Assignment 2:
Processing New York City Open Data

Due date: Sept. 28, 11:59PM EST.*

1 Summary

This assignment is designed to expose you to the use of open data. 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." New York City (NYC) has a treasure trove of open datasets that are available to download. You can explore them at https://nycopendata.socrata.com/.

For this assignment, you will work with the dataset consisting of the most popular baby names given in New York City in the four years from 2011 through 2014. The data is stored in a comma-separated-values (CSV) file[1] that has been downloaded, cleaned, and stored on our server for your convenience. You can find it in the directory

```
/data/biocs/b/student.accounts/cs235_sw/data/
```

You will be writing parts of a program that will process this dataset to answer various questions about the data, such as which name was the most popular name given to a baby girl in 2012 among Hispanic women, or what name was the most popular among all women in 2014, or how many boys born in 2013 were named Dylan. The exact types of questions will be discussed below.

This assignment gives you experience in working with existing code. You are given the interfaces to two different classes. In one case you have to write an implementation for that interface, and in the other, you are given the implementation as a binary file and you just need to write client code that uses the interface.

2 Objectives

The objectives of this first programming project are for you to master the following tasks:

- working with multi-file programs using separate compilation
- reading, understanding, and working with existing source code
- reading data from input files
- writing data to output files
- using and understanding command line arguments
- working with large data sets
- using arrays
- writing class interfaces and their implementations

Most, if not all, of the skills that you need to complete this project are based on the material covered in CSci 135, but it is possible that some of these skills need more practice than others, or that you forgot stuff. It is even possible that there might be certain topics that your class did not cover at all, or that you did not get much practice on, in which case you should view this as an opportunity to get up to speed now.

[1] Details regarding the format of a CSV file are given in Section 3.

3 Program Inputs

Your program will obtain its data from a file whose pathname is given as the first command line argument. It will also obtain the sequence of commands it must carry out from a file given as the second command line argument. There is no other input provided. In particular, there is no user interaction. To repeat, the program must get its input from command line arguments. It does not prompt for the name of a file, it does not assume the file has some fixed name hard-coded into it. It must process its command line!

3.1 Input Files

The program is given the pathnames of two text files as command line arguments. The first is the pathname of a file containing the data set in CSV (comma separated values) format. The second is the pathname of a file containing a list of commands that your program needs to perform. For example, if my program is named `proj1_sweiss`, and the data is stored in a file in my working directory named `babynames.csv` and the commands are in a file named `test_commands`, then I would type

```
proj1_sweiss babynames.csv test_commands
```

to run my program on the `babynames.csv` dataset using the commands in the set of commands in `test_commands`. It is an error if someone attempts to run the program without two file names. The program must check that two arguments were given on the command line and that each file can be opened for reading by the program. If there are any errors, such as missing arguments, arguments that do not exist, or that cannot be opened, the program must display a short but descriptive error message that indicates the nature of the error (such as "`the file mycommands cannot be opened.`") It is not enough to write something like "`could not open file`." The error message should be written to the standard error stream (`cerr` in C++, `stderr` in C). Your program will not be able to distinguish whether the file does not exist or whether the user does not have permission to open it. It is enough to report that it could not be opened, for whatever reason.

3.1.1 Data File

A CSV file is a file in which each line represents a record of some type, and within each line, the fields or members of that record are separated by commas. Because commas separate the fields, they do not appear within any field, unless the field itself is enclosed in quote characters. In this particular CSV file, no field contains commas, making parsing it easier.

The data file for this assignment was downloaded from the webpage
https://data.cityofnewyork.us/Health/Most-Popular-Baby-Names-by-Sex-and-Mother-s-Ethnic/25th-nujf
as a CSV file. It was then cleaned up, so that each line is in the proper format and there are no duplicate lines, and all lines have the same case. Each line in the file contains six comma-separated fields:

```
BRTH_YR,GNDR,ETHCTY,NM,CNT,RNK
```

representing a single data record with the following meaning:

`BRTH_YR` (integer) The year in which babies with the given name were born

`GNDR` (string) The gender of the babies with the given name

`ETHCTY` (string) The ethnicity of the mothers of the babies with the given name

`NM` (string) The actual baby name

`CNT` (integer) The number of babies with the given name, born in the given year, with the given gender and mother's ethnicity

`RNK` (integer) The rank of that name of all baby names in the given year, given mother's ethnicity, and baby gender, with 1 being the highest rank, i.e., the most frequently occurring name.

