EECS268:Lab5

Navigation

Home

Information

Syllabus
Schedule
Lecture Archive

Classwork

Labs
Submitting Work

Contents

Due time

This lab is due ~~1 week~~ 2 weeks from the start of your lab.

Overview

I recommend doing this lab in two phases. First, create and test an implementation of a List. Second, make an implementation of the WebHistory interface, using your List implementation.

Phase 1: Create and Test a List Implementation

Create an implementation of the List interface using a LinkedList. Make sure to...

1. Test each method as you go along
2. Compile VERY often (if you write even a simple getLength method, compile and test!)
3. I recommend writing a helper class to help speed up tests (e.g. a class that automates populating a LinkedList with value
4. Use the debugger!
   - The debugger can show you exactly what is happening with your list and help track down reasons for segfaults

What not to do...

- Skip testing to try to save time (it doesn't save time in the long run)

- Code an entire class or the entire lab before compiling for the first time

ListInterface

```
#ifndef LISTINTERFACE_H
#define LISTINTERFACE_H

template <typename T>
class ListInterface
{
    virtual ~ListInterface() {}

    /**
    * @pre The index is valid
    * @post The entry is added to the list at the index, increasing length by 1
    * @param index, position to insert at (1 to length+1)
    * @param entry, value/object to add to the list
    * @throw std::runtime_error if the index is invalid
    */
    virtual void insert(int index, T entry) = 0;

    /**
    * @pre The index is valid
    * @post The entry at given position is removed, reducing length by 1
    * @param index, position to insert at (1 to length)
    * @throw std::runtime_error if the index is invalid
    */
    virtual void remove(int index) = 0;

    /**
    * @pre The index is valid
    * @post None
    * @param index, position to insert at (1 to length)
    * @throw std::runtime_error if the index is invalid
    */
    virtual T getEntry(int index) const = 0;

    /**
    * @pre None
    * @post List is empty
    * @throw None
    */
    virtual void clear() = 0;

    /**
    * @pre The index is valid (must already exist)
    * @post Given entry overrides the entry at the given index
    * @param index, position to override at (1 to length)
    * @param entry, value/object to place at given index
    * @throw std::runtime_error if the index is invalid
    */
    virtual void setEntry(int index, T entry) = 0;

    /**
    * @pre None
    * @post None
    * @return the length of the list is returned
    */
     virtual int length() const = 0;
};
#endif
```

Phase 2: Web Browser History

You will create a class that mimics the behavior of your web browser's back button, forward button, and address bar.

Here is an interface for the BrowserHistoryInterface. You may not change or add to this interface. You must implement this interface, satisfying all the requirements described.

Your implementation can then be used by an Executive class.

```
#include "ListInterface.h"
#include <string>

class BrowserHistoryInterface
{
  public:
  /**
  * @post All memory allocated by the implementing class should be freed.
  *       This, as with all virtual destrucors, is an empty definition since we
  *       have no knowledge of specific implementation details.
  */
  virtual ~BrowserHistoryInterface(){}

  /**
  * @pre none
  * @post the browser navigate to the given url
  * @param url, a string representing a URL
  */
  virtual void navigateTo(std::string url) = 0;

  /**
  * @pre none
  * @post if possible, the browser navigates forward in the history otherwise it keeps focus
  *         on the current URL
  */
  virtual void forward() = 0;

  /**
  * @pre none
  * @post if possible, the browser navigates backwards in the history otherwise it keeps focus
  *         on the current URL
  */
  virtual void back() = 0;

  /**
  * @return the current URL
  */
  virtual std::string current() const = 0;

  /**
  * @pre The list being passed in is empty
  * @post The current browser history is copied into the given list
  * @param destination, an empty list of strings
  */
  virtual void copyCurrentHistory(ListInterface<string>& destination) = 0;
};
```

Reading the history from file

To build up your browser history, you will read in from file. The file name will come in on the command line.

Any given line of the file will contain one of the following entries:

File Entry	Description
NAVIGATE <URL>	Navigates the browser to the given URL. NOTE navigating to a URL retains all URLs accessible from going BACK, but any URLs that would have accessible from going FORWARD are now lost
BACK	A command indicating the web browser is redirected to the previous URL in the history. If there is no URL further back, then the browser stays on the current URL.
FORWARD	A command indicating the web browser is redirected to the next URL in the history. If there is no URL that is next, then the browser stays on the current URL.

