Table of Contents

Lab Exercise 9: Stack

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

1.Push 2.Pop 3.Display 4.Exit.

```
/************************************************************
//This program is developed by Hrishav Sharma (211B138)
/*******************************************************
#include<stdio.h>
#include<iostream>

using namespace std;

class Stack
{
int top;
int arr[50];
public:
   Stack()
{
      top=-1;
}

   void push();
   void pop();
   void view();
   int isEmpty();
   int isFull();
};

int Stack::isEmpty()
{
   return (top==(-1)?1:0);
}

int Stack::isFull()
{
   return ( top == 50 ? 1 : 0 );
}

void Stack::push()
{
   if(isFull())
   {
      cout<<"\nSTACK IS FULL { OVERFLOW }";
```

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

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

void Stack::view()
{
    if(isEmpty())
    {
            cout<<"\n STACK IS EMPTY [ UNDERFLOW ] ";
    }
    else
    {
    cout<<"\nSTACK :\n";
    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 Hrishav Sharma (211B138)

/***********************************************************************************
****
#include <iostream>
#include<malloc.h>
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 Hrishav Sharma (211B138)
/********************************************************************************
#include <iostream>
using namespace std;
bool isOperand(char x) {
  return (x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z');
}
string infixConversion(string postfix) {
  stack<string> infix;
  for (int i=0; postfix[i]!='\0'; i++) {
    if (isOperand(postfix[i])) {
      string op(1, postfix[i]);
      infix.push(op);
    } else {
      string op1 = infix.top();
      infix.pop();
      string op2 = infix.top();
      infix.pop();
      infix.push("{"+op2+postfix[i]+op1 +"}");
    }
  }
  return infix.top();
}
int main() {
  string postfix = "xyae+/%";
  cout<<"The infix conversion of the postfix expression '"<<postfix<<"' is : ";
  cout<<infixConversion(postfix);
  return 0;
}
```

```
************************************************************************************
```

Q4. WAP to implement tower of Hanoi puzzle

```
/**********************************************************************************
****
//This program is developed by Hrishav Sharma (211B138)
/**********************************************************************************
****
#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 ;
       cin>>N;

        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 created by Hrishav Sharma(211B138)

/***

```
#include <iostream>
using namespace std;
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert()
{
  int val;
  if (rear == n - 1)
  cout<<"Queue Overflow"<<endl;
  else {
    if (front == - 1)
    front = 0;
    cout<<"Insert the element in queue : "<<endl;
    cin>>val;
    rear++;
    queue[rear] = val;
  }
}
void Delete()
{
  if (front == - 1 || front > rear) {
    cout<<"Queue Underflow ";
    return ;
  } else {
    cout<<"Element deleted from queue is : "<< queue[front] <<endl;
    front++;;
  }
}
void Display()
{
  if (front == - 1)
  cout<<"Queue is empty"<<endl;
  else {
    cout<<"Queue elements are : ";
    for (int i = front; i <= rear; i++)
    cout<<queue[i]<<" ";
      cout<<endl;
  }
}
int main()
{
  int ch;
  cout<<"1) Insert element to queue"<<endl;
  cout<<"2) Delete element from queue"<<endl;
  cout<<"3) Display all the elements of queue"<<endl;
  cout<<"4) Exit"<<endl;
  do {
    cout<<"Enter your choice : "<<endl;
    cin>>ch;
    switch (ch) {
      case 1: Insert();
      break;
```

```
        case 2: Delete();
        break;
        case 3: Display();
        break;
        case 4: cout<<"Exit"<<endl;
        break;
        default: cout<<"Invalid choice"<<endl;
      }
  } while(ch!=4);
  return 0;
}
/*************************************************************************************
```

