SENG1120 – Data Structures
Trimester 2, 2024

Assignment *1* **Due Date** *Sunday 16 June 2024 @ 23:59*

Purpose *To measure the student's ability to implement and/or use a linked list structure to solve an underlying problem in C++.*

Overview

This assignment will test your ability to implement and use a linked list data structure to implement a (simulated) web browser. This browser will allow users to visit and navigate a history of websites visited, along with maintain a list of bookmarked favourite sites, using a linked list.

You will be provided with a main program that includes a menu and prompt to control and test your implementation. Additionally, the main program will be able to run a list of commands from a file. You will also be provided with appropriate header files and a `makefile`, leaving you to implement only the `Node`, `LinkedList`, and `Browser` classes.

This assignment is worth 100 marks and accounts for 15% of your final grade. Late submissions are subject to the rules specified in the Course Outline.

The Supplied Files

This section gives an overview of the files that you are provided as part of this assignment. There are quite a few, so you are recommended to take some time to understand the overall structure of the program.

- `main.cpp` – contains the main function and other functions that allow you to interactively test the functions you implement in playlist. **This file should not be modified.**

- `node.h` – contains the header for the `Node` class, which includes the instance variables and behaviours that your node should contain. **This file should not be modified.**

- `node.hpp` – contains (skeleton) template implementations for the `Node` class. This is where your implementation of the node is to be completed. **This file should be modified and be part of your submission.**

- `linked_list.h` – contains the header for the `LinkedList` class, which includes the instance variables and behaviours that your linked list should contain. **This file should not be modified.**

- `linked_list.hpp` – contains (skeleton) template implementations for the `LinkedList` class. This is where your implementation of the linked list is to be completed. **This file should be modified and be part of your submission.**

- `browser.h` – contains the header for the `Browser` class, which includes the instance variables and behaviours that your browser should contain. **This file should not be modified.**

- `browser.cpp` – contains (skeleton) implementations for the `Browser` class. This is where your implementation of the browser is to be completed. **This file should be modified and be part of your submission.**

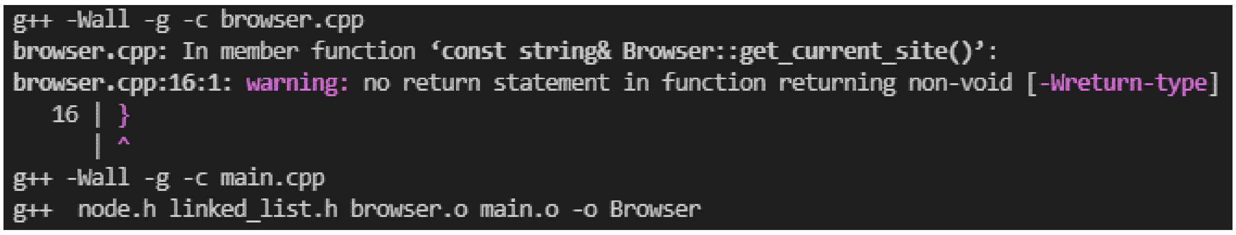
- `makefile` – the `makefile` for the project, which outlines the build recipe. The `-g` flag is set for you to make debugging and/or memory leak detection more convenient, should you wish to use these tools. **This file should not be modified.**

- `TestCommands.txt` – a text file containing a sequence of commands that can be used to run the program in file mode. This is simply a text file with valid commands, one per line. **This file is not really part of the program so you may modify if you wish.**

Running the Program

Of course, you will need to ensure the program is compiled prior to running it. You can use the supplied `makefile` for this – it will produce an executable called `Browser`.

Note: while the initial code you have been supplied will compile, it will produce a compiler warning (see Figure 1). In this case, the warning is indicating that we have a method expecting a reference to be returned, but we have not returned anything. In fact, all the methods you are meant to implement are either empty or return default values – you are provided only with a skeleton of the final program and are expected to complete the required implementations.



```
g++ -Wall -g -c browser.cpp
browser.cpp: In member function ‘const string& Browser::get_current_site()’:
browser.cpp:16:1: warning: no return statement in function returning non-void [-Wreturn-type]
   16 | }
      | ^
g++ -Wall -g -c main.cpp
g++  node.h linked_list.h browser.o main.o -o Browser
```

Figure 1. Compilation warning on initial compile.

