Chapter 8: Arrays

An **array** is a data structure that holds a fixed number of elements of the same **data type**. That is to say we can create an **array** named myNumbers, which holds *n* integers. All elements of the **array** can be accessed using the variable name myNumbers. This approach saves the programmer creating multiple individual variables for the same data. The **array** data structure also works well with loops.

The way this works behind the scenes is with the use of memory addresses. **Arrays** are nothing more than **pointers** and can be accessed in the same way. Don’t worry about pointers for the moment, they will be covered in Chapter 12: Pointers. It’s just something to bear in mind.

**One-Dimensional Arrays**

8.1 Example: Declaring Arrays

The most straightforward way to declare a variable is the following format:

DataType arrayName [ size ];

This creates an array with enough memory to hold *size* elements. The square brackets are important here.

An example of an array called myNumbers that holds 10 integers would look like this:

int myNumbers [ 10 ];

8.2 Example: Initialising Arrays

To set the value of an element in the array you follow the same process as any other variable. The only difference being that you need to inform the compiler which element you want to access. This is done using the square brackets again.

myNumbers [ 0 ] = 25;

myNumbers [ 1 ] = 600;

Note: When accessing elements of an array the positions run from 0 to size-1. So in the above example when accessing array elements they run from 0 to 9. 0 will give you the first element, 1 will give you the second and so on.

Note: If you use an element position that is outside of the range, you will get an out of bounds runtime error.

If you know the values of the elements then you can set this up at the declaration stage.

int myNumbers [ 10 ] = {25, 600, 2, 50, 1000, 75, 16, 8, 200, 99 };

It is also possible to omit the size of the array if you are giving the element values at declaration. In this case an array of the required size will be created.

float myOtherNumbers [ ] = {1.0f, 2.0f, 3.0f, 4.0f, 5.0f, 6.0f, 7.0f, 8.0f, 9.0f 10.0f };

**Program 18: Number Array**

This program will create an array and populate the elements. It will then output the array element and the value stored in a structured way using the **setw()** function. This function allows you to set how many characters to skip before moving onto the next output.. It stands for **set** **w**idth and takes an integer as its only parameter.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter8\_NumberArray”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “ArrayExample.cpp”
5. Replicate program listing 18.

#include <iostream>

using namespace std;

#include <iomanip>

using std::setw;

int main ()

{

int myNumbers[ 10 ];

// initialize elements of the array.

for ( int i = 0; i < 10; i++ )

myNumbers [ i ] = i;

// Output titles.

cout << "Array Element" << setw( 10 ) << "Stored Value" << endl;

// Output each array element's value .

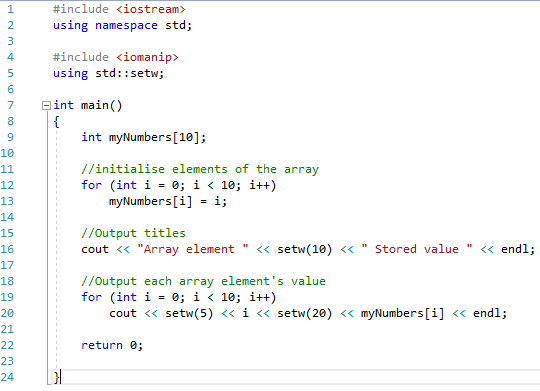
for ( int i = 0; i < 10; i++ )

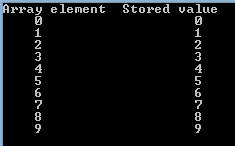
cout << setw( 5 )<< i << setw( 20 ) << myNumbers [ i ] << endl;

return 0;

}

Program Listing 18





**Multi-Dimensional Arrays**

We can extend the concept of arrays to hold more elements. The most common is a two-dimensional array. You can imagine this as a table. It is entirely possible to create arrays with more dimensions but this happens infrequently

8.3 Example: Declaring 2D Arrays

In much the same way as a single dimensional array is created, two dimensional arrays follow the same format, only now we add an additional size component.

DataType arrayName [ rows ][ columns ];

This creates an array with enough memory to hold (*rows* x *columns)* elements. To visualise this in a table take a look at Table 7.1: Array as Table.

