Programming Project 1: 2015 New York City Street Tree Census (Revised)

1 Overview

In 2016, New York City made public the results of *TreesCount!*, the *2015 Street Tree Census*, conducted by volunteers and staff organized by the NYC Department of Parks & Recreation as well as partner organizations. The data includes information about more than 680,000 trees on the streets of New York City. This is a large dataset, with over 683,000 lines of text totaling more than 193 MB of data. Although it will fit into your computer's memory, it will make many computers behave sluggishly, depending on what you do with the data. This first programming project is designed to give you practice in processing datasets in a piecewise way rather than by reading the entire data set into a single contiguous data structure such as a vector or an array. It will also familiarize you with the NYC Tree dataset, which will be the basis for a later project. Your program will consist of four components:

- A `Tree` class
- A `PseudoServer` class
- A `Bitcoin` class
- A main program

The `Tree` class will have a constructor whose single parameter is a line of text from the 2015 Street Tree Census dataset and which creates a `Tree` object from that single line of text. The `PseudoServer` class will simulate a server, maintaining an internal queue; it has two important methods, one of which reads a line of input from the data set and stores it in its internal queue, and the other delivering the data from the front of the queue to the client code that asks for it. The `Bitcoin` class, which has nothing to do with the digital currency of the same name, consists of objects that deliver random streams of zeros and ones, i.e., bits. The main program will open the input file named on the command line, and if successful, will process the file according to the detailed instructions described below.

2 Project Objectives

This project is designed with three objectives in mind:

- to give you exposure to and experience with large, open data sets. Open data sets are to data what open source software is to software. No one has proprietary rights to the data. You can download it and analyze it for free. Wikipedia has a good description of open data: "Open data is the idea that some data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control."
- to require that you compile and build your code on our Linux server by forcing you to use an object module compiled on our system, namely the `Bitcoin` class implementation, and
- to require that you write implementations of two other classes.

3 About The Data Set

The data set is part of the NYC OpenData website and can be found here:

`https://data.cityofnewyork.us/Environment/2015-Street-Tree-Census-Tree-Data/uvpi-gqnh`

You may find it interesting to take a look at an online visualization project based on an older New York City tree census data set at `http://www.cloudred.com/labprojects/nyctrees/`.

The NYC OpenData website for this tree census data gives you the means to download the data in various formats. Your program has to work with the *csv* format of the data. A file in *csv* format, in case you are not familiar with it, is a ***comma-separated-values*** file. A comma-separated-values file is a plain text file in which each line represents a single record, and within the line, commas separate the individual fields of the record. (Fields can also contain commas if they are within quoted strings, e.g., "Brooklyn, New York" is a single field.) Spreadsheet applications let you import csv files to view their contents by rows and columns. The tree data file that can be downloaded contains records for over 680,000 trees. Each row represents a single tree (or tree stump) and has 41 columns, which means that there are 41 different pieces of information for each tree. *The data set that is downloaded will have as its first row, the labels of its columns. For this assignment, you should delete that row, so that the program can assume all rows are actual data rows. (The version of the data set that I provide on the server has that first row deleted.)*

A detailed description of the meaning and form of every column[1] in that dataset can be found in the ***data dictionary*** described here:

`https://data.cityofnewyork.us/api/views/uvpi-gqnh/files/8705bfd6-993c-40c5-8620-0c81191c7e25?download=true&filename=StreetTreeCensus2015TreesDataDictionary20161102.pdf`

This data dictionary is also available on our server in the `resources` subdirectory of the `cs335_sw` directory. Each valid line in the dataset contains 41 columns. Some of these columns may be empty. An empty column is represented by two commas with no intervening characters. The columns are determined by the commas separating each entry. This means that a valid line has to contain at least 40 commas separating the entries (even if the entries are empty), and maybe more, if the fields contain embedded commas. While there should not be invalid lines in the file, if any are found, the program should handle them by ignoring them.

