Lab Exercise 8:

Linked list

Q1. Write to insert at the end of the linked list.

```
/*****************************************************************
   //This program is developed by mayank(211b179)
  /*****************************************************************.
#include <iostream>
#include <stdlib.h>
using namespace std;

struct node {
   int num;
   struct node * preptr;
   struct node * nextptr;
}*stnode, *ennode;


void Listcreation(int n);
void LinsertNodeAtEnd(int num);
void displayList(int a);

int main()
{
   int n,num1,a;
   stnode = NULL;
   ennode = NULL;

   cout<<" Input the number of nodes : ";
   cin>>n;
   Listcreation(n);
   a=1;
   displayList(a);
   cout<<" Input data for the last node : ";
   cin>>num1;
   LinsertNodeAtEnd(num1);
   a=2;
   displayList(a);
   return 0;
}

void Listcreation(int n)
{
   int i, num;
   struct node *fnNode;
```

```
   if(n >= 1)
   {
      stnode = (struct node *)malloc(sizeof(struct node));

      if(stnode != NULL)
      {
         cout<<" Input data for node 1: ";
         cin>>num;

         stnode->num = num;
         stnode->preptr = NULL;
         stnode->nextptr = NULL;
         ennode = stnode;
         for(i=2; i<=n; i++)
         {
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode != NULL)
            {
               cout<<" Input data for node "<< i<<": ";
               cin>>num;
               fnNode->num = num;
               fnNode->preptr = ennode;
               fnNode->nextptr = NULL;
               ennode->nextptr = fnNode;
               ennode = fnNode;
            }
            else
            {
               cout<<" Memory can not be allocated.";
               break;
            }
         }
      }
      else
      {
         cout<<" Memory can not be allocated.";
      }
   }
}

void LinsertNodeAtEnd(int num)
{
   struct node * newnode;

   if(ennode == NULL)
   {
      cout<<" No data found in the list!\n";
   }
   else
```

```
    {
       newnode = (struct node *)malloc(sizeof(struct node));
       newnode->num = num;
       newnode->nextptr  =  NULL;
       newnode->preptr  =  ennode;
       ennode->nextptr = newnode;
       ennode = newnode;
    }
}

void displayList(int m)
{
   struct node * tmp;
   int n = 1;
   if(stnode == NULL)
   {
      cout<<" No data found in the List yet.";
   }
   else
   {
      tmp = stnode;
      if (m==1)
      {
      cout<<"\n Data entered in the list are :\n";
      }
      else
      {
       cout<<"\n After insertion the new list are :\n";
      }
      while(tmp != NULL)
      {
         cout<<" node" << n<<": "<< tmp->num<<endl;
         n++;
         tmp = tmp->nextptr;
      }
   }
}
```

Q2. Write a program to print the element of linked list.

```
    /*****************************************************************

    //This program is developed by mayank (211b179)
   /*****************************************************************
#include <stdio.h>
#include <stdlib.h>
struct node {
  int data;
```

```
  struct node *next;
};
struct node* intoList(int data) {
  struct node* newnode = (struct node*)malloc(sizeof(struct node));
  newnode->data = data;
  newnode->next = NULL;
  return newnode;
}
void displayList(struct node *catchead) {
  struct node *temp;
  if (catchead == NULL) {
    printf("List is empty.
");
    return;
  }
  printf("elements of list are : ");
  temp = catchead;
  while (temp != NULL) {
    printf("%d ", temp->data);
    temp = temp->next;
  }
  printf("
");
}
int search(int key,struct node *head) {
  int index;
  struct node *newnode;
  index = 0;
  newnode = head;
  while (newnode != NULL && newnode->data != key) {
    index++;
    newnode = newnode->next;
  }
  return (newnode != NULL) ? index : -1;
}
int main() {
  int index;
  struct node* head = intoList(9);
  head->next = intoList(76);
  head->next->next = intoList(13);
  head->next->next->next = intoList(24);
  head->next->next->next->next = intoList(55);
  head->next->next->next->next->next = intoList(109);
  displayList(head);
  index = search(24,head);
  if (index >= 0)
    printf("%d found at position %d", 24, index);
  else
    printf("%d not found in the list.", 24);
```

```
 index=search(55,head);
 if (index >= 0)
   printf("%d found at position %", 55, index);
 else
 printf("%d not found in the list.", 55);}
```

Q3. WAP to insert at the beginning of the linked list.

```
   /*****************************************************************
   //This program is developed by mayank (211b179)

   /*****************************************************************
#include<iostream>
using namespace std;
class Node
{
   public:
   int data;
   Node *next;
};
void insertFront(Node** head, int data){
   Node* new_node = new Node();
   new_node->data = data;
   new_node->next = *head;
   *head = new_node;
   cout << "Inserted Item: " << new_node->data << endl;
}
void printList(Node* node){
       cout << "\nLinked List : " ;
       while(node!=NULL){
       cout << node->data << " ";
       node = node->next;
   }
   cout << endl;
}
int main(){
   Node* head = NULL;
   insertFront(&head,4);
   insertFront(&head,5);
   insertFront(&head,6);
   insertFront(&head,7);
   insertFront(&head,8);
   insertFront(&head,9);
   printList(head);
  return 0;
}
```


Q4. WAP to insert a node at specify position in a linked list.

```
    /*****************************************************************
    //This program is developed by MAYANK (211b179)
    /*****************************************************************.
#include <bits/stdc++.h>
using namespace std;
struct Node {
        int data;
        struct Node* next;
};
int size = 0;
Node* getNode(int data)
{
        Node* newNode = new Node();
        newNode->data = data;
        newNode->next =  NULL;
        return newNode;
}
void insertPos(Node** current, int pos, int data)
{
        if (pos < 1 || pos > size + 1)
                cout << "Invalid position!" << endl;
        else {
                while (pos--) {
                if (pos == 0) {
                                Node* temp = getNode(data);
                                temp->next = *current;
                                *current = temp;
                        }
                        else
                        current = &(*current)->next;
                }
                size++;
        }
}
void printList(struct Node* head)
{
        while (head != NULL) {
                cout << " " << head->data;
                head = head->next;
        }
        cout << endl;
}
int main()
{
        Node* head = NULL;
        head = getNode(3);
        head->next = getNode(5);
        head->next->next = getNode(8);
        head->next->next->next = getNode(10);
```

```
        size = 4;
        cout << "Linked list before insertion: ";
        printList(head);
        int data = 12, pos = 3;
        insertPos(&head, pos, data);
        cout << "Linked list after insertion of 12 at position 3: ";
        printList(head);

        data = 1, pos = 1;
        insertPos(&head, pos, data);
        cout << "Linked list after insertion of 1 at position 1: ";
        printList(head);
        data = 15, pos = 7;
        insertPos(&head, pos, data);
        cout << "Linked list after insertion of 15 at position 7: ";
        printList(head);
        return 0;
}
```

