

RSV Vaccine Effectiveness Analysis

Background

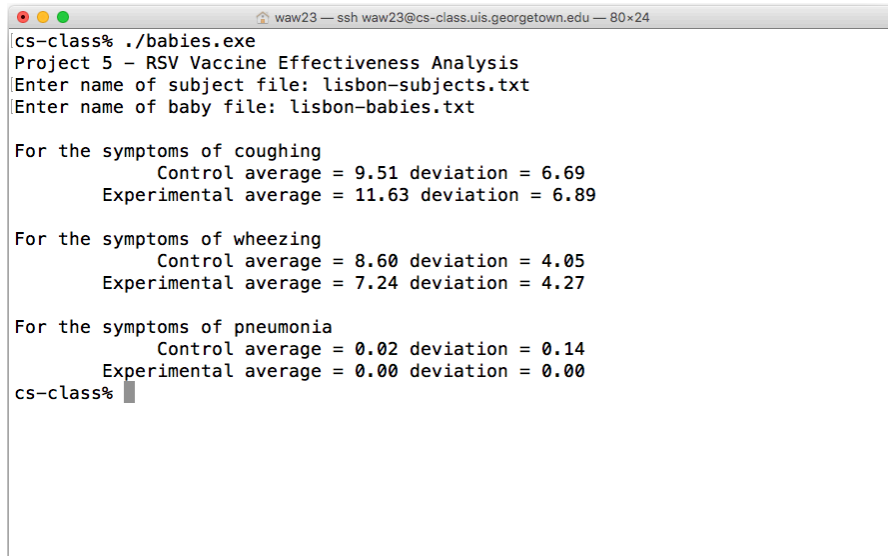
The setup for the RSV study has been successful and now real data is starting to come in about the patients we want to help. The babies have been monitored for symptoms that are consistent with RSV and we are ready to do an analysis of the data to help determine the efficacy of the vaccine.

Requirements

Your task will be to write a program that reads two files. One file contains subject data. The other file contains data about their babies. After the files have been read your program shall analyze the babies' symptoms to see if the vaccine is making a difference.

The subject file contains limited information about the subjects. Each row has the subject ID and 'C' or 'E' for which group she is in (control or experimental). Each row of the baby file has the mom's ID and three measures of symptoms: cough, wheezing, and pneumonia.

Read the subject data first and then the baby data. Your program must use the subject ID in the baby data to identify the correct mother and add the baby to the mother's object. Your program must then compute and display the appropriate statistics about the children's symptoms. Sample output from such a program is shown below.



```
waw23 — ssh waw23@cs-class.uis.georgetown.edu — 80x24
cs-class% ./babies.exe
Project 5 - RSV Vaccine Effectiveness Analysis
Enter name of subject file: lisbon-subjects.txt
Enter name of baby file: lisbon-babies.txt

For the symptoms of coughing
    Control average = 9.51 deviation = 6.69
    Experimental average = 11.63 deviation = 6.89

For the symptoms of wheezing
    Control average = 8.60 deviation = 4.05
    Experimental average = 7.24 deviation = 4.27

For the symptoms of pneumonia
    Control average = 0.02 deviation = 0.14
    Experimental average = 0.00 deviation = 0.00
cs-class%
```

User Interface

For this project, there will not be a menu of options. When the program begins, it shall output a very brief introduction and then prompt for the two input files. Error checking for file names is not required. Assume that valid file names are entered in the correct order. Once the user enters the file names, the program shall load the file data into specified data structures, perform the required analysis, and output the table of results.

Classes and Data Structures

Your program is not allowed to use vectors. Instead, we shall dynamically allocate memory for each required `Subject` object and store those objects in our own linked-list data structure. You will notice that the `Subject` class has many data members removed. There are also a few data members that have been added. One of the new data members is a pointer named `next` that points to another instance of the `Subject` class. A class that has a data member that is a pointer to an instance of the same class is called a *self-referential* class. Each `Subject` object shall store the memory address of the *next* object in the linked-list.

The `Trial` class is responsible for managing the linked-list of `Subject` objects. Its capabilities included adding objects to the linked-list, searching the linked-list for an object with a specified subject ID, and printing all objects on the linked-list.

