EECS402, Winter 2013, Project 2

Overview:
While histograms are a relatively simple construct, they are often very useful in measuring and visualizing the distribution of data contained within a sample set. For this project, you will implement a C++ program allowing a user to input values of sample sets, and then create histograms that are also defined by user input, such as the minimum value, maximum value, and number of bins.

All necessary input will come from the user, via the keyboard. The complete specifications for this project are given below.

Due Date and Submitting:
This project is due on **Friday, February 15 at 5:30pm**. Late submissions will not be accepted for any reason.

For this project, your solution must be fully contained in exactly one file. The filename should be your uniqname, followed by the number 2, with the extension cpp. Therefore, my submission would be named morgana2.cpp.

Detailed Description:
At this point in the course, you have been given an overview of basic object-oriented principles, and this project will be the first program in the course using the object-oriented property of encapsulation using C++ classes.

Two C++ classes will be required for the implementation of this project. The first class will contain information describing a sample set of data as well as functions that operate on that data. The name of this class must be "SamplingClass". Required attributes (data members) include: a single character identifier that is simply a way for the user to refer to a sample set of data (i.e. the data from set 'A', or the data from set '6', etc); an integer indicating the number of values that are in the sample set; and an array of double precision floating point values that will contain the actual sample values. In addition to the required functionality below, you may choose to implement additional member functions that are appropriate for your design.

Required functionality within the **SamplingClass** include the following **member functions**, which must be implemented using exactly the names and parameters as shown:

```
//This function reads attributes describing a SamplingClass object
//from standard input (i.e. "cin"). The user will be prompted for a
//character identifier, then will be allowed to enter as many
//floating point values as desired (up to the maximum allowed by
//the program) and will indicate completion by entering the special
//value "-999", at which point data value entry will cease, and
//this last special value will NOT be included in the sample set.
//The function returns true upon success, and false on failure.
bool readFromKeyboard();

//This function prints out the attributes of a SamplingClass object
//to the screen, including the identifying character, the total
//number of samples in the sample set, and each individual data
//value. Multiple data values will be printed on each line – the
//number per line will be defined in a global constant, with the
```

```
//possible obvious exception of the last line printed if the number
//of samples doesn't evenly divide the number to be printed per
//line. The function returns true for ANY sample set for which the
//user has entered at least an identifying character, false when
//called to print a sample set that the user has not initialized.
bool printToScreen();
```

The second required class will contain information describing a histogram as well as functionality that can be performed on or with a histogram. The name of this class must be "HistogramClass". Required attributes of this class include: a double precision floating value representing the minimum value of the range of data a user wishes to be stored in specified histogram bins; a double precision floating point value representing the maximum value of the range of data a user wishes to be stored in specified histogram bins; an integer representing the number of bins the range is to be separated into; and an array of integer values that contain the bin counts for the histogram object. In addition to the required functionality below, you may choose to implement additional member functions that are appropriate for your design.

Required functionality within the **HistogramClass** include the following **member functions**, which must be implemented using exactly the names and parameters as shown:

```
//This function allows the user to setup a histogram by reading
//values for the minimum bin value, maximum bin value, and the
//number of bins, which can be up to the maximum number of bins
//allowed by the program, which will be stored in an appropriately
//named global constant. The function returns true when the user
//enters valid values supported by the program, false otherwise.
bool setupHistogram();

//This function adds counts to a previously set-up histogram's bins
//appropriately. Data values are taken from the sampling parameter,
//and one bin count is incremented for each value in the sampling.
//Note: This function adds data to a histogram, and does NOT re-
//initialize the counts to zero when called! If a user wishes to
//start a new histogram with new counts, then setupHistogram() must
//be called. The function returns true on success, and false
//otherwise. One possible failure would be if the necessary objects
//haven't been setup/initialized by the user.
bool addDataToHistogram(SamplingClass &sampling);

//This function simply prints out the bin counts (and percentages)
//for each bin in the previously set up histogram. Specific output
//format can be seen in example outputs. Returns true on success,
//false otherwise.
bool printHistogramCounts();

//This function displays a graph-like representation of the
//histogram contents, allowing users to easily see how the sample
//data values are distributed in the bin range they have set up.
//Each bin will be indicated by a series of bars (equal-sign
//characters), one bar for each 2%. In other words, if a bin
//contains 12.5% of the total count in all bins, that bin would be
//displayed with 6 bars. Specific output format can be seen in
```

```
//example outputs. Returns true on success, false otherwise.
bool displayHistogram();
```

Often, users are interested in placing data values into histogram bins without being too concerned with outliers less than a minimum value or greater than a maximum value. Rather than just disregard these outliers, though, your program must maintain two histogram bins in addition to the user-defined bins: one for outliers that are less than the minimum value specified and one for outliers that are greater than the maximum specified value.

Floating Point Numbers, Computers, and Binning:
Floating point values (both float and double types) can act weird on computers. We'll discuss "floating point error" later in the course, but for now, to avoid any weirdness, you will need to take a special precaution in your solution when binning samples into the histogram. Rather than just checking if a sample is greater than a bin's calculated minimum value, you will check if the sample is greater than a bin's calculated value *plus a small tolerance*. This tolerance value should be maintained as a global constant, and will be set to 0.0000005. What this means is that instead of doing something like this:

```
if (val > binMinVal)
```

you would do this:

```
if (val > (binMinVal + HISTO_PRECISION)
```

where HISTO_PRECISION is a constant double set to 0.0000005.

