Preparatory data Structure (CSCI 591)

Project - V

Implementing a linked list with Recursive Function

Submitted By: Taddese Erba

March 3, 2020

St. Cloud state university
Department of Computer Science

Taddese Erba
Section – I
Project – Five
Due: March 03, 2020

Design Document

Introduction

A linked list is a linear data structure of objects that are stored at random memory locations and linked together by pointers. Like any other function, linked lists' operations can be implemented recursively. This project sought to implement linked lists using recursive functions.

This project will implement the methods and operations of Project Three using recursive functions wherever it applies. The project contains three distinctive files; the header file `list.h`, the implementation file `list.cpp`, and the test file `main.cpp` implementation file contains all the implementation for the classes. The `main.cpp`.

Data Structure

As briefly described in the introduction section of this document, this program has three distinct files. The `list.h` file contains all the declaration of the required functions and a few function decoration (implementation). It is the framework for `LinkedList` class implementation. It consists of nine private objects, the `struct Node` object which is used to lold the two main components of a node, the `Node * getNode(Item entry, Node * list)` function used to declare and initialize the nodes. The implementation of this function does not require recursive operation as it does not involve any looping. The other seven functions that declare the recursive functions for corresponding operations. In addition to the nine private objects, the header file contains twelve public functions which include two constructors, one destructor, eight operational functions, and one friend function.

Functions

As described in the Data Structure section of this document, there are twenty-one functions in this project. The first two functions, the `LinkedList()` and `LinkedList(const LinkedList&`

`source)` are constructors. The `LinkedList()`constructor function is used to initialize the class. The constructor `LinkedList(const LinkedList& source)` is a copy constructor that is used to copy the elements of the list. The function `void copyNew(const LinkedList& mynew)` implements the copy constructor recursively along with the function `void copy(Node*& source, Node * ptr)`. The third function, `~ LinkedList()`, is a destructor. It is used to delete all the nodes, deallocate the memory, and return it to the operating system. The `void re_Initialize()` function is used to re-initialize the linked lists to empty. This function is implemented non-recursively. The `void insert(Item entry)` along with the function `void rec_Insert(Item entry, Node*&p)` is used for the recursive implementation of the insert operation. The `void remove(Item target)` along with the function `void rec_Remove(Item target, Node*&p)` is used to recursively remove a node from the list. The `void operator = (LinkedList s)` is used to overload the assignment operator (=) to be used in the assignment operation involving copying the elements of the list. The `bool isEmpty()` function returns true if the list is empty or false otherwise. The `int listLength()` function together with `int length(Node * p)` will count each node in the list recursively and return the size of the list (number of nodes). The `bool isPresent(Item target)` will check if an item is in the list recursively using the recursive function `bool Present(Node*ptr, Item found)` and returns true if the target is found. If the item is not found it returns false to `main()`. The `Item kthValue(int numval)` function will return the k^{th} node of the list. If the node is not found, or the list is empty, it returns nothing. The `friend ostream& operator << (ostream& out_s, const LinkedList& l)` is a friend function that is used to overload the outstream operator (<<) for the purpose of printing all the elements of the list. It is implemented recursively and functions jointly with the recursive part `void LinkedList::write(ostream& out_s, Node*p)`. For convenience reasons, the friend function, along with its recursive counterpart, is implemented in the header file where it was declared but outside of the class `LinkedList`.

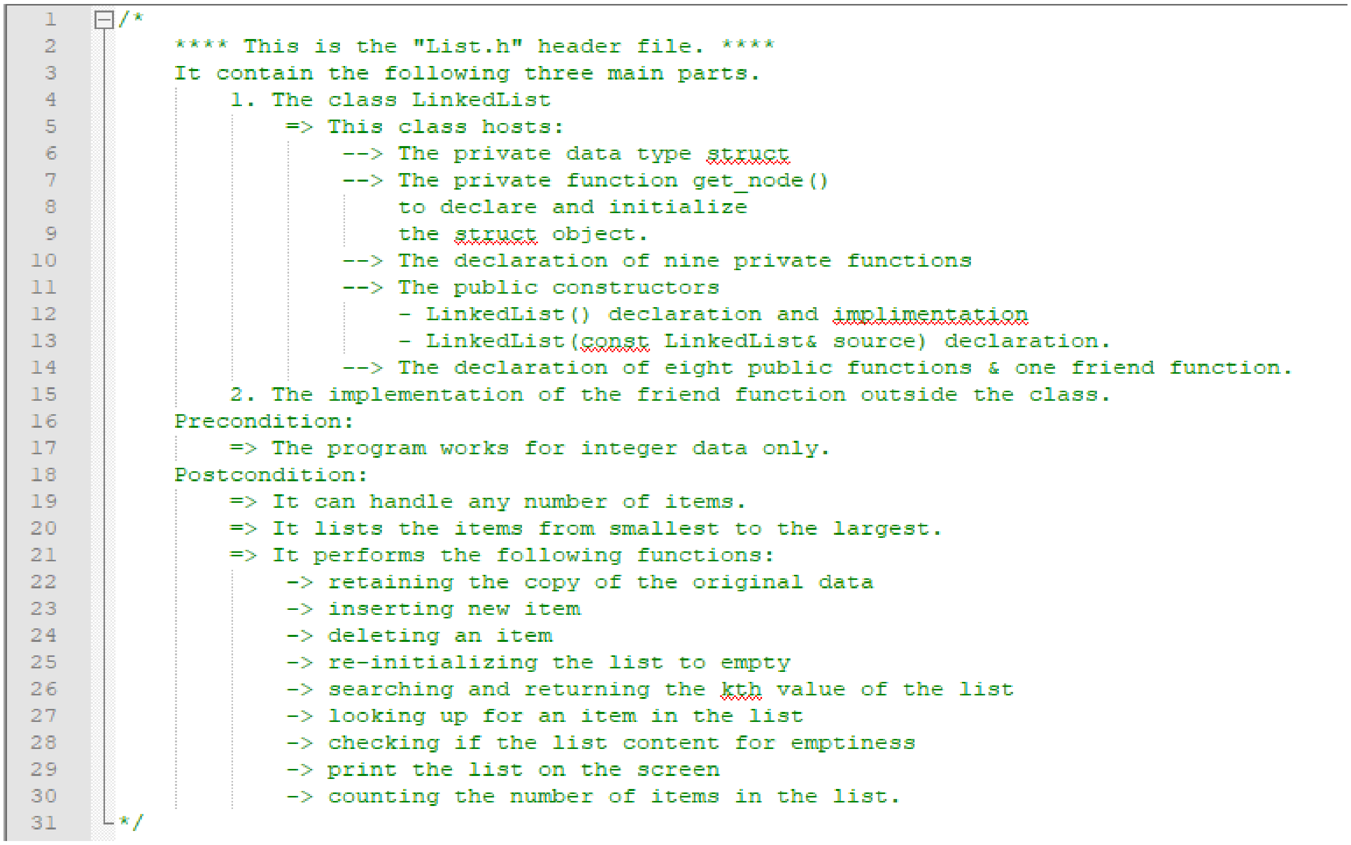
The Main Program

As a testing function, where the implementation is tested, there are many things going on in the `main`() function. To keep things simple, I will talk only about the main components of the main() function. The key frameworks in the main() function are the instantiation of the class `LinkedList` and representation of the key operations by a menu system. There are ten main menus from which the user can choose to perform an operation. The menus are represented by alphabets that are closely related to the operation followed by the name of the operation as in ***I -- Insert Item*** and ***R -- Remove Item***. The menus are continuously displayed after each operation until the user chooses to quit the program. A switch statement will track each choice of the user and perform the necessary operation accordingly. It may also worth mentioning the `bool searchArray(const char [], int, char)` function that is the part of the `main()` function that is used to search the array of constants that hold the alphabets designated to the menu. This enables that if the user enters a choice that is not available, the program can display the necessary message and exit the program.

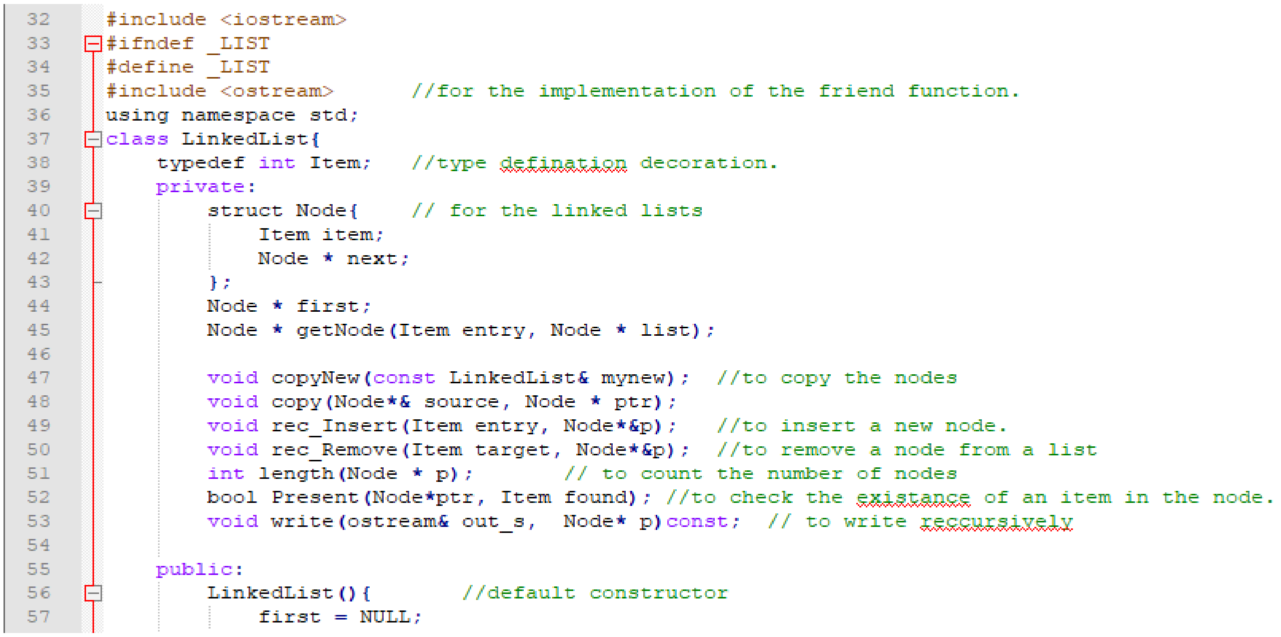
Taddese Erba
Section – I
Project – Five
Due: March 03, 2020

