Table of Contents

Customer Requirements: 2

Development Plan: 2

High Level Design Description: 2

Low Level Requirements: 3

- **Interval Wrapper (class IntervalCls<T>)** 3
- **IO Manager (class IOManagerBaseCls<T>):** 3
- **File Based IO Manager (class FileIOManagerCls : IOManagerBaseCls)** 4
- **Histogram Manager (class HistogramManager<T>):** 5
- **Histogram Application (class HistogramApplicationBaseCls)** 5
- **File Based Histogram Application (class FileHistogramApplicationCls : HistogramApplicationBaseCls)** 6
- **Error Logger (class ErrorLoggerCls)** 6

API USAGE 6

- **Build/Make** 6
- **Using the application as a Library** 7
- **Extending the Implementation** 7

Future Work 7

Customer Requirements:

Implement a program that defines a class **Histogram** that keeps a count of numbers in some intervals specified as constructor arguments (the intervals are of the form [3, 5.5), [7, 9.3), in other words, closed on the left and open on the right and the intervals might have gaps, but won't overlap, and are unsorted) to Histogram's constructor. The class must handle out-of-range values as well as flag potential input errors in the interval definitions.

This class must provide methods to print out the histogram and be template-ized (or use generics, depending on the programming language you choose) such that it can be used with either "floats" or "doubles".

(1) it must be an end to end program, i.e., and executable that reads the intervals and numbers either from a file or form the command line and printout the resulting histogram and outliers if any
(2) it must include a (GNU) make file (supporting "make", "make clean", "make test") or an equivalent build infrastructure if using Java/C# (e.g., ant's build.xml)
(3) it must include tests (normal input as well as input with failures)
(4) assume this is production-grade, so formatting, error handling (on the input data and elsewhere where needed), and documentation are all important

Development Plan:

- High level requirements *(customer requirements will be used)*
- High level design
- Low level requirements / Detailed design
- Test scenarios
- Test harness (test cases and input sets)
- Implementation
- Build Environment (makefiles)
- Test Execution
- Submission

High Level Design Description:

1 - There will be three main modules: IO Manager, Histogram Manager and Application
2 - IO Manager will be the base class to be used to get input from the user and provide output
3 - IO Manager will have methods for input validation
4 - IO Manager will provide virtual functions for getting and outputting data
6 - Histogram Manager will provide a mechanism for defining upper-lower bounds
7 - Histogram Manager will be responsible for range checking inputs
8 - Histogram Manager will provide a mechanism for storing the counts of entries in given intervals
9 - Histogram Manager will store out of bounds items for printing as outliers.
10 - Application will manage the connection between IO Manager and Histogram manager.
11 - Application will use IO Manager to get the interval and entry data and will feed Histogram Manager with this data.
12 - Application will use Histogram Manager to generate required output.
13 - Caught errors will be logged using an Error Reporter class

Low Level Requirements:

Interval Wrapper (class IntervalCls<T>)

IW-LLR-1- Interval Class will contain lower and upper boundaries and the count of entries that fall into that range (count field will be initialized to 0).

IW-LLR-2- Interval Class will be a template class and it will work on same type template with Histogram Manager

IO Manager (class IOManagerBaseCls<T>):

IOM-LLR-1- IO Manager will be a template based class. It will work on same type template with Histogram Manager. The template type will be used for input validation.

IOM-LLR-2- IO Manager will have a protected constructor. This will ensure that the derived classes will implement their own constructors which can be used to provide required parameters for that derived implementation

IOM-LLR-3- IO Manager will provide a method to decide on which kind of data is currently being read (Alternative method could be getting the input and deciding if it is an interval or entry):

```
virtual bool IsReadingIntervals()
```

IOM-LLR-4- IO Manager will provide a method to get Histogram boundaries from user:

```
virtual IntervalCls ReadInterval()
```

IOM-LLR-5- ReadInterval function will return lower and upper bounds in the template type

IOM-LLR-6- ReadInterval function will throw an InvalidArgument exception if the input cannot be parsed in the template type.

IOM-LLR-7- IO Manager will provide a method to get Histogram entries from user: ReadEntry

```
virtual bool ReadEntry(T&)
```

IOM-LLR-8- ReadEntry will return the histogram entry in the template type

IOM-LLR-9- ReadEntry will throw an InvalidArgument exception if the entry cannot be parsed in the template type.

IOM-LLR-10- ReadEntry will return true if the input could be read and parsed, false otherwise.

IOM-LLR-11- IO Manager will provide a method to initialize getting input: Initialize
virtual bool Initialize()

IOM-LLR-12- IO Manager will provide a method to check if input reception has ended
virtual bool IsEndOfInput()

IOM-LLR-13- IO Manager will provide a method to output the histogram
virtual void PrintHistogram(HistogramManager)

File Based IO Manager (class FileIOManagerCls : IOManagerBaseCls)

FBUIM-LLR-1- This class will inherit from IOManagerBaseCls.

FBUIM-LLR-2- This class will be used to read files with format with the following format:

```
Intervals:
<number>  <number>
<number>  <number>
…
Entries:
<number>
<number>
…
```

FBUIM-LLR-3- This class will have a constructor with a parameter, giving the name of the file to be used as input
FileIOManagerCls(char*)

FBUIM-LLR-4- Initialize function will be overridden to:

a. Open the file whose path is provided in constructor
b. Throw an exception if file cannot be opened for reading

FBUIM-LLR-5- A Boolean isReadingIntervals will be initialized with true and will be set to false when string “Entries:” is read from the file.