As an example, the line

```
2013,MALE,ASIAN AND PACIFIC ISLANDER,Dylan,45,24
```

represents the fact that 45 male babies born in 2013 to women who self-identified as "Asian and Pacific Islander" were named Dylan, and that of all male babies born in 2013 to such women, Dylan was the 24th most popular.

The only space characters in the lines are those found between the words of the ethnicity field.

3.1.2 Commands File

The file that contains commands has a single command per line. Each line is in one of the four forms described below. The exact meaning of each command is specified in section 4.

`print` *tofilename*

where *tofilename* is replaced by the name of a file to be created to hold the output.

`sort` *how*

where *how* is replaced by one of the words `byname` or `byrank`, whose meanings are explained below

`getmax` *gender year ethnicity*

where *year* is either a number between 2011 and 2014 or the digit 0,

gender is either `M`, `F`, or the digit 0,

ethnicity is either `W`, `B`, `H`, `A`, or the digit 0.

`getcount` *name year ethnicity*

where

name is any valid string of at most 16 characters,

year is either a number between 2011 and 2014 or the digit 0,

ethnicity is either `W`, `B`, `H`, `A`, or the digit 0.

Notice that the above descriptions do not describe *what the program is supposed to do* when it reads a line, but merely whether a line is in a legal form. If it is not syntactically correct, the program must report the error on the standard error stream and advance to the next line, and if it is correct, the program must process the given command.

3.2 Output Files

The program writes all correct output to one or more output files as specified by the `print` command, detailed below. All error messages generated by your code should be written to the standard error stream (`cerr` in C++, `stderr` in C.)

4 Program Behavior

If the data file and commands file are opened successfully, the program must then process each command one after the other. Each command must be processed according to the description presented below. When all commands have been processed, any open files should be closed and the program terminated.

`print`

Syntax: `print` *tofilename*

This command causes the data to be written to the given file in whatever order it is stored in when the command is issued, one record per line. If the file already exists, it is overwritten. The records should be written one per line, separating each field within a record with nothing but a single comma. The order of the fields should be the same as the order in the original file, i.e., as:

```
BRTH_YR,GNDR,ETHCTY,NM,CNT,RNK
```

sort

Syntax: `sort byname | sort byrank`

If `byname`, the records are sorted with four keys: first by birth year in ascending order, then by gender with FEMALE < MALE, then by ethnicity in ascending alphabetical order, then by name, in ascending alphabetical order.

If `byrank`, the records are sorted with four keys: first by birth year in ascending order, then by gender with FEMALE < MALE, then by ethnicity in ascending alphabetical order, then by rank, in descending numeric order.

In both cases, the records are sorted in place in the object storing the records.

getmax

Syntax: `getmax` *gender year ethnicity*

where

gender is either `M`, `F`, or the digit 0,

year is either a number between 2011 and 2014 or the digit 0,

ethnicity is either `W`, `B`, `H`, `A`, or the digit 0,

The `getmax` command is given three mandatory arguments. If no argument is the digit 0, then the `getmax` command finds the name whose count is the greatest for the given year, gender, and ethnicity and *appends the entire record to a file named* `results` *in the current working directory* using the same output format as is used in the print command described above.

The digit 0 is a ***wild-card*** for the given field. This means that `getmax` ignores that field when looking for the record with greatest count. If there are multiple records with a maximal count, then it writes all of them to the file, using the same format as the `print` command. For example, if the *year* is 2011 but *gender* is 0 and *ethnicity* is 0, then `getmax` writes the record of the baby born in 2011 with greatest count value, and if there are ties, then all records that are tied at that count are written. If the *year* is 2011 and the *gender* is F, but *ethnicity* is 0, then `getmax` writes the record of female babies born in 2011 with the greatest count.

getcount

Syntax: `getcount` *name year ethnicity*

where

name is any valid string of at most 16 characters,

year is either a number between 2011 and 2014 or the digit 0,

ethnicity is either `W`, `B`, `H`, `A`, or the digit 0.

