CMPT 225 - Lab 0: Hello World + Testing

In this lab you will write and test a simple C++ program.

All C++ programs start their execution at the **main** function. This is similar to Java, though different in that the C++ **main** function is not contained in a class.

Creating your main function

Your first task is to create a very simple **main** function that prints **Hello world!**.

Steps:

- You should be logged onto Ubuntu on a CISL workstation (our **target machine**) either in person or remotely.
- Open a Terminal window. To do so, look for the "Terminal" application and start executing it.
- At the command line, type **pwd** (print working directory) to see which directory you're in. Note that in the following examples, the string ending in **~$** is assumed to be the command line prompt, and should not be typed.

  ```
  uname@hostname:~$ pwd
  /home/uname
  ```

- Change directory to your **sfuhome**. Files stored here will persist on the network and will be accessible if you login to another CSIL workstation.

  ```
  uname@hostname:~$ cd sfuhome
  ```

- Make a directory in which you will store all your CMPT 225 work (labs and assignments) throughout the semester. Below, we call this new directory **cmpt-225**.

  ```
  uname@hostname:~/sfuhome$ mkdir cmpt-225
  ```

- Change to your **cmpt-225** directory.

  ```
  uname@hostname:~/sfuhome$ cd cmpt-225
  ```

- Make a directory for the files of this lab. Name it **Lab0**.

  ```
  uname@hostname:~/sfuhome/cmpt-225$ mkdir Lab0
  ```

- Change to this directory using **cd** (change directory):

  ```
  uname@hostname:~/sfuhome/cmpt-225$ cd Lab0
  ```

- Open a text/code editor such as "gedit", which can be opened at the command prompt using the command: **gedit &** . You can also use Visual Studio Code, Emacs, etc. For a list of editors, click on the menu option "Application Finder" then click on "Programming".
- Copy the following code into a new file you create in the editor:

  ```
  /*
   * hello.cpp
   *
   * Description: Solution to the good old "Hello world! problem.
   *
   * Author: AL
   * Last modified on: May 2024
   */

  #include <iostream>
  using std::cout;

  int main (void) {
      cout << "Hello world!";
      return 0;
  }
  ```

- Save this to a file called **hello.cpp** in your **Lab0** directory.
- Type **ls** (list) to see the contents of the current directory. **hello.cpp** should appear:

  ```
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ ls
  hello.cpp
  ```

Some explanations

Header comment block

We start the file with a **header comment block**, which documents various aspects of the content of the file:

- The name of the file
- The description of the content, i.e., of the program found in the file
- The author of the program
- The creation date or last modification date of the program

Include and using statements

To get input and print output in C++, use the **cin** and **cout** objects. **cin** is used for standard input (most often, from the keyboard) and **cout** for standard output (most often, to the computer monitor screen). Both objects are within the **iostream** library.

Your program (not the file) must start with **include** statements to import the necessary libraries, in this case **iostream**.

In addition, if your program is using objects such as **cout**, **cin**, **endl**, etc., which have already been declared somewhere else, you need to state so.

One way to achieve this is by placing the following lines:

```
#include <iostream>
using std::cout;
```

above the **main** function.

Another way to achieve this is to replace **using std::cout;** by **using namespace std;** as follows:

```
#include <iostream>
using namespace std;
```

However, some C++ software developers consider the use of **using namespace std;** to be *bad* practice. Here is an article that explains why.

Using cin and cout

To get input from the keyboard into a variable named **someInteger**:

```
int someInteger = 0;
cin >> someInteger;
```

To send output to the computer monitor screen:

```
cout << someInteger;
```

Note that this will not print the value of **someInteger** on a new line. It will actually print the value of **someInteger** wherever the cursor is located on the computer monitor screen. However, you can use **endl** to specify a newline. For example, the C++ statement below prints the value of **someInteger** on a new line along with some explanatory text (by the way, including such text is a good programming style - GPS - to adopt):

```
cout << endl << "The value of someInteger is: " << someInteger << endl;
```

Return values

By convention, the **main** function returns the value **0** if everything went well. Other values can be used to denote various forms of failure or output results.