Q5. WAP to delete a node from given position in a linked list.

```
    /*****************************************************************
     //This program is developed by MAYANK (211b179)
    /*****************************************************************.
#include<iostream>
using namespace std;
struct node
{
  int data;
  node * next;
};
 void insertNode(node **head,int data)
{
  node *temp = new node;
  temp->data = data;
  temp->next = *head;
  *head = temp;
void deleteNode(node **head,int position)
{
  node *temp = *head;
  node *prev;
  if(temp == NULL)
    return;
  if(position == 1)
  {
    *head = temp->next;
    free(temp);
    return;
  }
```

```
  for(int i=1 ; i!=position ; i++)
  {
     prev = temp;
     temp = temp->next;
  }
   if(temp == NULL)
  {
     cout<<"\nData not present\n";
     return;
  }
  else
  {
    prev->next = temp->next;
     free(temp);
  }
}
void display(node *head)
{
 while (head != NULL)
 {
    cout<<head->data<<"-> ";
    head = head->next;
 }
  cout<<"NULL";
}
int main()
{
 node* head = NULL;
 insertNode(&head, 72);
 insertNode(&head, 13);
 insertNode(&head, 59);
 insertNode(&head, 17);
 insertNode(&head, 33);
 insertNode(&head, 80);

 cout<<"Created Linked list is:\n";
 display(head);
 deleteNode(&head,3);
 cout<<"\n\nResultant Linked list is:\n";
 display(head);
 return 0;
}
```

Q6. WAP to print the elements in reverse order in a linked list.

```
   /*************************************************************
   //This program is developed by MAYANK (211b179)
```

```
   /*****************************************************************
#include <iostream>
struct node{
   int data;
   struct node* next;
   node(int x){
      data=x;
      next=NULL;
   }
};
void print_reverse(struct node* go){
   if(go==NULL){
      return;
   }
   print_reverse(go->next);
   std::cout<<go->data<<" "; }
int main() {
   struct node* head = new node(10);
   head->next = new node(20);
   head->next->next = new node(30);
   head->next->next->next = new node(40);
   head->next->next->next->next = new node(50);
   std::cout<<"Printing the linked list in reverse order:\n";
   print_reverse(head);
   return 0;
}
```

Q7. WAP to insert a node into a sorted doubly linked list.

```
    /*****************************************************************

    //This program is developed by MAYANK (211b179)
   /*****************************************************************
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node * prev;
   struct node * next;
}*head, *last;
void createList(int n);
void displayList();
void insertAtBeginning(int data);
void insertAtEnd(int data);
void insertAtN(int data, int position);
int main()
{
   int n, data, choice=1;
```

```
    head = NULL;
    last = NULL;
    while(choice != 0)
    {
        printf("1. Create List\n");
        printf("2. Insert node - at beginning\n");
        printf("3. Insert node - at end\n");
        printf("4. Insert node - at N\n");
        printf("5. Display list\n");
        printf("0. Exit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                printf("Enter the total number of nodes in list: ");
                scanf("%d", &n);
                createList(n);
                break;
            case 2:
                printf("Enter data of first node : ");
                scanf("%d", &data);
                insertAtBeginning(data);
                break;
            case 3:
                printf("Enter data of last node : ");
                scanf("%d", &data);
                insertAtEnd(data);
                break;
            case 4:
                printf("Enter the position where you want to insert new node: ");
                scanf("%d", &n);
                printf("Enter data of %d node : ", n);
                scanf("%d", &data);
                insertAtN(data, n);
                break;
            case 5:
                displayList();
                break;
            case 0:
                break;
            default:
                printf("Error! Invalid choice. Please choose between 0-5");
        }
    }

    return 0;
}
void createList(int n)
```

```
{
   int i, data;
   struct node *newNode;

   if(n >= 1)
   {
      head = (struct node *)malloc(sizeof(struct node));
      printf("Enter data of 1 node: ");
      scanf("%d", &data);
      head->data = data;
      head->prev = NULL;
      head->next = NULL;
      last = head;
      for(i=2; i<=n; i++)
      {
         newNode = (struct node *)malloc(sizeof(struct node));
         printf("Enter data of %d node: ", i);
         scanf("%d", &data);
         newNode->data = data;
         newNode->prev = last;
         newNode->next = NULL;
         last->next = newNode;
         last = newNode;
      }
   }
}
void displayList()
{
   struct node * temp;
   int n = 1;
   if(head == NULL)
   {
      printf("List is empty.\n");
   }
   else
   {
      temp = head;
      while(temp != NULL)
      {
         printf("DATA of %d node = %d\n", n, temp->data);
         n++;
         temp = temp->next;
      }
   }
}
void insertAtBeginning(int data)
{
   struct node * newNode;
   if(head == NULL)
```

```
    {
        printf("Error, List is Empty!\n");
    }
    else
    {
        newNode = (struct node *)malloc(sizeof(struct node));
        newNode->data = data;
        newNode->next = head;
        newNode->prev = NULL;
        head->prev = newNode;
        head = newNode;
    }
}
void insertAtEnd(int data)
{
    struct node * newNode;

    if(last == NULL)
    {
        printf("Error, List is empty!\n");
    }
    else
    {
        newNode = (struct node *)malloc(sizeof(struct node));
        newNode->data = data;
        newNode->next = NULL;
        newNode->prev = last;
        last->next = newNode;
        last = newNode;
    }
}
void insertAtN(int data, int position)
{
    int i;
    struct node * newNode, *temp;
    if(head == NULL)
    {
        printf("Error, List is empty!\n");
    }
    else
    {
        temp = head;
        i=1;
        while(i<position-1 && temp!=NULL)
        {
            temp = temp->next;
            i++;
        }
        if(position == 1)
```

```
      {
         insertAtBeginning(data);
      }
      else if(temp == last)
      {
         insertAtEnd(data);
      }
      else if(temp!=NULL)
      {
         newNode = (struct node *)malloc(sizeof(struct node));
         newNode->data = data;
         newNode->next = temp->next;
         newNode->prev = temp;
         if(temp->next != NULL)
         {
            temp->next->prev = newNode;
         }
         temp->next = newNode;

         printf("node inserted  at %d ", position);
      }
      else
      {
         printf("Error, Invalid position\n");
      }
   }
}
```

Q8. WAP to detect loop or cycle in a linked list.

```
 /*****************************************************************

    //This program is developed by MAYANK (211b179)
  /*****************************************************************
#include <bits/stdc++.h>
using namespace std;
struct Node {
        int data;
        struct Node* next;
};
void push(struct Node** head_ref, int new_data)
{
        struct Node* new_node = new Node;
        new_node->data = new_data;
        new_node->next = (*head_ref);
        (*head_ref) = new_node;
}
bool detectLoop(struct Node* h)
{
```

```
        unordered_set<Node*> s;
        while (h != NULL) {
                if (s.find(h) != s.end())
                return true;
                s.insert(h);
                h = h->next;
        }
        return false;
}
int main()
{
        struct Node* head = NULL;
        push(&head, 20);
        push(&head, 4);
        push(&head, 15);
        push(&head, 10);
        head->next->next->next->next = head;
        if (detectLoop(head))
                cout << "Loop Found";
        else
                cout << "No Loop";
        return 0;
}
```

