элементов массива с 1. Но в C++ массивы индексируются начиная с нуля. Первый элемент массива C++ обозначается как A [0]. Первый индекс массива C++ нулевой, поэтому последним элементом 15-элементного целочисленного массива A является A [14], а не A [15].

К сожалению, в C++ не проверяется выход индекса за пределы массива. C++ может предоставить доступ и к элементу A [500]. Более того, C++ позволит обратиться даже к A [-100]. Это можно объяснить с помощью следующей аналогии. Имеется улица, на которой 15 жилых домов. Если мы захотим найти 20-й дом, идя вдоль улицы и пересчитывая дома, то его просто не может быть. Тут могут быть заброшенные руины или, хуже того, дом, стоящий уже на другой улице. Чтение значения элемента A [20] может дать некоторое непредсказуемое значение или даже привести к ошибке нарушения защиты, а запись – к совершенно непредсказуемым результатам вплоть до полного краха программы.

Инициализация массива

Массив может быть инициализирован сразу во время объявления, например:

```
float A [4] = {1.0, 2.0, 3.0, 4.0}; // элементу A[0] присваивается
                                    //значение 1, A[1] – значение 2 и т.д.
```

Размер массива может определяться и количеством инициализирующих констант. Например, следующее объявление идентично представленному выше:

```
float A [] = {1.0, 2.0, 3.0, 4.0};
```

Все элементы массива можно инициализировать одним и тем же значением, указав его только один раз. Например, все 50 элементов массива A инициализируются значением 6:

```
int A [50] = {6};
```

Матрицы (многомерные массивы)

Иногда в некоторых приложениях приходится работать с последовательностями последовательностей, например, с таблицами, имеющими координаты – x и y.

В C++ матрицы определяются следующим образом:

```
int Matrix [10] [5]; /* Эта матрица может иметь 10 элементов в одном измерении и 5 в другом, что в сумме составляет 50 элементов*/
```

Matrix – 10-элементный массив, каждый элемент которого – массив из 5 элементов. Один угол матрицы обозначается Matrix [0] [0], а противоположный – Matrix [9] [4].

Матрицу можно инициализировать так же, как и массив:

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

– здесь фактически выполняется инициализация двух трехэлементных массивов: M [0] значениями 1, 2 и 3, а M [1] – значениями 4, 5 и 6.

Использование символьных массивов

В программе объявлен фиксированный массив символов, содержащий имя “Ivan”. Этот массив передается в функцию displayA вместе с его длиной.

```
// Выводит на экран массив символов
#include <cstdlib>
#include <iostream>

using namespace std;

// Объявления прототипов
void displayA (char stringA [], int sizeA);

int main (int argc, char *argv[])
{
    char myName [] = {'I', 'v', 'a', 'n'};
    displayA (myName, 4);
    cout << endl;
    system ("pause");
}

// displayA - отображает массив символов,
// по одному при каждой итерации
void displayA (char stringA [], int sizeA)
{
    for (int i = 0; i < sizeA; i++)
    {
        cout << stringA [i];
    }
}
```

```
Ivan
Для продолжения нажмите любую клавишу . . .
```

```
// Выводит на экран массив символов
#include <cstdlib>
#include <iostream>

using namespace std;

// Объявления прототипов
void displayA (char stringA []);
int main (int argc, char *argv[])
{
    char myName [] = {'I', 'v', 'a', 'n', 0};
    displayA (myName);
    cout << endl;
    system ("pause");
}

// displayA - посимвольно выводит на экран строку
void displayA (char stringA [])
{
    for (int i=0; stringA [i] != 0; i++)
    {
        cout << stringA [i];
    }
}
```

```
Ivan
Для продолжения нажмите любую клавишу . . .
```

Создание строки символов

Если в конце массива разместить специальный кодовый символ, то не потребуется передавать размеры массива (как это требуется в предыдущей программе). В C++ для этой цели зарезервирован нулевой символ.

Массив myName объявляется как массив символов с дополнительным нулевым символом в конце. Программа итеративно проходит по символьному массиву, пока не встретит нуль-символ.

Поскольку в этой программе функции displayA больше нет необходимости передавать длину символьного массива, использовать ее проще, чем в предыдущей программе. Включать нулевой символ в символьные массивы очень удобно, и в языке C++ он используется повсеместно. Для таких массивов даже придумано специальное имя:

Строка символов – это символьный массив с завершающим нулевым символом.

Выбор нулевого символа в качестве завершающего не был случаен. Это связано с тем, что в C++ только нулевое значение преобразуется в логическое значение false, а все остальные – в true. Это означает, что цикл for можно записать (что обычно и делается) следующим образом:

```
for (int i = 0; stringA [i]; i++)
```

Инициализировать строку в C++ можно с использованием двойных кавычек. Этот способ более удобен, чем тот, в котором используются одинарные кавычки для каждого символа. Следующие объявления идентичны:

```
char szMyName [] = “Ivan”;
char szMyName [] = {‘I’, ‘v’, ‘a’, ‘n’, ‘\0’};
```

Строка “Ivan” содержит 5, а не 4 символа (5-й – нулевой).

