# Department of Computing

# School of Electrical Engineering and Computer Science

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

**Class: BSCS 10AB**

# 

# Lab 03-Part(B) : Doubly Linked Lists

**Date: 05th October, 2021**

**Time: 10:00 am – 12:50 pm   
&  
 02:00 pm – 4:50 pm**

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Aftab Farooq

# MASHAL ASHFAQUE 337203

# CS 10-A

# Lab 03-Part(B): Doubly Linked List

**Introduction**

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

**Objectives**

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

**Tools/Software Requirement**

Visual Studio c++

**Helping Material**

Lecture slides, text book

**Description**

Your task is to implement the following operations of a doubly linked list:

* 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 tail end of the list
* InsertSorted(value); //If we want to maintain a sorted list, we should implement this function
* 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.

* int DeleteFront(value) ; Deletes front node of the list and also returns the value stored in it
* int DeleteLast(value) ; Deletes front node of the list and also returns the value stored in it
* 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;*

*ListNode \*prev;*

*};*

1. **Linked List**

Now, declare your main class LinkedList:

***class*** *DoublyLinkedList{*

***public****:*

*ListNode \*first; // special variable which stores address of head node.*

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

*}*

**Creating a LinkedList**

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

*DoublyLinkedList(){*

*first=NULL; last=NULL;*

*PredLoc\_=NULL; Loc\_=NULL;*

*}*

1. **Inserting value at Front:**

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

Insertion at front of a doubly linked list has two special cases:

1. Insertion into an empty list is a special case; you will have to update both the first and last pointer variables.

*first=newnode; last=newnode;*

1. Insertion at front of a list is a general case; you will have to update only the pointer variable first.

*newnode->next=first;*

*first->prev=newnode;*

*first=newnode;*

1. **Inserting value at Tail End:**

After reserving space for the new node and inserting value in its data part, you will have to link it at the end of the list. Insertion at tail end of a doubly linked list has two special cases:

1. Insertion into an empty list is a special case; you will have to update both the first and last pointer variables.

*first=newnode; last=newnode;*

1. Insertion at tail of a list is a general case; you will have to update only the pointer variable last.

*last->next = newnode;*

*newnode->prev=last;*

*last=newnode;*

1. **Search a Value Function**

* This function shall search a value in a list. If found, we will need to store two addresses:
  1. The Address of the node in which the searched value is found. We shall store it in a pointer variable named Loc\_. In case the searched value is not found in the list, we will store NULL in Loc\_.
  2. Moreover, we will store the address of the logical predecessor node of the value we are searching for. For this purpose, we shall use a pointer named PLoc.

1. **Search value in a list**

***Void search(value){***

*Initialize loc & ploc*

*Loc= address of head node*

*Ploc = address of logical predecessor of head node. Note that the first node has no predecessor. Thus, always initialize Ploc with NULL value.*

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

*While (loc!=NULL and loc.data < value){*

*Advance both ploc and loc*

*}*

*If(loc!null & loc.data!=value)*

*Loc=null; //as value is not found so set loc equal to null.*

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

**Table 1: Possible Values in Loc and Ploc after Call to Search() and their Interpretation**

|  |  |  |
| --- | --- | --- |
| **Ploc** | **Loc** | **Interpretation** |
| Null | Null | Value not found, and its logical position is at the front of the list |
| Null | Non-null | Value found in the head node of the list |
| Non-null | Non-null | Value found but node in the head node. As ploc is non-null, it might be in any node other than the head node |
| Non-null & ploc=last | Null | Value not found. Its logical position is at the end of the list. |
| Non-null | Null | Value not found. Its logical position is somewhere after first and before last 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 variable. This method should insert new node in the sorted list for all the three special cases:

1. Insertion at front of the list. This should be done when ploc=NULL after search
2. Insertion at tail end of the list. This should be done when ploc=last after search
3. General case insertion anywhere after first node and before last node.

***InsertSorted(value)****{*

*Search(value)*

***if*** *(value already exists)*

*Return without insertion and print a message*

***else*** *{*

***if*** *(position of value is as head node)*

*Insert value at front.*

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

*Insert value at tail end.*

***else*** *{ //insert after ploc.*

*Newnode->next= ploc->next;*

*Newnode->prev= ploc;*

*Ploc->next->prev= newnode;*

*Ploc->next = newnode;*

*}*

*}*

*}*

1. **Delete Front Node**

***If(*** *list is not empty){*

*ListNode \*temp = first;*

*first = first->next;*

*first->prev=NULL;*

*delete temp;*

*}*

1. **Delete last Node:**

***If(*** *list is not empty){*

*ListNode \*temp = last;*

*last = last->prev;*

*last->next=NULL;*

*delete temp;*

*}*

1. **Delete a Value**

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

*Delete(value){*

*//if empty return*

*Search(value)*

*If(value is found){ //check loc*

*If(value is in the head node){*

*//delete head node and free memory*

*}*

*Else if (value is in the last node)*

*//delete last node and free memory*

*Else {*

*//delete the node using ploc*

*ploc->next = loc->next;*

*loc->next->prev=ploc;*

*delete loc;*

*}*

*}*

1. **Destroy a Linked List:**

This method should destroy all nodes of a linked list making it empty. It should also free space allocated for all the nodes.

**Hint:** Save address of current head node in a temporary pointer variable. Advance start variable to second node so that it becomes new head node. Then, delete current head node using temporary pointer variable.

**Lab and Home Activities**

1. Write a function which rearranges order of the entire doubly linked list by reversing it.
2. Write a function which takes two values as input from the user and searches them in the list. If both the values are found, your task is to swap both the nodes in which these values are found. Note, that you are not supposed to swap values.
3. Write a function which rearranges the linked list by group nodes having even numbered and odd numbered value in their data part.

