COP 2535: Data Structures

Lab 01, Linked List

1. Read pages 51 - 60 in Mastering Algorithms with C
2. Implement the following program.
3. Upload the output of your execution as text.

```
// https://www.bogotobogo.com/cplusplus/linkedlist.php
#include <iostream>

using namespace std;

struct Node {
        int data;
        Node* next;
};

// only for the 1st Node
void initNode(struct Node* head, int n) {
        head->data = n;
        head->next = NULL;
}

// apending
void addNode(struct Node* head, int n) {
        Node* newNode = new Node;
        newNode->data = n;
        newNode->next = NULL;

        Node* cur = head;
        while (cur) {
                if (cur->next == NULL) {
                        cur->next = newNode;
                        return;
                }
                cur = cur->next;
        }
}

void insertFront(struct Node** head, int n) {
        Node* newNode = new Node;
        newNode->data = n;
        newNode->next = *head;
        *head = newNode;
}

struct Node* searchNode(struct Node* head, int n) {
```

```
        Node* cur = head;
        while (cur) {
                if (cur->data == n) return cur;
                cur = cur->next;
        }
        cout << "No Node " << n << " in list.\n";
}

bool deleteNode(struct Node** head, Node* ptrDel) {
        Node* cur = *head;
        if (ptrDel == *head) {
                *head = cur->next;
                delete ptrDel;
                return true;
        }

        while (cur) {
                if (cur->next == ptrDel) {
                        cur->next = ptrDel->next;
                        delete ptrDel;
                        return true;
                }
                cur = cur->next;
        }
        return false;
}

/* reverse the list */
struct Node* reverse(struct Node** head)
{
        Node* parent = *head;
        Node* me = parent->next;
        Node* child = me->next;

        /* make parent as tail */
        parent->next = NULL;
        while (child) {
                me->next = parent;
                parent = me;
                me = child;
                child = child->next;
        }
        me->next = parent;
        *head = me;
        return *head;
}

/* Creating a copy of a linked list */
void copyLinkedList(struct Node* node, struct Node** pNew)
{
        if (node != NULL)
    {
                *pNew = new Node;
                (*pNew)->data = node->data;
```

```
                (*pNew)->next = NULL;
                copyLinkedList(node->next, &((*pNew)->next));
        }
}

/* Compare two linked list */
/* return value: same(1), different(0) */
int compareLinkedList(struct Node* node1, struct Node* node2)
{
        static int flag;

        /* both lists are NULL */
        if (node1 == NULL && node2 == NULL) {
                flag = 1;
        }
        else {
                if (node1 == NULL || node2 == NULL)
                        flag = 0;
                else if (node1->data != node2->data)
                        flag = 0;
                else
                        compareLinkedList(node1->next, node2->next);
        }

        return flag;
}

void deleteLinkedList(struct Node** node)
{
        struct Node* tmpNode;
        while (*node) {
                tmpNode = *node;
                *node = tmpNode->next;
                delete tmpNode;
        }
}

void display(struct Node* head) {
        Node* list = head;
        while (list) {
                cout << list->data << " ";
                list = list->next;
        }
        cout << endl;
        cout << endl;
}

int main()
{
        struct Node* newHead;
        struct Node* head = new Node;

        initNode(head, 10);
        display(head);
```

```
addNode(head, 20);
display(head);

addNode(head, 30);
display(head);

addNode(head, 35);
display(head);

addNode(head, 40);
display(head);

insertFront(&head, 5);
display(head);

int numDel = 5;
Node* ptrDelete = searchNode(head, numDel);
if (deleteNode(&head, ptrDelete))
        cout << "Node " << numDel << " deleted!\n";
display(head);

cout << "The list is reversed\n";
reverse(&head);
display(head);

cout << "The list is copied\n";
copyLinkedList(head, &newHead);
display(newHead);

cout << "Comparing the two lists...\n";
cout << "Are the two lists same?\n";
if (compareLinkedList(head, newHead))
        cout << "Yes, they are same!\n";
else
        cout << "No, they are different!\n";
cout << endl;

numDel = 35;
ptrDelete = searchNode(newHead, numDel);
if (deleteNode(&newHead, ptrDelete)) {
        cout << "Node " << numDel << " deleted!\n";
        cout << "The new list after the delete is\n";
        display(newHead);
}
cout << "Comparing the two lists again...\n";
cout << "Are the two lists same?\n";
if (compareLinkedList(head, newHead))
        cout << "Yes, they are same!\n";
else
        cout << "No, they are different!\n";

cout << endl;
cout << "Deleting the copied list\n";
```

```
        deleteLinkedList(&newHead);
        display(newHead);
        return 0;
}
```