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 Hrishav Sharma(211B138).
/************************************************************************************
#include <iostream>
using namespace std;

int cqueue[100];
int front = -1, rear = -1, n=100;

void insertCQ(int val) {
  if ((front == 0 && rear == n-1) || (front == rear+1)) {
    cout<<"Queue Overflow";
    return;
  }
  if (front == -1) {
    front = 0;
    rear = 0;
  } else {
    if (rear == n - 1)
    rear = 0;
    else
    rear = rear + 1;
  }
  cqueue[rear] = val ;
}
void deleteCQ() {
  if (front == -1) {
    cout<<"Queue Underflow";
    return ;
  }
  cout<<"Element deleted from queue is : "<<cqueue[front]<<endl;

  if (front == rear) {
    front = -1;
    rear = -1;
  } else {
    if (front == n - 1)
    front = 0;
    else
    front = front + 1;
  }
}
void displayCQ() {
  int f = front, r = rear;
  if (front == -1) {
    cout<<"Queue is empty"<<endl;
    return;
  }
  cout<<"Queue elements are :";
  if (f <= r) {
    while (f <= r){
      cout<<cqueue[f]<<" ";
      f++;
    }
  } else {
    while (f <= n - 1) {
      cout<<cqueue[f]<<" ";
      f++;
    }
    f = 0;
    while (f <= r) {
      cout<<cqueue[f]<<" ";
      f++;
    }
  }
  cout<<endl;
}
int main() {

  int ch, val;
  cout<<"1)Insert";
  cout<<"2)Delete";
  cout<<"3)Display";
  cout<<"4)Exit";
  while(ch != 4)
  {
    cout<<"Enter choice : "<<endl;
    cin>>ch;
    switch(ch) {
      case 1:
      cout<<"Input for insertion: "<<endl;
      cin>>val;
      insertCQ(val);
      break;

      case 2:
      deleteCQ();
      break;

      case 3:
      displayCQ();
      break;

      case 4:
      cout<<"Exit";
      break;
      default: cout<<"Incorrect!";
    }
  }
  return 0;
}
```

****

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 Hrishav Sharma(211B138).
/***************************************************************************************
#include <iostream>
#include<malloc.h>
using namespace std;
struct node {
  int data;
  struct node *next;
};
struct node* front = NULL;
struct node* rear = NULL;
struct node* temp;
void Insert() {
  int val;
  cout<<"Insert the element in queue : "<<endl;
  cin>>val;
  if (rear == NULL) {
    rear = (struct node *)malloc(sizeof(struct node));
    rear->next = NULL;
    rear->data = val;
    front = rear;
  } else {
    temp=(struct node *)malloc(sizeof(struct node));
    rear->next = temp;
    temp->data = val;
    temp->next = NULL;
    rear = temp;
  }
}
void Delete() {
  temp = front;
  if (front == NULL)
   {
    cout<<"Underflow"<<endl;
    return;
  }
  else
  if (temp->next != NULL)
  {
    temp = temp->next;
    cout<<"Element deleted from queue is : "<<front->data<<endl;
    free(front);
    front = temp;
  } else {
    cout<<"Element deleted from queue is : "<<front->data<<endl;
    free(front);
    front = NULL;
    rear = NULL;
  }
}
void Display()

{
  temp = front;
  if ((front == NULL) && (rear == NULL))
   {
    cout<<"Queue is empty"<<endl;
    return;
  }
  cout<<"Queue elements are: ";
  while (temp != NULL)
  {
    cout<<temp->data<<" ";
    temp = temp->next;
  }
  cout<<endl;
}
```

```
int main()
{
  int ch;
  cout<<"1) Insert element to queue"<<endl;
  cout<<"2) Delete element from queue"<<endl;
  cout<<"3) Display all the elements of queue"<<endl;
  cout<<"4) Exit"<<endl;
  while(ch!=4)
   {
    cout<<"Enter your choice : "<<endl;
    cin>>ch;
    switch (ch)
    {
      case 1: Insert();
      break;
      case 2: Delete();
      break;
      case 3: Display();
      break;
      case 4: cout<<"Exit"<<endl;
      break;
      default: cout<<"Invalid choice"<<endl;
    }
  }
  return 0;
}
******************************************************************************
```