Code listing

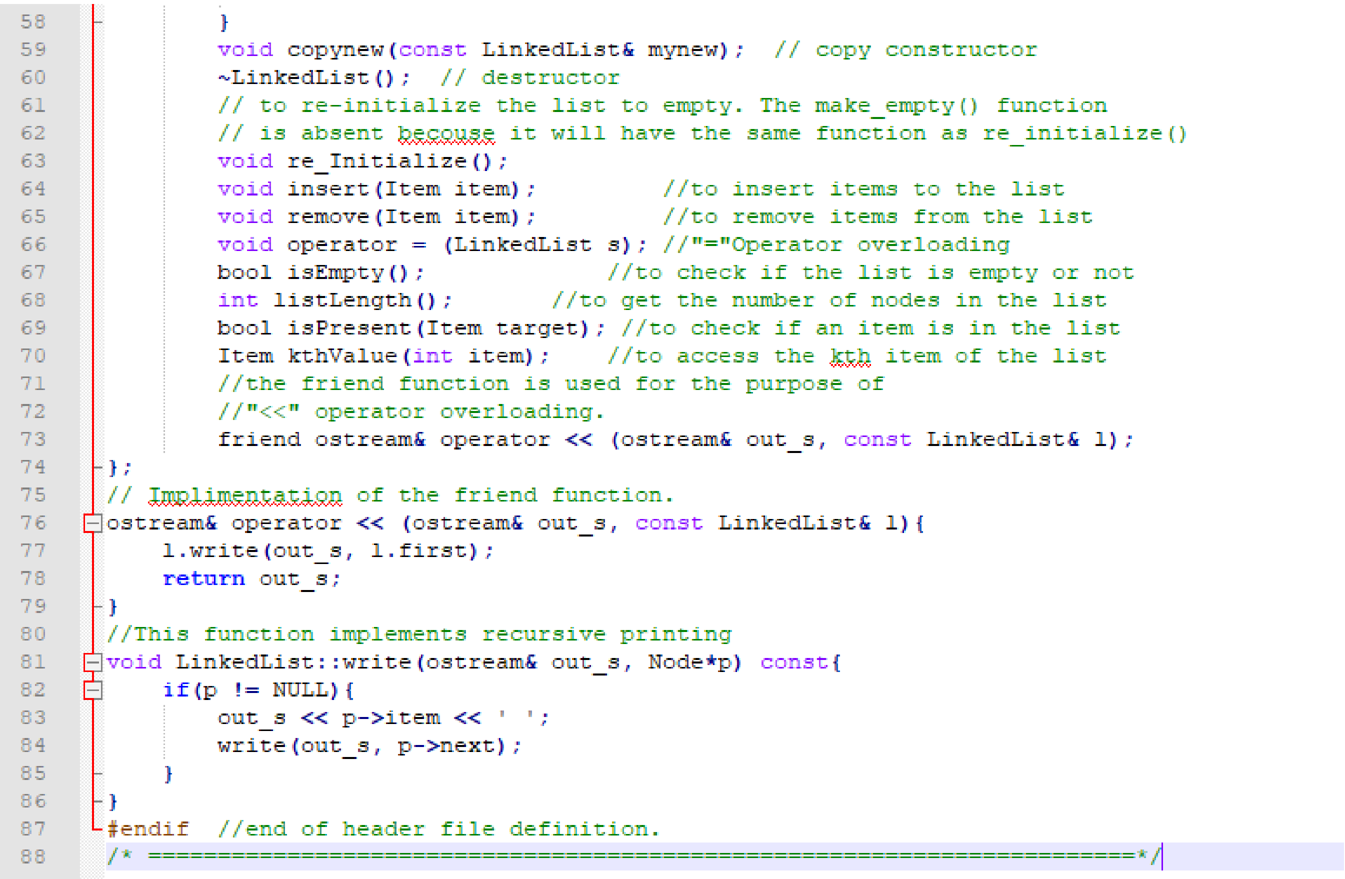
a. The header file (List.h)



```
/*
    **** This is the "List.h" header file. ****
    It contain the following three main parts.
        1. The class LinkedList
            => This class hosts:
                --> The private data type struct
                --> The private function get_node()
                    to declare and initialize
                    the struct object.
                --> The declaration of nine private functions
                --> The public constructors
                    - LinkedList() declaration and implimentation
                    - LinkedList(const LinkedList& source) declaration.
                --> The declaration of eight public functions & one friend function.
        2. The implementation of the friend function outside the class.
    Precondition:
        => The program works for integer data only.
    Postcondition:
        => It can handle any number of items.
        => It lists the items from smallest to the largest.
        => It performs the following functions:
            -> retaining the copy of the original data
            -> inserting new item
            -> deleting an item
            -> re-initializing the list to empty
            -> searching and returning the kth value of the list
            -> looking up for an item in the list
            -> checking if the list content for emptiness
            -> print the list on the screen
            -> counting the number of items in the list.
*/
```

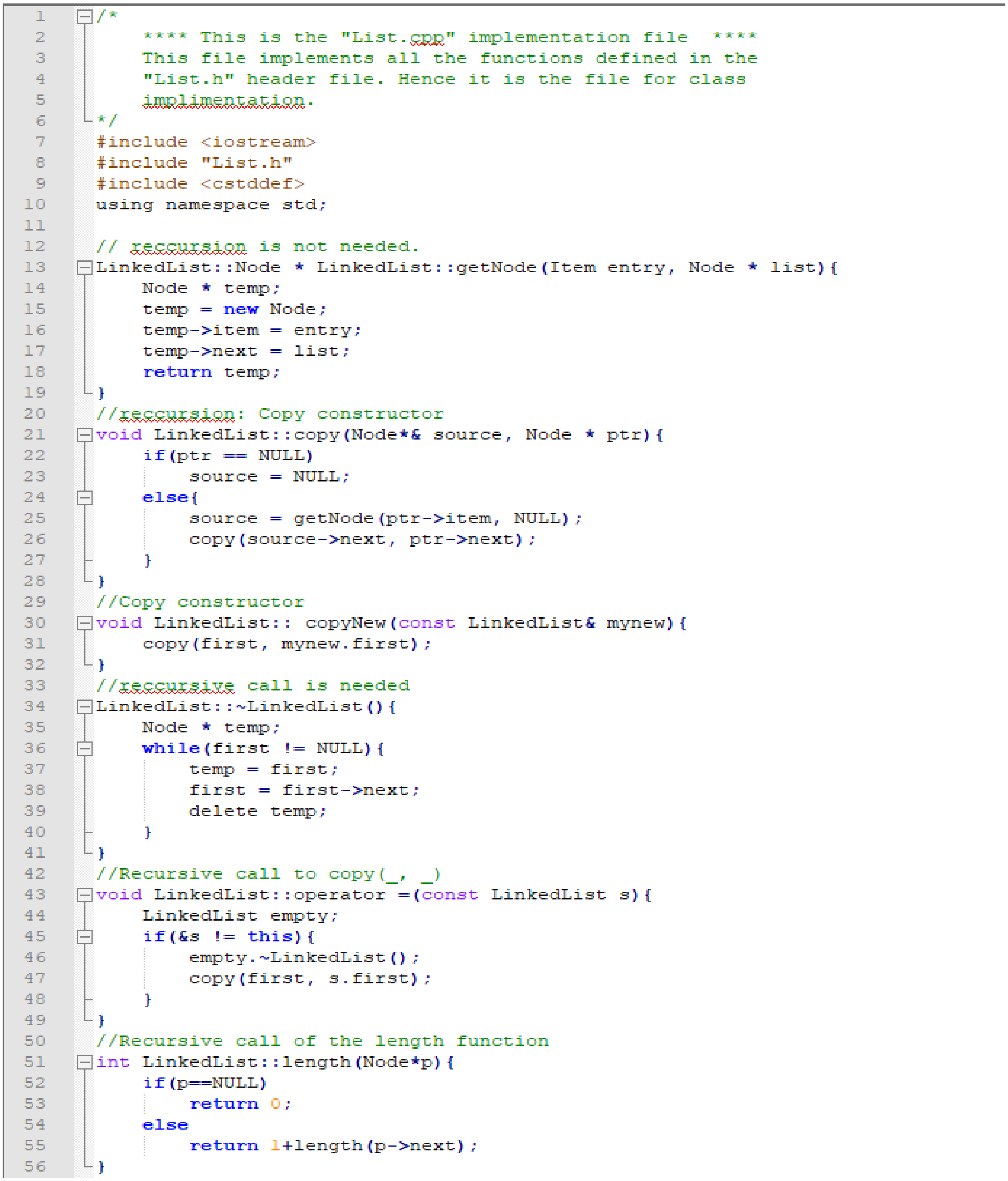


```
#include <iostream>
#ifndef _LIST
#define _LIST
#include <ostream>      //for the implementation of the friend function.
using namespace std;
class LinkedList{
    typedef int Item;   //type defination decoration.
    private:
        struct Node{    // for the linked lists
            Item item;
            Node * next;
        };
        Node * first;
        Node * getNode(Item entry, Node * list);

        void copyNew(const LinkedList& mynew);  //to copy the nodes
        void copy(Node*& source, Node * ptr);
        void rec_Insert(Item entry, Node*&p);   //to insert a new node.
        void rec_Remove(Item target, Node*&p);  //to remove a node from a list
        int length(Node * p);       // to count the number of nodes
        bool Present(Node*ptr, Item found); //to check the existance of an item in the node.
        void write(ostream& out_s,  Node* p)const;  // to write reccursively

    public:
        LinkedList(){       //default constructor
            first = NULL;
```