4 Main Program Invocation, Usage, and Behavior

The program is invoked from the command line and expects ***three*** command line arguments. The first specifies the ***csv*** file to be opened for reading and the second specifies the output file. *The third argument is a non-negative integer whose purpose will be explained below.* If there are fewer than three arguments, it is a usage error and the program must write an appropriate and meaningful error message onto the standard error stream, after which it must exit. If the input file that is specified does not exist or cannot be opened for some reason, the program must write an appropriate and meaningful error message onto the standard error stream and then exit. The second file does not need to exist, but if it does, the program must make sure that it can open it for writing and if it cannot, it must write an appropriate error message on the standard error stream. *If the third argument is not an integer, it is an error that must also be reported.*

Assuming that no usage errors occurred, the main program will simulate the simultaneous running of the `Tree` class and the `PseudoServer` by means of a virtual "coin toss." The virtual coin toss will be provided by a `Bitcoin` class, which is described below. You will not create this class; I have written it and will explain how you use it below. A `Bitcoin` object is an object that produces a stream of ones and zeros randomly. If the outcome of the `Bitcoin` toss is a 0, the main program will have the `PseudoServer` deliver a line of text to the `Tree` class constructor, which will construct a `Tree` object from it, after which the main program will write that object to the output file. If the outcome is a 1, the main program will ask the `PseudoServer` to read the next line from the input file and store it. This will happen repeatedly until the input file is exhausted.

Specifically, in each iteration of the main loop, `main()` flips a virtual coin that has two values: 0 or 1. If the coin flip is 0, the main program

1. calls the `PseudoServer`'s extraction method in order to have it get the string from the front of the queue. If the `PseudoServer`'s queue is empty, nothing more is done. Otherwise, it
2. calls the `Tree` object constructor, passing that string to it,
3. calls the `Tree` object's output method to create a different string representation of the `Tree` object (with fewer fields in a different order), and

[1] This description is missing the description of the column with index 14 that appears between the *user_type* and *root_stone* columns in the dataset.

4. writes the string delivered to it by the `Tree` object to the output file specified on the command line.

When the input file has been read completely, the main program will stop tossing the `Bitcoin` and will instead just perform the above steps repeatedly until the `PseudoServer` queue is empty.

If the `Bitcoin` produces a 1, the main program calls the `PseudoServer`'s `read()` method, to read the next line from the file. If the `PseudoServer` runs but the end of input has been reached, it returns false, which triggers an event in the main loop indicating that the input file has been read, although the queue filled by the `PseudoServer` may still have unprocessed data. (This is when the main program must stop flipping the coin.) If the end of the input file has not been reached, the `PseudoServer` reads the input and puts it on its queue.

4.1 Keeping and Reporting Statistics

The main program also keeps and reports statistics on the queue size. Specifically it will report the maximum queue size, the average queue size, the number of times that the `PseudoServer` tried to get the data from the front of the queue but found the queue to be empty, and the size of the queue immediately after the input file's last line was read. This report will be written to the standard output stream in the format:

```
average queue size: <n>
maximum queue size: <m>
empty queue count:  <c>
queue size on eof:  <e>
```

where `<n>`, `<m>`, `<c>`, and `<e>` are the actual numbers it obtains. ***The output should contain nothing else!***

To compute these statistics, it must get the queue size in each iteration. You must decide at what point within that iteration to get the queue size, remembering that the maximum must be the largest size that the queue reached. The average queue size should exclude the iterations that take place after the input file has been emptied. It is the total of the queue sizes divided by the number of iterations while the file is not empty.

