Program 5

Submit Assignment

Due Sunday by 11:59pm **Points** 60 **Submitting** a file upload **File Types** tar
Available after Mar 2 at 12am

C++ Linked List Implementation

Motivation

As we discussed in class, the data structures that you use to implement your program can have a profound impact on it's overall performance. A poorly written program will often need much more RAM and CPU time then a well-written implementation. One of the most basic data structure questions revolves around the difference between an array and a linked list. After you finish this assignment you should have a firm understanding of their operation.

NOTE: The submission guidelines for this assignment are slightly different from the previous ones. Be sure to understand the readme.txt requirement.

Problem Statement

For this assignment, you will implement a linked list class using pointers and object-oriented programming. Although the C++ STL (Standard Template Library) offers a linked list implementation, you must implement this program "from scratch" and cannot simply utilize the existing STL offerings (<list> or <forward_list>).

Your linked list will be designed to contain signed integers of type **int**.

Required Classes

You must implement the classes shown below (as well as the exact member functions that are listed). Note: It is okay to add additional functions or variables as desired. You cannot add extra parameters to the functions that are listed.

```
class Node {
public:
   int val;    // the value that this node stores
   Node *next; // a pointer to the next node in the list
   // you can add constructors or other functionality if you find it useful or necessary
}
```

Note: Node is being used akin to a struct (with public member variables). This is intentional so that you can easily modify the member variables from within the Linked_List class.

```
class Linked_List {
private:
   unsigned int length; // the number of nodes contained in the list
   Node *head; // a pointer to the first node in the list
   // anything else you need...
public:
   int get_length();
   // note: there is no set_length(unsigned int) (the reasoning should be intuitive)
   void print(); // output a list of all integers contained within the list
   void clear(); // delete the entire list (remove all nodes and reset length to 0)
   unsigned int push_front(int); // insert a new value at the front of the list (returns the new length of th
e list)
   unsigned int push_back(int); // insert a new value at the back of the list (returns the new length of the
 list)
   unsigned int insert(int val, unsigned int index); // insert a new value in the list at the specified index
(returns the new length of the list)
   void sort_ascending(); // sort the nodes in ascending order. You must implement the recursive Merge Sort a
lgorithm
   // Note: it's okay if sort_ascending() calls a recursive private function to perform the sorting.
   void sort_descending(); // sort the nodes in descending order
   // you can add extra member variables or functions as desired
}
```

Note that the sort_ascending() function must be implemented using the recursive **Merge Sort algorithm (https://en.wikipedia.org/wiki/Merge_sort)** . sort_descending() can utilize Merge Sort or a different algorithm (see extra credit).

You are also required to implement a function that counts the number of prime numbers within a Linked_List. This can be written as part of the Linked_List class or in some other class. For our purposes, a negative number is never considered to be prime.

Application Description

Once you have implemented the fundamental building blocks of a linked list you will use this functionality to build a simple application. Implement a program to replicate the following behavior:

Note: All of the program functionality must be implemented using your linked list. You should not be storing or copying the user input into an array, vector, or other types of data structures.

```
Please enter a number: 146
Do you want another num (y or n): y
Enter a number: 30
Do you want another num (y or n): y
Enter a number: 73
Do you want another num (y or n): y
Enter a number: 10
Do you want another num (y or n): y
```

```
Enter a number: -31
Do you want another num (y or n): n
Sort ascending or descending (a or d)? a
Your linked list is: -31 10 30 73 146
You have 1 prime number(s) in your list.
Do you want to do this again (y or n)? n
```

Other Program Requirements

- When sorting nodes, you may not swap the values between nodes and must change the pointers on the nodes to swap them.
- You must have a class for each of the following things: `Linked_List`, `Node`. As usual, it is completely fine to implement additional classes if you can defend their existence.
- Your program must be factored into interface, implementation, and application. Specifically, you should have one header file and one implementation file for each class, and you should have a single application file containing your `main()` function. If you choose to implement a template class, it is acceptable to insert both the header and implementation information into a corresponding .hpp file.
- You must provide a working Makefile in the TAR archive.
- The Big 3 must be implemented as appropriate.
- Your program may not have any memory leaks. It is recommended that you use Valgrind to occasionally test your program as you develop it (even when everything seems to be working).
- Segmentation faults/core dumps are not allowed (e.g. your program crashes).
- If your implementation does not follow the spirit of the assignment you will lose 30 points! You must implement a linked list for the storage of the data.
- This program is due immediately prior to the start of finals week. As a result, you will not schedule the usual demo with a TA. Instead, the TAs will grade this homework on their own. For this reason, your TAR file is required to include an extra text file named **readme.txt** which will contain your name, a description of the program, and any additional information that the TA should know when grading your assignment. In addition, if you implement the extra credit work, the readme.txt file **MUST** list the extra credit work that you did.

