North South University
CSE-225L(Data Structures & Algorithm)
Summer - 2018
Lab-07 (Sorted List – Linked List Based)

Class “ListNode”

listnode.h

```
#ifndef LISTNODE_H_INCLUDED
#define LISTNODE_H_INCLUDED
#include <iostream>
#include <string>
using namespace std;

class ListNode{

private:
    string name,number;
public:
    ListNode *nextNode;
    ListNode(string,string);
    string GetName();
    string GetNumber();
    int CompareToName(string);
    int CompareToNumber(string);

};

#endif // LISTNODE_H_INCLUDED
```

listnode.cpp

```
#include "listnode.h"

ListNode::ListNode(string n,string num)
{
    name = n;
    number = num;
    nextNode = NULL;
}

string ListNode::GetName()
{
    return name;
}

string ListNode::GetNumber()
```

```
{
    return number;
}

int ListNode::CompareToName(string n)
{
    return name.compare(n); // returns 0 if found equal, -ve if n
                           // greater than name, +ve if n smaller
}

int ListNode::CompareToNumber(string num)
{
    return number.compare(num);// returns 0 if found equal, -ve
                           // if n greater than name, +ve if n
     smaller
}
```

<u>Class “LinkedList”</u>

<u>linkedlist.h</u>

```
#ifndef LINKEDLIST_H_INCLUDED
#define LINKEDLIST_H_INCLUDED
#include "listnode.h"

class LinkedList{

private:
  ListNode *head;// will always point to the head/front of the
                      // list
  ListNode *nextItem;//will be used to traverse the nodes of the
                          //list
public:
    LinkedList();// constructor
    ~LinkedList();// destructor
    void MakeEmpty();
    void InsertNode(string,string);
    void ResetList(); // resets the nextItem pointer to the front
    bool SearchByName(string);
    bool SearchByNumber(string);
    void DeleteNode(string); //will delete the ListNode whose
                                // name is equal to this function's
                                // parameter
    void PrintList();

};
```

#endif // LINKEDLIST_H_INCLUDED
linkedlist.cpp

#include "linkedlist.h"

```
LinkedList::LinkedList()
{
    head = NULL;
    nextItem = NULL;
}

LinkedList::~LinkedList()
{
    MakeEmpty();
}

void LinkedList::MakeEmpty()
{

    ListNode *temp;
    while(head != NULL)
    {
        temp = head;
        head = head->nextNode;
        delete temp;
    }

}



void LinkedList::InsertNode(string name,string number)
{
    ListNode *newNode;

    newNode = new ListNode(name,number); // creating &
    // initializing the new node with the parameter data

    bool moreToSearch; // will be used to find the appropriate
    // insertion place for the new node

    ResetList();

    nextItem = head;

    ListNode *predLoc = NULL; // predLoc will always point to the
    //previous node of nextItem
    moreToSearch = (nextItem != NULL);



    while(moreToSearch)
    {
        if(nextItem->CompareToName(name)<0)
        {
            predLoc = nextItem;
            nextItem = nextItem->nextNode;
            moreToSearch = (nextItem != NULL);
        }
        else
            moreToSearch = false;
    }

    if(predLoc == NULL) // for the first node
    {
        newNode->nextNode = head; // for first node, value of
                                  // head is null here
        head = newNode;
    }
    else
    {
        newNode->nextNode = nextItem;
        predLoc->nextNode = newNode;
    }

}

void LinkedList::ResetList()
{
    nextItem = NULL;
}


bool LinkedList::SearchByName(string n)
{

    ResetList();
    nextItem = head;

    bool searchFlag = false;

    while(nextItem!=NULL)
    {
        if(nextItem->CompareToName(n)==0)
        {
            searchFlag = true;
            break;
        }
        else{
            nextItem = nextItem->nextNode;
        }

    }
    ResetList();

    return searchFlag;

}

bool LinkedList::SearchByNumber(string num)
{
    ResetList();
    nextItem = head;

    bool searchFlag = false;

    while(nextItem!=NULL)
    {
        if(nextItem->CompareToNumber(num)==0)
        {
            searchFlag = true;
            break;
        }

        else{

            nextItem = nextItem->nextNode;
        }


    }

    ResetList();

    return searchFlag;

}

void LinkedList::DeleteNode(string name)
{

    if(SearchByName(name)==false)
    {
        cout<<"Node with name = ";cout<<name;cout<<" does not
        exist"<<endl;
    }

    else
    {

        ResetList();

        nextItem = head;

        if(nextItem->CompareToName(name)==0)// item to be deleted
                                           //is at the list's
     beginning
        {
            ListNode *tempPtr = head;
            head = head->nextNode;
            delete tempPtr;
        }

        else
        {

            while(nextItem->nextNode->CompareToName(name)!=0)
            {
               nextItem = nextItem->nextNode;
            }

            // now nextItem is pointing to the node before the
            // ListNode containing the user to be deleted

            ListNode *tempPtr;

            tempPtr = nextItem->nextNode; // tempPtr pointing to
                                         //the node to be deleted


            nextItem->nextNode = nextItem->nextNode->nextNode;
           // pointing to the ListNode which was just after the
           // node containing the 'name' to be deleted

            delete tempPtr;

        }

        ResetList();
    }
```

```
}


void LinkedList::PrintList()
{

    ResetList();
    nextItem = head;

    string name,number;

    while(nextItem!=NULL)
    {
        name = nextItem->GetName();
        number = nextItem->GetNumber();

        cout<<"Name:";cout<<name;
        cout<<" & Number:";cout<<number<<endl;
        nextItem = nextItem->nextNode;
    }

}
```