CISC 3130 Assignment 1: Linked Lists

For now, submit these solutions on BB. Submit a single picture showing that all of your code runs as well (use multiple terminals). Once you finish this, I recommend putting it up on GitHub.

1. A regular linked list contains a head, nodes with data and a pointer to the next node as well as a tail. For this assignment we will be changing the code that creates a linked list in order to get a new and interesting data structure. **Before beginning the below questions, implement a linked list on your own** and make sure you understand what you did (don’t just copy and paste). Your linked list only needs an insert function and a traverse function at this point.
2. There is a type of linked list called a circular linked list, where the tail of the linked list is also the head (in graph theory this is called a cycle or circuit btw). Do we need to modify a linked list’s implementation code to get this behavior? If we do, how do we change the code and if we don’t, what should we do to get a circular linked list?
3. Change the node structure in the linked list so that it accepts two pointers instead of one: the first pointer will point to the next node on the list, the second pointer will point to the node after the next node.
   1. Create a traversal function which traverses through the list by skipping every odd node (the first part of this question is a major help here) and printing out the data values in the even nodes.
   2. Create a traversal function which prints the value in the current node and then prints the value in the node that is 3 spaces from the current node (array positions: 0, 3, 6, 9 etc.).
4. Modify 3 so that one pointer points to the node before the current node and the other points to the node after the current node. This is called a doubly linked list.
   1. Write a traversal function which traverses the list and once it reaches the tail, it traverses the list in reverse.
      1. What would happen if your list was circular?
5. Modify your linked list so that it now has these features:
   1. There is a way to delete nodes based on what data value is in the node. If the value doesn’t exist return “this data value doesn’t exist”.
   2. A way to insert data in a sorted order: if you insert a node into the list, it’ll traverse the list and find the place where the node should go based on its data value. [ex: node\_to\_insert = 5 -- current\_list = 3,4,6,7,11 -- this will insert the node between 4 and 6]. Note that the pointer values will need to change.