Extra Credit

In addition to the requirements above, you may earn extra credit as follows:

- (5 points) Implement the assignment using a template class. Use the template class to implement the original assignment (with type **int**). Utilize the exact same template class to implement a second version that operates with input of type **unsigned int**. When your application executes, give the user the option to utilize the **int** or **unsigned int** version of your code. If you implement this extra credit, be sure to describe this in the readme.txt file. Note: if your code is implemented correctly, even a gigantic prime number (such as 4294963943) should be detected (as long as it fits into an **unsigned int**).

- (5 points) Implement sort_descending() using a recursive Selection Sort algorithm. If you implement this extra credit, be sure to describe this in the readme.txt file.

Implementation Details

For this assignment you have additional freedom to design your own implementation. There are certain baseline requirements as specified in this document. However, any additional member functions and member variables are up to you, as long as the spirit of the assignment is followed.

PLEASE... take the time to design your program on paper. Use scratch paper with flow charts or pseudo-code. In particular, think of how your program will work when the various nodes are added or removed. What needs to happen when a link is added? What happens when a link is deleted? Consider the edge cases that arise when you add or delete nodes at the beginning and end of the list. These are the sorts of questions that should be answered during the design stage, long before you start writing code.

Taking the time to do a thorough program design can save you hours of frustrating debugging time!

Programming Style/Comments

In your implementation, make sure that you include a file header for every .cpp, .h, or .hpp file. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. As usual, please refer to the style guidelines for this class if you have any questions regarding the proper formatting.

```
/******************************************************
** Program: linked_list.cpp
** Author: Your Name
** Date: 03/15/2020
** Description:
** Input:
** Output:
******************************************************/
```

When you compile your code, it is acceptable to use C++11 functionality in your program. For this final programming assignment you are welcome to research and utilize smart pointers if you desire (although they may complicate your design).

In order to submit your homework assignment, you must create a TAR archive that contains your readme.txt, source code, and Makefile. This tar file will be submitted here to Canvas. In order to create the tar file, use the following command:

```
tar -cvf assign5.tar list_of_any_.h,_.cpp,_or_.hpp_files readme.txt Makefile
```

Special Notes

This is the final programming assignment for this class and you are not required to demo this code. Instead, the TAs will grade your code on their own during finals week. This is why it is important for you to create a readme.txt with any additional information that you feel the TAs should know.

C++ Linked List Rubric

Criteria	Ratings	Pts
This criterion is linked to a Learning Outcome Program Header / Good indentation & Use of whitespace / Function Documentation At a minimum, header should contain author's name (1pt) and a description of the program. (1pt) Conditional blocks of code should always be indented. (2pt) Every function contains it's own initial block comment (or multiple lines of comments prior to the function definition) that provides the reader with an explanation of the function's purpose (2 pts). -1 pt for each function that is missing a header (up to 3 point penalty).		6.0 pts
Included readme.txt TAR file contains readme.txt with author's name, a description of the program, and any additional information that the TA should know. Any extra credit work should also be described here.		3.0 pts
Determine Prime numbers in a linked list Correctly implement a function that counts the number of prime numbers within a Linked_List. The student code can report the total number of prime numbers or the unique number of primes.		6.0 pts
Correct function behaviors (1) get_length(), print(), clear(), push_front(), push_back() 3 pts each		15.0 pts
Correct function behaviors (2) insert() 3 pts for correctly insert a node, 2 pts for returning the new length of the list		5.0 pts
Correct function behaviors (3) sort_ascending(), sort_descending() 8 pts each (-6 pts if sort_ascending() is not implemented using recursive merge sort algorithm)		16.0 pts
Application Implement an application program to replicate the behavior provided in the requirement.		6.0 pts
Use Big 3 appropriately It's okay if students don't "need" the big 3 in their application. A specific class does not "need" the big three if that class does not utilize dynamic memory.		3.0 pts
Extra Credit 1 (5 pts) Implement the assignment using a template class. Program must allow user to select and use "unsigned int" mode. Only receives 3 out of 5 pts if program does not detect 4294963943 as prime.		0.0 pts

Criteria	Ratings	Pts
Auto Deductions -30 Not utilizing linked list to follow the spirit of the assignment -12 swap just values but not pointers when sorting nodes; -6 did not use required classes and functions with exact same names (-1 pt for each unmatched name found, 6 points at max) -6 memory leaks (use valgrind and sliding scale for how many/bad it is), -6 no makefile or tar, -6 use of globals or libraries not introduced in class, -6 functions over 20 lines (sliding scale for how many/bad it is), -12 didn't separate files		0.0 pts
Extra Credit 2 (5 pts) sort_descending() is implemented using a recursive Selection Sort algorithm. Non-recursive Selection Sort can receive 1 pt.		0.0 pts
Total Points: 60.0		