The program will operate in two different modes: prompt (interactive) mode and file mode, each described below. In prompt mode, the user is presented with a prompt to enter commands in sequence. In file mode, the user can supply a text file, which contains a list of commands that will be executed – this can be used to test a particular sequence of commands without having to type them every time.

Important Notes: The program does not validate web addresses. The program is also not equipped to handle spaces in the web address provided. Rather, the program interprets spaces as the separator character between different command arguments. If you wish to supply your own data, ensure that you do not include spaces in web addresses.

Prompt (Interactive) Mode

To run in prompt mode, the program should be executed as normal using the command:

```
./Browser
```

When the program is first run, you will be presented with a welcome message and a prompt. The prompt will allow you to enter commands to interact with your browser. Initially, the program will run, but not successfully given that you will not have implemented the functionality. However, you are recommended to have a look through the code, particularly the header files, to better understand the requirements for the various methods you must write.

Figure 2 shows the list of commands that the program can interpret and execute. The parsing and execution of these commands is handled in the main file. Thus, your concern is only to implement the underlying functionality to provide the intended behaviour.

Figure 3 shows an example of the program operating in prompt mode, where the user has entered the commands listed after the prompts `Enter Command:`

File Mode

To run in file mode, the program will be executed with an argument containing the name of the file. To run in file mode, the program will be executed as:

```
./Browser TestCommands.txt
```

The list of commands in the supplied `TestCommands.txt` is minimal and does not run an exhaustive set of tests. However, you may supply the path to any text file as the argument. In fact, you are encouraged to develop your own set of tests, which may be easiest to do with a preset list of commands in a text file.

Note: Executing the program in file mode should produce the exact same output as running the corresponding commands, in sequence, in prompt mode.

Figure 4 shows an example of the program operating in file mode, where the user has supplied the default `TestCommands.txt` file. **Note**: this figure does not show the entirety of the output, only a sample.

The Tasks

Your tasks can be summarised as the implementation of three classes, namely `Node`, `LinkedList`, and `Browser`. For each class, you are provided with the header file and skeleton implementations (i.e., method stubs that allow the program to compile, but are otherwise nonfunctional). **You are not permitted to change the supplied header files and must implement all methods listed in the header files, irrespective of whether they are used in the program.**

Node

The Node class is a templated version of a node object and should be implemented as discussed in the lectures. For a full list of methods required and some important details, you should examine the `node.h` file and its associated documentation.

LinkedList

The LinkedList class is a templated version of a doubly linked list using two sentinel nodes and should be implemented as discussed in the lectures. For a full list of methods required and some important details, you should examine the `linked_list.h` file and its associated documentation.

Browser

The Browser class contains the main logic of the browser and uses your LinkedList class to supply the underlying functionality. That is, this class must use the public interface provided by your LinkedList class to perform the various operations that support the list of commands given in Figure 2. **You are not permitted to use `std::list` or any other collection in the `std` namespace**. For a full list of methods required and some important details, you should examine the `browser.h` file and its associated documentation.

At various points, you are asked to throw an exception under certain circumstances. For the purposes of this assignment, you can pick an appropriate exception from the standard exceptions (see here for a list: std::exception - cppreference.com.) and throw this when appropriate. One example that may be particularly relevant is std::out_of_range (std::out_of_range - cppreference.com), which you could throw using a statement such as:

```
throw std::out_of_range("Custom message");
```

You should not need to import any additional libraries to make this work!

Marking

As mentioned previously, this assignment is worth 100 marks and accounts for 15% of your final grade. Late submissions are subject to the rules specified in the Course Outline.

Your submission will be assessed on both correctness and quality. This means, in addition to providing a correct solution, you are expected to provide readable code with appropriate commenting, formatting, best practices, memory management, etc. **Code that fails to compile will result in a zero for the functionality correctness section.** Your code may be tested on functionality not explicitly shown in the supplied demo file (i.e., using a different `main.cpp` file). Hence, you are encouraged to test broader functionality of your program before submission.

Marking criteria will accompany this assignment specification and will provide an indicative guideline on how you will be evaluated. **Note**: the marking criteria is subject to change, as necessary.

