Tutorial 1 of 10

Important:

- No late tutorials will be accepted.
- If there are specific instructions for making a function, please follow them exactly. That means that
  - function names
  - function return types
  - parameter types and order

  should all be **EXACTLY** as described. If the script can't read it, you will receive 0 for that part.
- Your **Tutorial 1** code will be marked by a Python script. You are being given the script so that you may make sure your code runs correctly. As such, submissions with improper files, configuration, or function signatures will not be accepted.

1 Submission Instructions

Download tutorial1.zip. Place it into your working directory and unzip it. There is the test script `t1test.py` and a folder "tutorial1". Inside the "tutorial1" folder is a Makefile and a short C++ program `p1.cc` that you can make and run if you like.

You will write or modify 5 files in the "tutorial1" directory. These files are `p1.cc`, `p2.cc`, `power.cc`, `p3.cc`, and `Makefile`.

Once you have finished and tested your code, you will zip the "tutorial1" directory into a file "tutorial1.zip". If you are doing this in the course vm, you must do this from the command line. Open a terminal in the folder that contains "tutorial1" (that means if you use the `ls` command you should see the "tutorial1" folder and `test1.py`). Use the command `zip -r tutorial1.zip tutorial1`. This will zip your file and update it if you change the contents. **Please note the use of `-r`. If you do not include `-r` you will be submitting an empty folder.** Submit "tutorial1.zip" to Brightspace by the deadline. DO NOT USE .tar OR .tar.gz FILES. Use .zip files only please.

2 Testing Your Tutorial With `t1test.py`

`t1test.py` is a test script that is very similar to what will be used to mark your tutorial (basically we will change the input and expected output). So the mark you see here should be the mark you receive, as long as you did not hard code output. You should run `t1test.py` from just outside the "tutorial1" folder (exactly as it was when you unzipped it). Open a command line in the folder containing `t1test.py`. You may have to make it executable, so type `chmod +x t1test.py`. You may run the script with the "unzip" step, or, if you have not zipped your files yet you may supply a "-nozip" argument.

To have the script unzip `tutorial1.zip` and then test your code, run `./t1test.py`. To skip the unzip step use `./t1test.py -nozip`. When your tutorial is being officially marked we expect a zipped file.

Running this script will generate a file "results.txt". This will have your program output, the script output, and the mark.

3 Learning Outcomes

This tutorial introduces some very basic concepts, such as variables, input and output, and input and output parameters. It is also meant to give you practice compiling and running in C++.

4 Instructions

4.1 Makefile

Your Makefile should have compilation instructions for each of the three executables described below. For example to compile `p1` you would use:

```
p1:  p1.cc
   g++ -o p1 p1.cc
```

In addition your Makefile should contain an `all` command that compiles all three executables and a `clean` command that removes all three executables.

4.2 File p1.cc

Write a `main` function that

1. asks the user for their name
2. takes user input
3. prints "Hello <username>!" to the console.

   The following could be an example interaction. Red text represents user input.

   ```
   >Hi!  What is your name?  Darryl

   >Hello Darryl!
   ```

   In the Makefile, compile this into an executable file `p1`.

4.3 File p2.cc

Write a `main` function that asks the user for two integers. Compute the product of those integers, then output the answer (the input should be included in the output). For example:

```
>Please enter two integers:  5 10

>The product of 5 and 10 is 50!
```

In the Makefile, compile this into an executable file `p2`.

4.4 File power.cc

Make a function `void power(int a, int b, int& c)`. a and b are input parameters. c is an output parameter. Your function should compute the value a^b (probably using a `for-loop`) and store it in c. Make sure that the function signature is precisely what is described above. You should compile this into object code.

4.5 File p3.cc

Write a `main` function. This function should ask the user for two integers. It should then call the `power` function from `power.cc` to compute the first integer taken to the power of the second integer. It should then print the output to the console. For example:

```
>Please enter two integers:  5 3

>5 to the power 3 is 125!
```

In the Makefile, compile this into an executable file `p3`. You will have to link `power.o` to do this. Currently the script does not test if you are compiling to object code and linking, but the script that I use for marking will copy in a file `test.cc` and will attempt to link and use `power.o` and the `power` function.