Q9. WAP to create the doubly linked list of n nodes.

```
    /*****************************************************************

    //This program is developed by MAYANK (211b179)
   /*****************************************************************
#include <stdio.h>
struct node{
   int data;
   struct node *previous;
   struct node *next;
};
struct node *head, *tail = NULL;
void addNode(int data) {
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
   if(head == NULL) {
      head = tail = newNode;
      head->previous = NULL;
      tail->next = NULL;
   }
   else {
        tail->next = newNode;
```

```
        newNode->previous = tail;
           tail = newNode;
           tail->next = NULL;
    }
}
int countNodes() {
    int counter = 0;
    struct node *current = head;
        while(current != NULL) {
        counter++;
        current = current->next;
    }
    return counter;
}
void display() {
    struct node *current = head;
    if(head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Nodes of doubly linked list: \n");
    while(current != NULL) {
        //Prints each node by incrementing pointer.
        printf("%d ", current->data);
        current = current->next;
    }
}
int main()
{
    addNode(1);
    addNode(2);
    addNode(3);
    addNode(4);
    addNode(5);
    display();
    printf("\nCount of nodes present in the list: %d", countNodes());
    return 0;
}
```

Q10. Write a menu driven program for implementing doubly linked list. 1. To insert new node at beginning, 2. To insert new node after specified position 3. To insert new node at the end 4. To delete the node from beginning 5. To delete after specified position 6. To delete from the end.


```
/*************************************************************
```

```
    //This program is developed by MAYANK (211b179)
   /*************************************************************
#include <stdio.h>
#include <stdlib.h>
struct node {
        int info;
        struct node *prev, *next;
};
struct node* start = NULL;
void traverse()
{
        if (start == NULL) {
                printf("\nList is empty\n");
                return;
        }
        struct node* temp;
        temp = start;
        while (temp != NULL) {
                printf("Data = %d\n", temp->info);
                temp = temp->next;
        }
}

void insertAtFront()
{
        int data;
        struct node* temp;
        temp = (struct node*)malloc(sizeof(struct node));
        printf("\nEnter number to be inserted: ");
        scanf("%d", &data);
        temp->info = data;
        temp->prev = NULL;
        temp->next = start;
        start = temp;
}
void insertAtEnd()
{
        int data;
        struct node *temp, *trav;
        temp = (struct node*)malloc(sizeof(struct node));
        temp->prev = NULL;
        temp->next = NULL;
        printf("\nEnter number to be inserted: ");
        scanf("%d", &data);
        temp->info = data;
        temp->next = NULL;
        trav = start;
        if (start == NULL) {
                start = temp;
```

```
        }
        else {
                while (trav->next != NULL)
                        trav = trav->next;
                temp->prev = trav;
                trav->next = temp;
        }
}
void insertAtPosition()
{
        int data, pos, i = 1;
        struct node *temp, *newnode;
        newnode = malloc(sizeof(struct node));
        newnode->next = NULL;
        newnode->prev = NULL;
        printf("\nEnter position : ");
        scanf("%d", &pos);
        if (start == NULL) {
                start = newnode;
                newnode->prev = NULL;
                newnode->next = NULL;
        }

        else if (pos == 1) {
        insertAtFront();
        }
        else {
        printf("\nEnter number to be inserted: ");
        scanf("%d", &data);
        newnode->info = data;
        temp = start;
                while (i < pos - 1) {
                        temp = temp->next;
                        i++;
                }
                newnode->next = temp->next;
                newnode->prev = temp;
                temp->next = newnode;
                temp->next->prev = newnode;
        }
}
void deleteFirst()
{
        struct node* temp;
        if (start == NULL)
                printf("\nList is empty\n");
        else {
                temp = start;
                start = start->next;
```

```
                if (start != NULL)
                        start->prev = NULL;
                free(temp);
        }
}
void deleteEnd()
{
        struct node* temp;
        if (start == NULL)
                printf("\nList is empty\n");
        temp = start;
        while (temp->next != NULL)
                temp = temp->next;
        if (start->next == NULL)
                start = NULL;
        else {
                temp->prev->next = NULL;
                free(temp);
        }
}
void deletePosition()
{
        int pos, i = 1;
        struct node *temp, *position;
        temp = start;
        if (start == NULL)
                printf("\nList is empty\n");
        else {
                printf("\nEnter position : ");
                scanf("%d", &pos);
                if (pos == 1) {
                        deleteFirst();
                        if (start != NULL) {
                                start->prev = NULL;
                        }
                        free(position);
                        return;
                }
                while (i < pos - 1) {
                        temp = temp->next;
                        i++;
                }
                position = temp->next;
                if (position->next != NULL)
                position->next->prev = temp;
                temp->next = position->next;
                free(position);
        }
}
```

```
int main()
{
	int choice;
	while (1) {

		printf("\n\t1 To see list\n");
		printf("\t2 For insertion at"
			" starting\n");
		printf("\t3 For insertion at"
			" end\n");
		printf("\t4 For insertion at "
			"any position\n");
		printf("\t5 For deletion of "
			"first element\n");
		printf("\t6 For deletion of "
			"last element\n");
		printf("\t7 For deletion of "
			"element at any position\n");
		printf("\t8 To exit\n");
		printf("\nEnter Choice :\n");
		scanf("%d", &choice);
		switch (choice) {
		case 1:
			traverse();
			break;
		case 2:
			insertAtFront();
			break;
		case 3:
			insertAtEnd();
			break;
		case 4:
			insertAtPosition();
			break;
		case 5:
			deleteFirst();
			break;
		case 6:
			deleteEnd();
			break;
		case 7:
			deletePosition();
			break;

		case 8:
			exit(1);
			break;
		default:
			printf("Incorrect Choice. Try Again \n");
```

```
                        continue;
                    }
            }
            return 0;
}
```

Q11. WAP to create circular linked list of n nodes.

```
    /*****************************************************************

    //This program is developed by MAYANK (211b179)
    /*****************************************************************
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct node{
   int data;
   struct node *next;
};
 int count = 0;
struct node *head = NULL;
struct node *tail = NULL;
void add(int data){
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
   if(head == NULL){
      head = newNode;
      tail = newNode;
      newNode->next = head;
   }else {
      tail->next = newNode;
      tail = newNode;
      tail->next = head;
   }
}
void countNodes() {
   struct node *current = head;
   do{
      count++;
      current = current->next;
   }while(current != head);
   printf("Count of nodes present in circular linked list: %d",count);
}
int main()
{
   add(1);
   add(2);
   add(4);
```

```
   add(1);
   add(2);
   add(3);
  countNodes();
  return 0;
}
```

