# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BSCS 11 A&C**

# 

# Lab 03-Part(A) : Circular Linked Lists

**Date: 04 October, 2021**

**Time - 02:00 pm – 4:50 pm**

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Hadaiq Ahmad

# 

# Lab 03-Part(A): Circular Linked List

**Introduction**

This lab will introduce students with the practical implementation of Circular Linked list with its operations.

**Objectives**

The objective of this lab session is to acquire skills in working with Circular linked lists.

**Tools/Software Requirement**

Visual Studio c++, Eclipse C++ IDE

**Helping Material**

Lecture slides, text book

**Description**

The task is to first implement the following operations:

* bool IsEmpty(); // checks whether the list is empty or not. Returns true if empty and false otherwise.
* InsertAtFront(value); // takes input from a user and inserts it at front of the list
* InsertAtEnd(value); // takes input from a user and inserts it at the tail end of the list
* Search(value); This function shall search value in a list. If found, we will need to store two addresses:
  1. Address of the node in which the searched value is found in a pointer variable named Loc\_; we will store NULL in Loc\_ in case value is not found.
  2. Address of the node which is logical predecessor of value in a list.

The Search() provides functionality for other operations such as insertion in a sorted list, deleting a value, modifying a value, printing it etc.

* InsertSorted(value); //If we want to maintain a sorted list, we should implement this function
* Delete(value); // searches value and then deletes it if found.
* DestroyList(); // Distroys all nodes of the list leaving the list in empty state.

**Declare Node Class:** The data structure that will hold the elements of the list is called **Node.** Declare it as follows:

***class*** *ListNode{*

***public****:*

***int*** *data;*

*ListNode \*next;*

*};*

**Declare class Linked List:** Now, declare your main class LinkedList:

***class*** *CircularLinkedList{*

***public****:*

*ListNode \*list; // special variable which stores address of last node.*

*ListNode \*PredLoc\_; //to be used by Search(value) method to store address of logical predecessor of value in a list.  
 ListNode \*Loc\_; //to be used by Search(value) method to store address of the node containing the searched value in a list. If it is not found it contains NULL.*

*}*

1. **Creating a LinkedList**

In order to create an empty list, assign NULL value to **list** pointer variable.

*CircularLinkedList(){*

*list=NULL;*

*PredLoc\_=NULL;*

*Loc\_=NULL;*

*}*

1. **Bool IsEmpty() function**

By checking content of the special pointer variable list, this function should return true value if the list is empty and false otherwise.

1. **Inserting a value at the Front of a list**

First, Reserve space for a new node to be inserted in the list by creating object of class ListNode and storing its address in a temporary pointer variable.

*ListNode \*newnode =* ***new*** *ListNode();*

Now store value in data part of the new node: *newnode->data=value;*

Finally, link newnode at the front of the linked. Note that there are two special cases for Insertion at Front:

1. Insertion into Empty List. Use the following statements for linking new node:

*newnode->next=newnode;*

*list=newnode;*

1. Insertion in an existing list. Following statements will link new node at the front of an existing list:

*newnode->next=list->next;*

*list->next=newnode;*

1. **Inserting a value at the Tail end of a list:**

First, Reserve space for a new node to be inserted in the list by creating object of class ListNode and storing its address in a temporary pointer variable.

*ListNode \*newnode =* ***new*** *ListNode();*

Now store value in data part of the new node: *newnode->data=value;*

Finally, link newnode at the tail end. Note that there are two special cases for Insertion at the tail end:

1. Insertion into an empty list. Use the following statements for linking a new node:

*newnode->next=newnode;*

*list=newnode;*

1. Insertion in an existing list. Following statements will link new node at the tail end of an existing list:

*newnode->next=list->next;*

*list->next=newnode;*

*list=newnode;*

1. **void PrintList()**

***if****( list not empty){*

*ListNode \*temp=list->next;*

***do****{*

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

*temp=temp->next;*

*}* ***while****(temp!=list->next);*

*cout<<****endl****;*

*}*

***else***

*cout<<"List is Empty";*

1. **Void Search( value)**

* This function shall search a value in a list. If found, we will need to store two addresses:
  1. Address of the node in which the searched value is found in a pointer variable named Loc\_; we will store NULL in Loc\_ in case the searched value is not found.
  2. Address of the node which is the logical predecessor of value in the list.

