Lab 3 **Linked Lists**

3.1. Introduction

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

3.1.1. Why Linked List?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.

2) Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.

For example, in a system, if we maintain a sorted list of IDs in an array id[].

```
id[] = [1000, 1010, 1050, 2000, 2040]
```

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

3.1.2. Advantages over arrays

1) Dynamic size

2) Ease of insertion/deletion

3.1.3. Drawbacks

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation. Read about it here.

2) Extra memory space for a pointer is required with each element of the list.

3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

3.2. Types of Linked List

There are three common types of Linked List.

Singly Linked List
Doubly Linked List
Circular Linked List

Singly Linked List

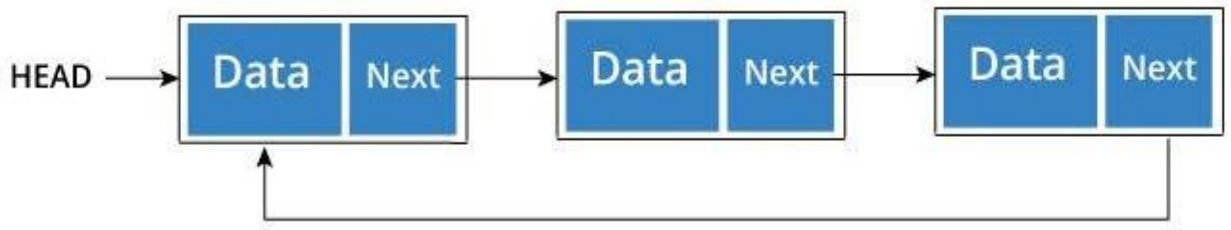
It is the most common. Each node has data and a pointer to the next node.

Doubly Linked List

We add a pointer to the previous node in a doubly linked list. Thus, we can go in either direction: forward or backward.

Circular Linked List

A circular linked list is a variation of linked list in which the last element is linked to the first element. This forms a circular loop.



A circular linked list can be either singly linked or doubly linked.

- for singly linked list, next pointer of last item points to the first item
- In doubly linked list, prev pointer of first item points to last item as well.

3.3. Single Linked List Operations

Now that you have got an understanding of the basic concepts behind linked list and their types, it's time to dive into the common operations that can be performed.

Two important points to remember:

- head points to the first node of the linked list
- next pointer of last node is NULL, so if next of current node is NULL, we have reached end of linked list.

In all of the examples, we will assume that the linked list has three nodes 1 --->2 --->3 with node structure as below:

```
struct node
{
  int data;
  struct node *next;
};
```

3.3.1. How to traverse a linked list

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

When temp is NULL, we know that we have reached the end of linked list so we get out of the while loop.

```
struct node *temp = head;
printf("\n\nList elements are - \n");
while(temp != NULL)
{
     printf("%d --->",temp->data);
     temp = temp->next;
}
```

The output of this program will be:

```
List elements are -
1 --->2 --->3 --->
```

3.3.2. How to add elements to linked list

You can add elements to either beginning, middle or end of linked list.

Add to beginning

- Allocate memory for new node
- Store data

- Change next of new node to point to head
- Change head to point to recently created node

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
newNode->next = head;
head = newNode;
```

Add to end

- Allocate memory for new node
- Store data
- Traverse to last node
- Change next of last node to recently created node

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
newNode->next = NULL;
struct node *temp = head;
while(temp->next != NULL){
  temp = temp->next;
}
temp->next = newNode;
```

Add to middle

- Allocate memory and store data for new node
- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
struct node *temp = head;
for(int i=2; i < position; i++) {
    if(temp->next != NULL) {
        temp = temp->next;
    }
}
newNode->next = temp->next;
temp->next = newNode;
```

3.3.3. How to delete from a linked list

You can delete either from beginning, end or from a position.

Delete from beginning

- Point head to the second node

```
head = head->next;
```

Delete from end

- Traverse to second last element
- Change its next pointer to null

```
struct node* temp = head;
while(temp->next->next!=NULL){
  temp = temp->next;
}
temp->next = NULL;
```

Delete from middle

- Traverse to element before the element to be deleted
- Change next pointers to exclude the node from the chain

```
for(int i=2; i< position; i++) {
    if(temp->next!=NULL) {
        temp = temp->next;
    }
}
temp->next = temp->next->next;
```

3.3.4. Complete program for linked list operations

Here is the complete program for all the linked list operations we learnt till now. Lots of edge cases have been left out to make the program short.

Example 3-1 Linked Lists Implementation

Code

`#include<iostream>`
`using namespace std;`
 	
`struct Node{`
`  int value;`
`  Node *next;`
 	
`  Node(){`
`  this->next = NULL;`
`  }`
`};`
 	
`class SingleLinkedList{`
`  private:`
`  Node *head;`
 	
`  public:`