Submission

Your submission should be made using the Assignment 1 link in the Assignments section of the course Canvas site. Assignment submissions will not be accepted via email. Incorrectly submitted assignments may be penalised.

Your submission should include only the completed versions of `node.hpp`, `linked_list.hpp`, and `browser.cpp`. You do not need to include any other files in your submission. Be sure that your code works with the supplied files. Do not change the files we have supplied as your submission will be expected to work with these files – when marking your code, we will add the required files to your code and compile it using the supplied `makefile`.

Compress the required files into a single .zip file, using your student number as the archive name. Do not use .rar, .7z, .gz, or any other compressed format – these will be rejected by Canvas. For example, if your student number is c9876543, you would name your submission:

c9876543.zip

Remember that your code will conform to C++ best practices, as discussed in lecture, and should compile and run correctly using the supplied `makefile` in the standard environment (i.e., the Debian virtual machine). **If you have developed your solution using any other environment, be sure to test it with the VM prior to submission.** There should be no segmentation faults or memory leaks during or after the execution of the program. Please see the accompanying marking criteria for an indicative (but not final) guideline to how you will be evaluated.

Helpful Tips

A few tips that may help you along your journey.

1. Work incrementally – don't try to implement everything at once. Given that the supplied code will compile, you have every reason to test as you develop rather than waiting until the end. In other words, make sure each method works before proceeding to the next.

2. Remember that you can debug your program in VS Code. See Lab3a for a brief guide. Of course, the name of the executable in your `launch.json` file will be different than the example in the lab, but this gives you a great way to inspect your data structures during the program execution. You will need to think carefully about what you would

expect the data to look like, especially given that we are using sentinel nodes (i.e., you should not expect `head` and `tail` for your linked list to have valid data). See **Error! Reference source not found.** for an example of what the `current` variable might look like in a linked list.

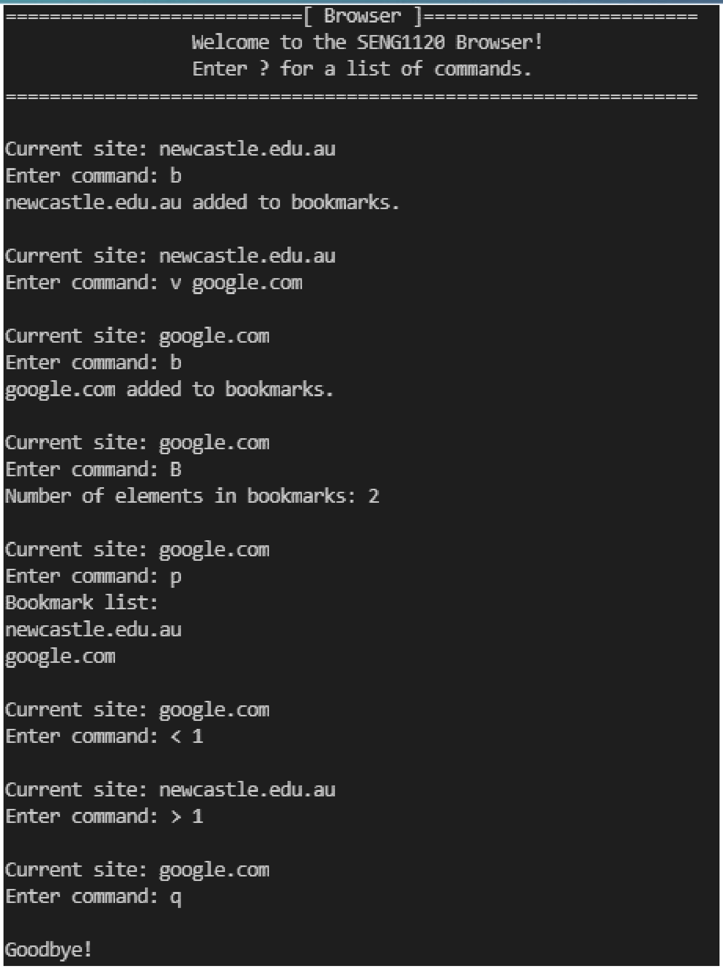
3. You will be expected to supply a program with no memory leaks. You are encouraged to use `valgrind` to assess whether you have any leaking memory in your program.

