Chapter 9: Strings

**String** is a data type we have not yet touched on. It allows for a collection of characters to be contained within a single variable. **String** was introduced as part of Standard C++, but before this we used the C-style character string using **char arrays**. We will be taking a look at both approaches below.

**Char Array**

The **char array** approach to strings comes from the C programming language. The way it works is that we create an array of **char** data type, which is terminated with a null character ‘\0’

To declare a string using this approach we have two options. The first is to categorically state what each character in the array is and add the terminating null character ourselves:

char testString[ 21 ] = { ‘T’, ‘h’, ‘i’, ‘s’, ‘ ‘, ‘i’, ‘s’, ‘ ‘, ‘a’, ‘ ‘, ‘t’, ‘e’, ‘s’, ‘t’, ‘ ‘, ‘s’, ‘t’, ‘i’, ‘n’, ‘g’, ‘\0’ };

Note: The size of the array is 1 bigger than the size of the string. This is to cater for the null character.

You can of course miss out the size as in array declarations, but can then use the double quotes as shown here:

char testString[ ] = “This is a test string”;

**Program 23: Char Array**

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter9\_CharArray”
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 23.

#include <iostream>

using namespace std;

int main ()

{

char testString1[21] = { ‘T’, ‘h’, ‘i’, ‘s’, ‘ ‘, ‘i’, ‘s’, ‘ ‘, ‘t’, ‘e’, ‘s’, ‘t’, ‘ ‘, ‘s’, ‘t’, ‘i’, ‘n’, ‘g’, ‘ ‘, ‘1’, ‘\0’ };

char testString2[ ] = “This is test string 2”;

cout << "testString1: " << testString1 << endl;

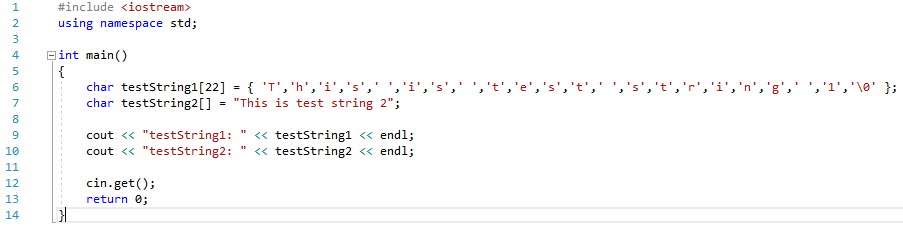
cout << "testString2: " << testString2 << endl;

cin.get();

return 0;

}

Program Listing 23





Screenshot 23

Within the cstring header there are a variety of functions, which can be used on null terminated strings. We will cover a few as detailed in Table8.1: CString Functions and in the examples below, but please take a look at: http://www.cplusplus.com/reference/cstring/ for further details.

|  |  |
| --- | --- |
| **Function** | **Description** |
| Strcpy\_s(s1, s2) | Copies string s2 into string s1. |
| Strcat\_s(s1, s2) | Concatenates s2 on to the end of s1. |
| strlen(s1) | Returns the length of s1. |
| strcmp(s1, s2) | Returns 0 is s1 and s2 match.  Returns less than 0 if s1 < s2.  Returns greater than 0 if s1 > s2. |

Table 9.1: String Functions

Note: In all of the below examples it is assumed that the program has included the <string.h> header like so:

#include <string.h>

Note: As Visual Studio is update by Microsoft often, functions become depreciated. If you get a warning of a depreciated function, you will need to investigate how to use the replacement.

9.1 Example: Copying Character Strings

To copy the contents from one char array to another you can use the **strcpy()** function. The first char array parameter passed in will be the one copied to, the second char array passed in is the one to be duplicated.

char stringToCopy[] = “TestString”;

char stringToChange[10];

cout << “Before copy” << endl;

cout << “stringToCopy contains “ << stringToCopy << endl;

cout << “stringToChange contains “ << stringToChange << endl;

strcpy\_s( stringToChange, stringToCopy );

cout << “After copy” << endl;

cout << “stringToCopy contains “ << stringToCopy << endl;

cout << “stringToChange contains “ << stringToChange << endl;

In this example we copied the entire string. It is possible to copy a portion of the string using the **strncpy\_s()** function.