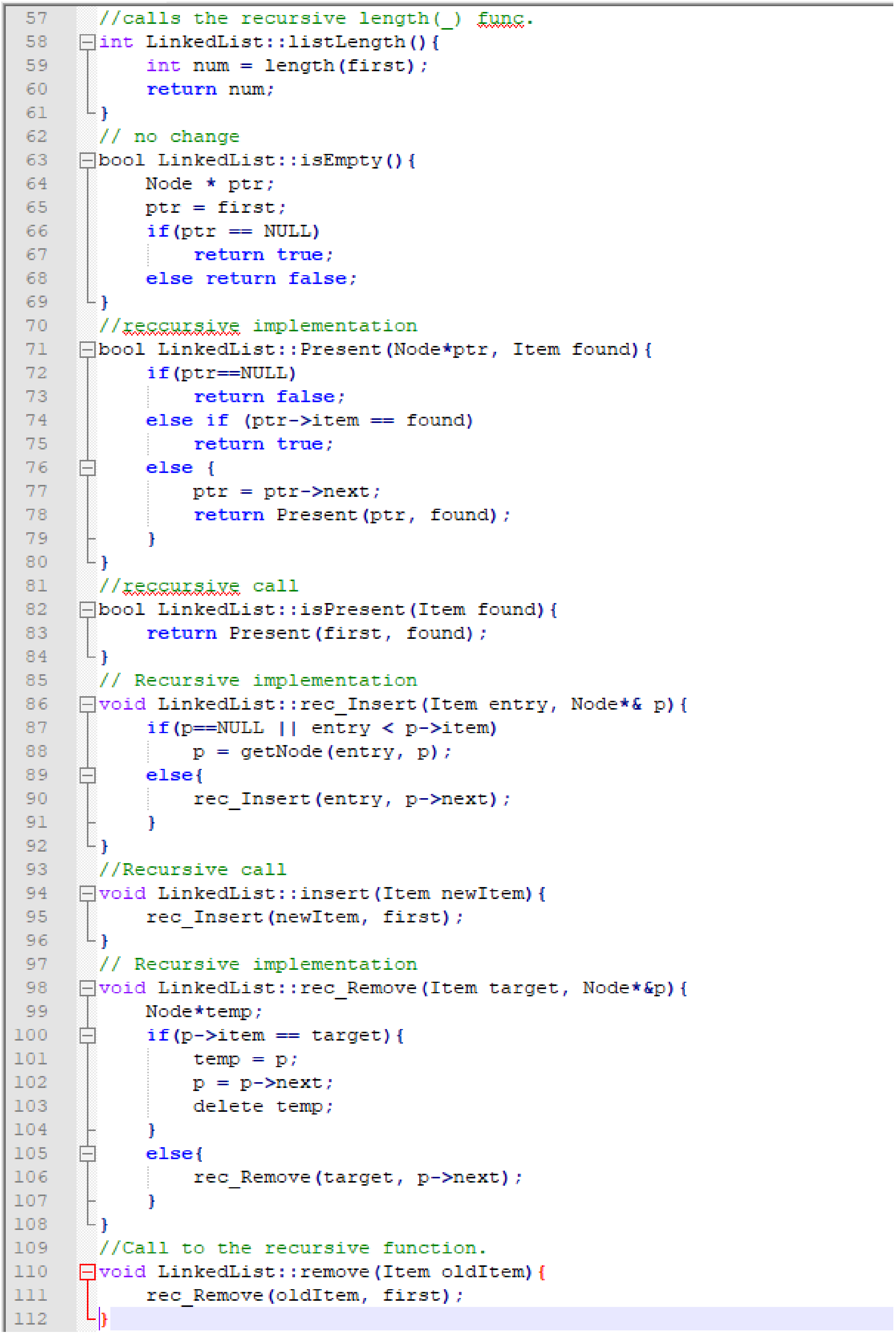
```
        }
        void copynew(const LinkedList& mynew);  // copy constructor
        ~LinkedList();  // destructor
        // to re-initialize the list to empty. The make_empty() function
        // is absent becouse it will have the same function as re_initialize()
        void re_Initialize();
        void insert(Item item);         //to insert items to the list
        void remove(Item item);         //to remove items from the list
        void operator = (LinkedList s); //"="Operator overloading
        bool isEmpty();             //to check if the list is empty or not
        int listLength();       //to get the number of nodes in the list
        bool isPresent(Item target); //to check if an item is in the list
        Item kthValue(int item);    //to access the kth item of the list
        //the friend function is used for the purpose of
        //"<<" operator overloading.
        friend ostream& operator << (ostream& out_s, const LinkedList& l);
};
// Implimentation of the friend function.
ostream& operator << (ostream& out_s, const LinkedList& l){
    l.write(out_s, l.first);
    return out_s;
}
//This function implements recursive printing
void LinkedList::write(ostream& out_s, Node*p) const{
    if(p != NULL){
        out_s << p->item << ' ';
        write(out_s, p->next);
    }
}
#endif  //end of header file definition.
/* =======================================================================*/
```

b. The implementation file (`List.cpp`)

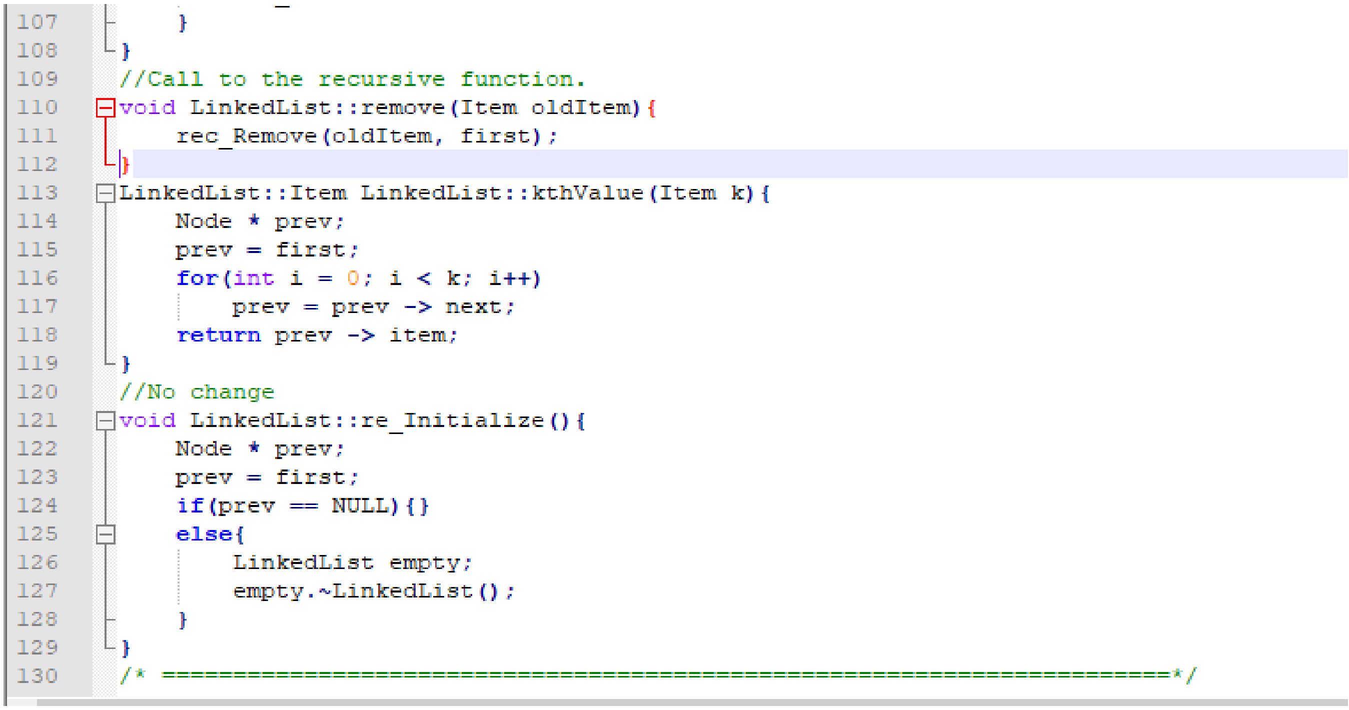


```
/*
    **** This is the "List.cpp" implementation file   ****
    This file implements all the functions defined in the
    "List.h" header file. Hence it is the file for class
    implimentation.
*/
#include <iostream>
#include "List.h"
#include <cstddef>
using namespace std;

// reccursion is not needed.
LinkedList::Node * LinkedList::getNode(Item entry, Node * list){
    Node * temp;
    temp = new Node;
    temp->item = entry;
    temp->next = list;
    return temp;
}
//reccursion: Copy constructor
void LinkedList::copy(Node*& source, Node * ptr){
    if(ptr == NULL)
        source = NULL;
    else{
        source = getNode(ptr->item, NULL);
        copy(source->next, ptr->next);
    }
}
//Copy constructor
void LinkedList:: copyNew(const LinkedList& mynew){
    copy(first, mynew.first);
}
//reccursive call is needed
LinkedList::~LinkedList(){
    Node * temp;
    while(first != NULL){
        temp = first;
        first = first->next;
        delete temp;
    }
}
//Recursive call to copy(_, _)
void LinkedList::operator =(const LinkedList s){
    LinkedList empty;
    if(&s != this){
        empty.~LinkedList();
        copy(first, s.first);
    }
}
//Recursive call of the length function
int LinkedList::length(Node*p){
    if(p==NULL)
        return 0;
    else
        return 1+length(p->next);
}
```