Q12. WAP to count the number of nodes in circular linked list if only start

pointer of circular linked list is given.

```
    /*************************************************************

    //This program is developed by MAYANK (211b179)
   /*************************************************************
#include <bits/stdc++.h>
using namespace std;
struct Node {
        int data;
        Node* next;
        Node(int x)
        {
                data = x;
                next = NULL;
        }
};
struct Node* push(struct Node* last, int data)
{
        if (last == NULL) {
                struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
                temp->data = data;
                last = temp;
                temp->next = last;
                return last;
        }
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = data;
        temp->next = last->next;
        last->next = temp;
        return last;
}
int countNodes(Node* head)
{
        Node* temp = head;
        int result = 0;
        if (head != NULL) {
                do {
                        temp = temp->next;
```

```
                result++;
            } while (temp != head);
        }

        return result;
}

int main()
{
        Node* head = NULL;
        head = push(head, 12);
        head = push(head, 56);
        head = push(head, 2);
        head = push(head, 11);
        cout << countNodes(head);
        return 0;
}
```

Lab Exercise 9:

Stack

Q1. Write a menu-driven program to implement stack using array with following options:

}


```
void Stack::push()
{
   if(isFull())
   {
      cout<<" OVERFLOW ";
   }
   else
```

```
    {
        int i;
        cout<<"\nEnter an element :: ";
        cin>>i;
        ++top;
        arr[top]=i;
        cout<<"\nInsertion successful.\n";
    }
}

void Stack::pop()
{
    int num;
    if(isEmpty())
    {
            cout<<"UNDERFLOW";
    }
    else
    {
    cout<<"\nDeleted item is : "<<arr[top]<<"\n";
    top--;
    }
}

void Stack::view()
{
    if(isEmpty())
    {
            cout<<" UNDERFLOW ";
    }
    else
    {
    for(int i=top;i>=0;i--)
    {
            cout<<arr[i]<<"\n";
    }
    }
}


int main()
{
    Stack s;
    int ch;
    ch=0;
    while(ch!=4)
    {
        cout<<"\n1. Push\n";
        cout<<"2. Pop\n";
```

```
        cout<<"3. Display\n";
        cout<<"4. Quit\n";
        cout<<"\nEnter your Choice :: ";
        cin>>ch;

        switch(ch)
        {
           case 1:
              s.push();
              break;

           case 2:
              s.pop();
              break;

           case 3:
              s.view();
              break;

           case 4:
              ch=4;
              cout<<"\nPress any key .. ";
              break;

           default:
              cout<<"\nWrong Choice!! \n";
              break;
        }
     }

    return 0;
}
```

Q2. Write a menu-driven program to implement stack using linked list with

following options: 1.Push 2.Pop 3.Display 4.Exit.

```
    /*****************************************************************

    //This program is developed by MAYANK (211b179)
   /*****************************************************************
#include <iostream>
using namespace std;
struct Node {
  int data;
  struct Node *next;
};
```

```
struct Node* top = NULL;
void push(int val) {
  struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
  newnode->data = val;
  newnode->next = top;
  top = newnode;
}
void pop() {
  if(top==NULL)
  cout<<"Stack Underflow"<<endl;
  else {
    cout<<"The popped element is "<< top->data <<endl;
    top = top->next;
  }
}
void display() {
  struct Node* ptr;
  if(top==NULL)
  cout<<"stack is empty";
  else {
    ptr = top;
    cout<<"Stack elements are: ";
    while (ptr != NULL) {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
    }
  }
  cout<<endl;
}
int main() {
  int ch, val;
  cout<<"1) Push in stack"<<endl;
  cout<<"2) Pop from stack"<<endl;
  cout<<"3) Display stack"<<endl;
  cout<<"4) Exit"<<endl;
  do {
    cout<<"Enter choice: "<<endl;
    cin>>ch;
    switch(ch) {
      case 1: {
        cout<<"Enter value to be pushed:"<<endl;
        cin>>val;
        push(val);
        break;
      }
      case 2: {
        pop();
        break;
      }
```

```
      case 3: {
        display();
        break;
      }
      case 4: {
        cout<<"Exit"<<endl;
        break;
      }
      default: {
        cout<<"Invalid Choice"<<endl;
      }
    }
  }while(ch!=4);
  return 0;
}
```

Q3. WAP to convert an expression from postfix to infix.

```
/*****************************************************************
//This program is developed by MAYANK (211b179)
/*****************************************************************
#include <bits/stdc++.h>
using namespace std;

bool isOperand(char x)
{
return (x >= 'a' && x <= 'z') ||(x >= 'A' && x <= 'Z');
}
string getInfix(string exp)
{
        stack<string> s;
        for (int i=0; exp[i]!='\0'; i++)
        {
                if (isOperand(exp[i]))
                {
                string op(1, exp[i]);
                s.push(op);
                }
                else
                {
                        string op1 = s.top();
                        s.pop();
                        string op2 = s.top();
                        s.pop();
                        s.push("(" + op2 + exp[i] +op1 + ")");
                }
        }
```

```
        return s.top();
  }
int main()
  {
        string exp = "ab*c+";
        cout << getInfix(exp);
        return 0;
}
```

Q4. WAP to convert an expression from infix to postfix.

```
    /*****************************************************************
    //This program is developed by MAYANK (211b179)
    /*****************************************************************.
struct stack
{
char elem[MAX];
int top;
};
struct stack stk;
void convert(char *infix, char *postfix);
int prcd(char op1, char op2);
void create();
void push(char op);
char pop(int *und);
int empty();
int full();
int isopnd(char ch);
int isoprtr(char ch);
int main()
{
char ch, infix[MAX], postfix[MAX];
create();
printf("Enter the infix expression\n");
scanf("%s", infix);
convert(infix, postfix);
printf("\n\nThe postfix expression is :\n");
printf("%s\n", postfix);
getch();
}
return(0);
}
void convert(char *infix, char *postfix)
{
int i, pos=0, over, und, n;
char ch, op;
for(i=0; (ch=infix[i]) != '\0' ; ++i)
{
if(isopnd(ch))
```

```
{
postfix[pos++] = ch;
}

else if(isoprtr(ch))
{
op = pop(&und);

while(!und && prcd(op, ch))
{
postfix[pos++] = op;
op = pop(&und);
}
if(!und)
push(op);
if(und || ch != ')')
push(ch);
else
pop(&und);
}
else
{
printf("\n\nThe infix expression is not valid\n");
getch();
return(0);
}
}
while(!empty())
{
postfix[pos++] = pop(&und);
}
postfix[pos++] = '\0';
}
int prcd(char op1, char op2)
{
if(op1 == '(' || op2 == '(')
return 0;
if(op2 == ')')
return 1;
if(op1 == '^')
{
if(op2 == '^')
return 0;
else
return 1;
}
if(op1 == '/' || op1 == '*')
{
if(op2 == '^')
```

```
return 0;
else
return 1;
}
else
{
if(op2 == '^' || op2 == '/' || op2 == '*')
return 0;
else
return 1;
}
}
void create()
{
stk.top = -1;
}
void push(char op)
{
stk.elem[++(stk.top)] = op;
}
char pop(int *und)
{
if(empty())
{
*und = 1;
return('0');
}
*und = 0;
return(stk.elem[stk.top--]);
}
int empty()
{
if(stk.top == -1)
return 1;
else
return 0;
}
int full()
{
if(stk.top == MAX - 1)
return 1;
else
return 0;
}
int isopnd(char ch)
{
if((ch>=48 && ch<58) || (ch>64 && ch<=90) || (ch>96 && ch<=122))
return 1;
else
```

```
return 0;
}
int isoprtr(char ch)
{
if(ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^' || ch == '(' || ch == ')')
return 1;
else
return 0;
}
```