9.2 Example: Concatenating Character Strings

To copy the contents from one char array and attach it to the end of another you can use the **strcat()** function. The first char array parameter passed in will be the one copied to, the second char array passed in is the one to be duplicated.

char stringToCopy[] = “TestString”;

char stringToChange[20] = “This is a “;

strcat\_s( stringToChange, stringToCopy );

cout << “Before concatenation” << endl;

cout << “stringToCopy contains “ << stringToCopy << endl;

cout << “stringToChange contains “ << stringToChange << endl;

strcat\_s( stringToChange, stringToCopy );

cout << “After concatenation” << endl;

cout << “stringToCopy contains “ << stringToCopy << endl;

cout << “stringToChange contains “ << stringToChange << endl;

9.3 Example: Character String Length

To determine the length of a string there is a function for that too. Use the **strlen()** function as shown below:

char testString[] = “TestString”;

cout << “Length of testString is “ << strlen(testString ) << endl;

9.4 Example: Comparing Character Strings

To determine whether two char arrays match simply pass the two strings through as parameters to the **strcmp()** function. This will only return a result of 0 if they are the same. Any other result means they do not match.

char string1[] = “TestString1”;

char string2[] = “TestString2”;

if ( strcmp( string1, string2 ) == 0 )

cout << “Matching strings “ << endl;

**Program 24: Initials**

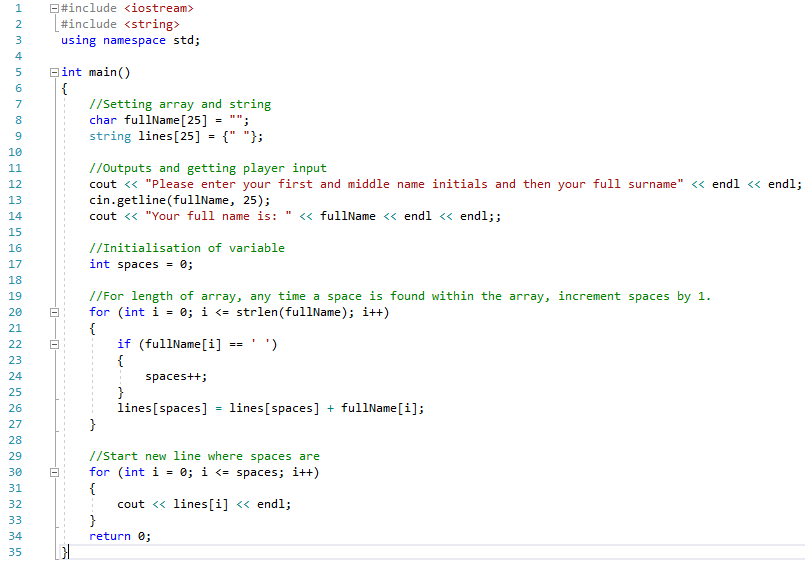
Write a program to prompt the user to enter in a single line their first name initial, followed by a space, their middle name initial followed by a space and the entirety of their surname. Store this in a char array.

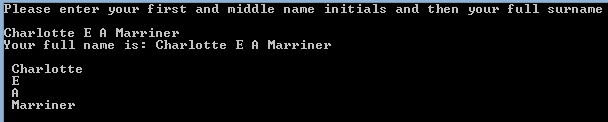
The program should then output the first initial on one line, then the middle initial on a separate line and the surname on a line of its own.

Note: This program is to demonstrate the use of char arrays. Ensure that you complete this program using a char array and that the data input by the user is stored in a single char array. You can separate out the various parts of the name into separate arrays afterwards, but the initial read from the console should put the entirety of the input into a single char array.

**Tips**

You will need to loop through your characters to find the first occurrence of a space. This should give you the information you require to be able to access the middle name initial.





**String Approach**

The **string** approach is an alternative way of dealing with a string of characters. Using **string** you are in fact using an underlying class, which is found in the **string.h** header. You must include this header at the top of any source file that uses **string**. **String** introduces a host of new functions, some of which are described below, others that you will need to research on your own.

