ECE 2036 Lab 1: Introduction to Software Objects

Assigned: Aug 24/25 2015
Due: September 1, 2015 by 11:59 PM
Reading: Deitel& Deitel Chapter 2-4
Student Name: _____

Check Off/Score Part 1: __ ___ Check Off/Score Part 2: __ ____Check Off/Score Part 3: _ ____

Blaise Pascal was a French mathematician and physicist who lived in the mid 1600's. He died when he was only 39 years old, but in that short time he made many contributions in the field of physics and mathematics. For example, you might have heard of Pascal's triangle, which is related to his study of generating binomial coefficients (i.e. the coefficients for expanding the binomial expression $(x+y)^n$). Eleven years earlier, Monsieur Pascal in 1642 as a teenager started building mechanical calculating machines – in fact he was one of the first to do this! This lab is inspired by Monsieur Pascal to help introduce a few programming concepts of modern computing machines!

Part 0: (0%) Accessing Deepthought

You will need to use a linux cluster called "deepthought" to create, run, and turnin your code for this lab. Please use Appendix A to help you get access to this cluster. Please note that you will need to use some linux commands to interact with the operating system of this cluster. We have included some basic commands in this Appendix A, but you should be able find many tutorials online if you need to go a little deeper.

Part 1: (30%) *Calculate Pascal Triangle*

For this first part of the lab, we will start with the use of basic C/C++ commands to calculate and display Pascal's triangle. If you are unfamiliar with the form of this triangle please educate yourself with Wikipedia's page on the subject. For your program, you will prompt the user for the size of the triangle to create. Make sure you use I/O manipulators and insertion stream operators to print to your terminal as discussed in class (i.e. do not use `printf`). We would suggest that you create a basic global function to calculate the factorial iteratively using for loops (we will discuss recursion later in the course). Appendix B shows sample input and output for this program. For submission on deepthought, please name the file that contains your program `lab1_part1.cc`. The submission instructions are in Appendix A.

Part 2: ***(40%) Make a Pascaline Software Object***

A key object-oriented feature of C++ are `class` definitions, which are used to create software objects. In this part of the lab, you will create an object in software that represents a simple adding machine that is, at a high level, generally similar to the basic mechanical calculator created by Blaise Pascal in 1642, which is called the Pascaline (see figure below).

For this lab, you will need to create the `class Pascaline` based on the skeleton code in Appendix C. Although this is not exactly how the Pascaline operated, for your Pascaline object you will need three private integer data members, *register1, register2,* and *registerResult*. Also you will need the following member functions for your object, which are *setRegister1(int), setRegister2(int), setRegisterResult(int), int getRegister1(), int getRegister2(), int getRegisterResult(), void addRegisters(), void clearRegisters(), void getInputValues(), and void displayOutputValue()*.

You will include the implementation of the member functions **OUTSIDE** the class definition; however, in this lab your entire program will be in one text file, which you should call `lab1_part2.cc`. As explained in section 3.6 of the textbook, we will start to separate files in a more sophisticated way to include a class interface in a header file with a separate implementation file for class member functions.

Programming Requirements Part 2:

1. Create a member function, `clearRegisters()`, that sets `register1`, `register2`, and `registerResult` to zero. This function has no values passed to it and has a void return type.

2. Create a constructor that initializes the data members of the `Pascaline` object to zero.

3. Create a member function, `addRegisters()`, which will add the value of the two registers, `register1` and `register2`, and put the sum into `registerResult`. This function has no values passed to it and has a void return type.

4. You will need to create the member function, `getInputValues()`, which should prompt the user for the inputs in the following way:

   ```
   Please input the value of register1:
   Please input the value of register2:
   ```

   This function has no values passed to it and has a void return type.

