

**NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY**

**School of Electrical Engineering and Computer Sciences**

**DATA STRUCTURES AND ALGORITHM**

**LAB# 3**

**SUBMITTED BY:**

**FATIMA SEEMAB (291310)**

**CLASS + SECTION: BS CS 9B**

**Date of Submission: 4Nov , 2020.**

1. **Linked List**

#include <iostream>

using namespace std;

struct ListNode

{// A structure is declared with three data fields

int data;

ListNode\* prev;

ListNode\* next;

};

class LinkList

{

public:

//ListNode structure pointer variable is declared

ListNode\* start;

ListNode\* last;

ListNode\* loc;

ListNode\* ploc;

// Constructure to initialize LinkList Pointers

LinkList()

{

start = NULL;

last = NULL;

loc = NULL;

ploc = NULL;

}

1. **Inserting value at Front:**

//Function to tell us whether a list is empty or not

bool isEmpty()

{

return (start == NULL);

}

//A method to insert an element at front of list

void InsertAtFront(int value)

{//ListNode object is created

ListNode\* newnode = new ListNode();

//Method to check whether a list is empty

if (isEmpty())

{

start = newnode;

last = newnode;

//for insertion in previous list we have to update both start and last pointers

}

else

{ //if list contains some element

newnode->next = start;

//newnode is inserted before the start variable

start->prev = newnode;

//start pointer previous has address of new node

start = newnode;

//now start contains the address of newly inserted node

}

//to fill data in newnodee initialized portion

newnode->data = value;

}

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

//A method to insert any element at last of end

void InsertAtLast(int value)

{ // ListNode object is created

ListNode\* newnode = new ListNode();

//Method to check whether a list is empty

if (isEmpty())

{

start = newnode;

last = newnode;

//for insertion in previous list we have to update both start and last pointers

}

//if list contains some element

else

{ //last pointer next field is pointed to newnode

last->next = newnode;

//newnode previous data field is pointed to last

newnode->prev = last;

//last is pointed to new data field

last = newnode;

}

newnode->data = value;

}

1. **Search a Value Function**

//Method to search a value in link list

void Searchvalue(int value)

{//Pre Condition:if list is empty or not

if (isEmpty())

{

cout << "There is no value in list";

return;

}

//loc is pointed to start

loc = start;

//ploc is initialized to Null

ploc = NULL;

//loop iterates till list ends and value not found

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

{

//As value is found the function returns

if (loc->data == value) { return; }

//ploc points to previous of loc

ploc = loc;

//loc is pointed to its next

loc = loc->next;

}

//Condition if loop breaks bcz desired value becomes smaller than the values in list

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

{//loc is pointed to Null to show value is not found

loc = NULL;

}

}

1. **Insertion in a Sorted List**

//Method to Insert value in a sorted list

void InsertSorted(int value)

{ //searches value in list

Searchvalue(value);

//Condition for value duplicate

if (loc != NULL)

{

cout << "value already in list";

return;

//returns method when value is in list

}

else

{//if value not found

//value logical position is at start

if (ploc == NULL)

{//value vis inserted at front

InsertAtFront(value);

return;

//loop returns

}

else

{//if logical position is not in head node

//checks if logical poistion is at last

if (ploc == last)

{//value is inserted at last

InsertAtLast(value);

}

else

{// if value logical positionin somewhere mid of list

ListNode\* newnode=new ListNode();

//A newnode is created

//it is inserted after the ploc

newnode->next = ploc->next;

ploc->next = newnode;

//To update the previous field of ploc

newnode->prev = ploc;

// value is inserted in its data part

newnode->data = value;

printLinkList();

}

}

}

}

1. **Delete Front Node**

//Method to delete head node

void DeleteFrontNode()

{//Temp pointer of listNode is created

ListNode\* temp = new ListNode();

//temp is pointed to start

temp = start;

//check for list is empty or not

if (isEmpty()) { return; }

//if lsit contains any element

else {

//start points to its next node

start = start->next;

//to make next of start new start we make its previous null

start->prev = NULL;

//temp which points to initial headnode will be deleted

delete temp;

}

}

1. **Delete Last Node**

void DeleteLastNode()

{//Temp pointer of listNode is created

ListNode\* temp = new ListNode();

//temp is pointed to last

temp = last;

