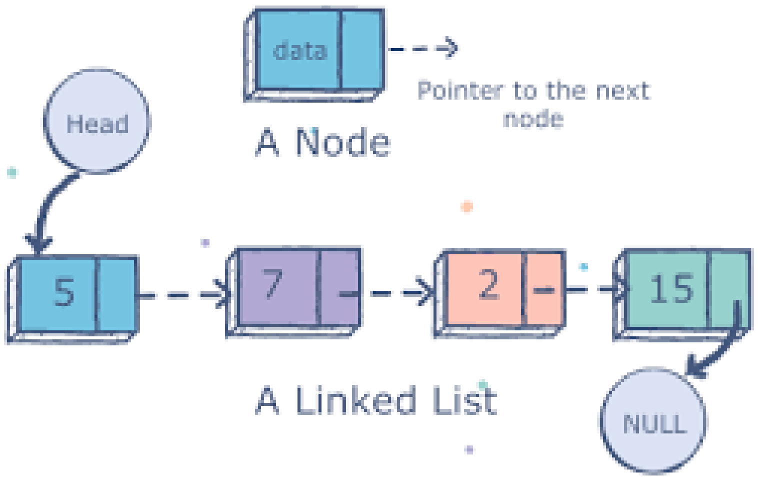
CS 162-010 Assignment #5 Recursion & Linked List

Design Document due: Sunday, 5/29/2022, 11:59 p.m. (Canvas)

Assignment due: Sunday, 6/5/2022, 11:59 p.m. (TEACH)

NO DEMO NEEDED FOR THIS ASSIGNMENT



Goals:

- Practice good software engineering design principles:
    - Design your solution before writing the program
    - Develop test cases before writing the program
- Practice Object-Oriented Programming
- Implement recursion
- Implement a singly linked list with basic functionalities
- Implement different sorting algorithms and analyze complexity
- Practice file separation and create Makefile
- Use functions to modularize code to increase readability and reduce repeated code

Motivation

As we discussed in class, the data structures that you use to implement your program can have a profound impact on its 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 homework you should have a firm understanding of their operation.

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

Introduction & Problem Statements

This assignment has two parts. In part 1, you will be implementing fractals using recursion. In part 2, you will use the skeleton code to implement a singly linked list.

Part 1: Warm-up: Recursive Fractals

Examine this pattern of asterisks and blanks, and write a recursive function called **`pattern()`** that can generate patterns such as this:

pattern(3, 2);

```
      *
    * * *
      *
```

pattern(5, 1);

```
      *
    * * *
      *
  * * * * *
      *
    * * *
      *
```

pattern(7, 0);

```
      *
    * * *
      *
  * * * * *
      *
    * * *
      *
* * * * * * *
      *
    * * *
      *
  * * * * *
      *
    * * *
      *
```

With recursive thinking, the function needs only about 10 lines of code (including two recursive calls). Your function prototype should look like this:

// Description:
// The longest line of the pattern has **`n`** stars beginning in column **`col`** of the output.
// Precondition: **`n`** is an positive odd number.
// Postcondition: A pattern based on the above example has been printed.
`void pattern(int n, int col);`

No error handling needed for this part. Assume that user will provide you a positive odd int as **n**, and a non-negative int as **col**.

Hint: Think about how the pattern is a fractal. Can you find two smaller versions of the pattern within the large pattern? Here is some code that may be helpful within your function:

```
    // A loop to print exactly col columns
    for (int i = 0; i < col; i++) cout << "  ";

    // A loop to print n asterisks, each one followed by a space:
    for (int i = 0; i < n; i++) cout << "* ";
```

Part 2: Linked List Implementation

In part 2, you will implement a linked list class using pointers and object-oriented programming as well as analyzing the complexity of the algorithms that you used when implementing the linked list. 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 (i.e., the use of <list> or <forward_list> are not allowed!).

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

Here are the files you can start with:

wget http://classes.engr.oregonstate.edu/eecs/spring2022/cs162-010/assignments/linked_list.zip

Required Class:

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)
   void push_front(int); // insert a new value at the front of the list
   void push_back(int); // insert a new value at the back of the list
   void insert(int val, unsigned int index); // insert a new value in the list at the specif
ied index
   void sort_ascending(); // sort the nodes in ascending order. You must implement the recur
sive Merge Sort algorithm
   // 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
};