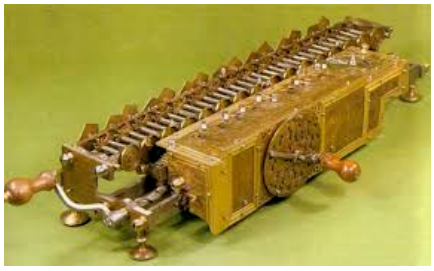
5. You will need to create the member function, `displayOutputValues()`, which displays the results of the calculation as seen below. This output example below assumes that the contents of `register1` is 3 and the contents of `register2` is 4. This function has no values passed to it and has a void return type.

   ```
   ***Pascaline Result: 3 + 4 = 7
   ```

6. We are providing some skeleton code to get you started; however, please note that there are a few little surprises (i.e. errors) in the code that you will need to find. This skeleton code appears in Appendix C. Make sure your output looks like the examples in Appendix D.

Part 3: ***(30%) Make LeibnizWheel Object***

About 30 years later, Leibniz developed a mechanical multiplier called a "Leibniz Wheel". We would like you to create a second class called `LeibnizWheel`. It will have the same set and get functions of the `Pascaline` class in part 2, but this will have a `multiplyRegisters()` member function instead of the `addRegisters()` member function in the Pascaline class. Also, `DisplayOutputValue()` for the `LeibnizWheel` class should display the product in the `registerResult()` in the `LeibnizWheel` object. The sample output appears in Appendix D.



Programming Requirements Part 3:

1. In this program, which you should call `lab1_part3.cc`, you will need to make an instance of TWO objects to get the sample output as indicated below (i.e. a Pascaline object and a LeibnitzWheel object should be included in this program). You will have to create a main function that manipulates both objects to produce the desired functionality that is illustrated in Appendix D.
2. Create a member function, `clearRegisters()`, that sets `register1`, `register2`, and `registerResult`, to zero. This function has no values passed to it and has a void return type.
3. Create a constructor that initializes the data members of the `LeibnizWheel` object to zero.
4. For this `LeibnizWheel` class, create a member function, `multiplyRegisters()`, which will multiply the value of the two registers, `register1` and `register2`, and assigns the product to `registerResult`. This function has no values passed to it and has a void return type.
5. You will need to create the member function, `getInputValues()`, which should prompt the user for the inputs in the following way:

   ```
   Please input the value of register1:
   Please input the value of register2:
   ```

   This function has no values passed to it and has a void return type.
6. You will need to create the member function, `displayOutputValues()`, which displays the results of the calculation as seen below. This output example below assumes that the contents of `register1` is 3 and the contents of `register2` is 4. This function has no values passed to it and has a void return type.

   ```
   ***Leibniz Wheel Result: 3 * 4 = 12
   ```

Part 4: ***(0%) Basic Grading Program***

In Appendix D, contains some simple code to compare two text files. We talked about some basic file I/O of text files in class, and this is a simple example of using text file manipulation to quickly check your results. We will provide you with the input files and output files that you can use for each section. For example, when you run your code at the prompt you can use "I/O redirection" in linux to automatically populate input requests and dump terminal output into a output file. Using the input.dat provided to you, at the command prompt type the following:

```
./part1 < input.dat > answer1.dat
```

Appendix A: Accessing Deepthought and Submitting Your Code

ACCESSING LINUX DEEPTHOUGHT CLUSTER (SERVER)

The CoC web page discussing this "deepthought" cluster is:

```
https://support.cc.gatech.edu/facilities/instructional-labs/deepthought-cluster
```

However to access this cluster you need certain software on your laptop or desktop system, as described below.

Linux or Mac OS X. If you use either Linux (any variation) or Mac OSX (any recent variation), you already have the necessary software. Both Linux and OSX include the "Secure Shell" (ssh) program, as well as the *X-Windows* graphical server that we will use for later assignments.

Windows. The Georgia Tech Office of Information Technology (*OIT)* maintains a web page of software that can be downloaded and installed by students and faculty. That web page is:

```
http://software.oit.gatech.edu
```

From that page you will need to install either X-Win32 2012 (with SSH) or SecureCRT 7.2.3. The first is recommended as it includes the X-Windows server we will need later, but SecureCRT will be easier to install and will suffice for this assignment.

To access this software you will first have to log in with your Georgia Tech user name and password, then answer a series of questions regarding export controls.