**Void search(value){**

Initialize loc & ploc to NULL before starting search.

Start search only if the list is non-empty. Before doing so, set Loc\_ to first node and PLoc equal to last node.

Loc= address of head node

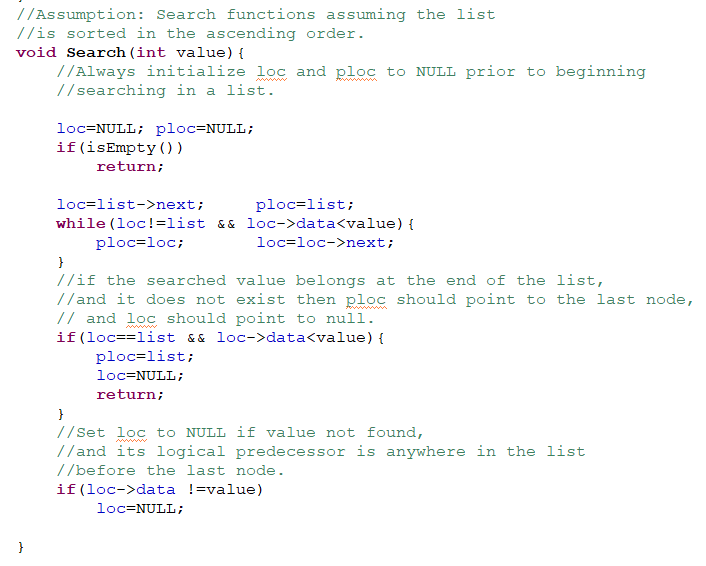
Ploc = address of logical predecessor of head node i.e. last node in this case.

For the moment assume that we are maintaining a list sorted in ascending order. Search value until we logical position of the value is passed or we have searched the entire list.

//Search value until:

// 1 : we reach its logical place in the list, or

// 2 : we reach the end of the list



After execution of search(value) method, there are four possible combinations of loc and ploc

|  |  |  |
| --- | --- | --- |
| Ploc | Loc | Interpretation |
| Null | Null | List is empty. |
| list | list | Value found in a list of length 1. |
| list | Null | Value not found. In a sorted list, its logical position is:   * At the front end, if the value is smaller than the first element of the list. * At the tail end, if the value is larger than the last element of the list. |
| Non-null & !=list | Null | Value not found. Its logical position is somewhere between head and tail nodes. |
| list | Non-null | Value found in the head node of the list |
| !=list | Non-null | Value found but not in the head node. As ploc is not equal to last, it might be in any node other than the head node |
|  |  |  |

1. **Insertion in a Sorted List**

For the moment, assume duplications are not allowed in the list. You have to insert value after call to search function by considering the above mentioned four possible combinations of loc and ploc pointer variables.

*InsertSorted(value){*

*Search(value)*

*If(value already exists)*

*Return without insertion and print a message*

*Else{*

*if (list is empty)*

*//insert statements here*

*else if (position of value is as last node)*

*Insert value at the tail end.*

*else*

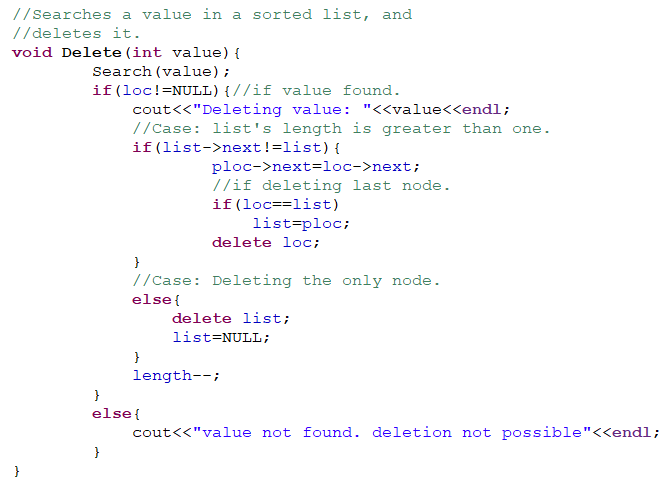
*Insert after ploc.*

*}*

*}*

1. **Delete a Value**

Find value using search method and if a node containing the searched value is found, then delete it from the linked list. Also free the allocated memory.