Class Declarations

You are required to use the class declarations below. Do not modify the identifiers of data members, class methods, or member function parameters. Do not add parameters to member functions or alter the order of parameters. Do not add any data members to the given class declarations. You may add methods, but only if they enhance your solution. Any additional class methods shall not replace the functionality of existing class methods.

```
class Baby
{
    friend istream& operator>>( istream &in, Baby &b );
    friend ostream& operator<<( ostream &out, Baby b );

public:
    Baby( );

    double get_cough();
    void set_cough( double );

    int get_wheezing();
    void set_wheezing( int );

    int get_pneumonia();
    void set_pneumonia( int );

    void report();

    unsigned int get_mom_id();
    void set_mom_id(unsigned int);

private:
    unsigned int mom_id; // must match one of the Subjects
    double cough;
    int wheezing;
    int pneumonia;

}; // END Baby class declaration
```

```

class Subject
{
    friend istream& operator>>( istream &in, Subject &ps );
    friend ostream& operator<<( ostream &out, Subject s);

private:
    unsigned int subject_id; // mom's id
    char group;             // e or c
    Subject *next;          // points to next in list in Trial
    Baby baby;              // points to embedded Baby object

public:
    Subject();

    void set_subject_id( unsigned int i ) { subject_id = i; }
    void set_group( char c ) { group = c; }
    void set_next( Subject *ps ) { next = ps; }
    unsigned int get_subject_id() { return subject_id; }
    Subject *get_next() { return next; }
    void set_baby( Baby pb ) { baby = pb; }
    Baby get_baby() { return baby; }

    void report();

    char get_group() { return group; }

}; // END Subject class declaration


class Trial
{
private:
    Subject *head_subject; // points to start of linked list of Subjects
    string subject_filename; // record which data is used for this trial
    string baby_filename;    // ... same (these are not used here but ...)

public:
    Trial(); // constructor
    ~Trial(); // destructor

    int load_subject_file(string); // prompt for name and read it
    int load_baby_file(string);    // prompt for name and read it

    void add_subject( Subject * ); // add Subject object to the 'linked-list'

    Subject* find_subject_by_id( unsigned int ); // for matching baby to mom

    void print_list(); // see the data

    void display_statistics(); // show avg + dev, e vs. c, for 3 symptoms

private:
    double average(char, char); // args: group (e or c), symptom code (c, w, or p)
    double deviation( char, char, double ); // same args, plus average
    void display_factor( char ); // shows both e and c for given symptom code

}; // END Trial class declaration

```

Sample Executable

As before, there is a sample executable that you may use to verify your calculations and output. This executable is not provided on Blackboard but may be copied from Professor Shield's server account. To copy the file to your account log onto the CS-CLASS server and type the following at the command prompt:

```
cp ~clay/babies.exe ./babies.exe
```

NOTE: This creates a copy of the program on your account. It is possible that the file on Professor Shield's account will be updated without notice. This could happen if any bugs and/or errors are reported. Before testing your code against the sample executable please rerun the command above to make sure you have the most recent version on your account.

Input Data

The list of new data files for this project is:

- almaty-subjects.txt
- almaty-babies.txt
- bandung-subjects.txt
- bandung-babies.txt
- bern-subjects.txt
- bern-babies.txt
- conakry-subjects.txt
- conakry-babies.txt
- lima-subjects.txt
- lima-babies.txt
- lisbon-subjects.txt
- lisbon-babies.txt
- surat-subjects.txt
- surat-babies.txt

You may download the zip file from Blackboard that contains all of these data files. Alternatively, you may copy the files directly from Professor Shield's server account. To do so, type the following at the command prompt:

```
cp ~clay/baby-files/* ./
```

This will copy all of the files to your current directory.

How To Begin

Copy and paste the given class declarations into a new source code file. Write function stubs for ALL member functions of both classes. This should be done prior to class on November 27th. Once that is done you may begin writing the implementation code in any order you decide.

Submission Details

Post to Blackboard a .cpp file containing your source code. Locate the assignment Project #5 on Blackboard and attach/upload your file. Do not post your executable file. You should ensure that your source file compiles on the server and that the executable file runs and produces the correct output. Use the following file name for your file: <netID>P5.cpp. Due date is no later than the end-of-day (11:59pm) on Wednesday, December 6th. Late submissions will be penalized 2.5 points for each 15 minutes late. If you are over 10 hours late you may turn in the project to receive feedback but the grade will be zero. In general requests for extensions will not be considered.

Academic Integrity

Your code is an individual project and all work must be your own. Refer to the guidelines specified in the *Academic Honesty* section of this course syllabus or contact me if you have any questions. Include the following comments at the start of your source code file:

```
/**
 * <netID>P5.cpp
 *
 * COSC 051 Fall 2017
 * Project #5
 *
 * Due on: 6 DEC 2017
 * Author: <your name>
 *
 * In accordance with the class policies and Georgetown's
 * Honor Code, I certify that, with the exception of the
 * class resources and those items noted below, I have neither
 * given nor received any assistance on this project.
 *
 * References not otherwise commented within the program source code.
 * Note that you should not mention any help from the TAs, the professor,
 * or any code taken from the class textbooks.
 */
```

These comments must appear **exactly** as shown above. The only difference will be values that you replace where there are "place holders" within angle brackets such as <netID> which should be replaced by your own netID. For example, I would replace <netID>P5.cpp with waw23P5.cpp.

Grading

A grade rubric will be published separately.

(this page intentionally left blank)