CSE 240 – Assignment 5

Points: 50 pts

Topics:

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

Overall Specifications:

You are to create a Doubly-LinkedList data structure from scratch. This should be Templated so that any data could be stored within it.

You are creating a data structure here from scratch. You may use std::string (#include <string>) if needed. You may use other built-in structures to support your algorithms but you are creating your own LinkedList.

The primary goal of this assignment is to **make** a LinkedList that you can use and re-use. The data structure itself is the majority of the specifications grade.

Specifications Scoring Breakdown:

You are eligible for these points in the **Specifications** part of the rubric if you accomplish:

- Templated Linked List & Zombie Conga makes you eligible for 100% of the Specifications points
- Adjusted grading:
    - A fixed-type Linked List penalizes -15%
    - Not having the Zombie Conga penalizes -15%
        - You should provide a thorough test of your linked list functionality
        - Note: We will be running your Linked List through our own test cases
    - This means you can create an Integer based Linked List with a **THOROUGH** test file and still be eligible for 70% of the Specifications points.
        - A Templated Linked List without Zombie Conga is eligible for 85% of the Specifications points
        - A Zombie Linked List with a Zombie Conga is eligible for 85% of the Specifications points

Eligibility level	Templated Linked List	Fixed Type List (Zombie/int)	Zombie Conga	Thorough Test
100%	X		X	
85%	X			X
85%		X	X	
70%		X		X

The rest of the rubric is graded as normal.

Programming Assignment:

Instructions:

In this assignment, you will create your code from scratch. Stay within the bounds of what we've covered in class. You are to use functions in this assignment.

All of the algorithms should be coded by you. Code the algorithms from scratch. Don't just find a library to solve the problems for you.

Filename:

Filenames are dependent on which options you complete:

Templated Linked List (with Zombie Conga)

- hw5_<lastname>_<firstname>.cpp
- linkedlist_<lastname>.hpp
- zombie_<lastname>.h
- zombie_<lastname>.cpp
- Makefile

Untemplated Linked List (with Zombie Conga)

- hw5_<lastname>_<firstname>.cpp
- linkedlist_<lastname>.h
- linkedlist_<lastname>.cpp
- zombie_<lastname>.h
- zombie_<lastname>.cpp
- Makefile

Templated Linked List (NO Zombie Conga)

- linkedlist_<lastname>.hpp
- hw5_<lastname>_<firstname>.cpp
- Makefile

Untemplated Linked List (NO Zombie Conga)

- linkedlist_<lastname>.h
- linkedlist_<lastname>.cpp
- hw5_<lastname>_<firstname>.cpp
- Makefile

Makefile:

- The makefile is required so we can easily compile your code
- Your makefile should produce an executable named `exe`

Library allowances:
You may use std::string (#include <string>) if necessary.

Code Style:
For yours, the instructors', and the graders' sake, keep your code well organized. Use proper indentation, variable names, and segmentation.

** BIG GIANT NOTE – TEMPLATES AND FILES **

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

You should still forward declare Classes above and then implement the Methods below.

Your .hpp file should have both, your LinkedList and Node classes, and their method definitions. Remember to use your :: operator correctly. Feel free to use friendship if needed.

Tutorial – Exception Handling

With data structures, often we must deal with PEBCAK errors (Problem Exists Between Chair and Keyboard). If we include a method in a Linked List that has the user add "at an index" we must account for that user giving a bad value, such as a negative number or one that is too large.

C++ has added exception handling to the std:: namespace and it is fairly similar to Java exception handling.

Syntax:

```
try
{
      //code goes here
}
catch(/*exception type*/)
{
      //catch code
}
```

We raise an exception with the *throw* command:

```
throw <message>;
```

A big difference between Java and C++ is that you can throw just about anything.

```
throw "an error happened";
throw std::out_of_range();
throw -1;
```

Examples:

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

int divide(int numerator, int denominator)
{
    if(denominator = 0)
    {
        throw std::runtime_error("Can't
divide by zero!");
    }

    return numerator / denominator;
}

int main(int argc, char const *argv[])
{
    int numerator;
    int denominator;
    cout << "Enter two values: ";
    cin >> numerator >> denominator;

    try
    {
        int test = divide(numerator,
denominator);
    }
    catch(const std::runtime_error& err)
    {
        std::cerr << err.what() << '\n';
    }

    return 0;
}
```

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

int divide(int numerator, int denominator)
{
    if(denominator = 0)
    {
        throw "Can't divide by Zero!";
    }

    return numerator / denominator;
}

int main(int argc, char const *argv[])
{
    int numerator;
    int denominator;
    cout << "Enter two values: ";
    cin >> numerator >> denominator;

    try
    {
        int test = divide(numerator,
denominator);
    }
    catch(const char* messsage)
    {
        std::cerr << messsage << '\n';
    }

    return 0;
}
```

std namespace exceptions

Exception	**Description**
std::exception	Exception and parent class of all standard C++ exceptions.
std::bad_alloc	Generally thrown by new.
std::bad_cast	Generally thrown by dynamic_cast.
std::bad_typeid	Generally thrown by typeid.
std::bad_exception	Useful device to handle unexpected exceptions.
std::logic_failure	Can be detected by reading code.
std::runtime_error	Cannot be detected by reading code.
std::domain_error	Thrown when using a mathematically invalid domain.
std::invalid_argument	Thrown when using invalid arguments.
std::length_error	Thrown when a large std::string is created.
std::out_of_range	Thrown by the at method.
std::overflow_error	Thrown when a mathematical overflow occurs.
std::range_error	Thrown when attempting to store an out-of-range value.
std::underflow_error	Thrown when a mathematical underflow occurs.

For the sake of Exception Handling in this assignment, you should be able to use the std namespace exceptions

Zombie Conga Party!

Description:

You are going to create a silly Zombie Conga Line using your Linked List.

Each node is going to store a Zombie in it. Zombies can be Red, Yellow, Green, Blue, Magenta and Cyan.

Every turn you will randomly generate a Zombie object and an action. You will then perform that action using the Zombie object as the parameter for that action.

Linked List

You will create a Doubly-LinkedList Class and a Node Class.

The LinkedList should contain the following methods in its public interface:
NOTE: T stands for the template data type, so T data means the variable of type T (whatever T is)

- Constructor
- Destructor
- AddToFront(T data):void – create a node containing T data and add it to the front of the list
- AddToEnd(T data):void – create a node containing T data and add it to the end of the list
- AddAtIndex(T data, int index):void – create a node containing T data and add it to the list at index. The new node containing the data will be the #index node in the list. *Throws std::out_of_range* exception if index is less than zero or greater than the size of the list
- AddBefore(Node<T>*, T data):void – create a node containing T data and add it before a particular node
- AddAfter(Node<T>*, T data):void – create a node containing T data and add it after a particular node
- RemoveFromFront():T – Delete first item and return its contents
- RemoveFromEnd():T – Delete last item and return its contents
- RemoveTheFirst(T data):void – find first instance of T data and remove it
- RemoveAllOf(T data):void – find each instance of T data and remove it
- RemoveBefore(Node<T>*):T – delete the node before a particular node, return its contents
- RemoveAfter(Node<T>*):T – delete the node after a particular node, return its contents
- ElementExists(T data):bool – Returns a T/F if element exists in list
- Find(T data):Node<T>* – Look for data in the list, return a pointer to its node
- IndexOf(T data):int – returns an index of the item in the list (zero-based), returns -1 if the item is not in the list.
- RetrieveFront:T – returns the data contained in the first node, *does not delete it*
- RetrieveEnd:T – returns the data contained in the last node, *does not delete it*
- Retrieve(int index):T – returns the data contained in node # index, *does not delete* it. *Throws std::out_of_range* exception if index is less than zero or greater than the size of the list
- PrintList:void – Loop through each node and print the contents of the Node
- Empty:void – Empty out the list, delete everything
- Length:int – How many elements are in the list

Linked List contd

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

- Properties
  - -data:T
  - -next:Node<T>*
  - -previous:Node<T>*
- Methods
  - +Node()
  - +Node(T)
  - +Node(T, Node<T>*)
  - +getData():T
  - +setData(T):void
  - +getNext():Node<T>*
  - +setNext(Node<T>*):void
  - +getPrevious():Node<T>*
  - +setPrevious(Node<T>*):void
- Friend (optional)
  - LinkedList class

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

Zombie Class

- Properties
  - -type:char
- Methods
  - +Zombie()
  - +Zombie(char)
  - +getType():char
  - +operator==:bool
- Friend
  - ostream& operator<<(ostream&, const Zombie&)

The Zombie class is very simple, don't give it any more work to do.

Conga (class?)

You can create a class for your Conga or you can create a set of functions.

Spelling it out for you:

You should create the following classes:

1. LinkedList
2. Node
3. Zombie

Your Nodes should store Zombies.