Lab 11: Trees

Q1.. WAP to check whether given tree is a binary search tree or not.

```
/*************************************************************************************
//This program is created by Hrishav Sharma(211B138)
/*************************************************************************************
#include<iostream>
using namespace std;
struct Node
{
   int data;
   struct Node *left, *right;
};
```

```
Node* newNode(int data)
{
   Node* temp = new Node;
   temp->data = data;
   temp->left = temp->right = NULL;
   return temp;
}
int getMin(struct Node* node);
int getMax(Node* root);
int isBST(struct Node* node)
{
  if (node == NULL)
   return 1;
  if (node->left != NULL && getMax(node->left) > node->data)
   return 0;
  if (node->left != NULL && getMin(node->right) < node->data)
   return 0;
  if (!isBST(node->left) || !isBST(node->right))
   return 0;
  return 1;
}
int getMin(Node *root)
    {
       if(root==NULL)
       return INT_MAX;
      int res=root->data;
      int left=getMin(root->left);
      int right=getMin(root->right);
      if(left<res)
      {
         res=left;
      }
      if(right<res)
      {
         res=right;
      }
      return res;
    }
int getMax(Node* root)
{
   if (root == NULL)
       return INT_MIN;

   int res = root->data;
   int lres = getMax(root->left);
   int rres = getMax(root->right);
   if (lres > res)
       res = lres;
   if (rres > res)
       res = rres;
   return res;
}
int main(){
   Node* root = newNode(10);
   root->left = newNode(6);
   root->right = newNode(4);
   root->left->left = newNode(11);
   root->left->right = newNode(2);

   if(isBST(root))
   cout<<"Binary Tree is BST";
   else
   cout<<"Binary Tree is not Bst";
   return 0;
}
```

****

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

```
/*************************************************************************************
//This program is created by Hrishav Sharma(211B138)
/*************************************************************************************
#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);
  return 0;
}
```

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

```
/*************************************************************************************
//This program is created by Hrishav Sharma(211B138)
/*************************************************************************************
#include<stdio.h>
#include<malloc.h>

struct node{
    int data;
    struct node* left;
    struct node* right;
};

struct node* createNode(int data){
    struct node *n;
    n = (struct node *) malloc(sizeof(struct node));
    n->data = data;
    n->left = NULL;
    n->right = NULL;
    return n;
}

void preOrder(struct  node* root){
    if(root!=NULL){
        printf("%d ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(struct  node* root){
    if(root!=NULL){
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }
}

void inOrder(struct  node* root){
    if(root!=NULL){
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }
}

int isBST(struct  node* root){
    static struct node *prev = NULL;
    if(root!=NULL){
        if(!isBST(root->left)){
            return 0;
        }
        if(prev!=NULL && root->data <= prev->data){
            return 0;
        }
        prev = root;
        return isBST(root->right);
    }
    else{
        return 1;
    }
}

struct node * search(struct node* root, int key){
    if(root==NULL){
        return NULL;
    }
    if(key==root->data){
        return root;
    }
    else if(key<root->data){
        return search(root->left, key);
    }
    else{
        return search(root->right, key);
    }
}

int main(){
         struct node *p1 = createNode(3);
    struct node *p2 = createNode(6);
    struct node *p3 = createNode(1);
    struct node *p4 = createNode(4);
    p->left = p1;
    p->right = p2;
    p1->left = p3;
    p1->right = p4;
    struct node* n = search(p, 10);
    if(n!=NULL){
    printf("Found: %d", n->data);
    }
    else{
        printf("Element not found");
    }
    return 0;
}
```