Taddese Erba
Section – I
Project – Five
Due: March 03, 2020

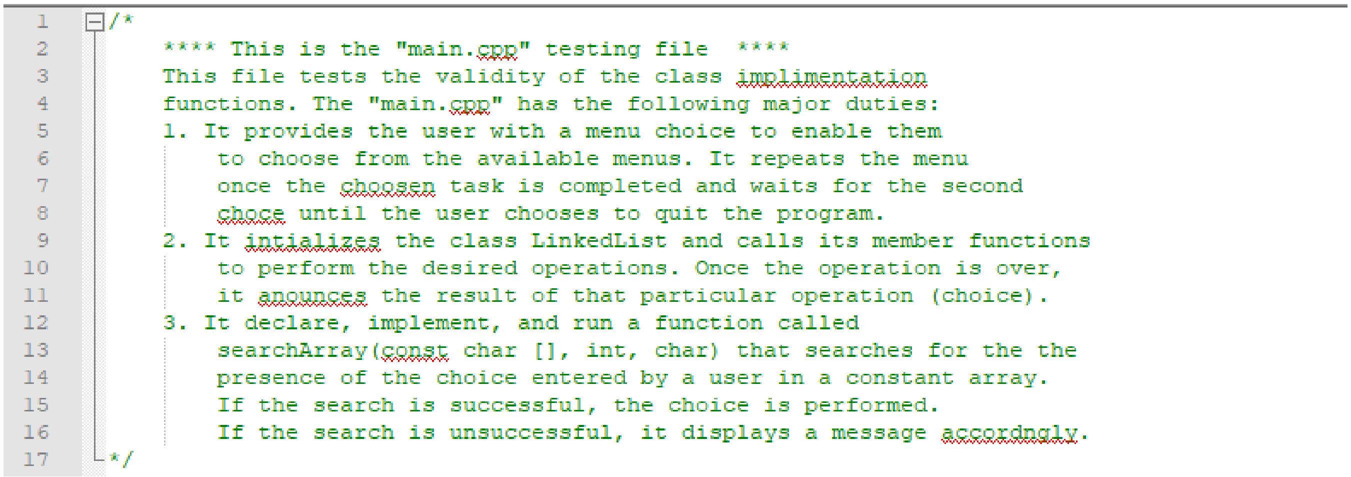


```
//calls the recursive length(_) func.
int LinkedList::listLength(){
    int num = length(first);
    return num;
}
// no change
bool LinkedList::isEmpty(){
    Node * ptr;
    ptr = first;
    if(ptr == NULL)
        return true;
    else return false;
}
//reccursive implementation
bool LinkedList::Present(Node*ptr, Item found){
    if(ptr==NULL)
        return false;
    else if (ptr->item == found)
        return true;
    else {
        ptr = ptr->next;
        return Present(ptr, found);
    }
}
//reccursive call
bool LinkedList::isPresent(Item found){
    return Present(first, found);
}
// Recursive implementation
void LinkedList::rec_Insert(Item entry, Node*& p){
    if(p==NULL || entry < p->item)
        p = getNode(entry, p);
    else{
        rec_Insert(entry, p->next);
    }
}
//Recursive call
void LinkedList::insert(Item newItem){
    rec_Insert(newItem, first);
}
// Recursive implementation
void LinkedList::rec_Remove(Item target, Node*&p){
    Node*temp;
    if(p->item == target){
        temp = p;
        p = p->next;
        delete temp;
    }
    else{
        rec_Remove(target, p->next);
    }
}
//Call to the recursive function.
void LinkedList::remove(Item oldItem){
    rec_Remove(oldItem, first);
}
```



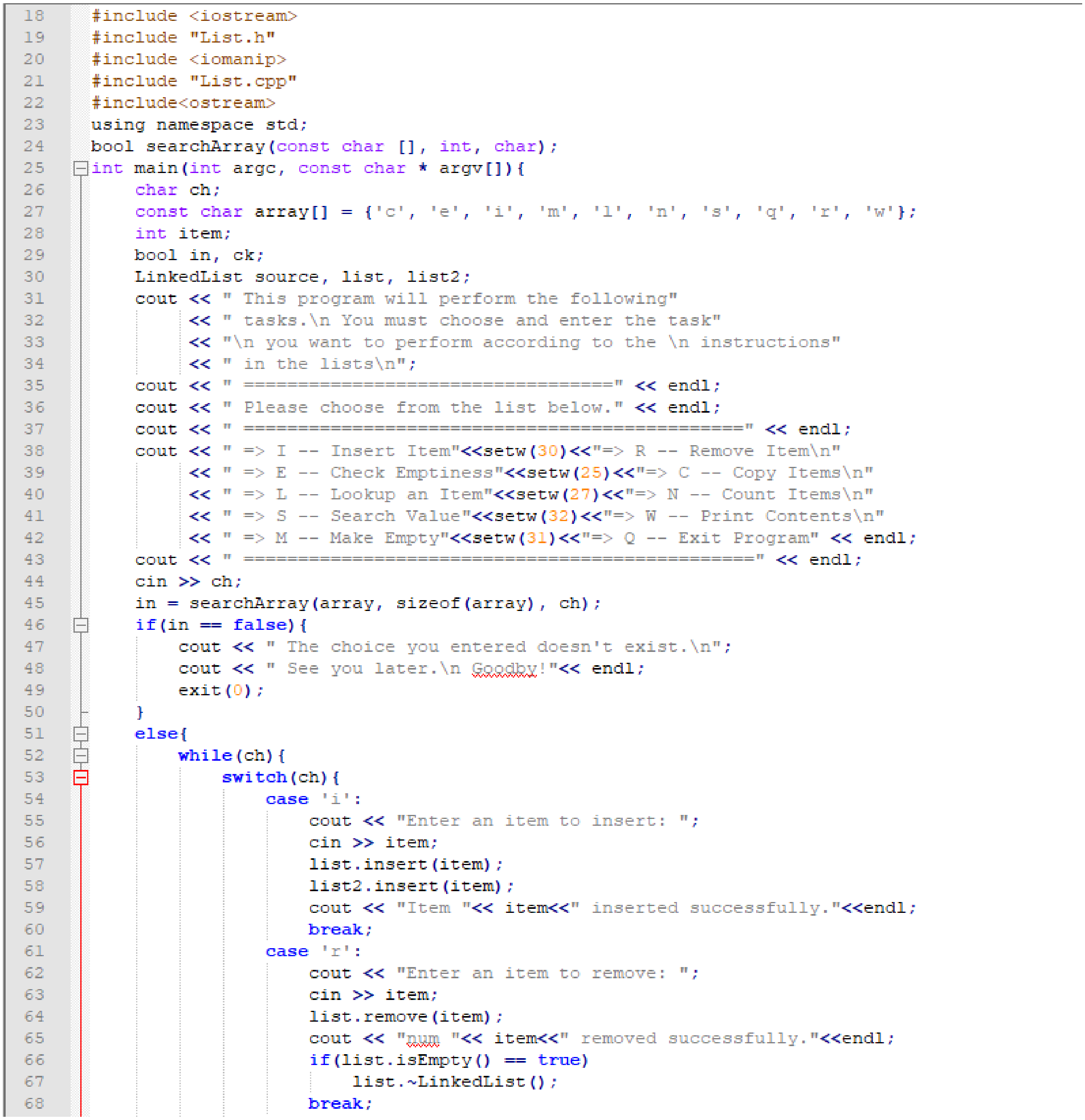
```
    }
}
//Call to the recursive function.
void LinkedList::remove(Item oldItem){
    rec_Remove(oldItem, first);
}
LinkedList::Item LinkedList::kthValue(Item k){
    Node * prev;
    prev = first;
    for(int i = 0; i < k; i++)
        prev = prev -> next;
    return prev -> item;
}
//No change
void LinkedList::re_Initialize(){
    Node * prev;
    prev = first;
    if(prev == NULL){}
    else{
        LinkedList empty;
        empty.~LinkedList();
    }
}
/* =======================================================================*/
```

c. The testing file (main.cpp)