Your LinkedList is made of Nodes.

Actions

If you chose to make a Conga class, then these actions should be methods of your Conga class. Methods don't need the list passed in since the linked list should be a property of the Conga class.

If you are not making a Conga class, then these actions should be functions.

- Engine!
  - This zombie becomes the first Zombie in the conga line
  - Suggested function:
    - void engine_action(LinkedList<Zombie> list, Zombie randomZomb)
- Caboose!
  - This zombie becomes the last zombie in the conga line
  - Suggested function:
    - void caboose_action(LinkedList<Zombie> list, Zombie randomZomb)
- Jump in the Line!
  - This zombie joins the conga line at position X where X <= length of the linked list
  - Suggested function:
    - void jump_in_action(LinkedList<Zombie> list, Zombie randomZomb)
- Everyone Out!
  - Remove all matching zombies from the linked list
  - Suggested function:
    - void everyone_out_action(LinkedList<Zombie> list, Zombie randomZomb)
- You Out!
  - Remove the first matching zombie from the linked list
  - Suggested function:
    - void you_out_action(LinkedList<Zombie> list, Zombie randomZomb)
- Brains!
  - Generate two more matching Zombies and add one to the front (engine_action), one to the end (caboose_action) and one to the middle (round down).
  - Suggested function:
    - void brains_action(LinkedList<Zombie> list, Zombie randomZomb)
- Rainbow Brains!
  - Perform an engine_action on the zombie that was generated
  - Add one of each zombie color to the end via caboose_action in this order: Red, Yellow, Green, Blue, Cyan, Magenta
  - Suggested function:
    - void rainbow_action(LinkedList<Zombie> list, Zombie randomZomb)
- Making new Friends!
  - Find the first Zombie of this color in line.
    - Do a coin flip - if rand() % 2 == 0 then insert before, else insert after.
  - If no Zombie of that color exists, then perform caboose_action on the zombie
  - Suggested function:
    - void friends_action(LinkedList<Zombie> list, Zombie randomZomb)

Notes
These actions are external to the Linked List. They are accomplished by calling Linked List methods.

These actions are not part of your Zombie class. Zombies are very simple classes and have no responsibility for Conga actions.

Setting up the List:
Set up the initial Conga Line by running these actions:

1. Run a Rainbow Brains! Action
2. Run 3 Brains actions

User Interface:

Command line argument:
Add a command line option -s <integer> to allow the user to provide a seed for the random number generator. This seed value should be given to srand(<int>).

If no -s option is given, then seed the random number generator with time:
srand(time(0));

Console input:

- Ask the user how many rounds they want to run.
- Run the conga party for that many rounds
  - Start with round 0
  - Every time `round % 5 == 0` – delete the first and last zombie in the linked list
  - Choose your action with rand() % 8
    - Keep the actions in the order show previously
  - Choose your zombie with rand() % 6
    - Use the order: Red, Yellow, Green, Blue, Cyan, Magenta
  - Run that action with your chosen zombie
- If the conga line ever empties completely due to an action tell the user that the Party is Over.
- Once the number of rounds has finished. Ask the user if they want to continue the party or end.
- If they choose to continue ask them for a new number of rounds to run.

Output:

Each round you'll output the entire Linked List. You can represent each zombie as a single character corresponding to their color (R, Y, G, B, M, C).

You'll show the zombie generated and the action generated.

Then you'll show the outcome of the action.

The output should follow this pattern (see sample output for example):

```
Round: <round number>
Size: <list size> :: [<R|Y|G|B|M|C>]= …
New Zombie: [<R|Y|G|B|M|C>] -- Action: [<Engine!|Caboose!|Jump In!|Everyone Out!|You
Out!|Brains!|Rainbow!|New Friends!>]
The conga line is now:
Size: <list size> :: [<R|Y|G|B|M|C>]= …
**************************************************
```

Hints:

- Start by building an Integer version of the doubly linked list. It will get you points if you can't get other things to work and it will be easier to transition to the template after getting all the functionality of the data structure solid.
- Build robust constructors and use them! Your Node and your Zombie will benefit highly from good constructors.
- Do yourself a favor and overload the == operator in your Zombie. It will make life easier!
- Overload the cout for your Zombie. It'll make your printList method easier.
- You might consider making the Conga its own class so you can make each of the actions a method in the conga class.

Extra Credit:

+3 – Color your Zombies in the output.

