Phuc Hong Le

CSS 342

Professor Ahmed Awad

November 25, 2017

**Assignment 5 Documentation**

**The member of list1 before initialize:**

**The linked list is currently empty.**

**The member of list1 after the insertion:**

**20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |**

**The member of list2 after the append:**

**1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |**

**The member of list after using copy constructor:**

**20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |**

**The member of list2 after removing some data:**

**2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |**

**The member of list1 after removing some data:**

**20 | 18 | 17 | 15 | 14 | 13 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |**

**The member of list1 after iteratively reversing all the data:**

**1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 13 | 14 | 15 | 17 | 18 | 20 |**

**The member of list2 after iteratively reversing all the data:**

**9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |**

**The member of list1 after recursively reversing all the data:**

**1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 13 | 14 | 15 | 17 | 18 | 20 |**

**Program ended with exit code: 0**

Design and approach for solving problem:

INSERT:

Inserting to the list by the position and the data given in the parameter. I approached this problem by splitting the insert into 4 different situations.

1. The first one is the position at the first node of the list where I just simply change the head Node to this new Node and set the next of the new Node to the previous head Node.
2. The second situation is when it inserts at the end of the list which I changed the tail Node to the new Node and set the next Node to Null. Then, I will set the tail Node to be that newly inserted Node.
3. The third one is insertion at any position. I have to make the list iterate through other Node in order to reach the position Node. This is where I set the new Node’s previous Node as the Node before the position Node and the new Node’s next Node as the Node at the position. Then I have to set up the next and previous Node of the next Node and the previous Node again for appropriate format.
4. The function will return 0 if successfully insert. If not, it will return 1 because of the conditions in the fourth situation.

APPEND:

Appending a data is inserting the data at the end of the list. I make the append Node’s next Node as Null because we will be putting it in the end of the list. Then I will set the data for the append Node. After that, I set the current tail Node’s next Node (which is Null at the moment) to be the append Node and set the tail Node to that append Node. The other possibility is the append Node is considered the first Node of the list (the list is empty or not initialized yet). Therefore, I set the tail and head Node to the same append Node because this is the only Node in the list.

REMOVE:

Removing a data at a particular position is based on the same concept of the insert function. There will be 4 situations:

1. The first one is used to check the conditions of the position to be removed is correct or not. If not, the function will return 1. On the other hand, it will return 0, if the function is successfully removed the data.
2. The second is deletion at first. I have to remove the current head Node as it is at the position 1. I need to set the head Node to a temporary Node and move the head Node to the next Node as I will be removing it.
3. The third is deletion at last. In order to do so, I have to iterate through the list to that specific position and get the previous Node of the Node at that position. After having the previous Node, I have to set its next Node to the next Node of the Node at that position.
4. The fourth is deletion at a specific position. I have to move through each Node in the list until I reach the node’s position. Then I need to connect the previous Node and the next Node of that current Node together. This will link those two Nodes together without linking to the Node at the position we’re going to remove.

CONSTRUCTOR:

The constructor for this linked list is simply set the head Node and the tail Node to Null. The size of the list is 0 due to the list is empty.

COPY CONSTRUCTOR:

The copy constructor will loop through an existing linked list Node by Node and copy the data from each Node to the current linked list’s Node.

DESTRUCTOR:

The destructor will remove all the Node in the linked list until the list is empty.

PRINT MEMBER:

The print member is going to loop through this linked list and put all the data to the command line. If the linked list is empty, the print member will print out an alert showing that the list is currently empty.

Differences and Similarities between Iterative and Recursive Reverse:

ITERATIVE:

Reverse the order of the linked list iteratively, I have to iterate through the list and swap the previous Node and current Node position each iteration. The previous Node will be set to Null at start because it is going to be the next Node of the last Node in the reversed linked list, later on. The current Node is the going to move from the head Node of the current linked list to the last Node. Each iteration, the previous Node will become the current Node and the current Node will be its own next Node. In every loop, the next Node of current will be saved in a temporary Node. Then, the next Node of the current Node will be set to the previous Node. After that, it will move to the next Node (which is saved in the temporary Node before it changes). If the current Node starts from the head Node, set it as the tail Node. When the loop is done, set the head Node to the previous Node of the current Node because the current Node is now Null.

RECURSIVE:

The recursive method is also had the same implementation with the iterative method. However, it required to have 2 parameters in order to move recursively through the end of the linked list. The first parameter is the previous Node of the current Node; the second parameter is the head Node of the current list or the current Node where it is currently on while it is doing the recursion. The previous Node value always set as Null because the recursion will start from the beginning to last exactly like the iterative method. After the first recursion, the previous Node will change value to the current Node and the head Node will change value to its next Node. The concept of changing the head Node and the tail Node is also the same as the iterative method.