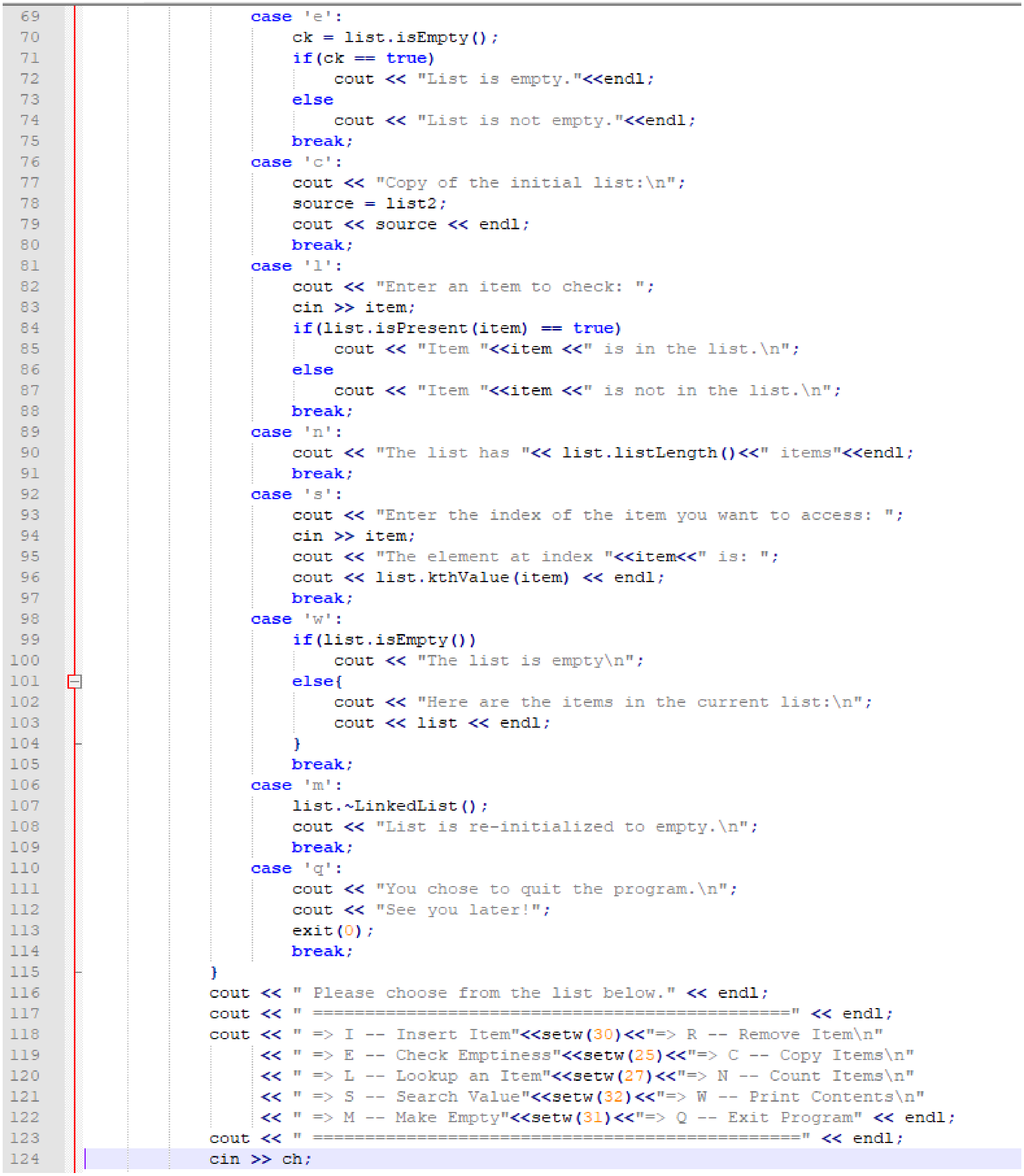


```
/*
    **** This is the "main.cpp" testing file  ****
    This file tests the validity of the class implimentation
    functions. The "main.cpp" has the following major duties:
    1. It provides the user with a menu choice to enable them
       to choose from the available menus. It repeats the menu
       once the choosen task is completed and waits for the second
       choce until the user chooses to quit the program.
    2. It intializes the class LinkedList and calls its member functions
       to perform the desired operations. Once the operation is over,
       it anounces the result of that particular operation (choice).
    3. It declare, implement, and run a function called
       searchArray(const char [], int, char) that searches for the the
       presence of the choice entered by a user in a constant array.
       If the search is successful, the choice is performed.
       If the search is unsuccessful, it displays a message accordngly.
*/
```

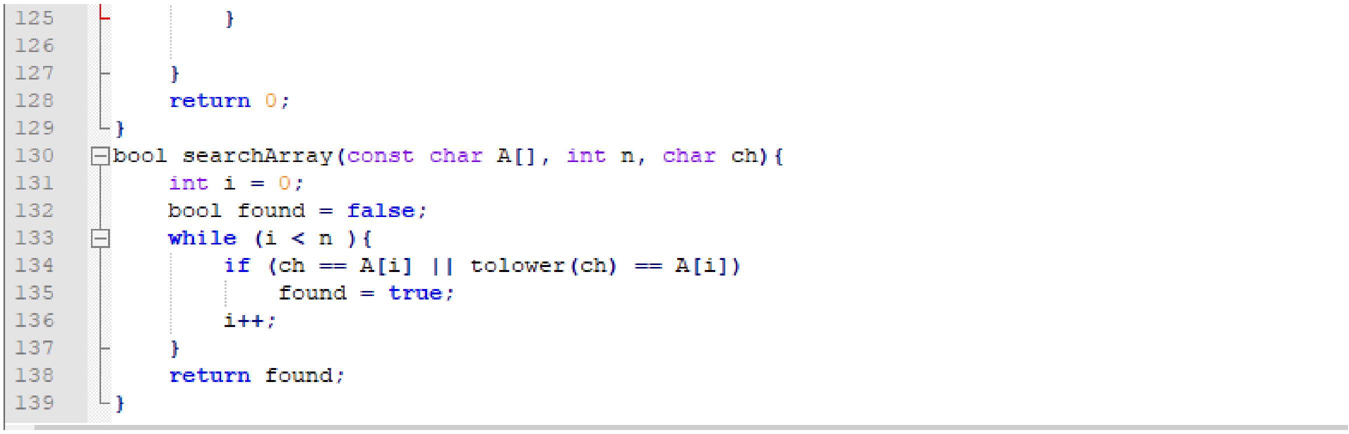
Taddese Erba
Section – I
Project – Five
Due: March 03, 2020



```
#include <iostream>
#include "List.h"
#include <iomanip>
#include "List.cpp"
#include<ostream>
using namespace std;
bool searchArray(const char [], int, char);
int main(int argc, const char * argv[]){
    char ch;
    const char array[] = {'c', 'e', 'i', 'm', 'l', 'n', 's', 'q', 'r', 'w'};
    int item;
    bool in, ck;
    LinkedList source, list, list2;
    cout << " This program will perform the following"
         << " tasks.\n You must choose and enter the task"
         << "\n you want to perform according to the \n instructions"
         << " in the lists\n";
    cout << " ==================================" << endl;
    cout << " Please choose from the list below." << endl;
    cout << " ===============================================" << endl;
    cout << " => I -- Insert Item"<<setw(30)<<"=> R -- Remove Item\n"
         << " => E -- Check Emptiness"<<setw(25)<<"=> C -- Copy Items\n"
         << " => L -- Lookup an Item"<<setw(27)<<"=> N -- Count Items\n"
         << " => S -- Search Value"<<setw(32)<<"=> W -- Print Contents\n"
         << " => M -- Make Empty"<<setw(31)<<"=> Q -- Exit Program" << endl;
    cout << " ===============================================" << endl;
    cin >> ch;
    in = searchArray(array, sizeof(array), ch);
    if(in == false){
        cout << " The choice you entered doesn't exist.\n";
        cout << " See you later.\n Goodby!"<< endl;
        exit(0);
    }
    else{
        while(ch){
            switch(ch){
                case 'i':
                    cout << "Enter an item to insert: ";
                    cin >> item;
                    list.insert(item);
                    list2.insert(item);
                    cout << "Item "<< item<<" inserted successfully."<<endl;
                    break;
                case 'r':
                    cout << "Enter an item to remove: ";
                    cin >> item;
                    list.remove(item);
                    cout << "num "<< item<<" removed successfully."<<endl;
                    if(list.isEmpty() == true)
                        list.~LinkedList();
                    break;
```



