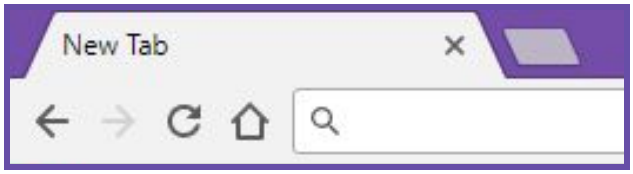
Project 2: Browser History

CPSC 131 Fall 2018



Introduction

Modern browsers keep track of browsing history which allows users to recall previously visited web pages using the back arrow button. When a user navigates back to a previous web page, they also have the option to navigate forward in the navigation history using the forward arrow button. However, if the user navigates back to a previous page and then goes to a new web page, all web pages that previously could be accessed with the forward arrow button are discarded. This behavior is well suited for Linked Lists. In this project you will write C++ code to model this behavior.

For a good video overview of this, check out this link.

In addition, when asked, your program should respond with every web page visited and when it was visited. Since we do not know how many sites the user will visit, we will have a second Linked List that will keep track of every site visited.

You are given a simple text file of a user's browsing history containing their actions: *back*, *forward*, or *new*. If their action is listed as *new*, it will be followed with a web address. Simulate their browsing history based on this text file.

Objective

You are given partial implementations of two classes. **`Webpage`** is a class that holds both the web page url as well as the time it was first visited. The time visited is the number of seconds from the UNIX epoch, 00:00 Jan 1, 1970 UTC. C++ has a variable type that can handle this, named time_t.

`BrowserHistory` is where the bulk of your work will be done. **`BrowserHistory`** stores both a linked list representation of the user's navigation history, as well as an overall history of all sites they've visited. It will be able to read the history from a text file.

The text file will have 3 basic commands: new, back, and forward. Back and forward will simulate the user pressing the back and forward arrow buttons of their browser. New will be followed by the web page's url and the time the site was visited.

You are to complete the implementations of these classes, adding public/private member variables and functions as needed.

You are encouraged to use the C++ Standard Library containers (such as std::list, and std::list::iterator) for this project.

Your code is tested in the provided **`main.cpp`**.

Source Code Files

You are given "skeleton" code files with many blank areas. Your assignment is to fill in the missing parts so that the code is complete and works properly when tested.

- **`Webpage.h`**: Stores both a web page's url and the time it was visited.
- **`BrowserHistory.h`**: stores both the user's navigation history as well as their full history of sites visited.
    - This class contains a method to read item information from a text file.
- **`main.cpp`**: The entry point to the application. The main() function will test the output of your functions. This is already completed but feel free to change it for your own testing (during grading we will use the original main file).

- **`README.md`** file. You must edit this file to include the name and CSUF email address of each student in your group. Do this even if you are working by yourself. We need this information so that we can enter your grades into Titanium. For example, if your group includes students Ada Lovelace and Charles Babbage, your **`README.md`** should be in this format:

```
Group members:
Ada Lovelace adalovelace@csu.fullerton.edu
Charles Babbage charlesbab@csu.fullerton.edu
```

Hints

Start by implementing the Webpage class, then the BrowserHistory class. It can be overwhelming working on the BrowserHistory class so start with the constructor, then the visitSite() function, then the back() and forward() functions.

Remember the BrowserHistory class will include two linked lists, one for the navigation history and one for the full history of every site visited. It will also need an iterator or pointer to point to a specific position in the linked list.

Iterators are very similar to pointers. Both iterators and pointers can be tricky. Make sure you're keeping track of whether you're talking about an address or the object at that address. Remember to use the **->** operator!

Obtaining and submitting code

We will be using GitHub Classroom to distribute the skeleton code and collect your submissions. This requires you to have an account on github.com. If you are having trouble look at the following:

1. Read Understanding the GitHub Flow and Hello World at GitHub Guides.
2. Read the instructions below for your chosen development environment.

Click the assignment link to fork your own copy of the skeleton code to your PC. One student from a group clicks on the link below and forms a new team. The team name must begin with your section number (e.g., 2-Brians-team). This student then invites his/her project partner as an "Outside collaborator" in the settings menu.

Do not fork your repository to your personal github account. Your code should have a URL like https://github.com/CSUF-CPSC-131-Fall2018/project1-2-brians-team, NOT https://github.com/brian/project1-2-brians-team.

https://classroom.github.com/g/OqgM3v5p

(If you get an error when clicking the assignment link, chances are that your repository was still created successfully. Check your github.com page and you will see you are a member of the CSUF-CPSC-131-Fall2018 organization)

Here is a video that shows the steps: https://www.youtube.com/watch?v=-52quDR2QSc
Then edit your code locally as you develop it. As you make progress, commit and push your changes to your repository regularly. This ensures that your work is backed up, and that you will receive credit for making a submission. Don't wait until the deadline to learn how to push code!

Development environment

The test platform is Linux with the g++ -std=c++14 compiler. For this reason, the recommended development platform is Linux.

Linux development environment

To attempt to compile the test program, use the following command:

```
$ clang++ -g -std=c++14 *.cpp -o test
```

To attempt to run the compiled test program, use the following command:
`$ ./test`

Automated Testing

We are also piloting the use of a continuous integration web service that automatically tests your code and gives real-time feedback. This service is provided courtesy of Travis CI.

The Travis service will wait for you to push code to GitHub. After you do, it will try to compile and build your code. It will then send you an email describing the outcome. The email also has a link to a dashboard web page you can view to see a detailed log of what worked, or didn't. This way you can get immediate objective feedback about how well your code is working. We recommend pushing your code to GitHub every time you reach a milestone and would like feedback on your progress.

You can check the results of automatic testing on Travis at https://travis-ci.com/ (same login as your github account).

Grading Rubric

Your grade will be comprised of two parts, *Form* and *Function*.

Function refers to whether your code works properly as tested by the main function (80%).
Form refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

Deadline

The project deadline is Sunday, October 7, 11:59 pm.

You will be graded based on what you have pushed to the main branch of your GitHub repository as of the deadline. Commits made after the deadline will not be considered. Late submissions will not be accepted.

Your code must compile/build for it to be tested and graded.
If you only complete part of the project, make sure that it compiles before submitting.