5 The Bitcoin Class

The `Bitcoin` class defines a ***function object***. Its interface is

```
class Bitcoin
{
public:
    // Constructor for Bitcoin
    Bitcoin( int n = 0);

    // operator()  returns a random 0 or 1 with equal probability
    int  operator()();
};
```

Function objects have an overloaded application operator (`operator()`), which makes it possible to write code like this:

```
Bitcoin mycoin();
std::cout << mycoin();
```

In other words, you can call the `operator()` by writing the object name followed by `()` as if the object were a function. To use a `Bitcoin` object, include the header file, "`bitcoin.h`" in your main program. The `Bitcoin` constructor must be invoked using the third command-line argument's integer value as its single parameter. For example, if that argument's value is stored in a variable named `seed`, then your program would declare a `Bitcoin` named `mycoin` as follows:

```
Bitcoin mycoin(seed);
```

When testing the program, you should pass the same value to the `Bitcoin` constructor each time, which implies that you should invoke your program with a command like

```
$ myprogram  infile outfile  1000
```

which will set the seed to 1000 each time[2]. When you want the program to have different results each time, pass a different value to the constructor. The UNIX `date` command can be used to generate the current time expressed as an integer number of seconds:

```
$ date +%s
1518105108
```

So you can call your program like this:

```
myprogram infile outfile `date +%s`
```

which will initialize the `mycoin` object with the current time, expressed in seconds since "*the Epoch*" (00:00:00, UTC, January 1, 1970.) *The backquotes around the `date` command are a `bash` function - they cause the command to be replaced by what it would output to the standard output stream.*

6 The Tree Class

As noted above, the csv file has 41 fields, but your `Tree` object will store a subset of them. The `Tree` class represents a single tree on some street in New York City. The header file for the `Tree` class should be stored in a file named `tree.h`. The `Tree` class must encapsulate the following fields of the data set:

- `string spc_common`; the common name of the tree, such as "`white oak`" or a possibly empty string
- `int tree_id`; a non-negative integer that uniquely identifies the tree
- `integer tree_dbh`; a non-negative integer specifying tree diameter
- `string status`; a string, valid values: "`Alive`", "`Dead`", "`Stump`", or the empty string
- `string health`; a string, valid values: "`Good`", "`Fair`", "`Poor`", or the empty string
- `string address`: nearest estimated address to tree
- `string boroname`; valid values: "`Manhattan`", "`Bronx`", "`Brooklyn`", "`Queens`", "`Staten Island`"
- `int zipcode`; a positive five digit integer (This means that any number from 0 to 99999 is acceptable. The values that are shorter should be treated as if they had leading zeroes, i.e., 8608 represents zipcode 08608, 98 represents zip code 00098, etc.)
- `double latitude`; specifies GPS latitude of the tree point, in decimal degrees
- `double longitude`; specifies GPS longitude of the tree point, in decimal degrees

These must be private data members of the `Tree` class. All of the string data fields should store the data in the exact case (upper or lower) as it is in the original input file. The spatial coordinates are GPS coordinates that can be used to locate the trees on a map.

The `Tree` class must provide ***at least*** the following public methods. You may add other methods if you think they are necessary. In any case, the Tree class implementation file must be in a file named `tree.cpp` implementation file. All methods must be case insensitive when comparing string data.

Your program must use the public interface described below. It cannot modify it in any way. Your implementation file must implement exactly what this interface file defines. You are free to define the private part in any way that you like, but if the public part deviates in any way it will be considered incorrect.

[2]Command-line arguments are passed to the program as C strings. To convert a string to an integer, you should use `strtol()`. You can read its man page for details on how to use it. Do not use `atoi()`, since `atoi()` has no error checking. You could also use an `istringstream`.

Method Syntax	Description
`Tree(const string & treedata);`	A constructor for the class that takes a string from a csv file.
`friend ostream& operator<< (ostream & os, const Tree & t );`	This prints a `Tree` object onto the given `ostream`. Each of the members of the `Tree` object should be printed, in the exact same order as they are described in the table above, e.g., with the tree `spc_common` name first, then the `tree_id`. Fields should be separated by commas in the output stream.
`string write();`	This creates a string from the data members of the `Tree` object, in the exact same order as they are described in the table above. The members in the string are separated by tab characters.