Q5. WAP to convert an expression from infix to prefix.

```
    /*****************************************************************
     //This program is developed by MAYANK (211b179)
    /*****************************************************************.
#include <bits/stdc++.h>
using namespace std;
bool isOperator(char c)
{
        return (!isalpha(c) && !isdigit(c));
}
int getPriority(char C)
{
        if (C == '-' || C == '+')
                return 1;
        else if (C == '*' || C == '/')
                return 2;
        else if (C == '^')
                return 3;
        return 0;
}

string infixToPostfix(string infix)
{
        infix = '(' + infix + ')';
        int l = infix.size();
        stack<char> char_stack;
        string output;
        for (int i = 0; i < l; i++) {
                if (isalpha(infix[i]) || isdigit(infix[i]))
                        output += infix[i];
                else if (infix[i] == '(')
                        char_stack.push('(');
                else if (infix[i] == ')') {
                        while (char_stack.top() != '(') {
                                output += char_stack.top();
                                char_stack.pop();
                        }
                        char_stack.pop();
```

```
		}
		else
		{
			if (isOperator(char_stack.top()))
			{
				if(infix[i] == '^')
				{
					while (getPriority(infix[i]) <= getPriority(char_stack.top()))
					{
						output += char_stack.top();
						char_stack.pop();
					}

				}
				else
				{
					while (getPriority(infix[i]) < getPriority(char_stack.top()))
					{
						output += char_stack.top();
						char_stack.pop();
					}

				}

				char_stack.push(infix[i]);
			}
		}
	}
	while(!char_stack.empty()){
		output += char_stack.top();
		char_stack.pop();
	}
	return output;
}

string infixToPrefix(string infix)
{
		int l = infix.size();
		reverse(infix.begin(), infix.end());
	for (int i = 0; i < l; i++) {
		if (infix[i] == '(') {
			infix[i] = ')';
		}
		else if (infix[i] == ')') {
			infix[i] = '(';
		}
	}
	string prefix = infixToPostfix(infix);
	reverse(prefix.begin(), prefix.end());
```

```
        return prefix;
}
int main()
{
        string s = ("x+y*z/w+u");
        cout << infixToPrefix(s) << std::endl;
        return 0;
}
```

Q6. WAP to evaluate postfix expression.

```
 /*****************************************************************

    //This program is developed by MAYANK (211b179)
  /*****************************************************************
#include<bits/stdc++.h>
using namespace std;
float scanNum(char ch){
  int value;
  value = ch;
  return float(value-'0');
}
int isOperator(char ch){
  if(ch == '+'|| ch == '-'|| ch == '*'|| ch == '/' || ch == '^')
    return 1;
    return -1;  }
  int isOperand(char ch){
    if(ch >= '0' && ch <= '9')
      return 1;
    return -1;
  }
  float operation(int a, int b, char op){
   if(op == '+')
      return b+a;
    else if(op == '-')
      return b-a;
    else if(op == '*')
      return b*a;
    else if(op == '/')
      return b/a;
    else if(op == '^')
      return pow(b,a); //find b^a
    else
  return INT_MIN;
}
float postfixEval(string postfix){
  int a, b;
  stack<float> stk;
  string::iterator it;
```

```
  for(it=postfix.begin(); it!=postfix.end(); it++){
      if(isOperator(*it) != -1){
       a = stk.top();
       stk.pop();
       b = stk.top();
       stk.pop();
       stk.push(operation(a, b, *it));
     }else if(isOperand(*it) > 0){
       stk.push(scanNum(*it));
     }
  }
  return stk.top();
}
main(){
  string post = "21+3*";
  cout <<postfixEval(post);
}
```

Q7. WAP to implement tower of Hanoi puzzle.

```
 /*****************************************************************

    //This program is developed by MAYANK (211b179)
  /*****************************************************************
#include <bits/stdc++.h>
using namespace std;
void towerOfHanoi(int n, char from_rod, char to_rod,char aux_rod)
{
        if (n == 0) {
                return;
        }
        towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
        cout << "Move disk " << n << " from rod " << from_rod << " to rod " << to_rod << endl;
        towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
int main()
{
        int N = 3;
        towerOfHanoi(N, 'A', 'C', 'B');
        return 0;
}
```

Lab Exercise 10:

Queue

Q1. Write a menu driven program to implement linear queue using array and switch-case with following options : 1.Insert 2.Delete 3.Display element at the front 4.Display all elements of the queue 5.Quit.

```
/*****************************************************************
  //This program is developed by MAYANK (211b179)
  /*****************************************************************.
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

int queue_arr[MAX];
int rear=-1;
int front=-1;

void insert(int item);
int del();
int peek();
void display();
int isFull();
int isEmpty();

int main()
{
    int choice,item;
    while(1)
    {
        printf("\n1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display element at the front\n");
        printf("4.Display all elements of the queue\n");
        printf("5.Quit\n");
        printf("\nEnter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
        case 1:
            printf("\nInput the element for adding in queue : ");
            scanf("%d",&item);
            insert(item);
            break;
        case 2:
            item=del();
```

```
                printf("\nDeleted element is %d\n",item);
                break;
            case 3:
                printf("\nElement at the front is %d\n",peek());
                break;
            case 4:
                display();
                break;
            case 5:
                exit(1);
            default:
                printf("\nWrong choice\n");
            }
        }

        return 0;
}
void insert(int item)
{
        if( isFull() )
        {
            printf("\nQueue Overflow\n");
            return;
        }
        if( front == -1 )
            front=0;
        rear=rear+1;
        queue_arr[rear]=item ;
}
int del()
{
        int item;
        if( isEmpty() )
        {
            printf("\nQueue Underflow\n");
            exit(1);
        }
        item=queue_arr[front];
        front=front+1;
        return item;
}
int peek()
{
        if( isEmpty() )
        {
            printf("\nQueue Underflow\n");
            exit(1);
        }
        return queue_arr[front];
```

```
}
int isEmpty()
{
	if( front==-1 || front==rear+1 )
		return 1;
	else
		return 0;
}
int isFull()
{
	if( rear==MAX-1 )
		return 1;
	else
		return 0;
}
void display()
{
	int i;
	if ( isEmpty() )
	{
		printf("\nQueue is empty\n");
		return;
	}
	printf("\nQueue is :\n\n");
	for(i=front;i<=rear;i++)
		printf("%d ",queue_arr[i]);
	printf("\n\n");
}
```