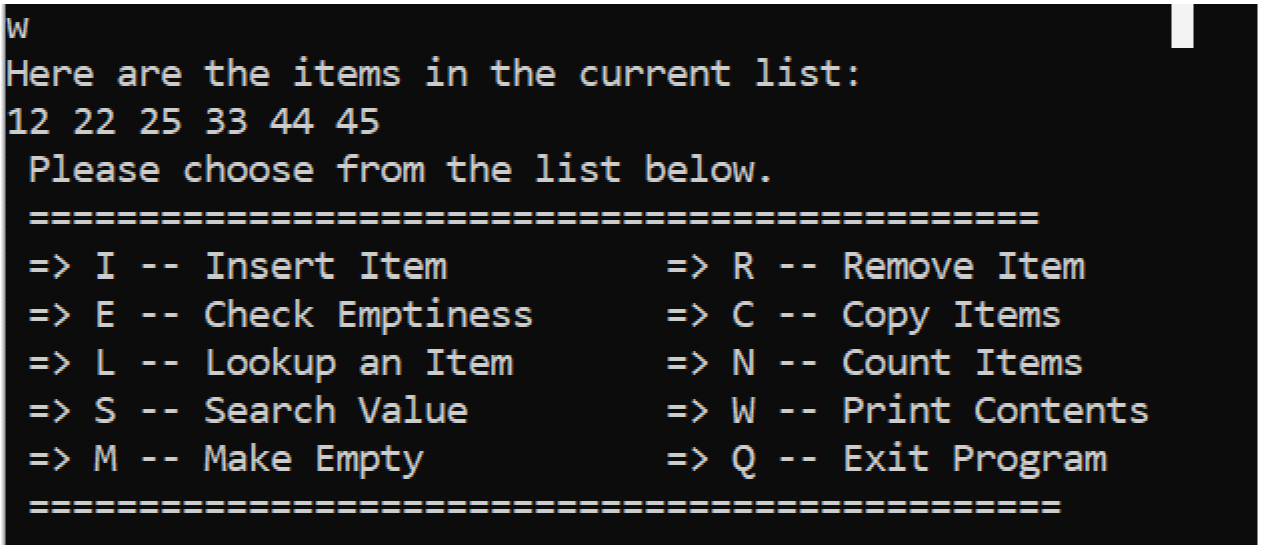
```
                case 'e':
                    ck = list.isEmpty();
                    if(ck == true)
                        cout << "List is empty."<<endl;
                    else
                        cout << "List is not empty."<<endl;
                    break;
                case 'c':
                    cout << "Copy of the initial list:\n";
                    source = list2;
                    cout << source << endl;
                    break;
                case 'l':
                    cout << "Enter an item to check: ";
                    cin >> item;
                    if(list.isPresent(item) == true)
                        cout << "Item "<<item <<" is in the list.\n";
                    else
                        cout << "Item "<<item <<" is not in the list.\n";
                    break;
                case 'n':
                    cout << "The list has "<< list.listLength()<<" items"<<endl;
                    break;
                case 's':
                    cout << "Enter the index of the item you want to access: ";
                    cin >> item;
                    cout << "The element at index "<<item<<" is: ";
                    cout << list.kthValue(item) << endl;
                    break;
                case 'w':
                    if(list.isEmpty())
                        cout << "The list is empty\n";
                    else{
                        cout << "Here are the items in the current list:\n";
                        cout << list << endl;
                    }
                    break;
                case 'm':
                    list.~LinkedList();
                    cout << "List is re-initialized to empty.\n";
                    break;
                case 'q':
                    cout << "You chose to quit the program.\n";
                    cout << "See you later!";
                    exit(0);
                    break;
            }
            cout << " Please choose from the list below." << endl;
            cout << " ===============================================" << endl;
            cout << " => I -- Insert Item"<<setw(30)<<"=> R -- Remove Item\n"
                 << " => E -- Check Emptiness"<<setw(25)<<"=> C -- Copy Items\n"
                 << " => L -- Lookup an Item"<<setw(27)<<"=> N -- Count Items\n"
                 << " => S -- Search Value"<<setw(32)<<"=> W -- Print Contents\n"
                 << " => M -- Make Empty"<<setw(31)<<"=> Q -- Exit Program" << endl;
            cout << " ================================================" << endl;
            cin >> ch;
```



```
        }

    }
    return 0;
}
bool searchArray(const char A[], int n, char ch){
    int i = 0;
    bool found = false;
    while (i < n ){
        if (ch == A[i] || tolower(ch) == A[i])
            found = true;
        i++;
    }
    return found;
}
```

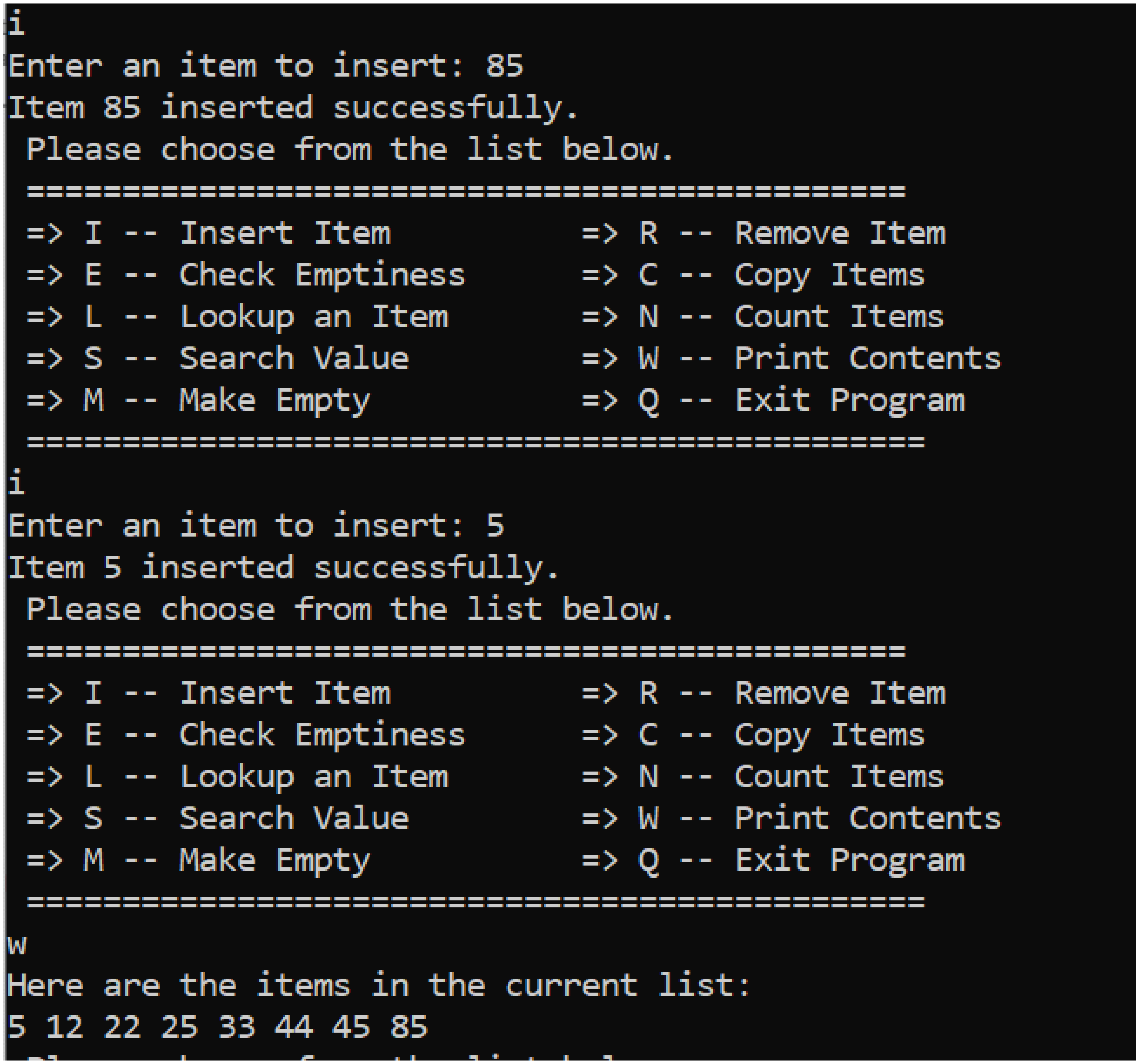
Test Results

Display befor the tests.



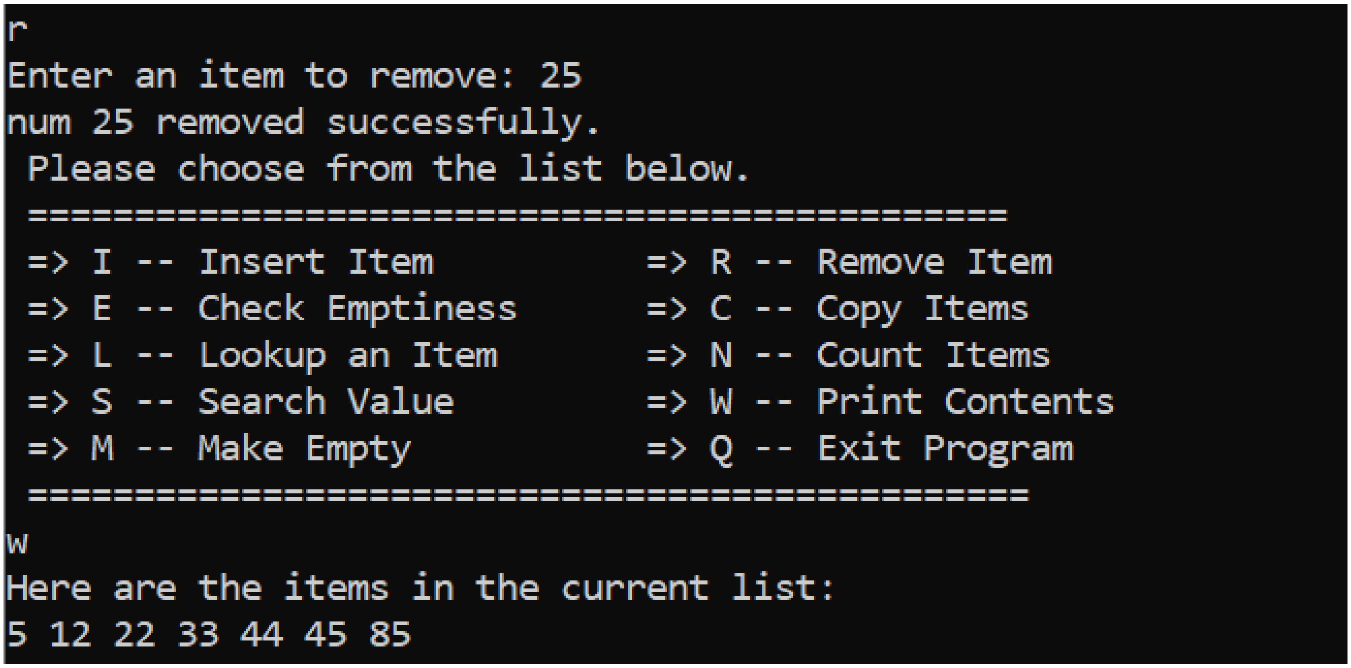
```
W
Here are the items in the current list:
12 22 25 33 44 45
 Please choose from the list below.
 ==============================================
 => I -- Insert Item          => R -- Remove Item
 => E -- Check Emptiness      => C -- Copy Items
 => L -- Lookup an Item       => N -- Count Items
 => S -- Search Value         => W -- Print Contents
 => M -- Make Empty           => Q -- Exit Program
 ===============================================
```

