*Chapter 7* Strings

So far we have looked at variables which store integers, floats and characters. Something which they all have in common is that they have a fixed size. For example:-

* a variable of type **int** is always of size 2 bytes (4 bytes in 32-bit operating systems – 8 bytes in a 64-bit OS) - whether you store the number 0, -32768 or +32767 or anything in between.
* similarly a **float** takes up 4 bytes,
* and **char**acter variables use up 1 byte.

Strings are different. A ***string*** is a collection of characters - some examples are:-

"This is your life!"

"Mama mia!"

"abc"

"xyz1234"

"6543"

" "

"A"

""

The main thing which makes strings different is that they can be ***any length*** ("How long is a piece of string?"). The ones above have lengths of 18, 9, 3, 7, 3, 1 and 0 (yes, we can have a zero-length string, and it does have its uses).

Whereas declaring an integer, a float or a character variable automatically tells the compiler how much memory to set aside for that variable, it would appear to be impossible to say in advance what size string data our programs might encounter, since the strings may be input from the user. The trick is to guess the maximum size of string likely to be input to the program, and declare that.

Notice whenever we quote an actual string in a program, it must be in double quotes **""** (remember when we quote a single character we use single quotes **''**). You have already come across strings because we’ve used them in **cout** statements - the part between double quotes e.g. :-

**cout >> "Please enter a number from 1 to 5";**

Strings can be any sequence of any characters you may find in the ASCII table, including letters, digits, punctuation and control characters.

So far so good. But now we meet a complication in the world of **C** programming. In the original C programming language the designers chose not to include a native (built-in) data type for strings. You could have strings but they were just **arrays of chars** – ie a sequence of individual characters. This is a valid way to think of strings and it did mean that anyone who understood arrays could manipulate strings very flexibly. But some simple string activities became unnecessarily complicated – for example the common need to compare two strings needed a library function – as did concatenation.

In **C++** this was addressed by including a **string class**. This was more in keeping with the object oriented (OO) nature of C++. We are going to use these newer C++-style strings (rather than the old C-style strings).

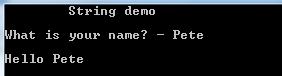
The next complication is that we need to include an extra library file in our programs. The standard iostream library which we have included in every one so far does not know about the string class. So we also need the string library:-

**#include <string>**

Here's a simple program to input and output a string:-

// String demo - C++ style strings

// PY Sept 2014



#include <iostream>

#include <string>

using namespace std;

void main()

{

string inName;

cout << '\t' << "String demo" << '\n' << '\n';

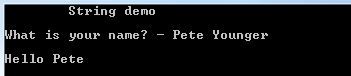
cout << "What is your name? - ";

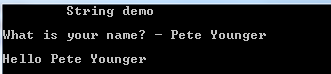
cin >> inName;

cout << endl << "Hello " << inName << endl << endl;

system("pause");

}

Try this. You should notice one problem. If you enter your full name then cin only accepts the first word. It stops at any space:-



To get round this we can use a function called getline() which handles cin:-

getline(cin, inName);

And another common problem is that getline() will fail to prompt because it reuses old data from the input buffer. A simple fix for this is to add a line before getline():-

cin.ignore();

getline(cin, inName);

# Finding the length of a string

It is often necessary to discover the length of a string (i.e. how many characters are in it, not including the null terminator). The function **length()** does this. For example:-

string inName;

int nameLen;

cout << '\t' << "String length demo" << '\n' << '\n';

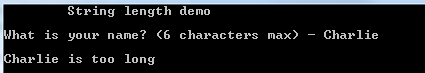
cout << "What is your name? (6 characters max) - ";

getline(cin, inName);

nameLen = inName.length();

if(nameLen > 6)

cout << endl << inName << " is too long " << endl << endl;



# Concatenating strings

Concatenating strings simply means 'stitching them together'. So, concatenating the strings "Somewhere " and "over the rainbow", gives "Somewhere over the rainbow". Note the inclusion of a space at the end of the first string so that the combination looks right.

In order to concatenate two C++ style strings we can simply use the + operator like this:-

string3 = string1 + string2;

The result of this is that the character array **string3** now has the combination string in it. If string1 was "**abc**" and string2 was "**def**" then string3 now contains "**abcdef**".

Now write a short program which gets a first name and a surname, concatenates them, and then displays "OK" if the resultant string is the same as your own full name.

Note that you should add a space between the forename and surname.

(Chap0701.cpp)