Q2. Write a menu driven program to implement circular queue using array and switch-case with following options : 1.Insert 2.Delete 3.Display element at the front 4.Display all elements of the queue 5.Quit.

```
/*************************************************************

//This program is developed by MAYANK (211b179)
/*************************************************************
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int cqueue_arr[MAX];
int front=-1;
int rear=-1;
void display( );
void insert(int item);
int del();
```

```
int peek();
int isEmpty();
int isFull();
int main()
{
	int choice,item;
	while(1)
	{
		printf("\n1.Insert\n");
		printf("2.Delete\n");
		printf("3.Peek\n");
		printf("4.Display\n");
		printf("5.Quit\n");
		printf("\nEnter your choice : ");
		scanf("%d",&choice);
		switch(choice)
		{
		case 1 :
			printf("\nInput the element for insertion : ");
			scanf("%d",&item);
			insert(item);
			break;
		case 2 :
			printf("\nElement deleted is : %d\n",del());
			break;
		case 3:
			printf("\nElement at the front is  : %d\n",peek());
			break;
		case 4:
			display();
			break;
		case 5:
			exit(1);
		default:
			printf("\nWrong choice\n");
		}
	}
	return 0;
}
void insert(int item)
{
	if( isFull() )
	{
		printf("\nQueue Overflow\n");
		return;
	}
	if(front == -1 )
		front=0;
```

```
        if(rear==MAX-1)
                rear=0;
        else
                rear=rear+1;
        cqueue_arr[rear]=item ;
}
int del()
{
        int item;
        if( isEmpty() )
        {
                printf("\nQueue Underflow\n");
                exit(1);
        }
        item=cqueue_arr[front];
        if(front==rear)
        {
                front=-1;
                rear=-1;
        }
        else if(front==MAX-1)
                front=0;
        else
                front=front+1;
        return item;
}
int isEmpty()
{
       if(front==-1)
                return 1;
        else
                return 0;
}
int isFull()
{
        if((front==0 && rear==MAX-1) || (front==rear+1))
                return 1;
        else
                return 0;
}
int peek()
{
        if( isEmpty() )
        {
                printf("\nQueue Underflow\n");
                exit(1);
        }
        return cqueue_arr[front];
}
```

```
void display()
{
	int i;
	if(isEmpty())
	{
		printf("\nQueue is empty\n");
		return;
	}
	printf("\nQueue elements :\n");
	i=front;
	if( front<=rear )
	{
		while(i<=rear)
			printf("%d ",cqueue_arr[i++]);
	}
	else
	{
		while(i<=MAX-1)
			printf("%d ",cqueue_arr[i++]);
		i=0;
		while(i<=rear)
			printf("%d ",cqueue_arr[i++]);
	}
	printf("\n");
```

Q3. Write a menu driven program to implement linear queue using linked list and

switch-case with following options : 1.Insert 2.Delete 3.Display element at the front

4.Display all elements of the queue 5.Quit.

```
/*****************************************************************
//This program is developed by MAYANK (211b179)
/*****************************************************************
#include <stdio.h>
#include <stdlib.h>
 struct node
{
   int info;
   struct node *ptr;
}*front,*rear,*temp,*front1;
 int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
```

```
void queuesize();
 int count = 0;
 void main()
{
    int no, ch, e;
     printf("\n 1 - Enque");
    printf("\n 2 - Deque");
    printf("\n 3 - Front element");
    printf("\n 4 - Empty");
    printf("\n 5  -  Exit");
    printf("\n 6 - Display");
    printf("\n 7 - Queue size");
    create();
    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            enq(no);
            break;
        case 2:
            deq();
            break;
        case 3:
            e = frontelement();
            if (e != 0)
                printf("Front element : %d", e);
            else
                printf("\n No front element in Queue as queue is empty");
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            queuesize();
            break;
        default:
            printf("Wrong choice, Please enter correct choice ");
```

```
            break;
        }
    }
}
void create()
{
    front = rear = NULL;
}
void queuesize()
{
    printf("\n Queue size : %d", count);
}
void enq(int data)
{
    if (rear == NULL)
    {
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear->ptr = NULL;
        rear->info = data;
        front = rear;
    }
    else
    {
        temp=(struct node *)malloc(1*sizeof(struct node));
        rear->ptr = temp;
        temp->info = data;
        temp->ptr = NULL;
        rear = temp;
    }
    count++;
}
void display()
{
    front1 = front;
   if ((front1 == NULL) && (rear == NULL))
    {
        printf("Queue is empty");
        return;
    }
    while (front1 != rear)
    {
        printf("%d ", front1->info);
        front1 = front1->ptr;
    }
    if (front1 == rear)
        printf("%d", front1->info);
}
```

```
void deq()
{
   front1 = front;
   if (front1 == NULL)
   {
      printf("\n Error: Trying to display elements from empty queue");
      return;
   }
   else
      if (front1->ptr != NULL)
      {
         front1 = front1->ptr;
         printf("\n Dequed value : %d", front->info);
         free(front);
         front = front1;
      }
      else
      {
         printf("\n Dequed value : %d", front->info);
         free(front);
         front = NULL;
         rear = NULL;
      }
      count--;
}
int frontelement()
{
   if ((front != NULL) && (rear != NULL))
      return(front->info);
   else
      return 0;
}
void empty()
{
    if ((front == NULL) && (rear == NULL))
      printf("\n Queue empty");
   else
     printf("Queue not empty");
}
```

Q4. WAP to implement priority queue with its basic operations.

```
/*************************************************************
//This program is developed by MAYANK (211b179)
/*************************************************************
```



```
#include <bits/stdc++.h>
```

```
using namespace std;
typedef struct node {
	int  data;
	int priority;
	struct node* next;
} Node;
Node* newNode(int d, int p)
{
	Node* temp = (Node*)malloc(sizeof(Node));
	temp->data = d;
	temp->priority = p;
	temp->next = NULL;
	return temp;
}
int peek(Node** head) { return (*head)->data; }
void pop(Node** head)
{
	Node* temp = *head;
	(*head) = (*head)->next;
	free(temp);
}
void push(Node** head, int d, int p)
{
	Node* start = (*head);
	Node* temp = newNode(d, p);
	if ((*head)->priority < p) {
		temp->next = *head;
		(*head) = temp;
	}
	else {
		while (start->next != NULL
			&& start->next->priority > p) {
			start = start->next;
		}
		temp->next = start->next;
		start->next = temp;
	}
}
int isEmpty(Node** head) { return (*head) == NULL; }
int main()
{
	Node* pq = newNode(4, 1);
	push(&pq, 5, 2);
	push(&pq, 6, 3);
	push(&pq, 7, 0);
	while (!isEmpty(&pq)) {
		cout << " " << peek(&pq);
		pop(&pq);
	}	return 0;    }
```