4. Start early! The longer you wait to start, the less time you will have to complete the assignment.

```
=====================================[ Commands ]=====================================
 v [url]
    Visit the specified URL.
< [steps]
    Move backward the specified number of steps.
> [steps]
    Move forward the specified number of steps.
r [url]
    Remove all history entries for the given URL.
b
    Bookmark/unbookmark the current URL.
c
    Clear the history, resetting to the homepage.
p
    Prints the bookmark list.
H
    Counts the number of elements in the history list.
B
    Counts the number of elements in the bookmark list.
V [index]
    Visits the bookmark with specified index, if it exists.
q
    Quit.
?
    Show this help menu.
======================================================================================
```

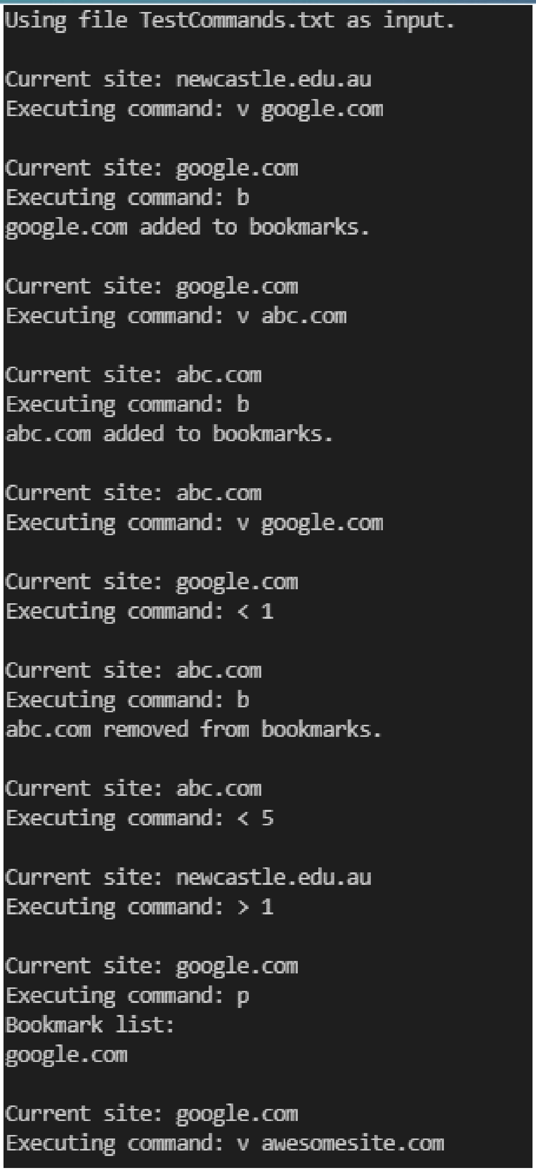
Figure 2. Listing of commands.

Good Luck!



```
===========================[ Browser ]=========================
                 Welcome to the SENG1120 Browser!
                 Enter ? for a list of commands.
===============================================================

Current site: newcastle.edu.au
Enter command: b
newcastle.edu.au added to bookmarks.

Current site: newcastle.edu.au
Enter command: v google.com

Current site: google.com
Enter command: b
google.com added to bookmarks.

Current site: google.com
Enter command: B
Number of elements in bookmarks: 2

Current site: google.com
Enter command: p
Bookmark list:
newcastle.edu.au
google.com

Current site: google.com
Enter command: < 1

Current site: newcastle.edu.au
Enter command: > 1

Current site: google.com
Enter command: q

Goodbye!
```

Figure 3. Sample run in prompt mode.



```
Using file TestCommands.txt as input.

Current site: newcastle.edu.au
Executing command: v google.com

Current site: google.com
Executing command: b
google.com added to bookmarks.

Current site: google.com
Executing command: v abc.com

Current site: abc.com
Executing command: b
abc.com added to bookmarks.

Current site: abc.com
Executing command: v google.com

Current site: google.com
Executing command: < 1

Current site: abc.com
Executing command: b
abc.com removed from bookmarks.

Current site: abc.com
Executing command: < 5

Current site: newcastle.edu.au
Executing command: > 1

Current site: google.com
Executing command: p
Bookmark list:
google.com

Current site: google.com
Executing command: v awesomesite.com
```

Figure 4. Sample run in file mode..