If you use termcolor (https://github.com/ikalnytskyi/termcolor) you must do std::cout << termcolor::colorize at the beginning of your program for the color to display correctly on grade scope, otherwise the graders won't see it and you won't get credit.

Sample output:

```
Round: 0
Size: 16 :: [R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]
New Zombie: [B] -- Action: [Engine!]
The conga line is now:
Size: 17 :: [B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]
**************************************************

Round: 1
Size: 17 :: [B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]
New Zombie: [M] -- Action: [Engine!]
The conga line is now:
Size: 18 :: [M]=[B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]
**************************************************

Round: 2
Size: 18 :: [M]=[B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]
New Zombie: [C] -- Action: [Caboose!]
The conga line is now:
Size: 19 :: [M]=[B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]=[C]
**************************************************

Round: 3
Size: 19 :: [M]=[B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]=[C]
New Zombie: [Y] -- Action: [Rainbow!]
The conga line is now:
Size: 26 ::
[Y]=[M]=[B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]=[C]=[R]=[G]=[B]=[Y]=[M]
=[C]
**************************************************

Round: 4
Size: 26 ::
[Y]=[M]=[B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[M]=[C]=[G]=[M]=[R]=[C]=[R]=[G]=[B]=[Y]=[M]
=[C]
New Zombie: [Y] -- Action: [Brains!]
The conga line is now:
Size: 29 ::
[Y]=[Y]=[M]=[B]=[R]=[M]=[G]=[R]=[R]=[G]=[G]=[R]=[M]=[B]=[Y]=[Y]=[M]=[C]=[G]=[M]=[R]=[C]=[R]=[G]=[B]
=[Y]=[M]=[C]=[Y]
**************************************************
```

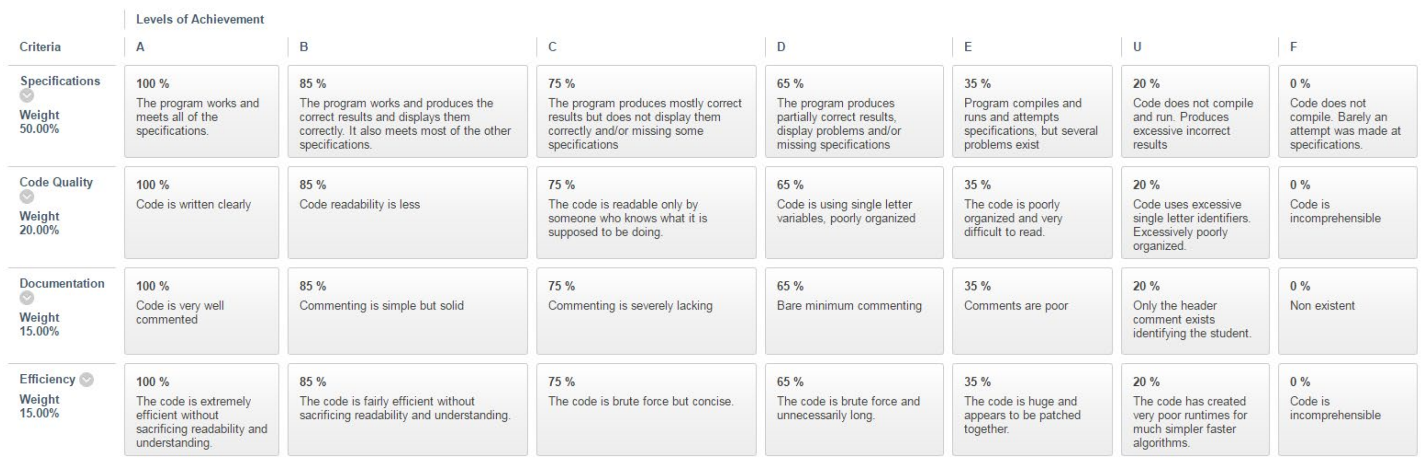
… continue for the # of rounds the user asks for

Grading of Programming Assignment

Turn your files into Gradescope. Make sure to follow the file naming guide so your project can be properly compiled and tested.

This project will have an auto grading component.

Rubric:



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_firstname4.zip.

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

- Makefile
- README.txt – instructions for using the makefile
- Your code files:
  - lastname_firstname_assn4.cpp
  - lastname_bst.h / lastname_list.h
  - If you didn’t template, also include the .cpp files for your .h files
  - Any other files that you created/need
- any other code/library files you create or use for the sake of the assignment
  - such as zombie.h/.cpp, conga.h/.cpp, wordentry.h/.cpp

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.