# Department of Computing

## **Name : Mahum Samar**

## **CMS ID : 290647**

**CS250: Data Structure and Algorithms**

## **Class: BSCS 9B**

# Lab 03 : part 02

# circular Linked Lists

# Course Instructor: Dr. Yasir Faheem

## **CODE**

/\* Name: Mahum Samar

Class: BSCS-9B

CMS ID: 290647

\*/

# include<iostream>

using namespace std;

class ListNode

{

public:

int data;

ListNode \*next;

};

class CircularLinkedList

{

public:

ListNode \*lastNode;

ListNode \*loc;

ListNode \*preloc;

bool isEmpty()

{

//method to check if the list is empty or not

return lastNode == NULL;

}

void InsertAtFront(int value)

{

//method to insert the value at front in the list

//creating the new node to store the value

ListNode \*newNode = new ListNode();

newNode -> data = value;

if(isEmpty())

{

//if new node is the only node in the list

newNode -> next = newNode;

lastNode = newNode;

}

else

{

//if there are already existing nodes in the list

newNode -> next = lastNode -> next;

lastNode -> next = newNode;

}

PrintList();

}

void InsertAtEnd(int value)

{

ListNode \*newNode = new ListNode();

newNode -> data = value;

if(!isEmpty())

{

//IF THERE ARE ALREADY EXISTING NODES IN THE LIST

newNode -> next = lastNode -> next;

lastNode -> next = newNode;

lastNode = newNode;

}

else

{

//if new node is the only node in the list

newNode -> next = newNode;

lastNode = newNode;

}

PrintList();

}

void SearchNode(int value)

{

//method to search the value in the list

//initializing the loc and preloc

loc = NULL;

preloc = lastNode;

if(isEmpty())

{

//if the list is empty.

cout << "The list is already empty." << endl;

return;

}

//incrementing loc and preloc

loc = lastNode -> next;

preloc = lastNode;

while(loc != lastNode && loc -> data < value)

{

//loop executes until not the end of the list or the

//logical location of the value is passed.

preloc = loc;

loc = loc -> next;

}

if(loc -> data != value)

{

//if the value is not found

if(loc -> data < value)

{

//to save the location of the lastNode

//if the logical location of the last node

//is after the lastNode

preloc = lastNode;

}

loc = NULL;

}

}

void PrintList()

{

if(!isEmpty())

{

//method to print the list on the screen

ListNode \*temp = lastNode -> next; //temp pointer variable to print the current node

do

{

//loop to print the data on the screen and incrementing temp until the whole list is printed.

cout << temp -> data << " ";

temp = temp -> next;

}while(temp != lastNode -> next); //executing stops when the 1st node is reached again.

cout << endl;

}

else

{

//if list is empty

cout << "List is Empty." << endl;

}

}

void InsertSorted(int value)

{

//method to insert the value at its logical position in the list.

SearchNode(value);

if(loc != NULL)

{

//if the value already exists.

cout << "This value already exists in the list." << endl;

}

else

{

if((preloc == NULL) || ((preloc == lastNode) && (lastNode -> next) -> data > value))

{

///if preloc is NULL, or at the last nodeor the 1st element is already greater then the value

//so value to be inserted is at the front of the list.

InsertAtFront(value);

}

if(preloc == lastNode && lastNode -> data < value)

{

//if preloc is at the lastNode and value is larger

//than lastNode value then insertion at the end of the list

InsertAtEnd(value);

}

else

{

// if insertion is in the middle of the list.

ListNode \*newNode = new ListNode();

newNode -> data = value;

newNode -> next = preloc -> next;

preloc -> next = newNode;

}

}

PrintList();

}

void Delete(int value)

{

//method to delete the specific value in the list if it exists/

SearchNode(value);

//if value is found

if(loc != NULL)

{

if(preloc == loc)

{

//if only single node

lastNode = NULL;

}

else

if(loc == lastNode)

{

//if the value is at the last node

preloc -> next = lastNode -> next;

lastNode = preloc;

}

else

{

//if value is in the middle of the list

preloc -> next = loc -> next;

}

delete loc;

}

else

{

//if value is not found.

cout << "Value not found." << endl;

}

PrintList();

}

void DestroyList()

{

//method to destroy the list both logically and physically

ListNode \*temp = lastNode -> next;

while(temp != temp -> next)

{

//loop to delete the nodes.

temp = lastNode -> next;

lastNode -> next = lastNode -> next -> next;

delete temp;

}

delete lastNode;

lastNode = NULL;

PrintList();

}

};

int main()

{

CircularLinkedList \*circularLinkedList = new CircularLinkedList();

cout << "Inserting 5 at front:" << endl;

circularLinkedList -> InsertAtFront(5);

cout << endl;

cout << "Inserting 4 at front:" << endl;

circularLinkedList -> InsertAtFront(4);

cout << endl;

cout << "Inserting 2 at front:" << endl;

circularLinkedList -> InsertAtFront(2);

cout << endl;

cout << "Inserting 6 at end:" << endl;

circularLinkedList -> InsertAtEnd(6);

cout << endl;

cout << "Searchong 2 in the List:" << endl;

circularLinkedList -> SearchNode(2);

if(circularLinkedList -> loc == NULL)

{

cout << "Value not found" << endl;

}

else

{

cout << "Value found" << endl;

}

cout << endl;

cout << "Inserting 3 at its logical position in ascending order.:" << endl;

circularLinkedList -> InsertSorted(3);

cout << endl;

cout << "Deleting 3 from the list:" << endl;

circularLinkedList -> Delete(3);

cout << endl;

cout << "Destroying the whole List" << endl;

circularLinkedList -> DestroyList();

cout << endl;

circularLinkedList -> PrintList();

cout << "Printing the whole list after destroying but nothing printed" << endl;

}

## **OUTPUT**

