CSE 240 – Assignment 7

Maximum points: 50 pts

Topics:

- C/C++ Syntax
- Pointers
- Functions
- Dynamic Allocation of Memory
- Data Structures: Linked Lists
- Object Orientation

Linked List Specifications:

You are to create a Linked List data structure from scratch. This linked list should be Templated so that any data could be stored within it.

For a maximum of B on specifications – you can make the linked list only store Integers.

The primary goal of this assignment is to make a Linked List that you can use and re-use. The Linked List itself is the majority of the specifications grade.

Specifications Scoring Breakdown:

Templated Linked List + Josephus Problem – 100% of specifications

Templated Linked List only – 80% of specifications

Integer Linked List + Josephus Problem – 80% of specifications

Integer Linked List only – 70% of specifications

BIG GIANT NOTE – TEMPLATES AND FILES

When you use a templated type in C++ ALL templated code must be done in the .h file. This means that ALL of your methods will be defined in the .h file as well as your class.

You should still forward declare Classes above then Methods below.

Your .h file should have your LinkedList class and your Node class and the method definitions for both. Remember to use your :: operator correctly.

Feel free to use friendship if needed.

You will create a Linked List Class and a Node Class/Struct

The Linked List should contain the following methods in its public interface:

- Constructor
- Destructor
- AddToFront(T data) – create a node containing T data and add it to the front of the list
- AddToEnd(T data) – create a node containing T data and add it to eh end of the list
- AddAtIndex(T data, int index) – create a node containing T data and add it to the list at index, return boolean for success or failure (optional: you could also return an integer with failure codes since this method can fail multiple ways)
- NextNode – Move the current pointer to the next node, wraps to front if it navigates past the end
- InsertAfterCurrent(T data) – Create a node containing T data and insert it after wherever the current pointer is pointing
- RemoveCurrent() – Delete the current item and return its contents
- RemoveAtIndex(int index) – delete the index # node in the list and return its contents
- RemoveFromFront() – Delete first item and return its contents
- RemoveFromEnd() – Delete last item and return its contents
- RemoveFirst(T data) – find first instance of T data and remove it
- RemoveAll(T data) – find each instance of T data and remove it
- ElementExists(T data) – Returns a T/F if element exists in list
- Find(T data) – Look for data in the list, return a pointer to its node
- IndexOf(T data) – returns an index of the item in the list (zero-based)
- RetrieveFront – returns the data contained in the first node, does not delete it
- RetrieveEnd – returns the data contained in the last node, does not delete it
- Retrieve(int index) – returns the data contained in node # index, does not delete it, returns null if index is out of bounds or data does not exist
- ToArray – Create an array from the contents of the list and return it
- Empty – Empty out the list, delete everything
- Length – How many elements are in the list

More methods private or public should be created as needed to facilitate the functionality of the Interface methods. If you feel your list needs more functionality, feel free to create it.

Node Class

- Constructor
- Destructor
- Getters & Setters

The node class should be fairly rudimentary. It should be templated so you can store anything in it.

Josephus Problem - Linked Lists

Description:

Josephus Problem

"There are people standing in a circle waiting to be executed. After the first man is executed, certain number of people are skipped and one man is executed. Then again, people are skipped and a man is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last man remains, who is given freedom.

The task is to choose the place in the initial circle so that you survive (are the last one remaining)."

-- Wikipedia, http://en.wikipedia.org/wiki/Josephus_problem

Assume that the number of people, P, in the circle may be any number between zero and one hundred.

Assume that every Nth person around the circle is killed each turn, where N is an integer between one and twenty.

Specifications:

- Create an application in C++ that uses a linked list to represent the circle of people, numbered from 1 to P.
- Acquire the values P and N from the user at runtime via console input.
- Output the the individual that survives the mass execution.

Flash Version

- http://webspace.ship.edu/deensley/flash/JosephusProblem.html

There are several Java applet versions of this problem to check your work as you debug.