1. **Destroy a Linked List:** This function should delete all nodes of a linked list one by one leaving the list in empty state.

**void** **DestroyList**(){

**if**(list not empty){

node\* temp;

**while**(more than 1 nodes in a list){

//set temp to point to the current head node

//delete the current head node

}

delete the last remaining node;

set list to NULL;

}

}

**Solution :**

*#include* <iostream>

*#include* <string>

*using* *namespace* std;

*class* Node

{

*public:*

    int data;

    Node \*next;

};

*class* CLL

{

    Node \*list;

    Node \*loc;

    Node \*ploc;

    CLL()

    {

        list = NULL;

        ploc = NULL;

        loc = NULL;

    }

    bool isEmpty();

    void insertAtFront(int);

    void insertAtTail(int);

    void printList();

    void Search(int);

    void insertedSorted(int);

    void Delete(int);

    void DestroyList();

};

bool CLL ::isEmpty()

{

*return* list == NULL;

}

void CLL ::insertAtFront(int data)

{

    Node \*newnode = new Node();

    newnode->data = data;

*// NOT EMPTY CASE*

*if* (!isEmpty())

    {

        newnode->next = list->next;

        list->next = newnode;

    }

*// EMPTY CASE*

*else*

    {

        list = newnode;

        list->next = newnode;

    }

}

void CLL ::insertAtTail(int data)

{

    Node \*newnode = new Node();

    newnode->data = data;

*// NOT EMPTY CASE*

*if* (!isEmpty())

    {

        newnode->next = list->next;

        list = newnode;

    }

*// EMPTY CASE*

*else*

    {

        list = newnode;

        list->next = newnode;

    }

}

void CLL ::printList()

{

*// NOT EMPTY CASE*

    Node \*p = list->next;

*if* (!isEmpty())

    {

*while* (p != list->next)

        {

            cout << p->data << endl;

            p = p->next;

        }

    }

*else*

    {

        cout << 'LIST IS EMPTY' << endl;

    }

}

void CLL ::Search(int value)

{

    loc, ploc = NULL;

*if* (isEmpty())

*return*;

    ploc = list;

    loc = ploc->next;

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

    {

        ploc = loc;

        loc = loc->next;

    }

*if* (loc == list && loc->data < value)

    {

        ploc = list;

        loc = NULL;

*return*;

    }

*if* (loc->data != value)

    {

        loc = NULL;

    }

}

void CLL::insertedSorted(int data)

{

    Search(data);

*if* (loc != NULL)

    {

        cout << "VALUE EXIST" << endl;

    }

*else*

    {

        Node \*newnode = new Node();

        newnode->data = data;

*if* (isEmpty())

        {

            list = newnode;

            list->next = list;

        }

*else* *if* (loc == list)

        {

            insertAtTail(data);

        }

*else*

        {

            ploc->next = newnode;

            newnode->next = loc;

        }

    }

}

void CLL::Delete(int data)

{

    Search(data);

*if* (loc != NULL)

    {

        cout << "DELETING THE VALUE :" << endl;

*if* (list != list->next)

        {

            ploc->next = loc->next;

*if* (loc == list)

            {

                list = ploc;

            }

            delete loc;

        }

*else*

        {

            delete list;

            list = NULL;

        }

    }

*else*

    {

        cout << "VALUE DOESN'T EXIST" << endl;

    }

}

void CLL::DestroyList()

{

*if* (!isEmpty())

    {

        Node \*temp;

*while* (list != list->next)

        {

            temp = list->next;

            list->next = temp->next;

            delete temp;

        }

        delete list;

        list = NULL;

    }

}

**Deliverables:**

Compile a single word document by filling in the solution part and submit this Word file on LMS. The name of word document should follow this format. i.e. **YourFullName(reg)\_Lab#.** This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS.

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.