Chapter 8: Arrays and Vectors

An array is a data structure that holds a fixed number of elements of the same data type. 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. Do not worry about pointers for the moment, they will be covered in a later chapter. It is just something to bear in mind.

**One-Dimensional Arrays**

**Declaring Arrays**

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



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



**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.



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.

Also, 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.

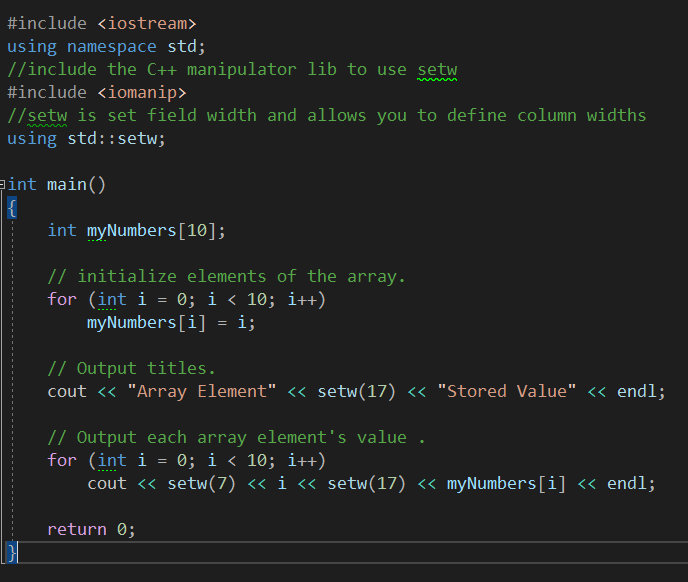


It is also possible to omit the size of the array if you are giving the element values at declaration. In this case an array of the required size will be created.



**Example array**

If you missed the lecture session, then please replicate the following program. 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 width and takes an integer as its only parameter.



**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.

**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.

****

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.

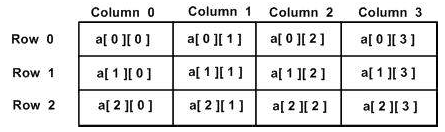


Table 7.1: Array as Table

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

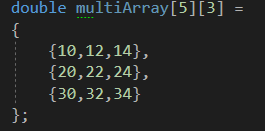
****

**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. 

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 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.



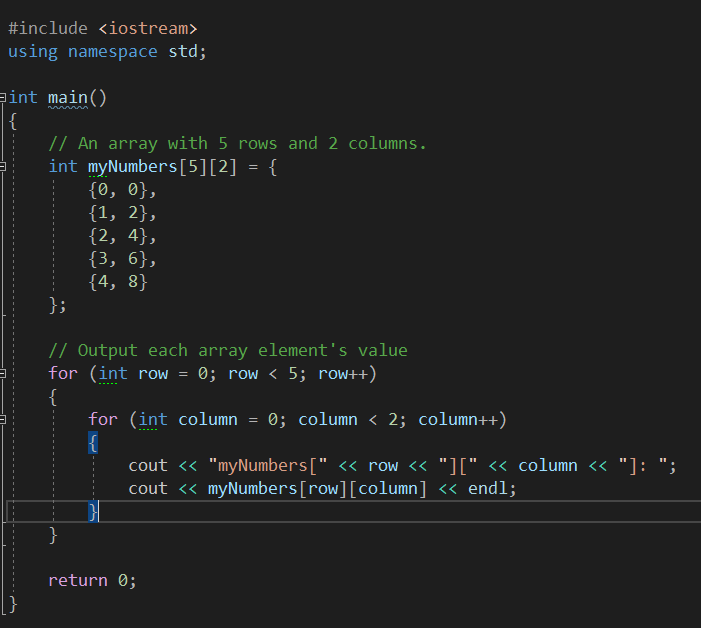
It is also possible to omit the nested curly braces and declare a two-dimensional array in a single line. This is less intuitive, and I would recommend declaring arrays in the previous format.