So, to create a variable of **string** type it is as simple as:

#include <string>

string testString = “TestString”;

Note: You MUST now use double quotes. Single quotes will create a syntax error.

When using a **string** type with the console, you can now read in an entire line of character in one go using the **getline()** function. Once you press return, the text entered will be retrieved and stored in the variable.

getline( cin, testString);

You can still use **cin** in its previous format, but the text read in will stop at the first whitespace it encounters.

cin >> testString;

Output to the console screen is handled as you would expect:

cout << testString;

**Program 25: Console Input / Output with string**

As an example of the functions described above, the following code snippet will create a string, read in from the console then output what is stored in the variable. To test the functionality the user needs to enter text including a whitespace.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter9\_StringInputOutput”
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 “StringExample.cpp”
5. Replicate program listing 25.

#include <iostream>

#include <string>

int main()

{

string testString = “”;

cin >> testString;

cout << testString;

getline( cin, testString);

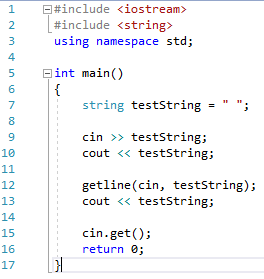
cout << testString;

cin.get();

return 0;

}

Program Listing 25





Screenshot 25

9.5 Example: String Concatenation / Appending

There are a couple of approaches to combining **string** objects. The first to be shown is the **append()** function, the second is using operators that have been overloaded such as + and +=.

string string1 = “Super”;

string string2 = “man”;

string1.append( string2 );

The above example will result in string1 containing the text “Superman”. The use of operators is shown next, which has the same end result.

string1 += string2;

Alternatively you could create a new string to hold the result:

string string3 = string1 + string2;

Note: Overloading Operators refers to changing the functionality of an operator depending on the type being used. For example int += int results in an int, string += string results in a string.

9.6 Example: String Comparisons

There are also a couple of possible approaches to determining if two **string** types match. The first is the **compare()** method, the second is an overloaded == operator. Both examples below will give the same result.

string string1 = “Example string”;

string string2 = “Test string”;

if( string1.compare( string2 ) == 0 )

cout << string1 + string2 + “ match” << endl;

if( string1 == string2 )

cout << string1 + string2 + “ match” << endl;

9.7 Example: Swapping Strings

**<string>** has a function called **swap()** and this allows you to swap the contents of one **string** for the contents of another.

string string1 = “string one”;

string string2 = “string two”;

string1.swap( string2 );

9.8 Example: Substrings

Another useful function is the **substr()**, which allows you to retrieve a particular portion from a string. The first integer parameter is the index in the string to start at, and the second integer parameter is how many characters from the first index.

string testString = “This is a test string.”;

cout << testString.substr( 10, 4 );

The above example will locate the ‘t’ and then take 4 characters. The resulting output will be “test”.

9.9 Example: Substrings - Finding

At times you will need to locate a particular phrase or character. **<string>** provides functionality for this. There are a few variants of the **find()** function and an example of each is shown below.

To find the location of the first occurrence of a phrase:

string testString = “String manipulation is cool! Is it not?”;

cout << “find( “is’ ) was found at << testString.find( “is” ) << endl;

To find the location of the first occurrence of a phrase starting at the end of the **string**, use the **rfind()** method.

string testString = “String manipulation is cool! Is it not?”;

cout << “rfind( “is’ ) was found at << testString.rfind( “is” ) << endl;

To locate the first / last occurrence of a character from a string you supply use the following examples.

string testString = “String manipulation is cool! Is it not?”;

int position = testString.find\_first\_of( “xziw” );

cout << “find\_first\_of( “xziw” ) found a << testString[ position ] << “ at position “ << position << endl;

The above example will find a ‘i’ at position 32.