Lab Exercise 11:

Trees

1. WAP to check whether given tree is a binary search tree or not.

```
  /*************************************************************
    //This program is developed by MAYANK (211b179)
   /************************************************************* .
#include<bits/stdc++.h>
#include<stdbool.h>
using namespace std;
class Node
{
        public:
        int key;
        Node *left, *right;
        Node *newNode(char k)
        {
                Node *node = ( Node*)malloc(sizeof( Node));
                node->key = k;
                node->right = node->left = NULL;
                return node;
        }
};
unsigned int countNodes(Node* root)
{
        if (root == NULL)
                return (0);
        return (1 + countNodes(root->left) +  countNodes(root->right));
}
bool isComplete ( Node* root, unsigned int index,unsigned int number_nodes)
{
        if (root == NULL)
                return (true);
        if (index >= number_nodes)
                return (false);
        return (isComplete(root->left, 2*index + 1, number_nodes) && isComplete(root->right, 2*index + 2,
number_nodes));
}
int main()
{
        Node n1;
        Node* root = NULL;
        root = n1.newNode(1);
        root->left = n1.newNode(2);
        root->right = n1.newNode(3);
        root->left->left = n1.newNode(4);
```

```
        root->left->right = n1.newNode(5);
        root->right->right = n1.newNode(6);
        unsigned int node_count = countNodes(root);
        unsigned int index = 0;
        if (isComplete(root, index, node_count))
                cout << "The Binary Tree is complete\n";
        else
                cout << "The Binary Tree is not complete\n";
        return (0);
}
```

Q2. WAP to implement inorder, preorder and postorder traversal in binary tree.

```
    /*****************************************************************

    //This program is developed by MAYANK (211b179)
   /*****************************************************************
#include <iostream>
using namespace std;
struct Node {
  int data;
  struct Node *left, *right;
  Node(int data) {
    this->data = data;
    left = right = NULL;
  }
};
void preorderTraversal(struct Node* node) {
  if (node == NULL)
    return;
  cout << node->data << "->";
  preorderTraversal(node->left);
  preorderTraversal(node->right);
}
void postorderTraversal(struct Node* node) {
  if (node == NULL)
    return;
  postorderTraversal(node->left);
  postorderTraversal(node->right);
  cout << node->data << "->";
}
void inorderTraversal(struct Node* node) {
  if (node == NULL)
    return;
  inorderTraversal(node->left);
  cout << node->data << "->";
```

```
 inorderTraversal(node->right);
}
int main() {
 struct Node* root = new Node(1);
 root->left = new Node(12);
 root->right = new Node(9);
 root->left->left = new Node(5);
 root->left->right = new Node(6);
 cout << "Inorder traversal ";
 inorderTraversal(root);
 cout << "\nPreorder traversal ";
 preorderTraversal(root);
 cout << "\nPostorder traversal ";
 postorderTraversal(root);
```

Q3. WAP to search a node in a given binary search tree.

```
/*****************************************************************
//This program is developed by MAYANK (211b179)
/*****************************************************************
#include <iostream>
using namespace std;
struct Node {
        int data;
        struct Node *left, *right;
        Node(int data)
        {
                this->data = data;
                left = right = NULL;
        }
};
bool ifNodeExists(struct Node* node, int key)
{
        if (node == NULL)
                return false;
        if (node->data == key)
                return true;
        bool res1 = ifNodeExists(node->left, key);
        if(res1) return true;
        bool res2 = ifNodeExists(node->right, key);
        return res2;
}
int main()
{
        struct Node* root = new Node(0);
        root->left = new Node(1);
        root->left->left = new Node(3);
```

```
	root->left->left->left = new Node(7);
	root->left->right = new Node(4);
	root->left->right->left = new Node(8);
	root->left->right->right = new Node(9);
	root->right = new Node(2);
	root->right->left = new Node(5);
	root->right->right = new Node(6);
	int key = 4;
	if (ifNodeExists(root, key))
		cout << "YES";
	else
		cout << "NO";

	return 0;
}
```

Q4. WAP to insert a node in a given binary search tree.

```
    /*************************************************************
    //This program is developed by MAYANK (211b179)
    /************************************************************* .
#include <bits/stdc++.h>
using namespace std;
struct Node {
	int key;
	struct Node *left, *right;
};
Node* newNode(int data)
{
	Node* temp = new Node;
	temp->key = data;
	temp->left = NULL;
	temp->right = NULL;
	return temp;
}
Node* insert(Node* root, int key)
{
	Node* newnode = newNode(key);
	Node* x = root;
	Node* y = NULL;
	while (x != NULL) {
		y = x;
		if (key < x->key)
			x = x->left;
		else
			x = x->right;
	}
	if (y == NULL)
		y = newnode;
```

```
        else if (key < y->key)
                y->left = newnode;
        else
                y->right = newnode;
        return y;
}
void Inorder(Node* root)
{
        if (root == NULL)
                return;
        else {
                Inorder(root->left);
                cout << root->key << " ";
                Inorder(root->right);
        }
}
int main()
{
        Node* root = NULL;
        root = insert(root, 50);
        insert(root, 30);
        insert(root, 20);
        insert(root, 40);
        insert(root, 70);
        insert(root, 60);
        insert(root, 80);
        Inorder(root);
        return 0;
}
```

