Lab6

Lab6

Assigned: Tuesday Feb 12, 2019
Due: Tuesday 26th at 11:59pm. NO late submissions.
NOTE: Lab6 has extra credit parts. In case you are done, please work on those parts for extra points.

~~**Late submissions will be accepted for two days after the deadline with a maximum penalty of 15% per day. For each day, submissions between 12am and 1am: 2% penalty; between 1 and 2am: 4%; 2 3am: 8%; and after 3am: 15%. No submissions accepted after Friday Feb. 15 at 4am.**~~

Please check the General Guidelines.

What you will practice in this lab

- How to define classes
- Using header files for definition of your classes
- Compiling a project with several different files
- Class constructors
- Function overloading
- Linked List
- Sorting Algorithm
- Writing a python wrapper

Using different files for object-oriented implementation

Up to now, your projects contained a single .cpp file. Obviously, for advanced problems with many lines of code, this cannot be the case. For a big project, different people may implement different classes, and they should all manage to make their codes work together. This is why we define separate classes in separate files. Please take some time to read about header files[1].

[1] You can refer to this cplusplus link: http://www.learncpp.com/cpp-tutorial/89-class-code-and-header-files/

Class Date

In this class, you will store information of a single date. Take a look into date.h and you can see there are two versions for set_date method with different type of arguments. When set_date is called in your code, the one with the matching form of arguments will be used. This is called “Function Overloading”.

bool Date::check_date()

We have already implemented this method for you (too many if-else, switch-cases which will make you bored!). The method checks if the current date is a valid one and returns true of this is the case and false otherwise.

bool Date::set_date(int d, int m, int y)

bool Date::set_date(string str)

These two methods set the date based on the input argument. In the second one, the input string should be of the form **M/D/YYYY**[1]. The code checks for the validity of the input date, and if its invalid, change it to the default date in Epoch[2] time, which is 1/1/1970. The return value is true of the argument is a valid date and false otherwise.

void Date::Date(int d, int m, int y)

void Date::Date(string str)

The constructor is very similar to the set_date function, only it does not return anything, and it’s called automatically when the object is created.

void Date::print_date(string type)

The method prints the date with different formats given by input argument string.

string Date::get_date()

This method implemented completely and is commented both in header and cpp files. You will see later why you need to use this method to have access to the private attributes of the Date class from other classes.

We have already implemented the code for you, but there is only one single bug in it (marked with TODO). Compile and test your code using the command[3] below, check the test output and it will definitely help you to find the bug. Fix it!

g++ test_date.cpp date.cpp -o test

---

[1] Correct examples: 1/13/2019, 5/6/1997, 11/11/423

[2] https://en.wikipedia.org/wiki/Unix_time

[3] Try to understand why we are compiling two .cpp files, and not any of the header .h files.

Class Connection

We start implementing a simple version of Connection in this lab and it will be completed in future labs. Take a look into connection.h, here are the main attributes of a connection (which is simply a person):

- First Name (data type is string)
- Last Name (date type is string)
- Date of birth (data type is Date)

Like before, we need set methods. One set method prompting the user for information has been already implemented, you need to implement the other set methods that gets a filename as argument (connection_template.txt has been provided for you as the template) and reads the information of a single connection and sets the attributes.

The parts that you need to complete for this class are marked as "TODO" in connection.cpp. We have also provided you with a test code. Compile your codes with the command below:

g++ test_connection.cpp connection.cpp date.cpp

Update: We missed the destructor for class Connection in the header template, please add this to your code:
Connection::~Connection(){delete birthdate;}

Class Network

We want to store our connections in a double sided Linked-List (LL). Look at the network header file and follow the steps below:

Review the completed code for push_front() and loadDB() to implement push_back() and saveDB(). The printDB() also has been already implemented. Use the test_network.cpp to test your code step by step. You need to come up with the compile command (what cpp files should you compile together for test_network?). Follow these steps:

- When you are implementing saveDB() you will see that it is necessary to go back to Date class and uncomment the get_date() method.
- networkDB.txt is given to you as a template. Just a few notes: loading will erase all the elements of the current LL, and then reads and adds the new ones. Don't forget the count variable!
- Test your code with the second part of test_network.cpp.
- Implement push_back(), test your code again.
- Implement search() and remove(), test your code.
- Implement the destructor.