```
            SingleLinkedList(){    //initially set the empty list
parameters
                  head = new Node;
            }


            void printList(){

                  cout<<endl<<"Linked List"<<endl<<endl;
                  cout<<"[head] -> ";
                  for(Node *tmp=head->next; tmp != NULL; tmp = tmp->next)
                  {
                        cout<<"["<<tmp->value<<"]"<<" -> ";
                  }
                  cout<<"null"<<endl<<endl;
            }


            void addStart(int v){

                  Node *tmp = new Node;   //create new node to be added
                  tmp->value = v;         //assign a value

                  tmp->next = head->next; //new node next pointer (tmp)
is now pointing to the same as head
                  head->next = tmp;      //head next pointer is updated
to a new node

            }

            void addEnd(int v){

                  Node *tmp = new Node;   //create new node
                  tmp->value = v;         //set the value

                  Node *n = new Node;
                  for(n = head; n->next != NULL ; n = n->next);
                  n->next = tmp;
                  tmp->next = NULL;
            }

            void addAfter(Node *prevNode, int val){

                  Node *tmp = new Node;
                  tmp->value =  val;

                  tmp->next=prevNode->next;
                  prevNode->next = tmp;

            }


            Node* find(int v, bool findPrev = false){

                  //pointer to the node that we wana find
                  Node *prevNode = NULL;
                  bool found = false;

                  for(prevNode = head; prevNode->next != NULL ; prevNode
= prevNode->next){
```

```
                        //if value is found
                        if(prevNode->next->value == v){
                              found = true;
                              break;
                        }

                  }

                  if(found){
                        if(findPrev)
                              return prevNode;
                        else
                              return prevNode->next;
                  }
                  else
                        return NULL;

            }

      bool remove(int value){

            Node *prevN;
            Node *tmp;

            //check for value existance
            if(find(value) != NULL){

                  //find previous node
                  prevN = find(value,true);

                  tmp = prevN->next;
                  prevN->next = prevN->next->next;
                  delete tmp;
                  return true;
            }
            else
            return false;
      }




};
int main(){
      SingleLinkedList list;
      int choice;
      int value;
      Node* node;
      bool again = true;
      while(again)
      {
            cout<<"========== Menu ==========="<<endl;
            cout<<"0. Exit"<<endl;
            cout<<"1. Display"<<endl;
            cout<<"2. addStart"<<endl;
            cout<<"3. addEnd"<<endl;
            cout<<"4. addAfter"<<endl;
            cout<<"5. Find Value"<<endl;
      cout<<"6. Remove Value"<<endl;
            cout<<"Enter choice: ";
```

```
            cin>>choice;

            switch(choice)
            {
                  case 0:  //exit loop
                        again = false;
                        break;
                  case 1: // display list
                        list.printList();
                        break;

                  case 2: // add start
                        cout<<"Enter the value that you want to add: ";
                        cin>>value;
                        list.addStart(value);
                        break;

                  case 3: // add end
                        cout<<"Enter the value that you want to add: ";
                        cin>>value;
                        list.addEnd(value);
                        break;

                  case 4: // add after


                        cout<<"Enter the value after which you want to
add: ";
                        cin>>value;
                        node = list.find(value); // find previous pointer

                        if(node == NULL)
                        {
                              cout<<"Value "<<value<<" not found!"<<endl;
                        }else{
                              cout<<"Enter the value that you want to
enter after "<<value<<": ";
                              cin>>value;
                              list.addAfter(node, value);
                        }
                        break;

                  case 5: // find value
                        cout<<"Enter the value that you want to find: ";
                        cin>>value;

                        if(list.find(value) != NULL)
                        {
                              cout<<"Value exists in the list"<<endl;
                        }else{
                              cout<<"Value does not exist in the
list"<<endl;
                        }
                        break;

                  case 6: // remove
                        cout<<"Enter the value that you want to remove:
";
                        cin>>value;

                        if(list.remove(value))
```

```
                        {
                              cout<<value<<" removed successfully"<<endl;
                        }else{
                              cout<<value<<" does not exist in the
list"<<endl;
                        }
                        break;
            }
            system("pause");
            system("CLS");
      }
}
```

Output

```
 ========== Menu ==========
 0. Exit
 1. Display
 2. addStart
 3. addEnd
 4. addAfter
 5. Find Value
 6. Remove Value
 Enter choice: 1

 Linked List

 [head] -> null
```

3.4. Example Problem

Example 3-2 Problem, Doubly Linked List Implementation

Problem Statement

Implement doubly linked list

Solution

```
#include<iostream>
using namespace std;

struct Node
{
      int value;
      Node *next, *previous;  //now we have 2 pointers

      Node()
      {
            this->next = NULL;
            this->previous = NULL;
      }
};

class doublyLinkedList
{
```

```
	private:
		Node *head;
		Node *tail;


	public:
		//initially set the empty list parameters
		doublyLinkedList()
		{
			head = new Node;
			tail = NULL;
		}

		void printList()
		{
			cout<<endl<<"Linked List"<<endl<<endl;
			cout<<"[head] <-> ";
			for(Node *tmp=head->next; tmp != NULL; tmp = tmp->next)
			{
				cout<<"["<<tmp->value<<"]"<<" <-> ";
			}
			cout<<"null"<<endl<<endl;
		}


		void addStart(int v)
		{
			Node *tmp = new Node;	//create new node to be added
			tmp->value    = v;		   //assign a value
			tmp->previous = head;	// pointing to head at
previously

			if(tail == NULL)
			{
				tail = tmp;
			}
			tmp->next = head->next; //new node next pointer (tmp)
is now pointing to the same as head
			head->next = tmp;	//head next pointer is updated
to a new node