Q5. WAP to delete a node from a given binary search tree.

```
    /*************************************************************
     //This program is developed by MAYANK (211b179)
    /************************************************************* .
#include <bits/stdc++.h>
using namespace std;
struct node {
        int key;
        struct node *left, *right;
};
struct node* newNode(int item)
{
        struct node* temp= (struct node*)malloc(sizeof(struct node));
        temp->key = item;
        temp->left = temp->right = NULL;
        return temp;
}
void inorder(struct node* root)
```

```
{
	if (root != NULL) {
		inorder(root->left);
		cout << root->key;
		inorder(root->right);
	}
}
struct node* insert(struct node* node, int key)
{
	if (node == NULL)
		return newNode(key);
	if (key < node->key)
		node->left = insert(node->left, key);
	else
		node->right = insert(node->right, key);
		return node;
}
struct node* minValueNode(struct node* node)
{
	struct node* current = node;
	while (current && current->left != NULL)
		current = current->left;
	return current;
}
struct node* deleteNode(struct node* root, int key)
{
	if (root == NULL)
		return root;
	if (key < root->key)
		root->left = deleteNode(root->left, key);
	else if (key > root->key)
		root->right = deleteNode(root->right, key);
	else {
			if (root->left==NULL and root->right==NULL)
			return NULL;
			else if (root->left == NULL) {
			struct node* temp = root->right;
			free(root);
			return temp;
		}
		else if (root->right == NULL) {
			struct node* temp = root->left;
			free(root);
			return temp;
		}
			struct node* temp = minValueNode(root->right);
		root->key = temp->key;
		root->right = deleteNode(root->right, temp->key);
	}
```

```
        return root;
}
int main()
{
        struct node* root = NULL;
        root =  insert(root,  50);
        root =  insert(root,  30);
        root =  insert(root,  20);
        root =  insert(root,  40);
        root =  insert(root,  70);
        root =  insert(root,  60);
        root = insert(root, 80);
        cout << "Inorder traversal of the given tree \n";
        inorder(root);
        cout << "\nDelete 20\n";
        root = deleteNode(root, 20);
        cout << "Inorder traversal of the modified tree \n";
        inorder(root);
        cout << "\nDelete 30\n";
        root = deleteNode(root, 30);
        cout << "Inorder traversal of the modified tree \n";
        inorder(root);
        cout << "\nDelete 50\n";
        root = deleteNode(root, 50);
        cout << "Inorder traversal of the modified tree \n";
        inorder(root);
        return 0;
}
```

Q6. Write the programs for following:

a. Determining the height of binary tree

```
    /************************************************************

    //This program is developed by MAYANK (211b179)
   /************************************************************
#include <stdio.h>
#include <stdlib.h>
typedef struct Node Node;
struct Node {
   int value;
     Node* left, *right;
};
Node* init_tree(int data) {
     Node* root = (Node*) malloc (sizeof(Node));
   root->left = root->right = NULL;
   root->value = data;
   return root;
```

```
}
Node* create_node(int data) {
   Node* node = (Node*) malloc (sizeof(Node));
   node->value = data;
   node->left = node->right = NULL;
   return node;
}
void free_tree(Node* root) {
     Node* temp = root;
   if (!temp)
      return;
   free_tree(temp->left);
   free_tree(temp->right);
   if (!temp->left && !temp->right) {
      free(temp);
      return;
   }
}
int tree_height(Node* root) {
   if (!root)
      return 0;
   else {
      int left_height  = tree_height(root->left);
      int right_height = tree_height(root->right);
      if (left_height >= right_height)
         return left_height + 1;
      else
         return right_height + 1;
   }
}

int main() {
   Node* root = init_tree(10);
   root->left = create_node(20);
   root->right = create_node(30);
   root->left->left = create_node(40);
   root->left->right = create_node(50);
   int height = tree_height(root);
   printf("Height of the Binary Tree: %d\n", height);
   free_tree(root);
   return 0;
}
```

b. Determining no. of nodes of binary tree

```
/*************************************************************
```

```
    //This program is developed by MAYANK (211b179)
  /*************************************************************
#include <bits/stdc++.h>
using namespace std;
class node {
public:
        int data;
        node* left;
        node* right;
};
int totalNodes(node* root)
{
        if (root == NULL)
                return 0;

        int l = totalNodes(root->left);
        int r = totalNodes(root->right);

        return 1 + l + r;
}
node* newNode(int data)
{
        node* Node = new node();
        Node->data = data;
        Node->left = NULL;
        Node->right = NULL;
        return (Node);
}
int main()
{
        node* root = newNode(1);
        root->left = newNode(2);
        root->right = newNode(3);
        root->left->left = newNode(4);
        root->left->right = newNode(5);
        root->right->left = newNode(9);
        root->right->right = newNode(8);
        root->left->left->left = newNode(6);
        root->left->left->right = newNode(7);
        cout << totalNodes(root);
        return 0;
}
```

c. Determining no. of internal nodes of binary tree.

```
 /*************************************************************
```



```
    //This program is developed by MAYANK (211b179)
  /*************************************************************
```

```
#include  <iostream>
using namespace std;
struct node
{
   int info;
   struct node *left, *right;
};
int count = 0;
class BST
{
   public:
         struct node *createnode(int key)
      {
         struct node *newnode = new node;
         newnode->info = key;
         newnode->left = NULL;
         newnode->right = NULL;
         return(newnode);
      }
      int internalnodes(struct node *newnode)
      {
         if(newnode != NULL)
         {
            internalnodes(newnode->left);
            if((newnode->left != NULL) || (newnode->right != NULL))
            {
               count++;
            }
            internalnodes(newnode->right);
         }
         return count;
      }
};
int main()
{
   BST t1,t2,t3;
   struct node *newnode = t1.createnode(25);
   newnode->left = t1.createnode(19);
   newnode->right = t1.createnode(29);
   newnode->left->left = t1.createnode(17);
   newnode->left->right = t1.createnode(20);
   newnode->right->left = t1.createnode(27);
   newnode->right->right = t1.createnode(55);
   cout<<"Number of internal nodes in first Tree are "<<t1.internalnodes(newnode);
   cout<<endl;
   count = 0;
   struct node *node = t2.createnode(1);
   node->right = t2.createnode(2);
   node->right->right = t2.createnode(3);
```

```
    node->right->right->right = t2.createnode(4);
    node->right->right->right->right = t2.createnode(5);
    cout<<"\nNumber of internal nodes in second tree are "<<t2.internalnodes(node);
    cout<<endl;
    count = 0;
    struct node *root = t3.createnode(15);
    cout<<"\nNumber of internal nodes in third tree are "<<t3.internalnodes(root);
    return 0;
}
```

e. Determining mirror image of binary tree.

```
    /*************************************************************

    //This program is developed by MAYANK (211b179)
   /*************************************************************
#include <bits/stdc++.h>
using namespace std;
struct Node {
        int data;
        struct Node* left;
        struct Node* right;
};
struct Node* newNode(int data)
{
        struct Node* node
                = (struct Node*)malloc(sizeof(struct Node));
        node->data = data;
        node->left = NULL;
        node->right = NULL;
        return (node);
}
void mirror(struct Node* node)
{
        if (node == NULL)
                return;
        else {
                struct Node* temp;
                mirror(node->left);
                mirror(node->right);
                temp = node->left;
                node->left = node->right;
                node->right = temp;
        }
}
void inOrder(struct Node* node)
{
        if (node == NULL)
```

```
                return;
        inOrder(node->left);
        cout << node->data << " ";inOrder(node->right);
}
int main()
{


    struct Node* root = newNode(1);root-
    >left = newNode(2);
    root->right = newNode(3); root->left-
    >left = newNode(4);
    root->left->right = newNode(5);
    cout << "Inorder traversal of the constructed"<< " tree is" << endl;inOrder(root);
    mirror(root);
cout << "\nInorder traversal of the mirror tree"inOrder(root);return 0;




}
```