# Department of Computing

# School of Electrical Engineering and Computer Science

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

**Class: BSCS 10AB**

# 

# Lab 02: Singly Linked Lists

**Date: 28th September, 2021**

**Time: 10:00 am – 12:50 pm**

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Aftab Farooq

# 

# Lab 02: Singly Linked List

**Introduction**

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

**Objectives**

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

**Tools/Software Requirement**

Visual Studio c++

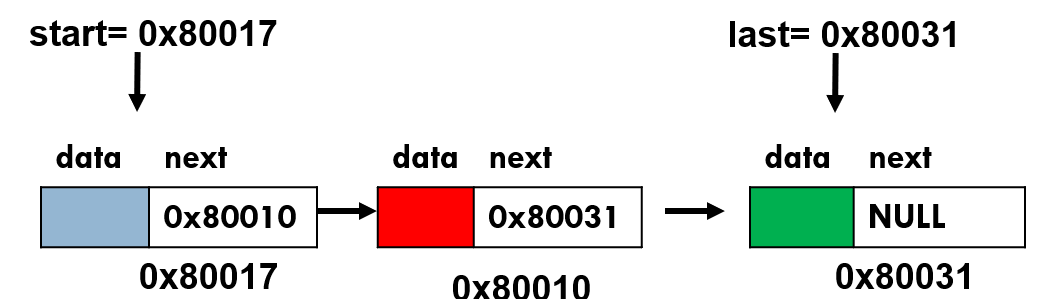
**Helping Material**

Lecture slides, text book

**Description**

The task is to first implement the following operations:

1. Create an empty linked list; // do so in the constructor.
2. bool IsEmpty(); // checks whether the list is empty or not. Returns true if empty and false otherwise.
3. InsertAtFront(value); // takes input from a user and inserts it at the front of a list
4. InsertAtEnd(value) ; // takes input from a user and inserts it at the tail end of a list
5. PrintList();
6. InsertSorted(value); //To maintain a sorted list, you shall implement this function. Note that if you are maintaining a sorted list then do not call InsertAtFront(value) and InsertAt(Front) functions in the main function.
7. 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.
      1. The Search() provides functionality for other operations such as insertion in a sorted list, deleting a value, modifying a value, printing it etc.
8. Delete(value); // searches value and then deletes it if found.
9. 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. Inside this class, you shall define all key functions to implement all operations of a linked list.

***class*** *LinkedList{*

***public****:*

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

*}*

1. **Creating a LinkedList**

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

*LinkedList(){*

*start=NULL;*

*PredLoc\_=NULL;*

*Loc\_=NULL;*

*}*

1. **Bool IsEmpty() function**

By checking content of the special pointer variable start/first, 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;*

If you are inserting in an empty list, you should update both the front and last pointer variables as follows:

*start=newnode;*

*last=newnode;*

Finally, link newnode at front of the linked list via following two statements:

*newnode->next=start;*

*start=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;*

If you are inserting in an empty list, you should update both the front and last pointer variables as follows:

*start=newnode;*

*last=newnode;*

Finally, link newnode at front of the linked list via following two statements:

*last->next=newnode;*

*last=newnode;*

1. **void PrintList()**

This function prints all elements of a linked list starting from the first one.

*ListNode \*temp = start;*

*If(list not empty){*

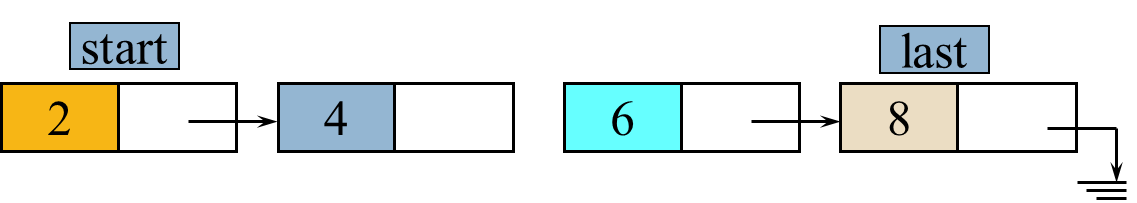
*While(not end of list){*

*cout<<temp->data<<” “;*

*// advance temp to successor node.*

*}*

*}*



1. **Void Search( value)**

As discussed in the class, we will implement a general-purpose search function which will provide functionality to other operations like insertion, deletion, modification etc. This function shall take a value as argument from the user and then search it in the list. You should use two node pointer variables namely ploc and loc in this function.

* 1. In the variable loc, save address of the node in which the searched value is found. In case, the searched value does not exist, save NULL value in loc.
  2. In the variable ploc, we shall store address of the logical predecessor node of the searched value.

**Void search(value){**

Initialize loc & ploc

Loc= address of head node

Ploc = address of logical predecessor of head node. Note that first node has no predecessor. So, we will always initialize ploc to NULL.

For the moment assume that we are maintaining a list sorted in the ascending order. Search value until either 1) we reach the end of the list or 2) logical position of the value is passed.