7 The PseudoServer Class

The `PseudoServer` provides three public methods: it reads the next line from a text file and places it into its internal queue, it removes the line from the front of the queue, and it reports on the size of the queue. The exact syntax of these methods is as follows:

Method Syntax	Description
`bool read(istream & is);`	If there is another line of text in the open input stream `is`, it returns `true` and puts the line onto the end of its internal queue. Otherwise it returns `false`.
`bool extract(string & s);`	If the queue is not empty, it removes the front element from the queue, stores it into the string parameter passed to it, and returns `true`. Otherwise it returns `false`.
`int queuesize();`	This returns the number of elements in the queue.

You must implement your own queue for this project; you are not allowed to use the C++ standard queue container from the library.

8 Project Organization

Your project must consist of the following files:

```
main.cpp
pseudoserver.h
pseudoserver.cpp
tree.h
tree.cpp
bitcoin.h
bitcoin.o
Makefile
```

Notice that there is no `bitcoin.cpp` file. You are not given the source code to it. It has been compiled on the `cslab` architecture and can only run there. The `bitcoin.o` file is stored on the server in the directory

```
/data/biocs/b/student.accounts/cs335_sw/resources/project1_files
```

You may also want to separate out your queue class in another pair of files, `queue.cpp` and `queue.h`, or you can embed the queue implementation in the `pseudoserver.cpp` file. *The `Makefile` should be the one that I put in the above directory, possibly modified to include instructions to compile `queue.cpp` and to link it to the rest of the executable.*

9 Testing Your Program

You should make sure that you are testing the program on a much smaller data set for which you can determine the correct output manually. You should create your own small test files for that purpose. (Feel free to share those with other students on Piazza.)

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

Be warned - do not try to use a large data set when writing and debugging the code. If you do, you will discover that it can be hours before you see results on typical laptops and desktop computers.

10 Programming Rules

Your program must conform to the programming rules described in the ***Programming Rules*** document on the course website. It is to be your own work alone. There are some very important requirements in that document, so do not ignore it.

11 Grading Rubric

The program will be graded based on the following rubric, based on 100 points.

- A program that cannot run because it fails to compile or link on a `cslab` host receives only 20%. This 20% will be assessed using the rest of the rubric below. A program that does not include instructions on how it is to be built, and which cannot be built by the standard default compile with no flags, will be treated as a program that does not compile, so be sure to include instructions on exactly how to build the executable.
- Meeting the behaviorial requirements of the assignment: 60%
- Design (modularity and organization): 15%
- Documentation: 20%
- Style and proper naming: 5%

This implies that a program that does not compile on a `cslab` host cannot receive more than 20 points.

12 Submitting the Assignment

This assignment is due by the end of the day (i.e. 11:59PM, EST) on February 26, 2018. Create a directory named named *`username`*`_project1`, where *username* is to be replaced by your CS Department network login name. Put all project-related source-code files into that directory. ***Do not place any executable files, data files, or object files other than bitcoin.o into this directory.*** You will lose 1% for each file that does not belong there, and you will lose 2% if you do not name the directory correctly[3].

Next, create a zip archive for this directory by running the `zip` command (on a `cslab` host, not `eniac`):

```
zip -r username_project1.zip ./username_project1
```

[3] I have scripts that process your submissions automatically and misnamed files force me to manually override them.

This will compress all of your files into the file named `username_project1.zip`. Do not use the `tar` compress utility.

The submit command that you will use is submit_cs335_project. It requires two arguments: the number of the project and the pathname of your file. Thus, if your file is named `username_project1.zip` and it is in your current working directory you would type

```
submit_cs335_project 1  username_project1.zip
```

The program will copy your file into the `project1` subdirectory

```
/data/biocs/b/student.accounts/cs335_sw/projects/project1/
```

and if it is successful, it will display the message, "`File ...  successfully submitted.`"

You will not be able to read this file, nor will anyone else except for me. But you can double-check that the command succeeded by typing the command

```
ls -l /data/biocs/b/student.accounts/cs335_sw/projects/project1
```

and making sure you see a non-empty file there.

If you put a solution there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.