//check for list is empty or not

if (isEmpty())

{

return ;

}

//if list contains any element

else

{

//last points to its prev node

last = last->prev;

//to make next of new last we make its next null

last->next = NULL;

//temp which points to initial last node will be deleted

delete temp;

}

}

1. **Delete Value**

//A method to delete value in list

void DeleteValue(int value)

{//We search value to find its location in list

Searchvalue(value);

//If value is not found

if (loc == NULL)

{

cout << "value is not found in list";

return;

}

else

{//If value is in head node

if (ploc == NULL)

{//we call this to delete front node

DeleteFrontNode();

}

else

{//If value is in last node

if (ploc == last)

{//we call this to delete last node

DeleteLastNode();

}

//If value is in somewhere mid

else

{//ploc next will point to next of loc

ploc->next = loc->next;

//loc next pointer previous field is updated

loc->next->prev = ploc;

//delete loc

delete loc;

}

}

}

}

1. **DestroyList**

//Method to destroy link list

void DestroyLinkList()

{//Precondition: Whether list is empty or not

if (!isEmpty())

{//A temporary pointer of structy ListNode is created

ListNode\* temp = new ListNode();

//loop iterates till the end of list

while (start != NULL)

{

temp = start;

start = start->next;

//delete everytimes the start of list

delete temp;

}

cout << "List is destroyed";

}

}

PrintLinkList

//Method to print link list

void printLinkList()

{//A listNode temp variable is created

ListNode\* temp;

//Pointed to start

temp = start;

//loop iterates till the end

while (temp != NULL)

{

cout << temp->data;

//Everytime temp is pointed to its next

temp = temp->next;

}

}

**Lab and Home Activities**

**Question 1**

Write a function which rearranges order of the entire doubly linked list by reversing it.

void ReversingLinkList()

{

// three pointers are created to iterate over the whole last,to store the initial start,to store the last iterated one

ListNode\* temp = start;

ListNode\* iteration = start;

ListNode\* previous = NULL;

//loop continues until whole list is accessed

while (temp->next != NULL)

{

temp = temp->next;

}

//At end of list start is pointed to the last one

if (temp == last)

{

start = temp;

}

//It will store till where the loop terminates

previous = temp->prev;

//temporary next is pointed to previous one

temp->next = previous;

//temp is again pointed to initial start

temp = iteration;

//loop continues till previous element not equals start

while (previous != iteration)

{//loop iterates till the pointer last accessed is reach

while (temp != previous)

{//temp is pointed to its next one

temp = temp->next;

}

if (temp != NULL)

{//previous contains the previous of last accessed one

previous = temp->prev;

}

//the pointer is pointed to its previous element

if (temp != NULL)

{//Swapping of nodes

temp->prev = temp->next;

temp->next = previous;

//again pointed to initial start

temp = iteration;

}

}

//When loop breaks last equals temp

if (temp == iteration)

{

if (temp != NULL) {

//swaps the nodes

temp->prev = temp->next;

temp->next = NULL;

//last points to temp

last = temp;

}

}

}

**Question 3**

Write a function which rearranges the linked list by group nodes having even numbered and odd numbered value in their data part.

//A method to saperate even nodes from odd ones

void saperate\_even()

{

ListNode\* temp = start;

ListNode\* ptemp\_odd = NULL;

ListNode\* ptemp\_even = NULL;

ListNode\* temp\_odd = NULL;

ListNode\* odd\_1 = NULL;

ListNode\* temp\_even = NULL;

ListNode\* even\_1 = NULL;

//Loop continues until whole list is accessed

while (temp != NULL)

{//checks if value is even

if (((temp->data) % 2) == 0)

{//if previously even element is found

if (ptemp\_even != NULL)

{//initialized to current pointer

ptemp\_even = temp\_even;

ptemp\_even->next = temp;

temp->prev = ptemp\_even;

}

else

{//if previously no even element is found

//to store 1st even of list

even\_1 = temp;

ptemp\_even = temp;

}

//current is stored

temp\_even = temp;

//points to next

temp = temp->next;

}

//checks if value is odd

else

{//if previously odd element is found

if (ptemp\_odd != NULL)

{

//initialized to current pointer

ptemp\_odd = temp\_odd;

ptemp\_odd->next = temp;

temp->prev = ptemp\_odd;