В соглашении об использовании имен для обозначения строк с завершающим нулем рекомендуется применять префикс sz.

Управление строками

Для работы со строками в C++ можно использовать стандартные библиотечные функции:

Название	Действие
int strlen (string)	Возвращает количество символов в строке (без учета нулевого символа)
char* strcat (target, source)	Присоединяет строку source к концу строки target
char* strcpy (target, source)	Копирует строку source в target
char* strncat (target, source, n)	Присоединяет не более n символов строки source к концу строки target
char* strncpy (target, source, n)	Копирует не более n символов строки source в target
char* strstr (source1, source2)	Находит первое вхождение строки source2 в source1
int strcmp (source1, source2)	Сравнивает две строки
int stricmp (source1, source2)	Сравнивает две строки без учета регистра символов

Чтобы использовать функции работы со строками, нужно добавить в начале программы директиву #include <string.h>.

```
// Concatenate - объединение двух строк, которые разделяются символом "-"
#include <cstdlib>
#include <iostream>
#include <locale>

using namespace std;

// Включаем файл, необходимый для использования функций работы со строками
#include <strings.h>

int main (int argc, char *argv[])
{
    setlocale (LC_ALL, "rus");

    // считываем первую строку
    char szString1 [256];
    cout << "Введите первую строку: ";
    cin >> szString1;

    // теперь вторую
    char szString2 [128];
    cout << "Введите вторую строку: ";
    cin >> szString2;

    // объединяем строки
    char szString [260];

    // копируем первую строку в буфер
    strncpy (szString, szString1, 128);

    // добавляем разделитель
    strncat (szString, " - ", 4);

    // теперь добавим вторую строку
    strncat (szString, szString2, 128);

    // и выведем результат на экран
    cout << "\n" << szString << endl;

    system ("pause");
    return 0;
}
```

```
Введите первую строку: one
Введите вторую строку: two

one - two
Для продолжения нажмите любую клавишу . . .
```

но

```
Введите первую строку: one one
Введите вторую строку:
one - one
Для продолжения нажмите любую клавишу . . .
```

и

```
Введите первую строку: one
Введите вторую строку: two three four five

one - two
Для продолжения нажмите любую клавишу . . .
```

считывает до нажатия клавиш Пробел или Enter

```cpp
// Concatenate - объединение двух строк, которые разделяются символом "-"
#include <cstdlib>
#include <iostream>
#include <locale>

using namespace std;

// Включаем файл, необходимый для использования функций работы со строками
#include <strings.h>

int main (int argc, char *argv[])
{
    setlocale (LC_ALL, "rus");

    // считываем первую строку
    char szString1 [256];
    cout << "Введите первую строку: ";
    cin.getline (szString1, 128);

    // теперь вторую
    char szString2 [128];
    cout << "Введите вторую строку: ";
    cin.getline (szString2, 128);

    // объединяем строки
    char szString [260];

    // копируем первую строку в буфер
    strncpy (szString, szString1, 128);

    // добавляем разделитель
    strncat (szString, " - ", 4);

    // теперь добавим вторую строку
    strncat (szString, szString2, 128);

    // и выведем результат на экран
    cout << "\n" << szString << endl;

    system ("pause");
    return 0;
}
```

```
Введите первую строку: one two three
Введите вторую строку: four five

one two three - four five
Для продолжения нажмите любую клавишу . . .
```

Функции strncpy() и strncat() в качестве одного из аргументов получают длину целевого буфера.

Вызов strncpy (szString, szString1, 128) означает “копировать в szString символы из szString1, пока не будет скопирован нулевой символ или пока не будет скопировано 128 символов”. Это не означает, что всякий раз будет копироваться 128 символов.

Существуют версии функций с передаваемой длиной буфера и без нее. Последние следует использовать, когда есть твердая уверенность, что переполнение целевого буфера возникнуть не может.

Тип String

ANSI C++ (стандарт) предоставляет программисту тип string, облегчающий работу с символьными строками (термин “строка” в C++ может означать как массив с завершающим нулевым символом, так и тип string). Тип string включает операции копирования, конкатенации, перевода строчных символов в прописные и т.п. функции. Они определены в заголовочном файле <string>.

```
// Concatenate - объединение двух строк,
// которые разделяются символом "-"
#include <cstdlib>
#include <iostream>
#include <locale>
#include <string>

using namespace std;

int main (int argc, char *argv[])
{
    setlocale (LC_ALL, "rus");

    // считываем первую строку
    string S1;
    cout << "Введите первую строку: ";
    cin >> S1;

    // теперь вторую
    string S2;
    cout << "Введите вторую строку: ";
    cin >> S2;

    // объединяем их в одном буфере
    string buffer;
    string divider = " - ";
    buffer = S1 + divider + S2;

    // и выведем результат на экран
    cout << "\n" << buffer << endl;

    system ("pause");
    return 0;
}
```

Здесь определены две переменные: S1 и S2. Эти переменные не имеют определенной длины – они могут расти и уменьшаться в зависимости от того, сколько символов в них находится, вплоть до всей оперативной памяти.

```
Введите первую строку: one two three
Введите вторую строку:
one - two
Для продолжения нажмите любую клавишу . . .
```