Semester Program 1: Simple Temporal Sequence Editor (STSE)

Data Structures and Analysis of Algorithms

Objectives

- To strengthen student's knowledge of C++ programming
- To give student experience reading and parsing strings of commands
- To give student experience in writing classes
- To give student experience using linked lists

Instructions

Movies, plays, albums, books, etc. all share a handful of common traits; the most prominent among these is they are often modeled as a sequence of events arranged over time. In this program, you will implement a simplistic tool for creating and managing a sequence of events. We will call this simple tool the STSE.

Advanced versions of a tool like this should support a robust set of editing, recording, and playback features. This one will not; for your version of this tool, you will should implement a text-based interface capable of handling the following commands:

exit – exits the program

load <file> - parses the contents of file as if they were entered in the text-based interface

display – displays every event in the sequence, along with their elapsed time

display until <time> - displays all the events until the indicated elapsed time.

find <event> - displays the specified event and its elapsed time, or informs the user that the event doesn't exist.

clear – clears the list of events

length – displays the total elapsed time for the sequence of events

remove <event> - removes the specified event from the sequence, or informs the user that the event doesn't exist.

append <event> <time> - inserts the specified event at the end of the list

prepend <event> <time> - inserts the specified event at the beginning of the list

insert <event> <time> before <loc> - inserts the specified event immediately before the location, or informs the user that loc doesn't exist

insert <event> <time> after <loc> - inserts the specified event immediately after the location, or informs the user that loc doesn't exist

Note: **<event>** and **<loc>** are single words used as names for an event
<time> is an integer representing play length in seconds.

Advanced versions of tools like this also use very robust data storage strategies. Your version should be implemented as a doubly linked list of events; each event should have an associated name and elapsed time.

Guidance

Don't write all of you code in one marathon session; start small and complete the project bit by bit. Start by implementing the node and a few basic methods for the stack; I suggest starting with the default constructors, prepend, and your method for displaying the contents of the list.

Once a few of your core methods work, implement the other methods one at a time. You show also consider implementing and testing the linked list before implementing the CLI with class Tokenizer. It is important to minimize your sources of error.

Grading Breakdown

Point Breakdown	
Structure	12 pts
The program has a header comment with the required information.	3 pts
The overall readability of the program.	3 pts
Program uses separate files for main and class definitions	3 pts
Program includes meaningful comments	3 pts
Syntax	16 pts
Implements Class List correctly	12 pts
Implements Class Node correctly	4 pts
Behavior	72 pts
Program produces the proper outputs for the following	
• Appending an event to the list	6 pts
• Prepending an event to the list	6 pts
• Inserting an event before an existing event	6 pts
• Inserting an event before an unknown event	3 pts
• Inserting an event after an existing event	6 pts
• Inserting an event after an unknown event	3 pts
• Displaying the length of the sequence of events in the list	6 pts
• Displaying the events in the list	6 pts
• Displaying the events in the list up till a give point in time	6 pts
• Searching for and displaying a selected event	6 pts
• Removing an existing event from the sequence	6 pts
• Removing an unknown event from the sequence	3 pts
• Clear the sequence and setting it to an empty sequence	3 pts

• Successfully executing a file using the load command	6 pts
Total Possible Points	**100pts**
Penalties	
Program does NOT compile	-100
Late up to 72 hrs	-10 per day
Late more than 72 hrs	-100

Header Comment

At the top of each program, type in the following comment:

/*

Student Name: <student name>

Student NetID: <student NetID>

Compiler Used: <Visual Studio, GCC, etc.>

Program Description:

<Write a short description of the program.>

*/

Example:

/*

Student Name: John Smith

Student NetID: jjjs123

Compiler Used: Eclipse using MinGW

Program Description:

This program prints lots and lots of strings!!

*/

Assignment Information

Due Date: 9/12/2021

Files Expected:

1. Main.cpp – File containing function main
2. Tokenizer.h - File containing the Tokenizer class definitions.
3. Tokenizer.cpp - File containing the code for the Tokenizer methods.
4. List.h – File containing the List class definitions
5. List.cpp – File containing the code for the List class methods.