9.10 Example: Substrings - Replacing

If you want to replace a character in a string with a different character then its as simple as using the **replace()** function. You will no doubt need to use the **find()** function first to locate the position of the character you want to replace. The first integer parameter is the position to start overwriting, the second parameter is how many characters to replace and the third is the string to replace with.

string testString1 = “String manipulation is cool! Is it not?”;

string testString2 = “No its not”;

cout << “testString1 contains: << testString1 << endl;

testString1.replace(28, 10, testString2);

cout << “testString1 now contains: << testString1 << endl;

9.11 Example: Substrings - Inserting

Replacing characters in a **string** is all well and good but what if you need to add some characters? That’s where the **insert()** function comes in useful.

string testString = “123456”;

string stringToInsert = “xx”;

testString.insert( 3, stringToInsert, 0, stringToInsrt.length()-1 );

The above example will result in testString containing “123xx456”.

Note: Before we move onto the string programs for you to implement it is important to note that there are many overloaded versions of the functions covered in this chapter, and a lot of others that have not been covered. It is up to you to do your own research to discover what they are and how they work.

**Program 26: Replacing Characters**

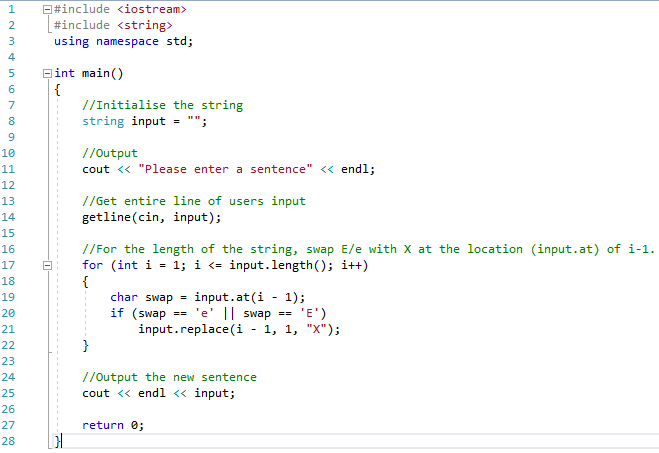
Write a program to prompt the user to enter in a sentence, which is then stored in a variable of string data type. Output the string they have entered to the console screen, then replace every occurrence of ‘e’ with an ‘X’.

The program should then output the altered sentence.

Note: This program is to demonstrate the use of the string data type.

**Tips**

You will need to use a loop to access the characters in the string.





**Program 27: String Manipulation**

Write a program to prompt the user to enter their name, which is then stored in a variable of string data type. Insert their name in between the ‘XX’ in the following string:

string testString = “Do you know who loves C++ XX does!”;

Output this string to the console. The program should then remove the two Xs and then output the string to the screen again.

The program should then ask the user to input another name (this one longer than the first). The program should then locate the first name in the test string and replace it with the new name.

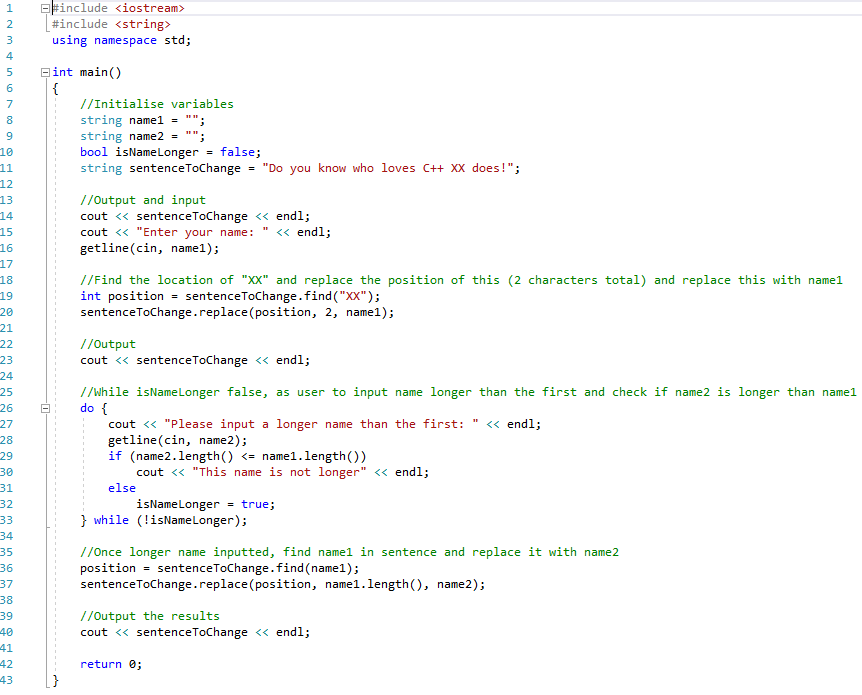
Output the altered string with the second name in place of the first to the console and screenshot the entire process.

