Week 1 Exercises : Sequential Programs

Jan 16th 2012

For the first week of the course we will look at simple programs that only have sequential control-flow. This means that they contain a simple list of statements. Each statement is a single step in the program. The first program that we will consider is the traditional starting point for learning to program:

```
#include<iostream>
using namespace std;
int main(int argc, char **argv)
{
  cout << "Hello World!\n";
}
```

Listing 1: Hello world program.

How does this relate to the overview of program steps that we saw in the lecture?

Input: None.

Output: The console is updated with new text.

Storage: A single piece of textual data.

Processing: A single effect that we can see, broken into smaller internal steps.

The single effect (an output) is a result of the *statement* within the program. The C++ language defines several types of statement that we can combine into *programs*. This statement is an *output* statement, we can recognise it because of the name `cout` (an abbreviation of console output) and the `<<` operator. The name on the left of the operator is the target of the output and the item on the right is a data value that is being output.

To understand this we need to break it smaller pieces: we learn what the small pieces are and what they do. For now we ignore everything outside of the braces `{...}` as it simply complicates things. Every program that we look at to begin with will contain the same lines outside of the braces: memorize them for now, later we will see what they are. The statements inside the braces are what we will look at first:

`cout`	The target of the output
`<<`	The output operator
`"Hello World!\n"`	A constant string
`;`	Statement terminator

We need to specify where the data will be output as there multiple possibilities, for now we will only look at the console, but later we will look at other possible targets for output such as files. This tells the compiler where the data should be output. The operator tells the compiler what we want the statement to do, in this case output data. The string is a type of data (value) that is the thing being output. In C++ string values are indicated by the quotes `"a string"`. The terminator tells the compiler that the statement has finished, this is necessary so that the compiler can tell which parts of the source text belong to which statement.

1. Type in the hello world program from Listing 1 and save it into a file called `hello.c`. Use the `cl.exe` compiler to compile it. If the compiler produces error messages then use them to discover the mistake in your typing and correct them until it does compile.

Many exercises in programming involve reusing programs that you have already written and modifying them to achieve a new result. For your next exercise you are going to reuse your existing program to make the same result in a new way. Remember, the `cout` line in your Hello World program does the work, while the rest are all a template that you will reuse. Listing 2 contains several variations of that one line:

```
cout << "Hello " << "World!\n";
cout << "Hello World!" << endl;
cout << "Hello "; cout << "World!\n";
cout << string("Hello ") + string("World!") << endl;
```

Listing 2: Replacement bodies for the Hello world program.

2. Write a separate program for each of the lines in Listing 2. In each case replace the `cout` line in Hello World with the line from Listing 2. After you try these programs you should be able to answer the question: what are the observable effects of outputting multiple strings in sequence, and how do they compare to outputting strings with the combined contents, or concatenated strings?

Echo is the name of a simple UNIX program. In a command line setting it is a program that reads input and then repeats it as output (or "echoes" what it sees). For the next exercise you will write a set of simple echo programs that repeat different types of data.

3. Write a program that inputs and then outputs a) an integer, b) a single character, c) a string.
4. Write a program that inputs three integers, and then outputs them in reverse order.
5. Write a program that inputs two strings and then joins them (concatenates) together. Output the concatenated string.
6. Write a program that inputs age, birthday, and name into appropriate types of variable. Output a welcome message that includes all of the information.
7. Combine the variables for age, birthday and name into a class declaration called `People`. Declare two `People` objects and input two sets of data, then output a pair of welcome messages.

So far you have seen a mixture of strings declared as `string` objects and constant string literals (the text between quotes). These are used in different ways in C++ programming and behave differently.

```
cout << "17" << endl;
cout << 17 << endl;
cout << string("17")+string("20") << endl;
cout << "17"+"20" << endl;
cout << 17+20 << endl;
```

Listing 3: Replacement bodies for the Hello world program.

8. Try each of the replacement lines from Listing 3 within your program template. After watching what happens you should be able to explain what is the difference between addition of numbers, addition of numbers in quotes, and addition of string objects made from numbers in quotes.
9. Try the statement `cout << setw(8) << 22 << endl;` in a program to see what it does. Write a program that inputs three integers and outputs them in a right justified 10-character wide column.
10. Input two numbers and add them together, outputting the result. Look for differences in behaviour between `int` and `float` variables.