User Interface:
The user interface for this program will be a simple menu-driven interface. The menu that will be presented to the user will look as follows:

```
1. Enter a sample set of data values
2. Print the contents of the current sample set
3. Reset / Provide values for setting up a histogram
4. Add the contents of current sample set to histogram
5. Print bin counts contained in histogram
6. View the histogram in graphical form
0: Exit the program
Your Choice:
```

Notice that each of these menu options correspond to required functionality described for the classes above.

After each operation completes, you will indicate whether the operation was successful or not, so that the user can be sure things went as he or she intended without having to assume things worked. Again, see the sample output to see how this is indicated.

Global Functions:
There is only one required global function for this project, which is described below. If you wish and it is appropriate for your design, you may choose to implement additional global functions.

```
//This function will present the program menu to the user, and
//request user input from standard input (i.e. "cin"). If the
//user enters an invalid choice, an error message is printed,
//and the menu is re-printed and the user is re-prompted. This
//continues until the user finally enters a valid choice, and
//that choice will be returned from the function.
int promptUserAndGetChoice();
```

Global Constants:
The use of global constants is required. Since global constants cannot be changed after initialization, they can and should be used as often as necessary. Some items that must be declared and used throughout your program as global constants include: the maximum size of the data set within a SamplingClass object (for this project, you should set this to 100); the maximum number of user bins within a histogram (for this project, you should set this to 100); the precision value to be used to partition actual histogram bin boundaries (for this project, set this to 0.0000005); the number of data set values to be printed on each line when the user requests a SamplingClass data set to be printed (for this project, set this to 5).

You should realize that I may change these values of these constants, re-compile your program, and ensure that it still works correctly with the new constant values, so you should perform extensive tests of this sort as well.

Additional Specifications / Information:
The program must exit "gracefully", by reaching the ONE return statement that should be the last statement in the main function - do not use the "exit()" function.

You may only access (read from OR write to) member variables of classes by using member functions of the same class. For example, any function that is not a member function of SamplingClass is not allowed to directly access any member variable (attribute) of a SamplingClass object – these attributes may only be accessed by calling one of the SamplingClass member functions. Enforce this by making all data member variables private in your implementation.

Any function parameters of a class type must be passed by reference. Parameters of type int, double, char, etc should be passed by value when appropriate, but all class type (HistogramClass, etc) parameters must be passed by reference.
The only library you may #include is <iostream>. No other header files may be included, and you may not make any call to any function in any other library (even if your IDE allows you to call the function without #include'ing the appropriate header file). Be careful not to use any function located in <cmath>.

Special Testing Requirement:
In order to allow me to easily test your program using my own main() function in place of yours, some special preprocessor directives will be required. Immediately before the definition of the main() function, (but after ***all*** other prior source code), include the following lines EXACTLY:

```
#ifdef ANDREW_TEST
#include "andrewTest.h"
#else
```

and immediately following the main() function, include the following line EXACTLY:

```
#endif
```

Therefore, your source code should look as follows:

```
library includes
program header
constant declarations and initializations
global function prototypes
class definitions
#ifdef ANDREW_TEST
#include "andrewTest.h"
#else
int main()
{
  implementation of main function
}
#endif
```
global function definitions
class member function definitions

Lines above in red are to be used exactly as shown. Other lines simply represent the location of those items within the source code file.

Error Checking Requirements:
You need not worry about data type related error checking in this program. You may assume that the user will enter data of the correct data type when prompted. Note: This is usually a horrible assumption. As you learn error handling techniques in this class, you will be required to use them, but for this project, just concentrate on implementing the functions as described.

Other error checking is required, however. Since you are making full use of arrays of a set size, you must assure that the user is never allowed to reference values outside the valid range of indices of any array. There are other problems that could come up, and you are expected to handle them appropriately. I will not list all potential errors that you must check for, but remember that you must determine what could go wrong and handle it preventing serious problems in the program.

Design and Implementation Details:
Remember that you are allowed to implement additional member functions and/or global functions as appropriate for your design. In my solution, I implemented a few additional member functions for both of the required classes that came in handy because I needed to perform the same functionality multiple times, so these additional functions minimized the amount of duplicate code in places. Your design is up to you – but it should be a logical and quality design.

A few things of note that you do *not* have to do follow. You do not need to keep track of multiple histograms or sample sets at any given time (your main function should only have one SamplingClass and one HistogramClass declared). Also you do not need to keep your data sorted – samples should be kept in the order they were input by the user.

As in project 1, implement this program in a piece-wise fashion starting with the basics and adding functionality as you go. You can certainly implement the entire SampleClass without ever even beginning the histogram class, and I would recommend doing so.

"Specific Specifications"
These "specific specifications" are meant to state whether or not something is allowed. A "no" means you definitely may NOT use that item. In general, you can assume that you should not be using anything that has not yet been covered in lecture (as of the first posting of the project).

- Use of Goto: No
- Global Variables / Objects: No
- Global Functions: Yes (as necessary)
- Use of Friend Functions / Classes: No
- Use of Structs: No
- Use of Classes: Yes – required!
- Public Data In Classes: **No** (all data members must be private)
- Use of Inheritance / Polymorphism: No
- Use of Arrays: Yes – required!
- Use of C++ "string" Type: No
- Use of C-Strings: No
- Use of Pointers: No
- Use of STL Containers: No
- Use of Makefile / User-Defined Header Files / Multiple Source Code Files: No
- Use of exit(): No
- Use of overloaded operators: No
- Use of float type: **No** (That is, all floating point values should be type double, not float)

Sample Program Output:
Sample outputs are provided on the course website that will allow you to see what is expected and to compare your program's output to. Important Note: The sample outputs show you what values are expected for a very limited number of cases. If your program produces the exact same output, that does NOT mean that you are finished. There are many cases that were not tested in the above output, including many error conditions that will be tested after submission.