**Example of 2D Array**

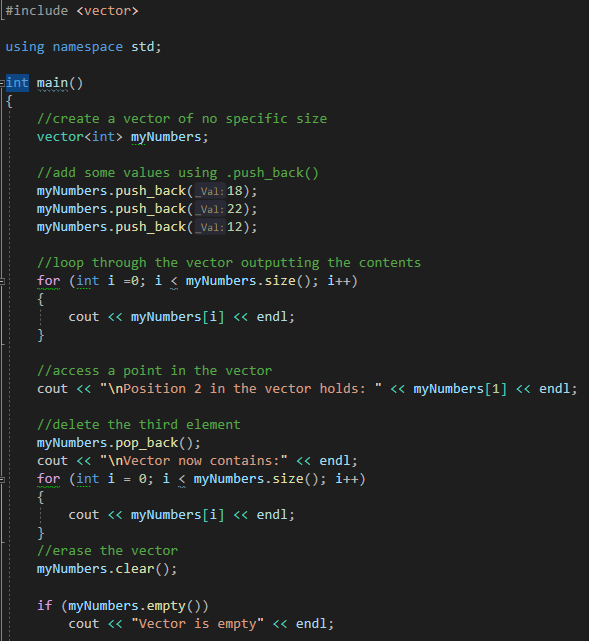
As with the single array above, if you missed the lecture please replicate the following program.

This program will define and initialise a 2D array and then output the results of the values held at each row and column.



**Example of Vectors:**

If you missed the lecture, please replicate the following program to get some familiarity with the use of vectors. We will be using them more next week.



**Program 21: 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.

**Things to consider:**

* How will you ask the user for input?
* How will you store the smallest element and its position?
* How will you loop through the array and compare numbers?
* How will you handle array range not being inclusive? I.E to match the user expected position being 1-10 instead of 0-9.
* Nested for loops are one option, you can also use one loop with a conditional inside. Research how to solve the problem if stuck.

**Program 21 Source Code:**

****

**Program 21 Screenshot:**

**Text

Description automatically generated**

**Program 22: Ordered Output.**

Write a simple C++ program that:

• Declares a one-dimensional array to hold 5 ints.

• Asks the user for 5 integers to fill the array.

• Outputs the array in descending order.

• Outputs the array ascending order.

• Outputs the largest element in the array and its position in the array.

**Things to consider:**

* Use a function to handle finding the biggest integer in the list.
* Research the C++ library <algorithm>. Sorting the ints into ascending and descending can be done several ways. A good programmer will always find and use the simplest method. Specifically read up on the **sort** method in the algorithm library.

**Program 22 Source Code:**

****

**Program 22 Screenshot:**

**Text

Description automatically generated**

**Program 23: Inventory**

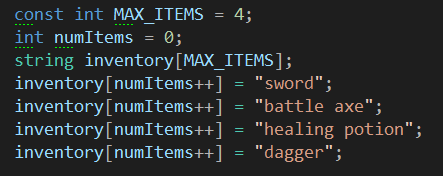
For this program you will be given a several starting variables and a complete array containing strings. Your job as the programmer is to inform the user, they have picked up a wizard’s staff and ask them if they would like to swap it for the dagger in their inventory.

The program should have two outcomes:

1. The player chooses to keep the staff.
   1. Update the contents of their inventory accessing the correct element. (Look at indexing)
   2. Output the contents of their inventory to screen. (Loops are best here)
2. The player chooses to leave the staff behind.
   1. Inform the player they have left the item behind
   2. Output the contents of their inventory

**Variables and Array initialisation**

This is one way this could be done, there are numerous ways, choose the method that suits you best, but this an alternative method for initialising as an example.



**Program 23 Source Code:**

****

**Program 23 Screenshots (must show both outputs):**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Program 24: Matching Pairs (challenging task)**

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.

Tips

1. Look at the below pseudo code to help guide you as to how to construct this program

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]

**Pseudo Code:**

Create a char array for your 10 board cards A-E twice over mixed up.

create a bool array for cards found and set this to equal false

^Hint: You'll need curly braces

create a variable to hold the attempt count

create two prototype functions for drawing the game board. One

takes no variables the other requires the two guesses to be passed

drawBoard

system("cls"); //this will clear the board of any turns

Tell player attempt count

Create a for loop

if cards found[i]

cout "[" << boardCards[i] << "]"

else

cout "[" << i + 1 << "]"

if i equal 4 split board

\*\*\*Main\*\*\*

create variables for score and a bool for playing

while the bool variable is true

draw the game board

Two ints to hold guess 1 and guess 2

ask for a selection and store it

Call the draw board that requires parameters, pass the guess

ask for second selection

do as above passing second guess

//Check results

Construct a way to check if the guesses match in an if statement

^Hint: boardCards[selectionA -1]

if match score++

cards found [selectionA -1] = true

cards found [B -1] = true

else

no match...

if score is equal to 5, player has won and end game

else attempts to be increased

drawBoard with parameters

system("cls"); //this will clear the board of any turns

Tell player attempt count

Create a for loop

if cards found[i] or i+1 == guess 1 or i+1 == guess 2

cout "[" << boardCards[i] << "]"

else

cout "[" << i + 1 << "]"

if i equal 4 split board

**Program 24 Source Code:**

**Program 24 Screenshots:**