*while (loc!=NULL and loc->data < value){*

*Advance both ploc and loc*

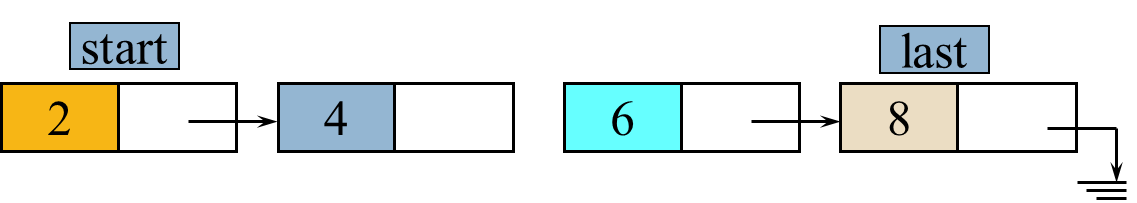
*}*

*If(loc!NULL & loc->data!=value)*

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

*} //end of search function.*

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



|  |  |  |  |
| --- | --- | --- | --- |
| **Search example** | **Ploc** | **Loc** | **Interpretation** |
| 1 | Null | Null | Searched value not found and its logical position is at the front of the list |
| 2 | Null | Non-null | Value found in the head node of the list |
| 4/6 | Non-null | Non-null | Loc=non-null implies the searched value has been found. Ploc=non-null implies the value is not in the head node; it might be in any node other than the head node.  **Special case:** loc ==last implies value found in the last node. |
| 10 | Non-null | Null | Loc=null implies searched value not found. Ploc=non-null implies its logical position is not at the front. |

1. **Insertion in a Sorted List**

For the moment, assume duplications are not allowed in the list. You have to insert new 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) //check using loc*

*Return without insertion and print a message*

*else{*

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

*Insert value at front.*

*Else //Insert newnode after ploc.*

*If insertion at the tail end, do update last.*

*Else insert value after ploc in the middle.*

*}*

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.

*Delete(value){*

*//if list empty then return*

*Search(value)*

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

*If(value is in the head node)//check ploc {*

*//delete head node and free memory*

*}*

*else{*

*//update link using ploc*

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

*//finally free memory using delete command.*

*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 the second node so that it becomes new head node. Then, delete current head node using the temporary pointer variable.

**Code:**

#include<iostream>

using namespace std;

class ListNode {

public:

int data;

ListNode\* next;

};

class LinkedList {

public:

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

int length = 0;

LinkedList() //constructor to create empty linked list

{

start = NULL;

preloc = NULL;

loc = NULL;

}

//other functions

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 the front of a list

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

void PrintList(); //prints all data in the list

void search(int value); //searches value entered by user, it will also be used in other functions like insertion etc

void InsertSorted(int value);//To maintain a sorted list, you shall implement this function.

//Note that if you are maintaining a sorted list then do not call InsertAtFront(value) and InsertAt(Front)

//functions in the main function

void Delete(int value); // searches value and then deletes it if found.

void DestroyList(); // Distroys all nodes of the list leaving the list in empty state.

};

bool LinkedList::isEmpty()

{

return start == NULL;

}

void LinkedList::InsertAtFront(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (isEmpty())

{

start = newnode;

last = newnode;

}

else

{

newnode->next = start;

start = newnode;

}

length++;

}

void LinkedList::InsertAtEnd(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (isEmpty())

{

start = newnode;

last = newnode;

}

else

{

last->next = newnode;

last = newnode;

}

}

void LinkedList::PrintList()

{

if (!isEmpty())

{

ListNode\* temp = start;

while (temp != NULL)

{

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

temp = temp->next;

}

}

else

{

cout << "List is Empty.\n";

}

}

void LinkedList::search(int value)

{

loc = start;

preloc = NULL;

if (isEmpty())

{

return;

}

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

{

preloc = loc;

loc = loc->next;

}

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

{

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

}

}

void LinkedList::InsertSorted(int value)

{

search(value);

if (loc == NULL)

{

if (preloc == NULL)

InsertAtFront(value);

else

{

ListNode\* newnode = new ListNode();

newnode->data = value;

newnode->next = preloc->next;

preloc->next = newnode;

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

last = newnode;

}

length++;

}

else

cout << value << " already exists in list. Duplication not allowed." << endl;

}

void LinkedList::Delete(int value)

{

if (isEmpty())

{

return;

}

search(value);

if (loc != NULL)

{

if (preloc == NULL)//value is in headnode

{

start = start->next;

}

else {

//update link using ploc

preloc->next = loc->next;

//finally free memory using delete command.

delete loc;

}

}

}

void LinkedList::DestroyList()

{

ListNode\* temp;

while (start != NULL)

{

temp = start;

start = start->next;

delete temp;

}

last = NULL;

}

int main()

{

LinkedList mylist;

//Implementing all functions

cout << "Insert some elements for list. To stop, press -1." << endl;

int f; cin >> f;

while (f != -1)

{

mylist.InsertSorted(f);

cin >> f;

}

cout << "Here's your list:" << endl;

mylist.PrintList();

cout << "\nNow try to enter a duplicate of a value in the list." << endl;

cin >> f;

mylist.InsertSorted(f);

cout << "Now delete a value from your list." << endl;

cin >> f; mylist.Delete(f);

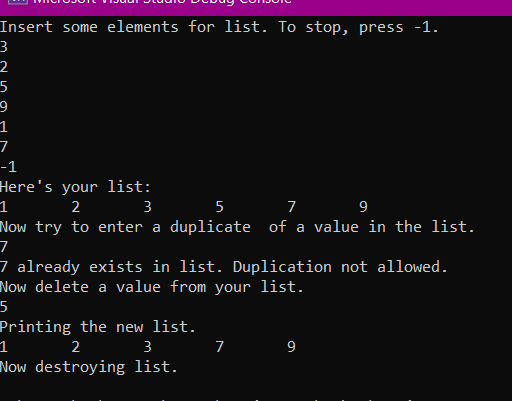
cout << "Printing the new list." << endl;

mylist.PrintList();

cout << "\nNow destroying list.\n";

mylist.DestroyList();

}



**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.