FBUIM-LLR-6- Function IsReadingIntervals will be overridden to return the field isReadingIntervals.

FBUIM-LLR-7- Function ReadInterval will be overridden to read the next line in file while skipping empty lines.

a. If the line is the first non-empty line and it is “Intervals:” it will be skipped.
b. If the line is in format [numberA, number), this function will create a new IntervalCls object if the numbers can be parsed, it will throw an exception if it can’t parse the files
c. If the line is “Entries:”, isReadingIntervals field will be set to false and null will be returned
d. If the end of file is reached, null will be returned and an isEndOfInput flag will be set to true.

FBUIM-LLR-8- Function ReadEntry will be overridden to read the next non-empty line in file as an entry of type T.

a. If EOF is reached, isEndOfInput flag will be set to true false will be returned.
b. If the line can be parsed as the template type, the reference parameter will be set to read value and true will be returned
c. If the line cannot be parsed, exception will be thrown.

FBUIM-LLR-9- IsEndOfInput method will be overridden to return isEndOfInput flag.

FBUIM-LLR-10- PrintHistogram method will be overridden to print the histogram information to a file named histogram.txt in the working directory. Output file will consist of lines that show the interval and the number of items that fall into that interval:

[number, number) : number

Histogram Manager (class HistogramManager<T>):

HM-LLR-1- Histogram Manager will be a template based class. The template type will be used for boundaries, entries and related data structures.

HM-LLR-2- Histogram Manager will provide a method for adding intervals:

void AddInterval(IntervalCls<T>)

HM-LLR-3- Intervals will be inserted into a hashtable<T, IntervalCls> (keys will be the lower boundaries)

HM-LLR-4- Histogram Manager will provide a method for adding entries:

void AddEntry(T)

HM-LLR-5- AddEntry function will check if the entry falls in one of the intervals.

a. The hashtable will be searched for the largest key that is smaller or equal to the entry value
b. The interval at the found key will be fetched
c. If the entry is smaller than the upper bound of the interval, the entry count of the interval will be incremented by one.
d. If the entry is larger than the upper bound of the interval, the entry will be added into a list of type T, which will be used to list the outliers

HM-LLR-6- Histogram manager will make the data structure of the interval tables publicly available.

Histogram Application (class HistogramApplicationBaseCls)

HA-LLR-1- Histogram Application will be a template based class

HA-LLR-2- Histogram Application will wrap a IO Manager and a Histogram Manager and will provide methods to set these fields.

HA-LLR-3- Histogram Application will provide a method to create the histogram: CreateHistogram

HA-LLR-4- CreateHistogram will make use of the IOManager to read the intervals and entries until end of input is reached.

HA-LLR-5- Histogram Application will provide a method to report the histogram results: ReportHistogram

HA-LLR-6- ReportHistogram function will call IOManager's PrintHistogram method with the Histogram Manager field.

File Based Histogram Application (class FileHistogramApplicationCls : HistogramApplicationBaseCls)

FBHA-LLR-1- File Based Histogram Application class will be a template based class and will extend from HistogramApplicationBaseCls.

FBHA-LLR-2- File Based Histogram Application class will be designed to remove the need to explicitly state the IO Manager and Histogram Manager fields. To do that:

a. File Based Histogram Application class will provide a constructor with a file name parameter
b. File Based Histogram Application class will set the IO Manager field of the base class with an instance of FileIOManagerCls with the filename given in the constructor
c. File Based Histogram Application class will instantaniate a HistogramManagerCls instance with the same template type.

Error Logger (class ErrorLoggerCls)

EL-LLR-1- Error Logger class will be a singleton class

EL-LLR-2- Error Logger class will receive logs as strings

EL-LLR-3- Error Logger class will append logs to error.txt file separated with new line characters

EL-LLR-4- Error Logger class will provide a method to clear the error.txt file

API USAGE

Build/Make

"make" command creates an executable called histogram
This executable can be called with a parameter giving an input file. If no parameter is provided, test.txt in the same folder will be used as the input.

"make test" command creates an executable called histogram_test
This executable does not need an argument. Test inputs under the Test folder are automatically used and compared.

"make clean" cleans the object and executable files.

For manual building, it must be noted that both main.cpp and TestMain.cpp files include a main function and it will cause a compilation error if they are built in the same project at the same time.

Using the application as a Library

It is enough to create a new instance of FileHistogramApplicationCls with required template type and input file path as parameter in constructor. After calling CreateAndReportHistogram function, resulting histogram will be printed in output.txt file.

```
FileHistogramApplicationCls<float>* app =
             new FileHistogramApplicationCls<float>(dirPath + "/input.txt");

app->CreateAndReportHistogram();
```

Extending the Implementation

To have a different input/output method, a new class inherited from IOManagerBaseCls should be implemented.
An instance of this class should be provided to HistogramApplicationBaseCls as the IOManager.

FileHistogramApplicationCls class used in this project is a wrapper which automatically creates a FileIOManagerCls and register it as the IOManager.

Future Work

Current tests run only on float as template type. Besides the automatic tests in the Test folders, code based tests can be written where data types and input files are explicitly stated.

Current tests do not use real maximum minimum boundaries for floats. These should be added instead of virtually big/small numbers.

Current tests do not have big files where the data structures could start having memory problems. These kinds of test inputs could be generated automatically or can be generated as a memory stream instead of physical files.

ErrorLogger right now is a too generic solution for errors and exceptional situations. Better solutions could be provided.

Requirement-Code mapping is not complete. Comments in the code can be enriched using the related requirement numbers.

Files in the makefile is hardcoded. These can be automatically generated using text templates.