HISTORY	Prints the current URL history to the screen using the following format:
	`Oldest`
	`==========`
	`<URL>`
	`<URL>  <==current (assuming this is the current URL)`
	`<URL>`
	`==========`
	`Newest`

Sample input file

```
NAVIGATE http://google.com
NAVIGATE http://reddit.com
NAVIGATE http://facebook.com
NAVIGATE http://myspace.com
HISTORY
BACK
BACK
HISTORY
FORWARD
FORWARD
FORWARD
FORWARD
HISTORY
BACK
BACK
BACK
NAVIGATE http://ku.edu
FORWARD
HISTORY
BACK
HISTORY
```

Output to screen:

```
Oldest
==========
http://google.com
http://reddit.com
http://facebook.com
http://myspace.com  <==current
==========
Newest

Oldest
==========
http://google.com
http://reddit.com   <==current
http://facebook.com
```

```
http://myspace.com
===========
Newest

Oldest
===========
http://google.com
http://reddit.com
http://facebook.com
http://myspace.com  <==current
===========
Newest

Oldest
===========
http://google.com
http://ku.edu       <==current
===========
Newest

Oldest
===========
http://google.com   <==current
http://ku.edu
===========
Newest
```

Libraries You May Include

- iostream
- fstream
- string
- stdexcept

Vectors or any other dynamic data structure that you don't build yourself is forbidden!

Rubric

- 20pts List Implementation
- 30pts Web Browser Implementation
- 20pts Modularity
  - Sensical class design
  - Completely object oriented (e.g. your main should invoke some kind of executive class)
- 10pts Stability
  - There should be zero segfaults even if the
  - NOTE: You can assume a properly formatted file
- 10pts Memory leaks: There should be no memory leaks. Use valgrind ./YourProgramName to verify
  - Any memory leaks will result in a loss of all 10 points!
  - Your destructors will play a key role in preventing memory leaks
- 10pts Comments and documentation:
  - @author, @file, @date, @brief (the author, file name, current date, and a brief description of a file) at the **top of every file!**
  - @pre, @post, @return, (Pre conditions, Post conditions, Return descriptions) in header files. Not required for cpp files
  - NOTE you don't need to comment the Makefile

Remember, if we type "make" into the console and your program fails to compile, we do not grade it.

Submission instructions

Send a tarball with all the necessary files (*.h, *.cpp, and makefile) in a tar file to your GTA email. Do not include any object (*.o) or executable file.

Creating a File Archive Using Tar

The standard Unix utility for created archived files is **`tar`**. Tar files, often called **tarballs**, are like zip files.

1. Create a folder to hold the files you will be sending in. The folder should be named like *LastName-KUID-Assignment-Number*:

   ```
   mkdir Smith-123456-Lab-0#
   ```

2. Now, copy the files you want to submit into the folder:
3. Tar everything in that directory into a single file:

   ```
   tar -cvzf Smith-123456-Lab-0#.tar.gz Smith-123456-Lab-0#
   ```
   That single command line is doing a number of things:
   - `tar` is the program you're using.
   - `-cvzf` are the options you're giving to `tar` to tell it what to do.
     - `c`: **create** a new tar file
     - `v`: operate in **verbose** mode (show the name of all the files)
     - `z`: **zip** up the files to make them smaller
     - `f`: create a **file**
   - `Smith-123456-Lab-0#.tar.gz`: the name of the file to create. It is customary to add the `.tar.gz` extension to tarballs created with the `z` option.
   - `Smith-123456-Lab-0#`: the directory to add to the tarball

Please note that it is **your** responsibility to make sure that your tarball has the correct files. You can view the contents of a tarball by using:

```
tar -tvzf filename.tar.gz
```

Emailing Your Submission

Once you have created the tarball with your submission files, email it to your TA. The email subject line **must** look like "[EECS 268] *SubmissionName*":

[EECS 268] Lab 0#

Note that the subject should be *exactly* like the line above. Do not leave out any of the spaces, or the bracket characters ("[" and "]"). In the body of your email, include your name and student ID.

- This page was last edited on 4 October 2021, at 15:11.