		}

		void addEnd(int v)
		{
			Node *tmp = new Node;	//create new node
			tmp->value = v;		//set the value
			tmp->previous = tail;	//tail is now the 2nd last node
so previously pointing to tail

			if(tail == NULL)	//checking for the empty list
			{
				tail = tmp;		//new node is also the tail
in case of only one node in the list
				head->next = tmp;
				tmp->previous = head;	//new node is previously
is pointing towards head
			}

			else
			{
```

```
                        tail->next = tmp;  //tail next pointer is updated
now which was null
                        tail = tail->next;  //set the new tail to tmp
i.e. new node
                  }

            }
            void addAfter(Node *prevNode, int val)
            {
                  Node *tmp = new Node;
                  tmp->value =  val;

                  //check for the end node
                  if(prevNode->next == NULL)
                  {
                        prevNode->next = tmp;
                        tmp->next = NULL;
                        tail = tmp;
                  }

                  else
                  {
                        prevNode->next->previous = tmp;
                        tmp->next=prevNode->next;
                        prevNode->next = tmp;
                  }
                  tmp->previous = prevNode;
            }

            Node* find(int v){

                  //pointer to the node that we wana find
                  Node *tmp = NULL;
                  bool found = false;

                  for(tmp = head->next; tmp != NULL ; tmp = tmp->next){

                        //if value is found
                        if(tmp->value == v){
                              found = true;
                              break;
                        }
                  }

                  if(found)
                        return tmp;
                  else
                        return NULL;

            }




            bool remove(int value){

            Node *tmp ;   //tmp node for traversing through the list
            Node *delNode;  //node to be deleted
            delNode = find(value);   //find the pointer of the deleted
node
```

```
             //check for value existance
            if( delNode != NULL){

                  for(tmp = head; tmp != NULL; tmp = tmp->next){
//traversing through the list

                        //if we 've reached to the previous node
                        if(tmp == delNode->previous){

                              //if tail is gonna delete
                              if(delNode->next == NULL){
                                    tmp->next = NULL;
                                    tail = tmp;
                              }

                              else {
                                    tmp->next = delNode->next;
                                    delNode->next->previous = tmp;

                              }

                        delete delNode;
                        break;
                        return true;
                        }
                  }
            }
            else
            return false;
      }







            void reversePrint(){

                        Node *tmp = new Node;
                        tmp = tail;
                        while(tmp != NULL){
                              cout<<tmp->value<<" ";
                              tmp = tmp->previous;
                        }
                        cout<<endl;
                  }
};
int main(){
      doublyLinkedList list;
      int choice;
      int value;
      Node* node;
      bool again = true;
      while(again)
      {
            cout<<"========== Menu ==========="<<endl;
            cout<<"0. Exit"<<endl;
            cout<<"1. Display"<<endl;
            cout<<"2. addStart"<<endl;
            cout<<"3. addEnd"<<endl;
            cout<<"4. addAfter"<<endl;
```

```
            cout<<"5. Find Value"<<endl;
      cout<<"6. Remove Value"<<endl;
            cout<<"Enter choice: ";
            cin>>choice;

            switch(choice)
            {
                  case 0:  //exit loop
                        again = false;
                        break;
                  case 1: // display list
                        list.printList();
                        break;

                  case 2: // add start
                        cout<<"Enter the value that you want to add: ";
                        cin>>value;
                        list.addStart(value);
                        break;

                  case 3: // add end
                        cout<<"Enter the value that you want to add: ";
                        cin>>value;
                        list.addEnd(value);
                        break;

                  case 4: // add after


                        cout<<"Enter the value after which you want to
add: ";
                        cin>>value;
                        node = list.find(value); // find previous pointer

                        if(node == NULL)
                        {
                              cout<<"Value "<<value<<" not found!"<<endl;
                        }else{
                              cout<<"Enter the value that you want to
enter after "<<value<<": ";
                              cin>>value;
                              list.addAfter(node, value);
                        }
                        break;

                  case 5: // find value
                        cout<<"Enter the value that you want to find: ";
                        cin>>value;

                        if(list.find(value) != NULL)
                        {
                              cout<<"Value exists in the list"<<endl;
                        }else{
                              cout<<"Value does not exist in the
list"<<endl;
                        }
                        break;

                  case 6: // remove
                        cout<<"Enter the value that you want to remove:
";
```

```
                        cin>>value;

                        if(list.remove(value))
                        {
                              cout<<value<<" removed successfully"<<endl;
                        }else{
                              cout<<value<<" does not exist in the
list"<<endl;
                        }
                        break;
            }
            system("pause");
            system("CLS");
      }
}
```

Output

```
 ========== Menu ==========
 0. Exit
 1. Display
 2. addStart
 3. addEnd
 4. addAfter
 5. Find Value
 6. Remove Value
 Enter choice: 1

 Linked List

 [head] -> null
```