GTL Windows (PuTTY option): If you are having trouble accessing OIT website. You can also use PuTTY. You can read the Wikipedia page on PuTTY at https://en.wikipedia.org/wiki/PuTTY. The download for the program is http://www.chiark.greenend.org.uk/~sgtatham/putty .

Using SSH. Once you have the proper software, simply open a terminal window (the procedure for this differs based on OS and software package), and connect to deepthought as follows :

```
ssh -l your-user-name-here deepthought-login.cc.gatech.edu
```

Note, some of the windows packages automatically connect for you and simply prompt for the password.

After successfully logging in, you can use any of the Linux commands to create directories, list files, edit source code, and build your programs. There are plenty of on-line resources about using Linux. At a minimum you will need ls, ls -la, mkdir, cd, make.

CREATING AND SUBMITTING LAB1 PROGRAM ON DEEPTHOUGHT

Creating Source Code. Once logged into deepthought, you will need to use one of several available text editors found on the deepthought cluster. Those are vi, vim, pico and emacs.

Basic Unix Commands. Please make a directory (i.e. "folder") called `Lab1` where you keep your files. To make this directory use the following command in your home directory:

```
mkdir Lab1
```

To go into this directory from your home directory, you can use the following command:

```
cd Lab1
```

To get back to your home directory you can use the command:

```
cd
```

Compiling Source Code. On the deepthought system we will be using the gnu g++ compiler. At the command prompt in your `Labl` directory use the following command to compile your source code and create an exectuable file called `partl`.

```
g++ labl_partl.cc -o partl
```

To run your program at the command prompt, type in the executable file name

```
./partl
```

Submitting Assignment on Deepthought. The system administrator for the deepthought cluster has created a script that you are to use to turn in your project. The scripts are found in /usr/local/bin, which should be in the search path for everyone. Depending on your teacher, from your home directory enter one of the following at the commands at your prompt:

```
davis-turnin Labl
```

or

```
riley-turnin Labl
```

This automatically copies everything in your `Labl` directory to a place that we can access (and grade) it.

APPENDIX B: Sample Input (Bolded) and Output for Part 1 (We will test your program with a level of 0, 1, 10 and 20)

Please input the size of the triangle: **10**

```
n =  0:  1
n =  1:  1   1
n =  2:  1   2   1
n =  3:  1   3   3   1
n =  4:  1   4   6   4   1
n =  5:  1   5   10  10  5   1
n =  6:  1   6   15  20  15  6   1
n =  7:  1   7   21  35  35  21  7   1
n =  8:  1   8   28  56  70  56  28  8   1
n =  9:  1   9   36  84  126 126 84  36  9   1
n = 10:  1   10  45  120 210 252 210 120 45  10  1
```

APPENDIX C: Partial Skeleton Code for Part 2

```
#include <iostream>
// Missing line
class Pascaline
{
public:

  //Missing line

  //member functions prototypes
  void setRegister1(int);
  void setRegister2(int);
  void setRegisterResult(int);
  int getRegister1();
  int getRegister2();
  int getRegisterResult();
  void clearRegisters();
  void addRegisters();
  void getInputValues();
  void displayOutputValue();

private:
   int register1;
// Missing line of code
  int registerResult;

//Could be syntax error around here!
} //end class Pascaline

//----------------------------------------
//---Implementation of the member functions
//----------------------------------------

//Insert Constructor implementation here

//member function to set register1
void Pascaline::setRegister1( int reg1)
{
//Missing code
}

//member function to set register2
void Pascaline::setRegister2( int reg2)
{
//Missing code
}

//member function to set resultRegister
void Pascaline::setRegisterResult( int regResult)
{
```

```
  registerResult = regResult;
}

//member function to get the register1 value
int Pascaline::getRegister1()
{
 //missing code
}

//member function to get the register2 value
int Pascaline::getRegister2()
{
  return(register2);
}

//member function to get the resultRegister value
int Pascaline::getRegisterResult()
{
//Missing line of code
}

//Implement clearRegisters() here
//It sets the value of all the registers equal to zero

//Implement addRegisters() here
//It will add register1 and register2 and put result in registerResult

//Implement getInputValues() here
//It will get the input data from the user

//Implement displayOutputValue() here
// It will display the output register and computation to the user

int main()
{
  //Instantiate your main object here
  Pascaline myPascaline;
  char input;

  //Now call the public member functions to use the object
  do {
  myPascaline.getInputValues();
  myPascaline.addRegisters();
  // Missing code

  // Missing code for formating to appear in Appendix C
  // Trying using an escape sequence in the string
  cout >> "Would you like to calculate again (Y or N)? ";
  cin << input;
  } while (input == 'Y');
  return 0;
}
```

