EECS402 Winter 2017 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.

Submission and Due Date

You will submit your program using an email-based submission system by attaching *one C++ source file* only (named exactly "project2.cpp"). Do not attach any other files with your submission – specifically, do *not* include your compiled executable, etc. The due date for the project is **Friday, February 17, 2017 at 4:00pm**. Early submission bonus and late submission penalties will be applied as described in the course syllabus.

No submissions will be accepted after the late submission final deadline. As discussed in lecture, please double check that you have submitted the correct file (the correct version of your *source code* file named exactly as specified above). Submission of the "wrong file" will not be grounds for an "extension" or late submission of the correct file, and will result in a score of 0.

A Little About Histograms

A histogram is a construct that considers a set of values and provides counts of the number of times those values fall within specified ranges. For a histogram, these ranges have equal extents and are typically referred to as "bins". As an example, let's say you have a set of data corresponding to the total number of runs your team scored in 16 baseball games:

- 5, 4, 4, 7, 0, 2, 0, 4, 1, 3, 6, 4, 5, 3, 12, 4

That is an interesting set of data, but, especially as the size of the data set gets bigger, its not especially easy to interpret statistically. A histogram could be used to "bin" up those values and be presented as a graph to answer a question like, "How often do we score >= 2 runs but less than 4? How often do we score >= 4, but less than 6?" Let's say we know we typically score between 2 and 8 runs - for this example, we'll make a histogram of values from 2 to 8, with a bin size of 2 (that is, each "bin" contains counts of two possible values).

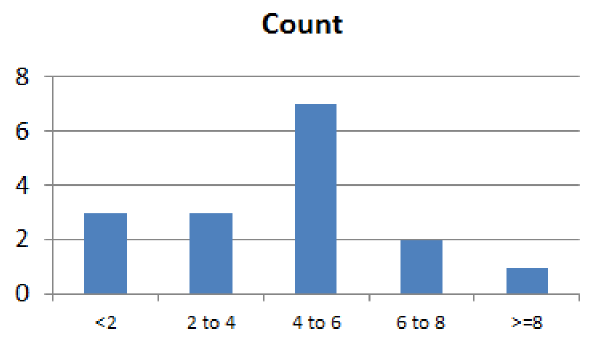
- Bin: >=2 and <4 Count: 3 (values 2, 3, 3)
- Bin: >=4 and <6 Count: 7 (values 5, 4, 4, 4, 4, 5, 4)
- Bin: >=6 and <8 Count: 2 (values 7, 6)
- Bin: Outliers <2 Count: 3 (values 0, 0, 1)

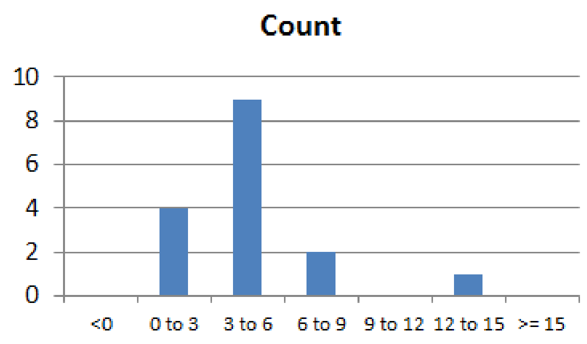
- Bin: Outliers >=8 Count: 1   (value: 12)

Of course, we could set up our histogram so each bin only held 1 possible value (bin size==1), or increase the range we consider non-outliers – for example, we could make a histogram with a bin size of 3 that ranges from 0 (inclusive) to 15 (exclusive). In this case, we'd have:

- Bin: >=0 and <3 Count: 4
- Bin: >=3 and <6 Count: 9
- Bin: >=6 and <9 Count: 2
- Bin: >=9 and <12 Count: 0
- Bin: >=12 and <15 Count: 1
- Bin: Outliers <0 Count: 0
- Bin: Outliers >=15   Count: 0

Typically, the real power of histograms come with visualization which usually means forming a chart for very easy visual interpretation of frequency. Here are example charts for the two histograms described above:





Detailed Description

At this point in the course, you have been given an overview of basic object-oriented principles, and this project will be 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". The second class will contain information describing a histogram that contains a number of bins, each of which contains a count of the number of samples from a sampling class object that fall within the bin. This name of this class must be “HistogramClass”.

While we will have a list of required member variables and member functions of both of these classes, *you may choose to implement additional member functions* and/or *include additional member variables* that are appropriate for your design.

