DUE DATE:

Friday, Nov 15, 2013, 11:59 pm

REQUIREMENTS

Project # 2

~~Using your aRandomNumber Generator class designed in project #1,~~

~~use inheritance to design two new classes. Those classes are to be~~

~~called aDie and aCoin. They are to simulate a die and a coin. The die~~

~~should produce a random number between 1 and 6 when tossed. The~~

~~coin should produce a random number that is 0 for heads and 1 for~~

~~tails when flipped.~~

~~The aDie class will have a method called roll(). The prototype for~~

~~Roll() is: int aDie::roll()~~

~~The aCoin class will have a method called flip(). The prototype for~~

~~flip() is: int aCoin::flip()~~

.

It is up to you to determine whether those methods should be defined

as const or not.

Your project will also design a histogram class. We will call that class

aHistogram.

For the histogram to know how many bins it might need you will

provide it with a method that specifies the range of expected numbers.

The prototype of this method will be: void **setRange(int low,**

**int high)** where low represents the lowest number to be expected and high

represents the highest number to be expected.

~~Your aRandomNumber Generator class will now have to provide two~~

~~methods that will specify the low and high numbers generated by the a~~

~~class.~~

~~The prototypes for those two methods are:~~

~~int aRandomNumberGenerator::getLowNumber()~~

~~int aRandomNumberGenerator::getHighNumber()~~

~~Again, it is up to you to determine whether those methods should be~~

~~defined as const or not.~~

All classes in the project should have proper constructors and destructors.

Your previous aRandomNumberGenerator class generates a random number

through a method called **generate()** with prototype:

int aRandomNumberGenerator::generate()

Determine its constness (const or not?)

The aHistogram class should see its range of potential numbers set

through the setRange(int low, int high) method discussed above and

should also update the appropriate bin count through a method called

**update()** with prototype:

void aHistogram::update(int number) where the argument number represents

the outcome of a call to aRandomNumberGenerator::generate().

The histogram class should also have a **display()** method to display its information.

The histogram class should also have a **clear()** method to clear all the

bins

void aHistogram::clear();

In your main program, you will create one die and one coin. You will

ask the user how many times s/he wishes to roll the die and how

many times s/he will flip the coin.

With that information in hand you will roll the die and flip the coin

the appropriate number of times and then use two separate histogram

objects to display the histograms of the die rolls and coin flips.

The graphical display of the histograms will be as you did in project #1

The histogram display should show the count for each possible outcome and then

a graph in the form of lines of ‘x’.

For example, assuming 1000 flips, the histogram for the die should look something like this:

Heads: xxxxxxxxxxxxxx….xx (490)

Tails : xxxxxxxxxxxxxx…xx (510)

.

.

Meaning that Heads appeared 490 times and Tails appeared 510 times

(your mileage may vary)

.

~~Please note that this histogram does not use~~

~~1,2,...6 as it does~~

~~for the die, but instead uses the full terms HEAD and~~

~~TAIL~~

Again, in the histogram displays ‘x’ represents some number of occurrences

of a particular face or coin side. Since you are likely to get 100’s of occurrences of a face or coin side, you will not be able to use

one x for one occurrence. Therefore you will have to scale your x to

represent some number of occurrences for each of the histogram.

The recommendation made for project #1 was that you find out for the

die and for the coin what the largest count was and then divide that

count by 50 (or something like that) so that each of the largest bin count

would be represented by a line of 50 x’s where each x would represent

1/50 th of the largest bin count.

This leads to the suggestion that your aHistogra

m class should provide

a method called **count(int randomNumber)** with prototype:

int aHistogram::count(int randomNumber)

Again, you decide on the constness of this method.

This method would return the number of occurrences of a particular

random number. Of course this would mean that you must make sure that given a request for the number of occurrences of a particular

number, you will have to make sure that you return the count for the

appropriate bin. In other words, the bin whose count you will return is

not necessarily the bin whose index is the random number you passed

as an argument.

~~Finally, since some of you seemed to have made that mistake, you~~

~~need to seed the random number ONLY ONCE!~~

**This means that you should have main ask the user**

**to provide a seed.** ~~In this way, you can always repeat a simulation run should something go wrong. This will also allow the T.A. to test your application under different conditions.~~

Guidelines:

You will need to make sure to use data hiding principles. Make sure

you use Public, Protected and Private access rights appropriately.

Not using data hiding principles will result in a 10% penalty

~~Make sure each class is declared and defined in separate header and~~

~~source files.~~

~~Not defining classes in separate files will result in a 50% penalty.~~

Make sure you have appropriate **constructors** and destructors.

Not having the appropriate constructors will result in a 10% penalty

~~Do not use a switch statement to update the counts in the histogram!~~

~~Using a switch statement in the updating method of your histogram~~

~~will result in a 30% penalty.~~

.

You will need to make sure you have selected the appropriate

constness for your methods. Not using the correct const attributes will

cost you 10% in penalty.

~~You MUST use inheritance in designing the aDie and aCoin classes. If~~

~~you do not use inheritance you will automatically receive a grade of 0.~~

Your code MUST compile! If your code doesn’t compile you will

automatically receive a grade of 0.

Your executable MUST run. If not, you will automatically receive a

grade of 0.

If your code doesn’t display the histogram and counts appropriately

you will suffer a penalty of at least 50% depending on the cause of the

error.

You will submit your project through the digital drop box feature of

blackboard.

You will clean (Build -> Clean) the project (to minimize its size) , zip your project and submit the zipped file. Please name your zip file by lastname1\_lastname2\_2.zip for each group.

.

Each group only needs to submit one project file.

If you have any questions, please ask them!