1. Tests result for insert() function.



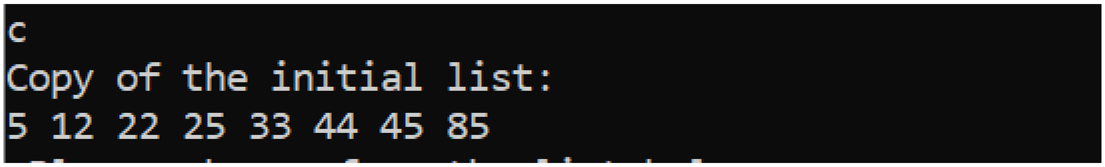
```
i
Enter an item to insert: 85
Item 85 inserted successfully.
 Please choose from the list below.
 ==============================================
 => I -- Insert Item          => R -- Remove Item
 => E -- Check Emptiness      => C -- Copy Items
 => L -- Lookup an Item       => N -- Count Items
 => S -- Search Value         => W -- Print Contents
 => M -- Make Empty           => Q -- Exit Program
 ===============================================
i
Enter an item to insert: 5
Item 5 inserted successfully.
 Please choose from the list below.
 ==============================================
 => I -- Insert Item          => R -- Remove Item
 => E -- Check Emptiness      => C -- Copy Items
 => L -- Lookup an Item       => N -- Count Items
 => S -- Search Value         => W -- Print Contents
 => M -- Make Empty           => Q -- Exit Program
 ===============================================
w
Here are the items in the current list:
5 12 22 25 33 44 45 85
 Please choose from the list below.
```

2. Tests result for remove() function.



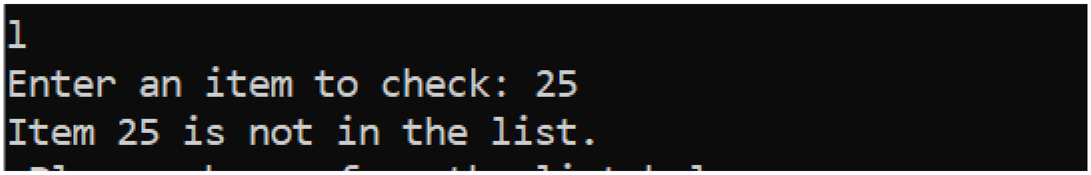
```
r
Enter an item to remove: 25
num 25 removed successfully.
 Please choose from the list below.
 ==============================================
 => I -- Insert Item            => R -- Remove Item
 => E -- Check Emptiness        => C -- Copy Items
 => L -- Lookup an Item         => N -- Count Items
 => S -- Search Value           => W -- Print Contents
 => M -- Make Empty             => Q -- Exit Program
 ===============================================
W
Here are the items in the current list:
5 12 22 33 44 45 85
```

3. Tests result for copy () function.

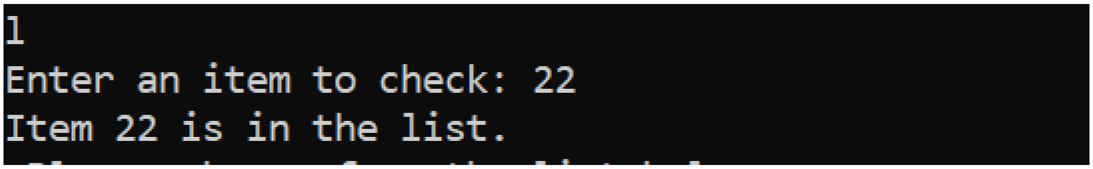


```
c
Copy of the initial list:
5 12 22 25 33 44 45 85
```

4. Tests result for isPresent() function.

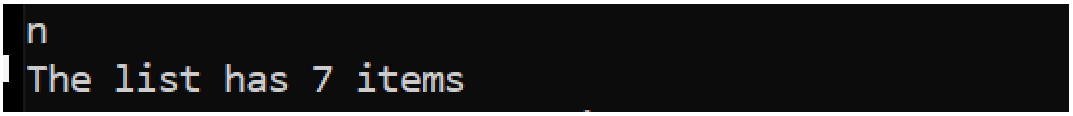


```
l
Enter an item to check: 25
Item 25 is not in the list.
```



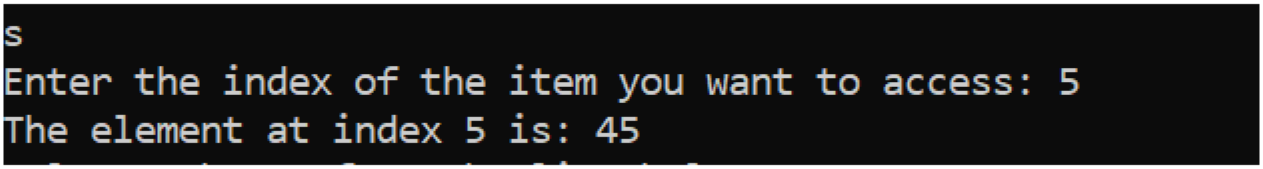
```
l
Enter an item to check: 22
Item 22 is in the list.
```

5. Tests result for listLength() function.



```
n
The list has 7 items
```

6. Tests result for kthValue() function.



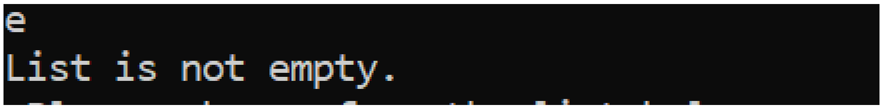
```
s
Enter the index of the item you want to access: 5
The element at index 5 is: 45
```

Note: If we try to access an index that is not in the list the function will exhaust searching that index and exit the program as shown in the above screenshot.

7. Tests result for write () function

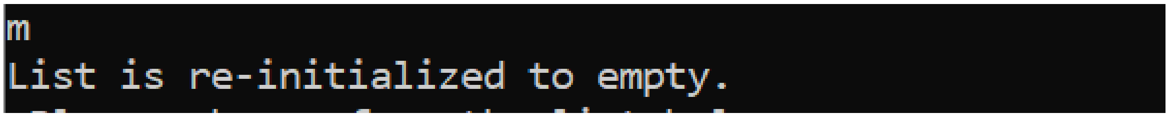
I am not running a separate test for the write () function since I am running it for almost every other step as part of checking the program to see if it is doing what it supposed to do. Please look at the end of the other programs where I occasionally run the write() function.

8. Tests result for checkEmpty () function

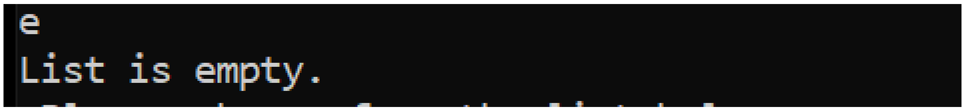


```
e
List is not empty.
```

9. Tests result for makeEmpty () function

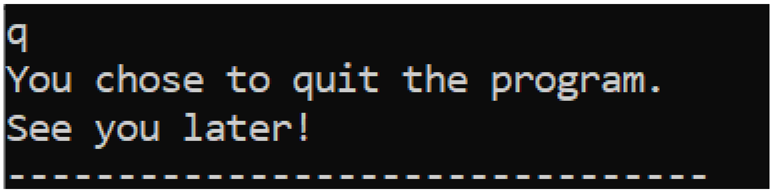


```
m
List is re-initialized to empty.
```



```
e
List is empty.
```

10. Tests result for quit



```
q
You chose to quit the program.
See you later!
----------------------------------
```

Moreover, if the user enters a choice that is not listed the program will announce that and exit as shown below.

