COP 2535: Data Structures

Lab 03, Circular Lists

1. Read pages 82 – 84 in Mastering Algorithms with C
2. Implement the following program.
3. Upload the output of your execution as text.

```
                //https://www.softwaretestinghelp.com/circular-linked-list/
#include<iostream>
using namespace std;

struct Node
{
    int data;
    struct Node* next;
};
//insert a new node in an empty list
struct Node* insertInEmpty(struct Node* last, int new_data)
{
    // if last is not null then list is not empty, so return
    if (last != NULL)
        return last;

    // allocate memory for node
    struct Node* temp = new Node;

    // Assign the data.
    temp->data = new_data;
    last = temp;

    // Create the link.
    last->next = last;

    return last;
}
//insert new node at the beginning of the list
struct Node* insertAtBegin(struct Node* last, int new_data)
{
    //if list is empty then add the node by calling insertInEmpty
    if (last == NULL)
        return insertInEmpty(last, new_data);

    //else create a new node
    struct Node* temp = new Node;

    //set new data to node
    temp->data = new_data;
```

```
    temp->next = last->next;
    last->next = temp;

    return last;
}
//insert new node at the end of the list
struct Node* insertAtEnd(struct Node* last, int new_data)
{
    //if list is empty then add the node by calling insertInEmpty
    if (last == NULL)
        return insertInEmpty(last, new_data);

    //else create a new node
    struct Node* temp = new Node;

    //assign data to new node
    temp->data = new_data;
    temp->next = last->next;
    last->next = temp;
    last = temp;

    return last;
}

//insert a new node in between the nodes
struct Node* insertAfter(struct Node* last, int new_data, int after_item)
{
    //return null if list is empty
    if (last == NULL)
        return NULL;

    struct Node* temp, * p;
    p = last->next;
    do
    {
        if (p->data == after_item)
        {
            temp = new Node;
            temp->data = new_data;
            temp->next = p->next;
            p->next = temp;

            if (p == last)
                last = temp;
            return last;
        }
        p = p->next;
    } while (p != last->next);

    cout << "The node with data " << after_item << " is not present in the list." << endl;
    return last;

}
//traverse the circular linked list
```

```
void traverseList(struct Node* last) {
    struct Node* p;

    // If list is empty, return.
    if (last == NULL) {
        cout << "Circular linked List is empty." << endl;
        return;
    }
    p = last->next; // Point to the first Node in the list.

    // Traverse the list starting from first node until first node is visited again

    do {
        cout << p->data << "==>";
        p = p->next;
    } while (p != last->next);
    if (p == last->next)
        cout << p->data;
    cout << "\n\n";
}

//delete the node from the list
void deleteNode(Node** head, int key)
{
    // If linked list is empty retun
    if (*head == NULL)
        return;

    // If the list contains only a single node,delete that node; list is empty
    if ((*head)->data == key && (*head)->next == *head) {
        free(*head);
        *head = NULL;
    }
    Node* last = *head, * d;

    // If key is the head
    if ((*head)->data == key) {
        while (last->next != *head) // Find the last node of the list
            last = last->next;

        // point last node to next of head or second node of the list
        last->next = (*head)->next;
        free(*head);
        *head = last->next;
    }

    // end of list is reached or node to be deleted not there in the list
    while (last->next != *head && last->next->data != key) {
        last = last->next;
    }
    // node to be deleted is found, so free the memory and display the list
    if (last->next->data == key) {
        d = last->next;
        last->next = d->next;
```

```
        cout << "The node with data " << key << " deleted from the list" << endl;
        free(d);
        cout << endl;
        cout << "Circular linked list after deleting " << key << " is as follows:" << endl;
        traverseList(last);
    }
    else
        cout << "The node with data " << key << " not found in the list" << endl;
}

// main Program
int main()
{
    struct Node* last = NULL;

    last = insertInEmpty(last, 30);
    last = insertAtBegin(last, 20);
    last = insertAtBegin(last, 10);
    last = insertAtEnd(last, 40);
    last = insertAtEnd(last, 60);
    last = insertAfter(last, 50, 40);
    cout << "The circular linked list created is as follows:" << endl;
    traverseList(last);
    deleteNode(&last, 10);
    return 0;
}
```