- http://www.wou.edu/~burtonl/josephus.html
    - This applet lets you choose how many people are in the circle and tells you who survives if every n die

Grading of Programming Assignment

The TA will grade your program following these steps:
(1) Compile the code. If it does not compile, If it does not compile you will receive a U on the **Specifications** in the Rubric.

(2) The TA will read your program and give points based on the points allocated to each component, the readability of your code (organization of the code and comments), logic, inclusion of the required functions, and correctness of the implementations of each function.

Rubric:

	Levels of Achievement						
Criteria	A	B	C	D	E	U	F
Specifications Weight 50.00%	100 % The program works and meets all of the specifications.	85 % The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	75 % The program produces mostly correct results but does not display them correctly and/or missing some specifications	65 % The program produces partially correct results, display problems and/or missing specifications	35 % Program compiles and runs and attempts specifications, but several problems exist	20 % Code does not compile and run. Produces excessive incorrect results	0 % Code does not compile. Barely an attempt was made at specifications.
Code Quality Weight 20.00%	100 % Code is written clearly	85 % Code readability is less	75 % The code is readable only by someone who knows what it is supposed to be doing.	65 % Code is using single letter variables, poorly organized	35 % The code is poorly organized and very difficult to read.	20 % Code uses excessive single letter identifiers. Excessively poorly organized.	0 % Code is incomprehensible
Documentation Weight 15.00%	100 % Code is very well commented	85 % Commenting is simple but solid	75 % Commenting is severely lacking	65 % Bare minimum commenting	35 % Comments are poor	20 % Only the header comment exists identifying the student.	0 % Non existent
Efficiency Weight 15.00%	100 % The code is extremely efficient without sacrificing readability and understanding.	85 % The code is fairly efficient without sacrificing readability and understanding.	75 % The code is brute force but concise.	65 % The code is brute force and unnecessarily long.	35 % The code is huge and appears to be patched together.	20 % The code has created very poor runtimes for much simpler faster algorithms.	0 % Code is incomprehensible

What to Submit?

You are required to submit your solutions in a compressed format (.zip). Zip all files into a single zip file. Make sure your compressed file is labeled correctly - lastname_firstname7.zip.

For this home assignment, the compressed file MUST contain the following:

- Makefile
- README.txt – instructions for using the makefile
- lastname_firstname7.cpp – where you main/josephus problem is
- lastname_linkedList.h – where your linked list/node code is
- any other code/library files you create or use for the sake of the assignment

No other files should be in the compressed folder.

If multiple submissions are made, the most recent submission will be graded, even if the assignment is submitted late.

Where to Submit?

All submissions must be electronically submitted to the respected homework link in the course web page where you downloaded the assignment.

Academic Integrity and Honor Code.

You are encouraged to cooperate in study group on learning the course materials. However, you may not cooperate on preparing the individual assignments. Anything that you turn in must be your own work: You must write up your own solution with your own understanding. If you use an idea that is found in a book or from other sources, or that was developed by someone else or jointly with some group, make sure you acknowledge the source and/or the names of the persons in the write-up for each problem. When you help your peers, you should never show your work to them. All assignment questions must be asked in the course discussion board. Asking assignment questions or making your assignment available in the public websites before the assignment due will be considered cheating.

*The instructor and the TA will **CAREFULLY** check any possible proliferation or plagiarism. We will use the document/program comparison tools like MOSS (Measure Of Software Similarity: http://moss.stanford.edu/) to check any assignment that you submitted for grading. The Ira A. Fulton Schools of Engineering expect all students to adhere to ASU's policy on Academic Dishonesty. These policies can be found in the Code of Student Conduct:*

http://www.asu.edu/studentaffairs/studentlife/judicial/academic_integrity.htm

ALL cases of cheating or plagiarism will be handed to the Dean's office. Penalties include a failing grade in the class, a note on your official transcript that shows you were punished for cheating, suspension, expulsion and revocation of already awarded degrees.