Taddese Erba
Section – I
Project – Five
Due: March 03, 2020

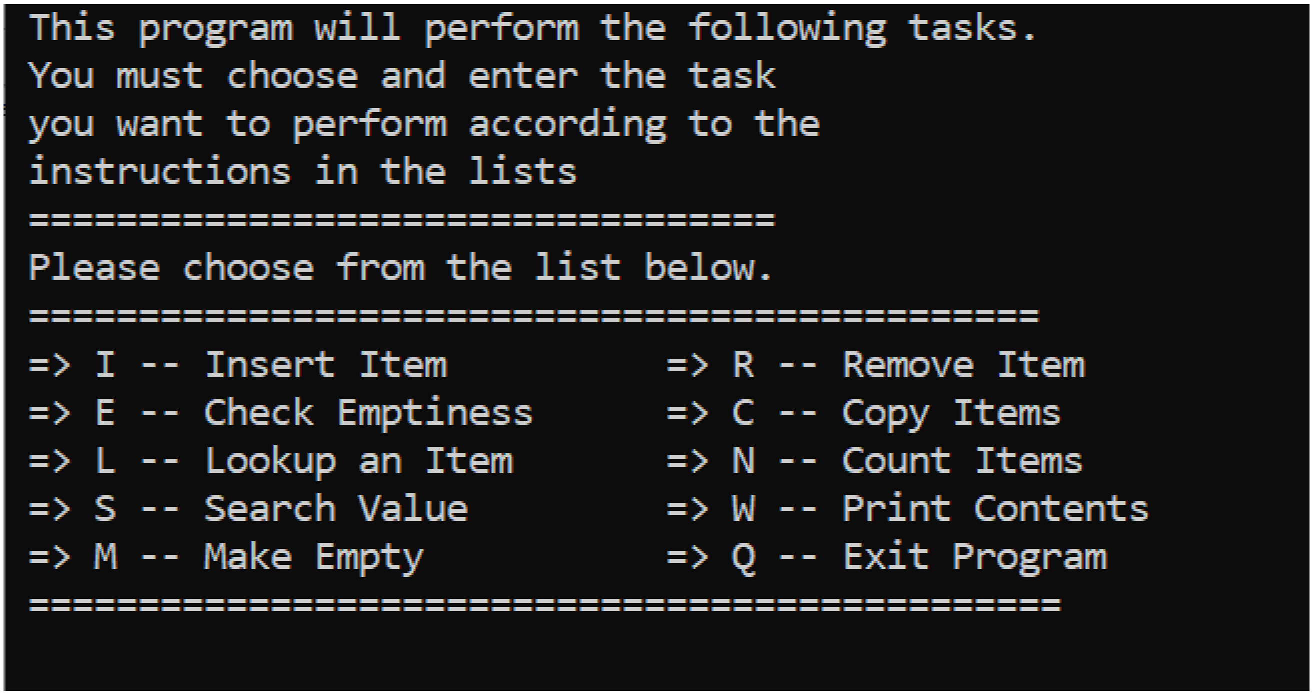
User document

This program can perform different tasks on a linked list as shown in the menu below. In order to run the program, you must perform the following steps.

- The program name is `main.cpp.` on the terminal enter the following command to compile and run the program.

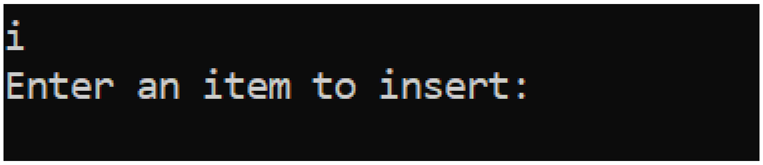
```
g++ -o main main.cpp
```

- The program will compile and open the following window:



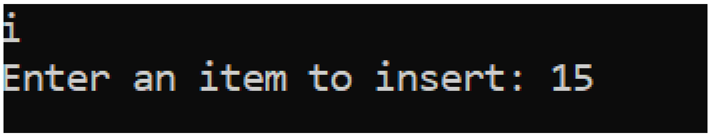
```
This program will perform the following tasks.
You must choose and enter the task
you want to perform according to the
instructions in the lists
==================================
Please choose from the list below.
==============================================
=> I -- Insert Item          => R -- Remove Item
=> E -- Check Emptiness      => C -- Copy Items
=> L -- Lookup an Item       => N -- Count Items
=> S -- Search Value         => W -- Print Contents
=> M -- Make Empty           => Q -- Exit Program
===============================================
```

- Once the window opens, make a choice from the displayed menu. For example to insert an item type i or I and then enter.



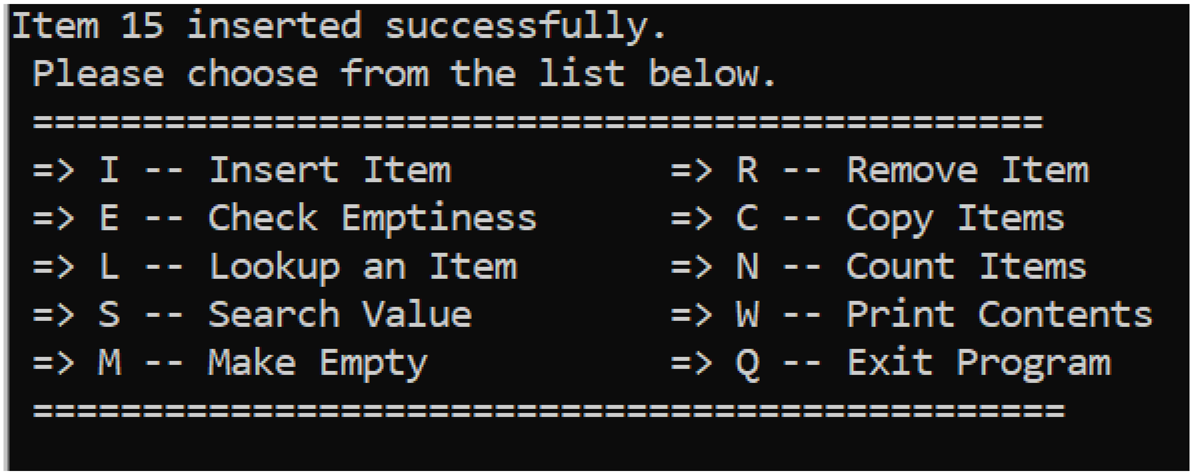
```
i
Enter an item to insert:
```

- Next, type the item you want to insert and then enter. For example, type 15 and enter.



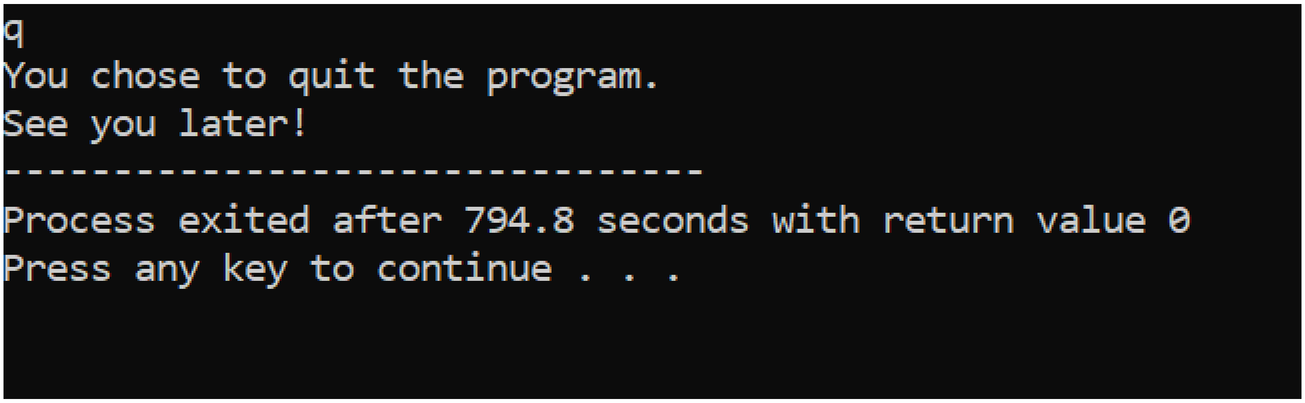
```
i
Enter an item to insert: 15
```

- The program will announce that the item is entered successfully and display the menu to make the next choice.



```
Item 15 inserted successfully.
 Please choose from the list below.
 ==============================================
 => I -- Insert Item           => R -- Remove Item
 => E -- Check Emptiness       => C -- Copy Items
 => L -- Lookup an Item        => N -- Count Items
 => S -- Search Value          => W -- Print Contents
 => M -- Make Empty            => Q -- Exit Program
 ===============================================
```

- If you want to repeat the insert repeat the above procedure; otherwise make the next selection.
- The program will perform in the same manner for all other tasks as for insert. Hence, all the other eight functions will perform in the same manner.
- Feel free to play around with the other choices (alphabets) and see what the program is meant to do.
- If you wish to exit the program, type q (Q) and enter.



```
q
You chose to quit the program.
See you later!
--------------------------------
Process exited after 794.8 seconds with return value 0
Press any key to continue . . .
```

- Now you can close the window.

Here are very important points while using this program

1. You must insert integer values only. If you try to enter something else other than an integer, the program may crash.
2. Do not try to access the ends of the list. If your list has 3 nodes only and you try to access the 4th node, the program will stop and exit.
3. The program will save a copy of the current list you are working with. You can just type c(C) and access that copy. Of course, once you exit the program that copy will not exits.

Taddese Erba
Section – I
Project – Five
Due: March 03, 2020

Summery

The project implements linked list operations such as inserting a new item, removing an item from the list, making the list empty, checking for the presence of an item, displaying a copy of the original item that contains all the elements of the list and so on recursively where every recursion can apply. This project applies the knowledge I gained from the lecture presented in the classroom and the knowledge I gained from reading different C++ data structure books.

This program can be made more useful by making it accommodate various types of data such as strings, characters, and double variables. This way, the program can do something important such as storing important records. Furthermore, data could also be made available as a file and a permanent copy of that file is kept with all current updates included while we still have the old data for reference.

By completing this project, I have gained a significant level of confidence and the necessary knowledge to work with linked lists and recursive function implementations. The only challenge I faced while completing this project is how to implement the recursive function for the out streaming function which I finally figured out how to do it. [1]

[1] This materials in this document is mostly from the previous project due to the close similarity of the two projects.