}

else

{//if previously no odd element is found

//to store 1st odd of list

odd\_1 = temp;

ptemp\_odd = temp;

}

temp\_odd = temp;

//points to next

temp = temp->next;

}

}

//for connection of odd to even if 1st element is odd

if (((start->data)%2)!=0)

{

if ((temp\_odd != NULL) && (even\_1 != NULL))

{

temp\_odd->next = even\_1;

temp\_even->next = NULL;

last = temp\_even;

}

} //for connection of odd to even if 1st element is even

else

{

if ((temp\_even != NULL) && (odd\_1 != NULL))

{

temp\_even->next = odd\_1;

temp\_odd->next = NULL;

last = temp\_odd;

}

}

printLinkList();

}

**Question # 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.

void swapping\_values(int value1, int value2)

{

//ListNode\* ploc1;

ListNode\* loc1;

//ListNode\* ploc2;

ListNode\* loc2;

ListNode\* temp = start;

//Searches value 1 and 2 in list respectively

Searchvalue(value1);

//loc and ploc for values are stored in variables like ploc1,loc1 and same for value2

//ploc1 = ploc;

loc1 = loc;

//Returns if one of the value is not present in list

if (loc1 == NULL)

{

cout << value1 << "is not found";

return;

}

Searchvalue(value2);

//ploc2 = ploc;

loc2 = loc;

if (loc2 == NULL)

{

cout << value2 << "is not found";

return;

}

//if user enters same values to be swapped method retuens

if ((loc1->prev == NULL) && (loc2->prev == NULL))

{

cout << "both are same values not allowed in linklist";

return;

}

//if one is head bode and the other is not last

if ((loc1->prev == NULL) && (loc2 != last))

{

//if both value are not consecutive

if (loc1->next != loc2)

{

//swaps the pointers

temp = loc2->next;

loc2->next = loc1->next;

loc2->prev->next =loc1;

loc1->next = temp;

//start is assigned to the first of list after swapping

start = loc2;

// loc1 is pointed by previous of loc2

loc2->prev =NULL;

}

//if both value are consecutive

else

{ //temp stores next to value2

temp = loc2->next;

//start is assigned to the first of list after swapping

start = loc2;

//swaps the pointers

loc2->next = loc1;

loc1->next = temp;

}

}

//if none is head bode and none is last

if ((loc1->prev != NULL) && (loc2 != last))

{//if both value are not consecutive

if (loc1->next != loc2)

{

//swaps the pointers

temp = loc2->next;

loc1->prev->next = loc2;

loc2->next = loc1->next;

loc1->next->prev = loc2;

loc2->prev->next = loc1;

loc1->next = temp;

loc2->prev = loc1->prev;

loc1->prev = loc2->prev;

}

//if both value are consecutive

else

{//swaps the pointers

temp = loc2->next;

loc2->next = loc1;

loc1->next = temp;

loc1->prev->next = loc2;

}

printLinkList();

}

//if none is head bode and one is last

if ((loc1->prev != NULL) && (loc2->next == NULL))

{

//if both value are not consecutive

if (loc1->next != loc2)

{

loc1->prev->next = loc2;

loc2->next =loc1->next;

loc2->prev->next = loc1;

//for making value1 the last of list

loc1->next = NULL;

//updates last pointer

last = loc1;

loc2->prev = loc1->prev;

loc1->prev = loc2->prev;

}

//if both value are consecutive

else

{

loc1->prev->next = loc2;

loc2->next = loc1;

//for making value1 the last of list

loc1->next = NULL;

//updates last pointer

last = loc1;

loc1->prev = loc2;

loc2->prev = loc1->prev;

}

printLinkList();

}

}

};

**MAIN PROCEDURE**

int main(){

LinkList\* l = new LinkList();

l->isEmpty();

l->InsertAtFront(4);

l->InsertAtFront(3);

l->InsertAtFront(2);

l->InsertAtLast(5);

l->InsertAtLast(6);

l->InsertSorted(15);

l->printLinkList();

l->DeleteFrontNode();

l->DeleteLastNode();

l->printLinkList();

l->ReversingLinkList();

l->saperate\_even();

l->printLinkList();

l->swapping\_values(2,3);

l->printLinkList();

l->DestroyLinkList();

}