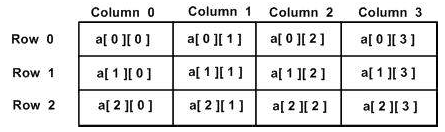


Table 7.1: Array as Table

An example of an array called myNumbers that holds 9 integers in a 5x3 table would look like this:

double myNumbers [ 5 ][ 3 ];

8.4 Example: Initialising 2D Arrays

To set the value of an element in the array you follow the same process as any other variable. The only difference being that you need to inform the compiler which element you want to access. This is done using the square brackets again.

myNumbers [ 0 ] [ 0 ] = 25.0f; // Sets the very first element to 25.0f.

myNumbers [ 4 ] [ 2 ]= 600.0f; // Sets element found at row 5 and column 3 to 600.0f.

Note: Remember when accessing elements of an array the positions run from 0 to size-1. So in the above example when accessing row 4 it is actually the 5th row. I.e. 0, 1, 2, 3, 4 is the 5th element when counting from zero.

If you know the values of the elements then you can set this up at the declaration stage.

double myNumbers [ 5 ] [ 3 ] = {

{ 10.0f, 12.0f, 14.0f },

{ 20.0f, 22.0f, 24.0f },

{ 30.0f, 32.0f, 34.0f }

};

It is also possible to omit the nested curly braces and declare a two-dimensional array in a single line. This is less intuitive and I’d recommend declaring arrays in the previous format.

double myNumbers [ 5 ] [ 3 ] = { 10.0f, 12.0f, 14.0f, 20.0f, 22.0f, 24.0f, 30.0f, 32.0f, 34.0f };

**Program 19: Multi-Dimensional Array**

This program will create an array and populate the elements. It will then output the array element and the value stored in a structured way using the **setw()** function. This function allows you to set how many characters to skip before moving onto the next output.. It stands for **set** **w**idth and takes an integer as its only parameter.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter8\_MultiDimensionalArray”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “MultiDimensionalArrayExample.cpp”
5. Replicate program listing 16.

#include <iostream>

using namespace std;

int main ()

{

// An array with 5 rows and 2 columns.

int myNumbers[5][2] = {

{0, 0},

{1, 2},

{2, 4},

{3, 6},

{4, 8}

};

// Output each array element's value

for ( int row = 0; row < 5; row++ )

{

for ( int column = 0; column < 2; column ++ )

{

cout << "mNumbers[" << row << "][" << column << "]: ";

cout << myNumbers[ row ][ column ]<< endl;

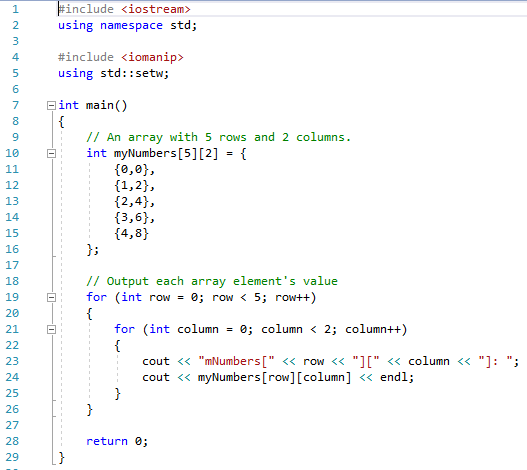
}

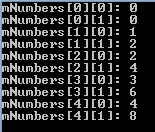
}

return 0;

}

Program Listing 19





**Program 20: Smallest Element**

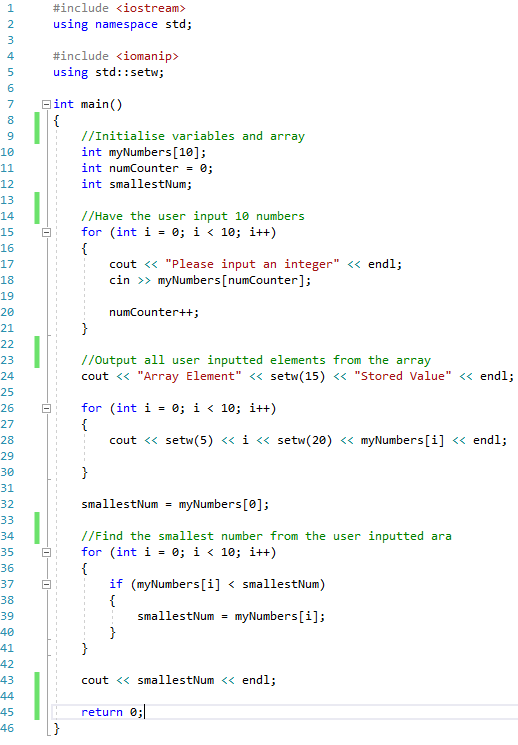
Write a program, which asks the user to input 10 integers. These values should be stored in an array for use later. Once all numbers have been entered, the program should find and output the smallest value and its position in the array.