**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. In case of any problems discuss it by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).

**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.

**CODE:**

#include <iostream>

using namespace std;

class ListNode

{

public:

int data;

ListNode\* next;

ListNode\* prev;

};

class DoublyLinkedList {

public:

ListNode\* first; // special variable which stores address of head node.

ListNode\* last; // special variable which stores address of the last node. ListNode\* preloc; //to be used by Search(value) method to store address of logical predecessor of value in a list.

ListNode\* preloc = NULL;

ListNode\* loc = NULL; //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.

int length = 0;

DoublyLinkedList() {

first = NULL; last = NULL;

preloc = NULL; loc = NULL;

}

bool IsEmpty(); // checks whether the list is empty or not. Returns true if empty and false otherwise

void InsertAtFront(int value); // takes input from a user and inserts it at front of the list

void InsertAtEnd(int value); // takes input from a user and inserts it at the tail end of the list

void printlist(bool);

void search(int value);

void InsertSorted(int value); //If we want to maintain a sorted list, we should implement this function

void Delete(int value);

void DestroyList();

};

bool DoublyLinkedList::IsEmpty()

{

return first == NULL;

}

void DoublyLinkedList::InsertAtFront(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (first == NULL)

{

first = newnode;

last = newnode;

}

else

{

newnode->next = first;

first->prev = newnode;

first = newnode;

}

}

void DoublyLinkedList::InsertAtEnd(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (IsEmpty())

{

first = newnode;

last = newnode;

}

else

{

last->next = newnode;

newnode->prev = last;

last = newnode;

}

}

void DoublyLinkedList::search(int value)

{

if (IsEmpty())

{

return;

}

loc = first;

preloc = NULL;

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

{

preloc = loc;

loc = loc->next;

}

if (loc != NULL && loc->data < value)

{

preloc = last;

loc = NULL;

}

}

void DoublyLinkedList::InsertSorted(int value)

{

search(value);

if (loc == NULL)

{

if (preloc == NULL) //position at front end

{

InsertAtFront(value);

}

else if (preloc == last) //insertion at tail end

{

InsertAtEnd(value);

}

else //insertion in middle

{

ListNode\* newnode = new ListNode();

newnode->data = value;

newnode->prev = preloc;

newnode->next = preloc->next;

preloc->next->prev = newnode;

preloc->next = newnode;

}

}

}

void DoublyLinkedList::Delete(int value)

{

search(value);

if (loc != NULL)

{

if (preloc == NULL)//deletion at front end

{

if (loc == last)//lenghth is 1}

{

first = NULL; last = NULL;

}

else {

first = first->next;

first->prev = NULL;

}

}

else //deletion after first node

{

if (loc == last)

{

last = preloc; preloc->next = NULL;

}

else

{

loc->next->prev = preloc;

preloc->next = loc->next;

}

}

//deallocated memory pointed to by loc

}

else

{

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

}

}

void DoublyLinkedList::printlist(bool direction)

{

if (!IsEmpty())

{

ListNode\* temp;

if(direction){//print in forward direction

temp = first;

while (temp != NULL)

{

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

temp = temp->next;

}

}

else //reverse direction printing

{

temp = last;

while (temp != NULL)

{

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

temp = temp->prev;

}

cout << endl;

}

}

else

{

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

}

}

void DoublyLinkedList::DestroyList()

{

ListNode\* temp;

while (first != NULL)

{

temp = first;

first = first->next;

delete temp;

}

last = NULL;

first = NULL;

}

int main()

{ //implementation of all functions

DoublyLinkedList mylist;

int len; int val;

cout << "How much length list you want to create?" << endl;

cin >> len;

for (int i = 0; i < len; i++)

{

cout << "Enter element no. " << i + 1 << endl;

cin >> val;

mylist.InsertSorted(val);

}

cout << endl << "Press 0 to view list in reverse. Press 1 to view in forward direction." << endl;

int dir; cin >> dir;

cout << endl << "The List:" << endl;

mylist.printlist(dir);

cout << endl << "Which element do you want to delete?" << endl;

cin >> val;

mylist.Delete(val);

cout << endl << "The New List:" << endl;

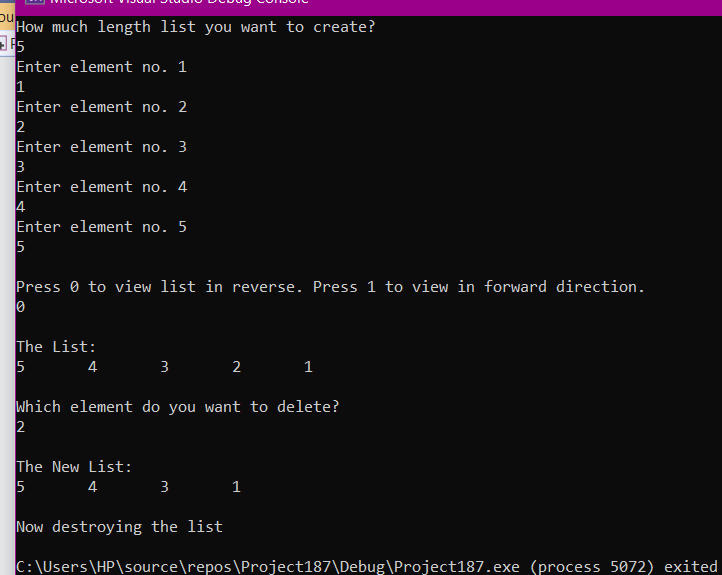
mylist.printlist(dir);

cout << endl << "Now destroying the list" << endl;

mylist.DestroyList();

}

**Output:**

****