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 integer values that will contain the actual sample values. In order for us to grade your project, there are some required member functions that you must implement exactly according to this specification.

```
SamplingClass()
```

An explicit default constructor (ctor). This ctor simply performs some functionality to put the sampling class in a state that you can use to recognize that it has not been properly set up with some form of input. After being created with this ctor, the sampling will not have any samples stored, since none are provided in the ctor parameter list.

```
SamplingClass(const char inIdChar,
              const int inNumSamples,
              const int inputSamples[])
```

A value ctor that allows the user to specify initial data that describes the new sampling object. The new sampling object's member variables (attributes) are set to the corresponding values provided as input to the ctor. However, if "inNumSamples" is not an allowed number of samples supported by this class, this function will print an informative error message, and the newly generated sampling object will be set as described in the explicit default ctor description. If "inNumSamples" is acceptable, the new sampling object's attributes will be set according to the values provided as input to the ctor.

```
bool readFromKeyboard()
```

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 integer values as desired (up to the maximum allowed by the program). The user will indicate they are done entering samples by entering the special value "-99999", at which point data value entry will cease, and this last special value will NOT be included in the sample set. This function is considered to "fail" if the user attempts to enter more than the allowed number of samples. Upon failure, an informative error message is displayed, and the function will return false – otherwise, the function will return true, indicating success.

```
bool printToScreen()
```

This function will print the current state of a sampling class object to the screen (i.e. "cout"). If the sampling object it is called on has been initialized, either directly, by a value ctor, or by reading data from stdin via the user, then the function will be considered successful and return true, otherwise it will print "ERROR: Can not print uninitialized sampling!" and return false (i.e. if the default ctor was used to create it, but wasn't modified in any way after that). When successful, 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. Each line will print that number of individual samples, 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. See the sample output for details about output formatting and such.

HistogramClass

Required attributes of this class include: an integer value representing the minimum value of the range of data a user wishes to be stored in specified histogram bins; an integer 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.

For this project, all of our samples will be integer values. Floating point values (both of type float and double) can act weird in computers and we'll discuss floating point error briefly later, but to avoid this for our first project, we will only store integers in our samplings and histograms. To further avoid the issue of floating point error, there is an additional requirement that the number of bins requested by the user must evenly divide the range of the histogram. In other words: If you have a histogram that stored values from 2 to 19, then the range of values spans 18 integers (since both endpoints are inclusive). If the user requests 9 histogram bins, that is acceptable since 9 divides into 18 evenly. If the user requests 7 histogram bins, though, that will be an error, as 18/7=2.571. Due to this restriction, your integer values will always be able to be placed in a specific histogram bin. In the example above (min=2, max=19, num bins = 9), each bin will hold counts of 2 integer values. The first bin will hold counts of the sample values of 2 and 3, the second bin will hold counts of 4 and 5, etc., up to the last bin holding counts for A18 and 19. Values less than 2 or greater than 19 are considered outliers in this case.

Often, users are interested in placing sample 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.

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

`HistogramClass()`
An explicit default constructor (ctor) that will ensure that all newly generated HistogramClass objects will be initialized to known state of not having been loaded with data.

`bool setupHistogram()`
This function allows the user to setup a histogram by obtaining 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, as specified in an appropriately named global constant. Note that the input values must be prompted for in this particular order (min value, max value, number of bins). All values are obtained from the user interactively using the keyboard (i.e. cin). The function returns true when the user enters valid values supported by the program, or false otherwise.

`bool addDataToHistogram(SamplingClass &sampling)`
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. In other words, the user could add multiple samplings to the same histogram by simply calling this method multiple times without calling setupHistogram in between. 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. Note: ideally, we would like to pass the “sampling” parameter by constant reference, but there’s an issue with that which we haven’t discussed yet, so for this project, leave it as non-const.

`bool printHistogramCounts()`

This function simply prints out the bin counts and percentages (as floating point values) for each bin in this previously set up histogram object. Specific output format can be seen in example outputs. Returns true on success, false otherwise.

`bool displayHistogram()`

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.

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.

`int promptUserAndGetChoice()`

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.

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 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); and the number that the user should enter to indicate they are done entering sample values (for this project, set this to -99999).

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 or multiple return statements in main.

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. You **must** 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 built-in types like int, double, char, etc should be passed by value when the function is not expected to potentially change the value, 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
global constant declarations and initializations
global function prototypes
class definitions (with prototypes only for member functions)

#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. Realize this exception is that you don't need to worry about data type (i.e. the user enters a character when your program expected them to enter an integer, etc).

Other error checking is required, however. Since you are making full use of arrays of a set size, you must ensure 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.

I suggest you implement this program (and all programs, really) in a piece-wise fashion starting with the basics and adding functionality as you go. You can certainly implement the entire SamplingClass (one function at a time, or a few statements at a time) without ever even beginning the histogram class, and I would recommend doing so.

Sample Outputs

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 provided output, including many error conditions that will be tested during grading.