The `getcount` command is given three mandatory arguments. If neither `year` nor `name` is the digit 0, then the `getcount` command finds the total number of babies who were given that name in the given year with the given ethnicity. The digit 0 is a ***wild-card*** for the given field. If year is 0, then all years are used in the calculation. If ethnicity is 0 then all ethnicities are used. So if both are 0, `getcount` computes the total number of babies whose name is the given name, across all years and ethnicities.

This command creates a line of the form "There were *count* babies born in the given years whose mother was of the given ethnicity with the name *name* ", where *count* is replaced by the amount computed, and *name* is the name argument. If a wildcard is used, an appropriate line should be created. In all cases, the line is appended to ***a file named*** `results` ***in the current working directory*** .

5 Implementation Constraints

You are provided the interface to a class named `Command` that represents a single command. That class provides the functionality to obtain the next command from the input command file and parse it. It stores the command's parameters in private data members but it also provides methods to get the type of command and the command's arguments. If it encounters any bad commands, it returns appropriate error values. The `Command` class interface and its implementation are stored on the server in two separate files named `command.h` and `command.o`. In particular you are not given the source code from which `command.o` was compiled. You will not have access to the source code of `command.o`, because it is not necessary. Your main program must use this `Command` class as its means of obtaining the commands. You need to read the `command.h` file to see what public methods it provides and design your program to use these methods. It is also included at the end of this document in Section 11.

You are also given the interface to a class named `Babyname`. The `Babyname` class represents a single `Babyname` record, i.e., the data contained in a single row of the CSV file. The interface includes a few public methods and four friend functions. If friend functions are a new concept for you, do not fear - they will be explained in class. Your job is to write an implementation for this interface. You are not allowed to alter the interface in any way. It contains all functions needed to implement an efficient solution to this problem. The interface is included at the end of this assignment in Section 10 but it is also contained in the header file `babyname.h` on the server, the exact location to be given in class. Your implementation must be named `babyname.cpp` or `babyname.cxx`. ***It is case-sensitive - if you change the case of the filename, points will be deducted.***

Your main program is responsible for opening input files, reading from and writing to files, and closing files. It is also responsible for checking the correctness of the command line arguments and issuing any error messages related to the command line. If you are not experienced in using command line arguments, please read my notes on the topic here:

`http://www.compsci.hunter.cuny.edu/~sweiss/resources/cmmdlineargs.pdf`.

Your program should create a class that represents the entire data set. This class should store the data in a hidden array or a C++ vector, and provide methods for populating the array from an input stream, sorting the array by the different sorting commands specified above, writing the array data to an output stream and getting the various maximal values needed by the `getmax` command. ***It should hide all details from the client (main) and should not read from or write to a file.*** I repeat, *it does not perform any file I/O; instead it must write to and read from streams.*

All class interfaces and implementations are to be in separate files from each other and from the main program. Thus, your project will have at least the following files:

a main file, `main.cpp`

an interface for the `Command` class, `command.h`

an object file for the `Command` class, `command.o`

an interface for the `Babyname` class, `babyname.h`

an implementation file for the `Babyname` class, `babyname.cpp`

an interface for the dataset class (`babycollection.h` or whatever you want to call it), and

an implementation file for that class, `babycollection.cpp`.

6 Advice

You should start right away! This program does not require you to write much code (more than you are used to perhaps, but less than future assignments will require), but it will take some time.

You should modularize your design so that you can test it regularly. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

You should backup your code after you have spent some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, anything that gives you a second (or maybe third copy). Murphy's Law is always in force - things can go wrong at the last minute, so do not take a chance on not having something to submit.

7 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. You may discuss the assignment with other people, such as your classmates and tutors but all work for this assignment must be entirely your own. Any sharing or copying of the assignment will be considered academic dishonesty. If you get significant help from anyone, you should acknowledge that help in your submitted documents, in which case your grade will be adjusted in proportion to the part that you completed on your own. You are responsible for every line in your program: you need to know what it does and why.

You should not use any data structures and features of C++ that have not been covered in this class or any of its prerequisite classes. If you have doubts about whether or not you are allowed to use certain language features, inquire first.

8 Grading Rubric

The program will be graded based on the following rubric.

- Compiling on a `cslab` machine: 30%
- Correctness of the implementation 40%
- Performance 5%
- Design (modularity and organization) 10%
- Documentation: 10%
- Style and proper naming: 5%

This implies that a program that does not compile on a lab computer cannot receive more than 70 points. Some implementations might be more efficient in terms of running time than others. Although you will not have learned enough to write this program in a very efficient way, there are certain programming decisions that can lead to very poor performance or to acceptable performance. The performance component of 5% will be given if the program is reasonably efficient.