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

```
/**************************************************************************************
//This program is created by Hrishav Sharma(211B138)
/**************************************************************************************
#include<stdio.h>
#include<malloc.h>

struct node{
    int data;
    struct node* left;
    struct node* right;
};

struct node* createNode(int data){
    struct node *n;
    n = (struct node *) malloc(sizeof(struct node));
    n->data = data;
    n->left = NULL;
    n->right = NULL;
    return n;
}

void preOrder(struct  node* root){
    if(root!=NULL){
        printf("%d ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(struct  node* root){
    if(root!=NULL){
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }
}

void inOrder(struct  node* root){
    if(root!=NULL){
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }
}

int isBST(struct  node* root){
    static struct node *prev = NULL;
    if(root!=NULL){
        if(!isBST(root->left)){
            return 0;
        }
        if(prev!=NULL && root->data <= prev->data){
            return 0;
        }
        prev = root;
        return isBST(root->right);
    }
    else{
        return 1;
    }
}

struct node * searchIter(struct node* root, int key){
    while(root!=NULL){
        if(key == root->data){
            return root;
        }
        else if(key<root->data){
            root = root->left;
        }
        else{
            root = root->right;
        }
    }
    return NULL;
}
```

```
void insert(struct node *root, int key){
  struct node *prev = NULL;
  while(root!=NULL){
    prev = root;
    if(key==root->data){
      printf("Cannot insert %d, already in BST", key);
      return;
    }
    else if(key<root->data){
      root = root->left;
    }
    else{
      root = root->right;
    }
  }
  struct node* new = createNode(key);
  if(key<prev->data){
    prev->left = new;
  }
  else{
    prev->right = new;
  }

}

int main(){
   struct node *p = createNode(5);
   struct node *p1 = createNode(3);
   struct node *p2 = createNode(6);
   struct node *p3 = createNode(1);
   struct node *p4 = createNode(4);

   // Linking the root node with left and right children
   p->left = p1;
   p->right = p2;
   p1->left = p3;
   p1->right = p4;

   insert(p, 16);
   printf("%d", p->right->right->data);
   return 0;
}
```

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

```
/*************************************************************************************
//This program is created by Hrishav Sharma(211b138).
//*************************************************************************************
#include <iostream>
using namespace std;
struct Node {
        int key;
```

```
    struct Node *left, *right;
};
struct Node* newNode(int key)
{
    struct Node* temp = new Node;
    temp->key = key;
    temp->left = temp->right = NULL;
    return temp;
};

void inorder(struct Node* temp)
{
    if (!temp)
        return;
    inorder(temp->left);
    cout << temp->key << " ";
    inorder(temp->right);
}

void deletDeepest(struct Node* root, struct Node* d_node)
{
    queue<struct Node*> q;
    q.push(root);
    struct Node* temp;
    while (!q.empty()) {
        temp = q.front();
        q.pop();
        if (temp == d_node) {
            temp = NULL;
            delete (d_node);
            return;
        }
        if (temp->right) {
            if (temp->right == d_node) {
                temp->right = NULL;
                delete (d_node);
                return;
            }
            else
                q.push(temp->right);
        }

        if (temp->left) {
            if (temp->left == d_node) {
                temp->left = NULL;
                delete (d_node);
                return;
            }
            else
                q.push(temp->left);
        }
    }
}

Node* deletion(struct Node* root, int key)
{
    if (root == NULL)
        return NULL;

    if (root->left == NULL && root->right == NULL) {
        if (root->key == key)
            return NULL;
        else
            return root;
    }

    queue<struct Node*> q;
    q.push(root);

    struct Node* temp;
    struct Node* key_node = NULL;
    while (!q.empty()) {
        temp = q.front();
        q.pop();

        if (temp->key == key)
            key_node = temp;

        if (temp->left)
            q.push(temp->left);

        if (temp->right)
            q.push(temp->right);
    }

    if (key_node != NULL) {
        int x = temp->key;
        deletDeepest(root, temp);
        key_node->key = x;
    }
    return root;
}
int main()
{
    struct Node* root = newNode(10);
    root->left = newNode(11);
    root->left->left = newNode(7);
    root->left->right = newNode(12);
    root->right = newNode(9);
    root->right->left = newNode(15);
    root->right->right = newNode(8);
```

```
        cout << "Inorder traversal before deletion : ";
        inorder(root);

        int key = 11;
        root = deletion(root, key);

        cout << endl;
        cout << "Inorder traversal after deletion : ";
        inorder(root);

        return 0;
}
**************************************************************************************
```