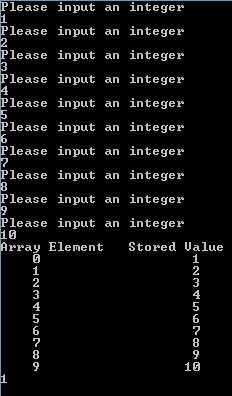
Note: This is an array exercise and the numbers MUST be stored in an array.

Run your program using the following values. The screenshot must show the use of these elements:

1 2 3 4 5 6 7 8 9 10

10 9 8 7 6 5 4 3 2 1

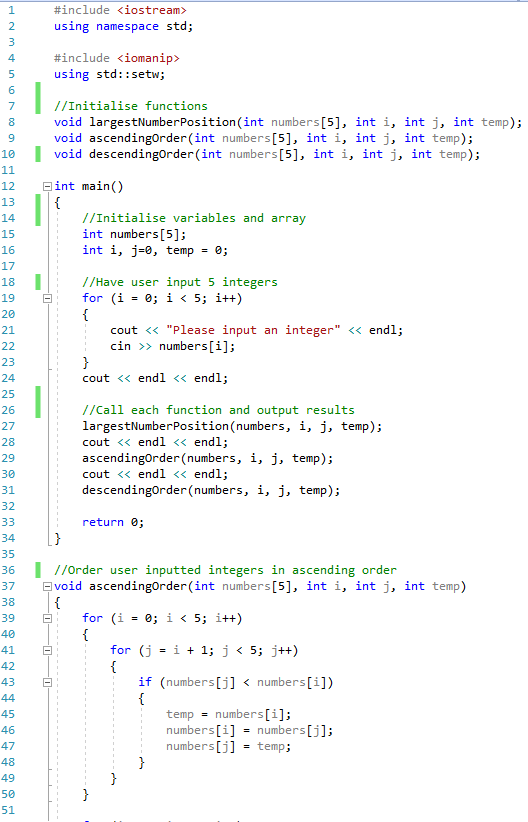


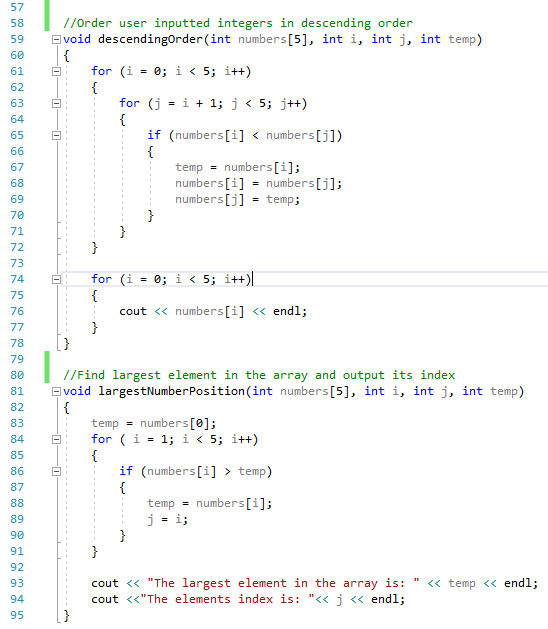


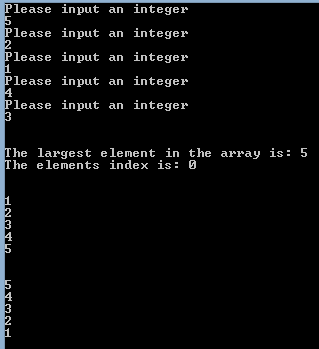
**Program 21: Ordered Output**

1. Write a simple C++ program that

* Declares an one dimensional array of integers of size 5.
* Asks the user for 5 integers to fill the array.
* Outputs the array in order.
* Outputs the array backwards.
* Outputs the largest element in the array and its position in the array.







**Program 22: Pairs**

The player selects 2 cards (one at a time) if they match the player gets a point and the card remain face up. The game continues until all cards have been turned.

Note: This program is to demonstrate the use of multi-dimensional arrays. Ensure that you complete this program using a 2d array.

**Tips**

1. Take a look at how Tic-Tac-Toe (This week’s appendix game) constructed its game board. Take a similar approach for your cards.
2. Use the system(“cls”) function.
3. Use the square brackets as the cards and have a number for selection.
4. Use letters as your card faces.

Example board: [1] [2] [3] [4] [5]

[6] [7] [8] [9] [10]

Example card faces: [A][B][C][D][E]