9 Submitting the Assignment

This assignment is due by the end of the day (i.e. 11:59PM, EST) on September 28, 2017. There is a directory in the CSci Department network whose full path name is
`/data/biocs/b/student.accounts/cs235_sw/projects/project1`.

That is where your submission will go. In order to put it there you must follow these steps. First you will create a zip file containing your source code, including the header file `command.h` and the object file `command.o`. To do this, create a directory named `proj1_`*`username`* where username is replaced by your login name, and put all of your files into that directory. Do not place anything else into this directory. ***You will lose 1 point for each file that does not belong there***. With all files in your directory, change directory so that `proj1_`*`username`* is within the current directory and run the command

```
zip -r proj1_username.zip ./proj1_username
```

This will compress the directory and all of the files within that directory into the file named `proj1_`*`username`*`.zip`. Then you will use the program `submit235project` to deposit the zip file into the submission directory. The program requires two arguments: the number of the assignment and the pathname of your zip file. For example, if your username on our system is `Bugs.Bunny` and your zip file is named `proj1_Bugs.Bunny.zip` and it is in your current working directory (e.g., your home directory) then you would type

```
/data/biocs/b/student.accounts/cs235_sw/bin/submit235project 1 proj1_Bugs.Bunny.zip
```

The program will create the file
`/data/biocs/b/student.accounts/cs235_sw/projects/project1/proj1_Bugs.Bunny.zip`.

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

```
ls -l /data/biocs/b/student.accounts/cs235_sw/projects/project1
```

and making sure you see your file and that its size is the same as the size of the original `proj1_Bugs.Bunny.zip`.

If you decide to make any changes and resubmit, just run the command again and it will replace the old file with the new one. Do not try to put your file into this directory in any other way - you will be unable to do this.

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.

10 The Babyname Class Interface

```
#include "command.h"

class Babyname
{
public:
    /** Babyname() - default constructor */
    Babyname();

    /** set() - set method for the class
     * assigns given values to the member variables, while validating values
     * method ensures that all parameters are within the allowed ranges
     * as defined in the specification
     * @param int       year      [in]  birth year
     * @param Gender    gender    [in]  baby's gender
     * @param Ethnicity ethnicity [in]  mother's ethnicity
     * @param string    name      [in]  name of baby
     * @param int       count     [in]  frequency (see comments)
     * @param int       rank      [in]  rank (see comments)
     * @pre   none
     * @post  object is assigned all valid values
     */
    void set( int       year,
              Gender    gender,
              Ethnicity ethnicity,
              string    name,
              int       count,
              int       rank
            );
    /** get() - get method for the class
     * retrieves values of the member variables, and passes to corresponding
     * parameters.
     * @param int       year      [out]  birth year
     * @param Gender    gender    [out]  baby's gender
     * @param Ethnicity ethnicity [out]  mother's ethnicity
     * @param string    name      [out]  name of baby
     * @param int       count     [out]  frequency (see comments)
     * @param int       rank      [out]  rank (see comments)
     * @pre   none
     * @post  object is assigned all valid values
     */
    void get( int       & year,
              Gender    & gender,
              Ethnicity & ethnicity,
              string    & name,
              int       & count,
              int       & rank
            );

    /** print() - prints the baby name data onto the given ostream
     *  assuming it is already open for writing, in the format specified
     * in the assignment.
     * @param ostream s   [inout]  ostream opened for  writing
```

```
     * @pre   the object has valid data
     * @post  if the object has valid data, then it is written to ostream
     *        in the format specified in the assignment and the ostream is updated
     */
    void print ( ostream & s );

    /** Several friend functions that perform similar tasks;
     * Each is given two Babyname objects by constant reference and
     * returns true or false depending on whether one is less than or equal to
     * the other.
     * is_less_by_rank(lt, rt) is true if and only if:
     *  birth year of lt < birth year of rt
     *     or they are equal and gender of lt < gender of rt,
     *     or they are equal and ethnicity of lt < ethnicity of rt,
     *     or they are equal and rank of lt < rank of rt.
     * is_less_by_name( lt, rt) has the same logic except in the last
     *     case when name replaces rank.
     * The two is_equal functions return true if and only if year, gender,
     *     and ethnicity are equal and in the case of is_equal_name, the names
     *     are identical, and in the case of rank, the ranks are identical.
     */
    friend  bool is_less_by_rank(const Babyname & lt, const Babyname & rt );
    friend  bool is_equal_rank  (const Babyname & lt, const Babyname & rt );
    friend  bool is_less_by_name(const Babyname & lt, const Babyname & rt );
    friend  bool is_equal_name  (const Babyname & lt, const Babyname & rt );


private:
    int       birth_year;        // 2011 through 2014
    Gender    baby_gender;       // male or female
    Ethnicity mother_ethnicity;  // asian, black, hispanic, or white
    string    baby_name;         // any valid string
    int       name_count;        // >= 0
    int       name_rank;         // > 0
};
```