unsigned int count_prime(Linked_List&); // count and return the number of prime numbers with
in the list
```

Note that the `sort_ascending()` function must be implemented using the recursive Merge Sort algorithm . `sort_descending()` can utilize Merge Sort or a different algorithm (see extra credit).

You are also required to implement `count_prime()` function that counts the number of prime numbers within a `Linked_List`. For our purposes, a negative number is never considered to be prime.

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 homework 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

---

Testing and `README.txt` Description

Once you have implemented the fundamental building blocks of a linked list, compile your code with `test_linked_list.cpp` and run it.

Instead of a demo, you will create `README1.txt` and `README2.txt` files for part 1 and part 2 that include the following information:

1. Your name and ONID

2. **Description**: One paragraph advertising what your program does (for a user who knows nothing about this assignment, does not know C++, and is not going to read your code). Highlight any special features.
3. **Instructions**: Step-by-step instructions telling the user how to compile and run your program. Each menu choice should be described. If you expect a certain kind of input at specific steps, inform the user what the requirements are. Include examples to guide the user.
4. **Limitations**: Describe any known limitations for things the user might want or try to do but that program does not do/handle.
5. **(Part 2 only) Extra credit**: If your program includes extra credit work, describe it here for the user.
6. **(Part 2 only) Complexity analysis**: For each of the following function, explain the algorithm you used and the Big O
    a. sort_ascending()
    b. sort_descending()
    c. count_prime()

Here is an example README.txt file (with first 5 items) for a hypothetical assignment (not this one):

http://classes.engr.oregonstate.edu/eecs/spring2022/cs162-010/assignments/README_example.txt

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. 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 zip archive.
- The Big 3 must be implemented as appropriate.
- Your program may not have any memory leaks. I strongly recommend 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 50 points! You must implement a linked list for the storage of the data.
- This program is due immediately prior to the start 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 zip file is required to include an extra text file named **README.txt** (see description above)

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 program header. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Make sure you review the style guidelines for this class , and begin trying to follow them, i.e. don't align everything on the left or put everything on one line!

```
/*******************************************************
** Program: linked_list.cpp
** Author: Your Name
** Date: 05/30/2022
** Description:
** Input:
** Output:
*******************************************************/
```

Design Document – Due Sunday 5/29/2022, 11:59pm on Canvas
Refer to the Example Design Document – Example_Design_Doc.pdf

You need to provide a design for both parts/programs!

Understanding the Problem/Problem Analysis:

- What are the required classes, member variables for each class, etc.?
- What assumptions are you making?
- What are all the tasks and subtasks in this problem?
- How would you test each functionality of the linked list you implemented?

Program Design:
At this point, I highly recommend you create a design using both flowcharts and pseudocode!!!

- What does the overall big picture of the problem? (Flowcharts)
  - How would you modularize the program?
  - How does each function interact with each other?

- What each function looks like? (Pseudocode)
    - What is the task of this function?
    - What are the steps to get the task done?
- What kind of bad input are you going to handle?

Based on your answers above, list the **specific steps or provide a flowchart** of what is needed to create. Be very explicit!!!

Program Testing:
Create a test plan with the test cases (bad, good, and edge cases). What do you hope to be the expected results?

- What are the good, bad, and edge cases for ALL input in the program? Make sure to provide enough of each and for all different inputs you get from the user.

Electronically submit your Design Doc (.pdf file!!!) by the design due date, on Canvas.

Program Code – Due Sunday, 6/5/2022, 11:59pm on TEACH

Additional Implementation Requirements:

- Your user interface must provide clear instructions for the user and information about the data being presented
- Your program must catch all required errors and recover from them.
- You are not allowed to use libraries that are not introduced in class, more specifically, you may not use the <algorithm> or <list> library in your program. Any searching or sorting functionality must be implemented "manually" in your implementation.
- Your program should be properly decomposed into tasks and subtasks using functions. To help you with this, use the following:
    - Make each function do one thing and one thing only.
    - No more than 15 lines inside the curly braces of any function, including main(). Whitespace, variable declarations, single curly braces, vertical spacing, comments, and function headers do not count.
    - Functions over 15 lines need justification in comments.
    - Do not put multiple statements into one line.
- No global variables allowed (those declared outside of many or any other function, global constants are allowed).
- No goto function allowed
- You must not have any memory leaks
- You program should not have any runtime error, e.g. segmentation fault
- Make sure you follow the style guidelines, have a program header and function headers with appropriate comments, and be consistent.

Compile and Submit your assignment

When you compile your code, it is acceptable to use C++11 functionality in your program. In order to support this, change your `Makefile` to include the proper flag.
For example, consider the following approach (note the inclusion of **`-std=c++11`**):

```
g++ -std=c++11 <other flags and parameters>
```

In order to submit your homework assignment, you must create a **zip file** that contains your `.h, .cpp, readme.txt,` and `Makefile` files. This zip file will be submitted to TEACH . In order to create the zip file, use the following command:

```
zip assign5.zip <list of all .h and .cpp files> makefile readme.txt
```

Special Notes

This is the final programming assignment for this course, 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.