Lab Work 8 Part 2 (linked list advanced)

Learning Objectives

- Learn how to create linked lists using OOP classes

Exercise 1 : Creating a linked list header file

```
// Specification file for the NumberList class
#ifndef NUMBERLIST_H
#define NUMBERLIST_H

class NumberList
{
private:
  // Declare a structure for the list
  struct ListNode
  {
    double value;         // The value in this node
    struct ListNode *next;  // To point to the next node
  };

  ListNode *head;           // List head pointer

public:
  // Constructor
  NumberList()
    { head = nullptr; }

  // Destructor
  ~NumberList();

  // Linked list operations
  void appendNode(double);
  void insertNode(double);
  void deleteNode(double);
  void displayList() const;
};
#endif
```

Exercise 2: Implementing the numberlist class

```
// Implementation file for the NumberList class
#include <iostream>  // For cout
#include "NumberList.h"
using namespace std;

//****************************************************
// appendNode appends a node containing the       *
// value pased into num, to the end of the list.   *
//****************************************************

void NumberList::appendNode(double num)
{
   ListNode *newNode;  // To point to a new node
   ListNode *nodePtr;  // To move through the list

   // Allocate a new node and store num there.
   newNode = new ListNode;
   newNode->value = num;
   newNode->next = nullptr;

   // If there are no nodes in the list
   // make newNode the first node.
   if (!head)
      head = newNode;
   else  // Otherwise, insert newNode at end.
   {
      // Initialize nodePtr to head of list.
      nodePtr = head;

      // Find the last node in the list.
      while (nodePtr->next)
         nodePtr = nodePtr->next;

      // Insert newNode as the last node.
      nodePtr->next = newNode;
   }
}

//**************************************************
// displayList shows the value                      *
// stored in each node of the linked list           *
// pointed to by head.                              *
//**************************************************

void NumberList::displayList() const
{
   ListNode *nodePtr;  // To move through the list

   // Position nodePtr at the head of the list.
   nodePtr = head;

   // While nodePtr points to a node, traverse
   // the list.
   while (nodePtr)
   {
      // Display the value in this node.
      cout << nodePtr->value << endl;

      // Move to the next node.
      nodePtr = nodePtr->next;
   }
}
```

```
//**************************************************
// The insertNode function inserts a node with     *
// num copied to its value member.                *
//**************************************************

void NumberList::insertNode(double num)
{
  ListNode *newNode;                                // A new node
  ListNode *nodePtr;                                // To traverse the list
  ListNode *previousNode = nullptr;   // The previous node

  // Allocate a new node and store num there.
  newNode = new ListNode;
  newNode->value = num;

  // If there are no nodes in the list
  // make newNode the first node
  if (!head)
  {
    head = newNode;
    newNode->next = nullptr;
  }
  else  // Otherwise, insert newNode
  {
    // Position nodePtr at the head of list.
    nodePtr = head;

    // Initialize previousNode to nullptr.
    previousNode = nullptr;

    // Skip all nodes whose value is less than num.
    while (nodePtr != nullptr && nodePtr->value < num)
    {
      previousNode = nodePtr;
      nodePtr = nodePtr->next;
    }

    // If the new node is to be the 1st in the list,
    // insert it before all other nodes.
    if (previousNode == nullptr)
    {
      head = newNode;
      newNode->next = nodePtr;
    }
    else  // Otherwise insert after the previous node.
    {
      previousNode->next = newNode;
      newNode->next = nodePtr;
    }
```

```
   }
}

//**************************************************
// The deleteNode function searches for a node     *
// with num as its value. The node, if found, is   *
// deleted from the list and from memory.          *
//**************************************************

void NumberList::deleteNode(double num)
{
   ListNode *nodePtr;       // To traverse the list
   ListNode *previousNode;  // To point to the previous node

   // If the list is empty, do nothing.
   if (!head)
      return;

   // Determine if the first node is the one.
   if (head->value == num)
   {
      nodePtr = head->next;
      delete head;
      head = nodePtr;
   }
   else
   {
      // Initialize nodePtr to head of list
      nodePtr = head;

      // Skip all nodes whose value member is
      // not equal to num.
      while (nodePtr != nullptr && nodePtr->value != num)
      {
         previousNode = nodePtr;
         nodePtr = nodePtr->next;
      }

      // If nodePtr is not at the end of the list,
      // link the previous node to the node after
      // nodePtr, then delete nodePtr.
      if (nodePtr)
      {
         previousNode->next = nodePtr->next;
         delete nodePtr;
      }
   }
}

//**************************************************
```

```
// Destructor                                          *
// This function deletes every node in the list.   *
//**************************************************

NumberList::~NumberList()
{
  ListNode *nodePtr;   // To traverse the list
  ListNode *nextNode;  // To point to the next node

  // Position nodePtr at the head of the list.
  nodePtr = head;

  // While nodePtr is not at the end of the list...
  while (nodePtr != nullptr)
  {
    // Save a pointer to the next node.
    nextNode = nodePtr->next;

    // Delete the current node.
    delete nodePtr;

    // Position nodePtr at the next node.
    nodePtr = nextNode;
  }
}
```

Exercise 3: implementing the linked list class in the main function

```
// This program demonstrates the deleteNode member function.
#include <iostream>
#include "NumberList.h"
using namespace std;

int main()
{
  // Define a NumberList object.
  NumberList list;

  // Build the list with some values.
  list.appendNode(2.5);
  list.appendNode(7.9);
  list.appendNode(12.6);

  // Display the list.
  cout << "Here are the initial values:\n";
  list.displayList();
  cout << endl;

  // Delete the middle node.
  cout << "Now deleting the node in the middle.\n";
  list.deleteNode(7.9);
  // Display the list.
  cout << "Here are the nodes left.\n";
  list.displayList();
  cout << endl;

  // Delete the last node.
  cout << "Now deleting the last node.\n";
  list.deleteNode(12.6);

  // Display the list.
  cout << "Here are the nodes left.\n";
  list.displayList();
  cout << endl;

  // Delete the only node left in the list.
  cout << "Now deleting the only remaining node.\n";
  list.deleteNode(2.5);

  // Display the list.
  cout << "Here are the nodes left.\n";
  list.displayList();
  return 0;
}
```