11 The Command Class Interface

```
#include <iostream>

using namespace std;

/*************************************************************************
                              Exported Types
*************************************************************************/

/** Command_type:
    An enumerated type to represent the different types of commands. This is
    more efficient than storing command types as strings. Notice that the last
    value of the type is num_Command_types, which is simply a count of how many
    values the type contains. It is a useful method of counting, because as long
    as you insert new values before it, it remains valid.
*/
typedef enum
```

```
{
    print = 0,
    sort_byname,
    sort_byrank,
    getmax,
    getcount,
    bad_command,
    null,
    num_Command_types
} Command_type;

/** Gender
    This is an enumerated type to represent two possible genders plus a
    wildcard value named "any" that means either gender.
*/
typedef enum
{
    female = 0,
    male,
    any_gender
} Gender;

/** Ethnicity
    This represents the different ethnic groups that might be associated to an
    individual name in the dataset. The wildcard "any" represents any ethnic
    group.
*/
typedef enum
{
    asian = 0,
    black,
    hispanic,
    white,
    any_ethnicity,
    num_ethnicities
} Ethnicity;

/*****************************************************************************
                              Command Class Interface
*****************************************************************************/

class Command
{
public:
    /** Command()  A default constructor for the class
     */
    Command ();

    /** get_next(istream & in) resets the values of the command object on
     * which it is called to the values found at the current read pointer of
     * the istream in, provided in.eof() is false.
     * @param istream in [inout]  an istream already opened for reading
     * @pre   istream in is open for reading and in.eof() is false
     * @post  If in.eof() is false on entry to this constructor, then
```

```
     *        the command is re-initialized to the values found in the input
     *        stream in, and the istream pointer is advanced to the next line.
     *        If the command is invalid, then when typeof() is called on it,
     *        it will return bad_command.
     *        If in.eof() is true on entry, then the Command_type is set
     *        to null and the remaining values are undefined.
     * @return true if the command was initialized to something other than a
     *        bad_command, and false otherwise.
     */
    bool get_next (istream & in );

    /** typeof() returns the type of the Command on which it is called.
     * @pre  None
     * @post None, as this is a const method
     * @return A value of Command_type, depending on the type of the command
           object.
     */
    Command_type type_of () const;

    /** args() sets the values of its parameters to the argument values of
     * the Command object on which it is called. If the Command object is a
     * bad_command, or if it is a sort_byname, sort_byrank, or null command
     * then the result is set to false and the remaining parameter values are
     * undefined. Otherwise, the Command_type should be getmax, getcount, or
     * print and the appropriate values are copied from the current values in
     * the Command object, meaning
     * if print, then the filename, and
     * if getmax, then the birth_year, gender, and ethnicity, and
     * if getcount, the name, birth_year, and ethnicity.
     * @pre  Command_type is initialized to a valid value
     * @post Either result == false or all members are
     *       set to the values in the object.
     */
    void  get_args (string    & outfile,
                    string    & b_name,
                    int       & b_year,
                    Gender    & m_gender,
                    Ethnicity & m_ethnicity,
                    bool      & result) const;

private:
    Command_type type;       // The type of the Command object
    string       filename;   // If a print command, the name of the file
    string       birth_name; // The name argument on the command line, if given
    int          birth_year; // The birth year argument on the command line
    Gender       gender;     // The gender argument passed on the command line
    Ethnicity    ethnicity;  // The ethnicity argument passed on the command line.

};
```