Programming 1: Tutorial 8

Introduction

For this tutorial you are going to be looking at C++ standard library strings.

You can either over write previous question solutions with new ones or you can have a project for each of the exercises – don't forget to make them blank applications.

Hopefully you remember the basic layout of a C++ program? If not refer to one of the previous tutorials, last weeks maybe more appropriate as you may wish to add functions to your programs.

It may also help if you go through a design stage for each exercise where you plan and puzzle through the logic before actually coding.

To complete these exercises you may need to reference the notes, either online or use the printed ones provided. Doing this isn't a bad thing; most programmers have to look up the syntax to make sure there code is correct.

There is sample code at the end of the tutorials, typing and running this code may help you when it comes to solving the exercises.

It is also important to understand what is happening with the questions, rather than just typing it into the compiler. If you look at it, run it and don't understand then just ask.

<u>Exercises</u>

<u>String Objects: Exercise 1</u>

The aim here is introduce you to string objects by getting you to use some of the functionality available.

Creating String Objects

The first thing to do in main() is to create 3 string objects in the 3 different ways available to us:

```
string string1 = “Welcome to“;
string string2(“Salford uni”);
string string3(3, !);
```

Word 1 and 2 are different methods of creating a strong object with the specified string. Word 3 creates a string object that has the specified character repeated a number of times.

Now print out to the screen the string objects. Note the following lines of code assume you have added using namespace std; at the top of your code.

```
cout << string1 << endl;
cout << string2 << endl;
cout << string3 << endl;
```

Now compile your code and see the results.

Concatenating

Concatenating is simply linking the string together. To do this we use the ‘+’ operator. The ‘+’ operator has been redefined, or overloaded, in for the context of dealing with strings so that rather than add them together in a mathematical sense it puts the strings together.
In your code now type the following:

```
string greeting = string1 + “ “ + string2 + string3;

cout << greeting << endl;
```

Now compile and run the code.

size() member function

We now look at using the ‘.’ Or member access operator to access the sting objects functions. One of those is the size function which returns the size of the string object concerned – spaces and all – as an unsigned integer.

Add the following code:

```
cout << “The number of characters in the string ‘greeting’
= “ << greeting.size() << endl;
```

Now re-compile and run the program.

Indexing
We can index arrays, we can also index strings in much the same way. Remembering as with arrays that if the size (or length) is n then the string or array is index from 0 to n-1. E.g. If size = 10, then the index is from 0 to 9. Indexing with string objects look identical to indexing with arrays. So if our array or string is called `word` then we index using the name, square brackets and a number: `word[0]` for instance access the first element.

I now want you to index the first character in the sequence of the string object greeting and change it to V.

Iteration
We can loop through string objects, accessing each character in turn. To do it we use the member function size(). In fact it is safer doing it this way than with arrays and it assures us that we are not going to have any out of bounds access.

Type the following:

```
for(int I = 0; I < greeting.size(); i++)
{
     cout << “Position “ << i << “ Character = “ <<
     greeting[i] << endl;
}
```

- *Question: What is occurring here?*

find() member function
The find member function checks to see whether a string literal or character is contained within the object whose find function you called.
The function returns the position number of the first occurrence where the `string` object for which you are searching begins in the calling `string` object. The position is based on the index so the first position is 0.
If the string or character you are searching for doesn’t exist in the calling string it returns a special constant which is defined in the string header file. We can access by using `string::npos`;

```
greeting.find(“uni”);
```

The following code above checks to see if the literal string “uni” is in the string object greeting, the position number of the first occurrence where the `string` object for which you are searching begins is returned.

Incorporate this code in to a `cout` statement so the position is printed to screen.

```
if(greeting.find(‘z’) != string::npos)
```

The code above checks to see if the character `z` is contained in the `greeting string` object. Complete the code so that you are informed if the letter is contained or not contained in the `string` object.

Other member functions
Other member functions can be looked at by checking out the following link:
http://www.cppreference.com/cppstring/

<u>Multidimensional Arrays: Exercise 2</u>

For this exercise I want you to declare a 2D array of dimensions 10x10. Initialise the array elements to 0 and then iterate through the whole array printing it out to the screen.
The print out should be in the form of a 2D array:

```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```