Compiling your main function

In order to run the program you just wrote, you need to compile it into an executable file. The C++ compiler we will use in this course is called **g++**.

- Run the compiler on the program contained in **hello.cpp** by typing:

  ```
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ g++ hello.cpp
  ```

  If there are errors in your code, the compiler will let you know. Otherwise, it will produce an executable file called *a.out*:

  ```
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ ls
  a.out hello.cpp
  ```

- Run this executable file by typing:

  ```
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ ./a.out
  Hello world!
  ```

  "." refers to the current directory, and the **"/"** character is used to separate directories and file names in a Linux file path.

- The name **a.out** is the default name given to executable files, but is not particularly descriptive. You can specify another name for an executable file produced by **g++** using the **-o** option:

  ```
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ g++ -o hello_world hello.cpp
  ```

- You can now run **hello_world** in a similar fashion:

  ```
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ ./hello_world
  Hello world!
  ```

Say hello

Modify **hello.cpp** to have the following behaviour:

- Obtain a single integer as input using **cin**.
- Print "Hello world!" (no quotes), unless the integer is **42**, in which case "The answer to the ultime question of life, the universe and everything!" (no quotes) must be printed.

Your program must not output anything else, such as prompts to the user (although, this would be very user friendly and a good programming style to adopt). Your program must create exactly the following output when it executes and the user enters **5** and when the user enters **42**:

```
uname@hostname:~/sfuhome/cmpt-225/Lab0$ ./hello_world
5
Hello world!
uname@hostname:~/sfuhome/cmpt-225/Lab0$ ./hello_world
42
The answer to the ultime question of life, the universe and everything!
uname@hostname:~$
```

Do not forget to modify the header comment block to reflect the new behaviour (description) of our program.

Testing

Let's now test this program.

- Download this zip file and unzip it in your **Lab0** directory:

  ```
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ unzip Lab0-Files.zip
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ ls
  1.er          2.in          a.out         hello_world   test.py
  1.in          2.er          hello.cpp     Lab0-Files.zip
  ```

  The files with the **.in** extension contain **input test data** and the files with the **.er** extension contain **expected results**. We shall talk about **input test data** and **expected results** in more details during our lectures when we talk about testing. The files **1.in** and **1.er** represent parts of Test Case 1, while the files **2.in** and **2.er** represent parts of Test Case 2.

- Have a look at the Python script **test.py**. Its first line, called the **shebang** (#!) line, tells Linux to use Python to run the script. Then, the Python script **test.py** opens the **.in** files, uses **input/output redirection** (>,<) to pass their content as input test data to your **hello.cpp** program. The Python script then captures the output of **hello.cpp**, i.e., the **actual results**, and saves these results into the files with the extension **.out**. Then it compares these **actual results** (the content of the **.out** files) against the **expected results** (the content of the **.er** files) using the command **diff**. If both results (expected and actual) defined as part of Test Case 1 are exactly the same, i.e., if **1.out** exactly matches **1.er**, then **hello.cpp** has passed Test Case 1. Ditto for Test Case 2!
- Run the program **test.py**:

  ```
  uname@hostname:~/sfuhome/cmpt-225/Lab0$ ./test.py
  ```

  If you are unable to run **test.py** because of permission problem, enter the following at the command line:

  ```
  $ chmod 755 test.py
  ```

  This command changes the permission of **test.py** such that it should now be **executable**. Try running **test.py** again. If you are curious about this command and how permissions work in Linux, feel free to ask the instructor, the TA or the Internet. :)

  If you have correctly modified **hello.cpp** as specified above, you will see:

  ```
  Running test 1... passed
  Running test 2... passed
  Passed 2 of 2 tests
  ```

  If not, debug, recompile and retest your program until you do!

Enjoy!