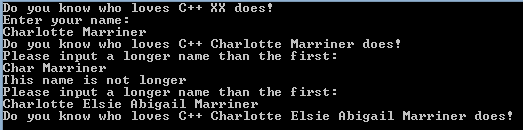
Note: The second name entered MUST be longer than the first name entered.

Note: This program is to demonstrate the use of the string data type.

**Tips**

Store the first users name in its own string variable. You can then use this to locate it again when replacing it with the second name.





**Program 28: Reversal**

Write a program that prompts the user for a positive integer of more than 1 digit and less than 11 digits (it must output a suitable error message if the input is not valid). Once the user has input a valid integer, your program will output the reverse of the integer and re-prompt for another integer, continuing the process until a suitable negative number is input.

For example:

Enter an integer of at least 2 digits or -1 to quit: 1

Your input is invalid, please try again.

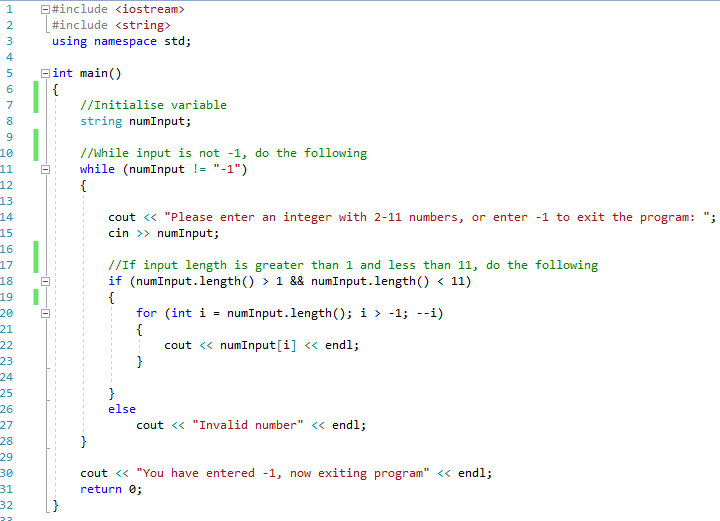
Enter an integer of at least 2 digits or -1 to quit: 123456

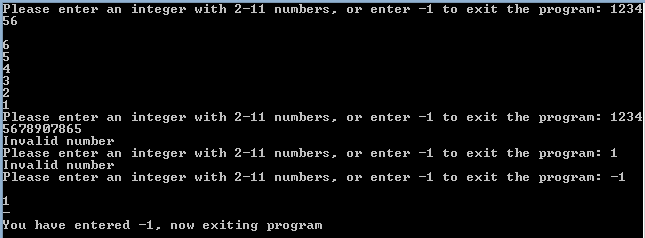
Your integer reversed is: 654321

Enter an integer of at least 2 digits or -1 to quit: -1

Program end.

Once your program is complete, test it with various inputs until you are confident it is functioning correctly. Your screenshot must show all possible conclusions.





**Program 29: Paper, Scissors, Rock**

Write a game of Paper, Rock, Scissors; where the user is prompted to enter their choice as a string (e.g. Paper or paper) and the computer's choice is generated randomly. The first to score 3 wins, is the overall winner.

At the conclusion of the game your program should output the result (who won) and the scores of both players.

Those not familiar with the game can learn a little about it here (<https://en.wikipedia.org/wiki/Rock-paper-scissors>)

When you have completed your program, take a screen shot of the last three plays and the concluding output (see below for an example).