LAB 12: Graph

Q1. WAP to perform BFS on graph.

```
//*************************************************************************************
This program is created by Hrishav Sharma(211b138).
//*************************************************************************************
#include<iostream>
#include<queue>
#define NODE 6
using namespace std;

typedef struct node {
  int val;
  int state;
}node;
```

```
int graph[NODE][NODE] = {
  {0, 1, 1, 1, 0, 0},
  {1, 0, 0, 1, 1, 0},
  {1, 0, 0, 1, 0, 1},
  {1, 1, 1, 0, 1, 1},
  {0, 1, 0, 1, 0, 1},
  {0, 0, 1, 1, 1, 0}
};

void bfs(node *vert, node s) {
  node u;
  int i, j;
  queue<node> que;

  for(i = 0; i<NODE; i++) {
    vert[i].state = 0;
  }

  vert[s.val].state = 1;
  que.push(s);

  while(!que.empty()) {
    u = que.front();
    que.pop();
    cout << char(u.val+'A') << " ";

    for(i = 0; i<NODE; i++) {
      if(graph[i][u.val]) {
        if(vert[i].state == 0) {
          vert[i].state = 1;
          que.push(vert[i]);
        }
      }
    }
    u.state = 2;
  }
}

int main() {
  node vertices[NODE];
  node start;
  char s;

  for(int i = 0; i<NODE; i++) {
    vertices[i].val = i;
  }

  s = 'B';
  start.val = s-'A';
  cout << "BFS Traversal: ";
  bfs(vertices, start);
  cout << endl;
}
```

Q2. WAP to perform DFS on graph.

```
// This program is created by Hrishav Sharma(211b138)
/**********************************************************************************
#include<iostream>
#include<stack>
using namespace std;
#define NODE 6

typedef struct node {
  int val;
  int state; //status
}node;

int graph[NODE][NODE] = {
  {0, 1, 1, 1, 0, 0},
  {1, 0, 0, 1, 1, 0},
  {1, 0, 0, 1, 0, 1},
  {1, 1, 1, 0, 1, 1},
  {0, 1, 0, 1, 0, 1},
  {0, 0, 1, 1, 1, 0}
};

void dfs(node *vertex, node start) {
  node u;
  stack<node> myStack;

  for(int i = 0; i<NODE; i++) {
    vertex[i].state = 0;   //not visited
  }

  myStack.push(start);
  while(!myStack.empty()) {
    //pop and print node
    u = myStack.top();
    myStack.pop();
    cout << char(u.val+'A') << " ";

    if(u.state != 1) {
      //update vertex status to visited
      u.state = 1;
      vertex[u.val].state = 1;

      for(int i = 0; i<NODE; i++) {
        if(graph[i][u.val]) {
          if(vertex[i].state == 0) {
            myStack.push(vertex[i]);
            vertex[i].state = 1;
          }
        }
      }
    }
  }
}

int main() {
  node vertices[NODE];
  node start;
  char s;

  for(int i = 0; i<NODE; i++) {
    vertices[i].val = i;
  }

  s = 'C';   //starting vertex C
  start.val = s-'A';
  cout << "DFS Traversal: ";
  dfs(vertices, start);
  cout << endl;
}
```

Q3

```
//****************************************************************************************
// This program is created by Hrishav Sharma(211b138)
//****************************************************************************************


#include <bits/stdc++.h>
using namespace std;
#define V 5
bool createsMST(int u, int v, vector<bool> inMST){
  if (u == v)
    return false;
  if (inMST[u] == false && inMST[v] == false)
    return false;
  else if (inMST[u] == true && inMST[v] == true)
    return false;
  return true;
}
void printMinSpanningTree(int cost[][V]){
  vector<bool> inMST(V, false);
  inMST[0] = true;
  int edgeNo = 0, MSTcost = 0;
  while (edgeNo < V - 1) {
    int min = INT_MAX, a = -1, b = -1;
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
        if (cost[i][j] < min) {
          if (createsMST(i, j, inMST)) {
            min = cost[i][j];
            a = i;
            b = j;
          }
        }
      }
    }
    if (a != -1 && b != -1) {
      cout<<"Edge "<<edgeNo++<<" : ("<<a<<" , "<<b<<" ) : cost = "<<min<<endl;
      MSTcost += min;
      inMST[b] = inMST[a] = true;
    }
  }
  cout<<"Cost of Minimum spanning tree ="<<MSTcost;
}
int main() {
  int cost[][V] = {
    { INT_MAX, 12, INT_MAX, 25, INT_MAX },
    { 12, INT_MAX, 11, 8, 12 },
    { INT_MAX, 11, INT_MAX, INT_MAX, 17 },
    { 25, 8, INT_MAX, INT_MAX, 15 },
    { INT_MAX, 12, 17, 15, INT_MAX },
  };
  cout<<"The Minimum spanning tree for the given tree is :
";
  printMinSpanningTree(cost);
  return 0;
}
```