APPENDIX D: Sample Input (Bold) and Output for Part 2 & 3

Sample Sessions Part 2:

Please input the value of register1: **5**
Please input the value of register2: **6**
***Pascaline Result: 5 + 6 = 11

Would you like to calculate again (Y or N)? **Y**

Please input the value of register1: **-5**
Please input the value of register2: **3**
***Pascaline Result: -5 + 3 = -2

Would you like to calculate again (Y or N)? **Y**

Please input the value of register1: **-19**
Please input the value of register2: **-9**
***Pascaline Result: -19 + -9 = -28

Would you like to calculate again (Y or N)? **Y**

Please input the value of register1: **22**
Please input the value of register2: **-12**
***Pascaline Result: 22 + -12 = 10

Would you like to calculate again (Y or N)? **N**

Sample Session Part 3:

Would you like to use the Leibniz Wheel (enter 'L') or the Pascaline (enter 'P')? **L**
Please input the value of register 1: **5**
Please input the value of register 2: **6**
***Leibniz Wheel Result: 5 * 6 = 30

Would you like to calculate again (Y or N)? **Y**

Would you like to use the Leibniz Wheel (enter 'L') or the Pascaline (enter 'P')? **P**
Please input the value of register 1: **-10**
Please input the value of register 2: **8**
***Pascaline Result: -10 + 8 = -2

Would you like to calculate again (Y or N)? **Y**

Would you like to use the Leibniz Wheel (enter 'L') or the Pascaline (enter 'P')? **F**
I am not sure what you want!

Would you like to calculate again (Y or N)? **Y**

Would you like to use the Leibniz Wheel (enter 'L') or the Pascaline (enter 'P')? **L**
Please input the value of register 1: **-9**
Please input the value of register 2: **-8**
***Leibniz Wheel Result: -9 * -8 = 72

Would you like to calculate again (Y or N)? **N**

APPENDIX E: Simple Grading Program in C++

(Please note that white space variations between files are ignored with this program)

```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char * * argv)
{
if (argc > 2)
{
  ifstream key(argv[1], ios::in);
  ifstream outputAnswer(argv[2],ios::in);
  char input_key;
  char input_answer;
  bool GETOUT =false, KEY_GETOUT =false, ANSWER_GETOUT= false, ERROR = false;
  int characterPosition = 0;

  while (!GETOUT)
  {
     if (key) //is valid then read in value
     {
       key >> input_key;
       characterPosition++;
     }
     else //file is not there or file is ended
     {
       KEY_GETOUT = true;
       GETOUT = true;
     }

     if (outputAnswer) //is valid then read in value
     {
       outputAnswer >> input_answer;
     }
     else //file is not there or file is ended
     {
       ANSWER_GETOUT = true;
       GETOUT = true;
     }

     if ((input_key != input_answer)||(KEY_GETOUT != ANSWER_GETOUT))
     //this also produces an error if files are not of the same length
     { cout << "You have an error at character position " << characterPosition;
       cout << ":" <<  input_answer << " - please try again" << endl;
       ERROR = true; GETOUT = true;
     }
  }

  if (!ERROR)
  {   cout << "Good job! You have perfect output!" << endl; }

  key.close();
  outputAnswer.close();
}
else
{
  cout << "You must input two filenames to compare!" << endl;
}

}//end main
```