Sort Class

In this part you are supposed to write a class called Sort. The Sort class gets the integer data from an input file, sorts it with an specified algorithm (among bubble, merge, quick, selection, radix sorts) in ascending order and writes the result (sorted data) in an output file with the same format of the input file (each integer in a separate line). The skeleton for the class has been provided for you in sort.h, you are allowed to add to the class declaration. The definitions of methods in Sort should be written in sort.cpp.

The main function (main.cpp) gets 4 arguments (using argv), in this order: name of the input file, name of the output file, number of integer values, and the name of the algorithm (the same as the name of the methods in Sort). You are supposed to

create an object of class Sort, call the specified sort algorithm, and store the results in the output. Here is an example of the command for running main:

./mySortProgramm.out input.txt output.txt 32 bubbleSort

The result of this part would be 3 different files: main.cpp, sort.cpp, sort.h

Python Wrapper

We want to compare the speed of different sorting algorithms in different situations. While running lower level language C++ is much faster than higher level python, it is much easier to make different experiments in python. This is why we are using C++ for the implementation of our sort algorithms, however, we would like to have our data flow to be in python.

The wrapper is responsible for compiling the C++ code, generating the input data, running the sort executable application (the result of complying the C++ code), and measure how long each algorithm takes for different data.

Read the instructions for the python code inside in wrapper.py, and pay attention to a few notes:

- Do not change the first 10 lines of the python code.

- The python code should only print one single line in the console.

- No hard-coding of parameters, use the PARAM dictionary for your parameters. We will change the PARAM dictionary and your code should work.

The result for this part would be one single "wrapper.py" file.

Main TA for this Lab

Saeed Abrishami

FAQ

Q: Are we allowed to use C++11?
A: No. We have already covered alternatives to stoi. Please convert your string to csttring (using c_str method) and then use atoi.

Q: Are we supposed to check for memory leaks?
A: Yes.

Q: Connection class does not have a destructor, and this may cause memory leak, what should we do?
A: We have email all the class to modify their code with this give destructor: Connection::~Connection(){delete birthdate;}

Q: Can you give us more information on what does "const Date& rhs" mean?
A: Check this reference, it explains pass constant reference in details: https://stackoverflow.com/a/5060218

Q: Is there a typo in Connection::Connection(...)?
A: Yes! Please change the function arguments to Connection::Connection(string f_name, string l_name, string bdate)

Source Files

Attached Files: Lab6_source_files.zip (7.603 KB)

It was so painful for myself to download the text files, so I compressed them in a single file!

Update: use the repo below to test your code. We will add more to these test cases and automatically grade your Lab.
Important Note: DO NOT create branches from this repo!
https://github.com/msabrishami/EE355_Sp2019_Lab5_test_student

Submission

Push all files (**date.h**, **date.cpp**, **connection.h**, **connection.cpp**, **network.h, network.cpp, sort.cpp, sort.h, wrapper.py**) into the github repo.

Don't forget a readme f ile:

- **Your readme file should be named as readme_<STUDENT-ID>.txt, please replace <STUDENT-ID> with your own 10-digit USC-ID.**
- Any non-working part should be clearly stated.
- The citations should be done carefully and clearly, e.g.: "to write my code, lines 27 to 65, I used the Djkstra's shortest path algorithm C++ code from the following website: www.SampleWebsite.com/..."
- Github link for submission: https://classroom.github.com/a/BEg0e1IY

Extra Credit - 25%

- Implement the double pivot quick sort algorithm. You can take a look into: https://learnforeverlearn.com/yaro_web/ (but we don't confirm if its the correct implementation).

- Design an experiment to find the complexity difference between classic quick sort and the double pivot one. You may need to use large unsorted datasets, and running the experiment several times. Try your code with different inputs (make them large enough) and measure the time it takes to finish sorting. Write a report with your results and theory behind it in a the file: <Lab6-QuickSort.txt>.