Lab 8:Linked List

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

```
\/*********************************************************************************
This program is created by Hrishav Sharma(211B138)
//*********************************************************************************
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;          // Data
   struct node *next; // Address
}*head;


void createList(int n);
void insertNodeAtEnd(int data);
void displayList();


int main()
{
   int n, data;
   printf("Enter the total number of nodes: ");
   scanf("%d", &n);
   createList(n);
    printf("\nData in the list \n");
   displayList();
   printf("\nEnter data to insert at end of the list: ");
   scanf("%d", &data);
   insertNodeAtEnd(data);

   printf("\nData in the list \n");
   displayList();

   return 0;
}
void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;

   head = (struct node *)malloc(sizeof(struct node));
   if(head == NULL)
   {
      printf("Unable to allocate memory.");
   }
   else
   {
      printf("Enter the data of node 1: ");
      scanf("%d", &data);
```

```
        head->data = data;
        head->next = NULL;

        temp = head;
        for(i=2; i<=n; i++)
        {
            newNode = (struct node *)malloc(sizeof(struct node));

            if(newNode == NULL)
            {
                printf("Unable to allocate memory.");
                break;
            }
            else
            {
                printf("Enter the data of node %d: ", i);
                scanf("%d", &data);

                newNode->data = data;
                newNode->next = NULL;
                temp->next = newNode;
                temp = temp->next;
            }
        }
    }
}

void insertNodeAtEnd(int data)
{
    struct node *newNode, *temp;

    newNode = (struct node*)malloc(sizeof(struct node));

    if(newNode == NULL)
    {
        printf("Unable to allocate memory.");
    }
    else
    {
        newNode->data = data; // Link the data part
        newNode->next = NULL;

        temp = head;

        // Traverse to the last node
        while(temp != NULL && temp->next != NULL)
            temp = temp->next;

        temp->next = newNode;
 printf("DATA INSERTED SUCCESSFULLY\n");
    }
```

```
}
void displayList()
{
   struct node *temp;
   if(head == NULL)
   {
      printf("List is empty.");
   }
   else
   {
      temp = head;
      while(temp != NULL)
      {
         printf("Data = %d\n", temp->data);
         temp = temp->next;
      }
   }
}
```

Q2. WAP to print the elements of a linked list.

```
//*************************************************************************************
// This Program is created by Hrishav Sharma(211B138)
//*************************************************************************************
#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 %d", 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 created by Hrishav Sharma(211b138)
//*****************************************************************************************
#include <stdio.h>
#include <stdlib.h>
struct node
{
int data;
struct node *next;
}*head;
void createList(int n);
void insertNodeAtBeginning(int data);
void displayList();
int main()
{
int n, data;
printf("Enter the total number of nodes: ");
scanf("%d", &n);
```

```
createList(n);
printf("\nData in the list ==>>\t");
displayList();
printf("\nEnter data to insert at beginning: ");
scanf("%d", &data);
insertNodeAtBeginning(data);
printf("\nData in the list \n");
displayList();
return 0;
}
void createList(int n)
{
struct node *newNode, *temp;
int data, i;
head = (struct node *)malloc(sizeof(struct node));
if(head != NULL)
{
printf("Enter the data of node 1: ");
scanf("%d", &data);
head->data = data;
head->next = NULL;
temp = head;
for(i=2; i<=n; i++)
{
newNode = (struct node *)malloc(sizeof(struct node));
if(newNode!= NULL)
18
{
printf("Enter the data of node %d: ", i);
scanf("%d", &data);
newNode->data = data;
newNode->next = NULL;
temp->next = newNode;
temp = temp->next;
}
}
}
}
void insertNodeAtBeginning(int data)
{
struct node *newNode;
newNode = (struct node*)malloc(sizeof(struct node));
if(newNode!= NULL)
{
newNode->data = data;
newNode->next = head;
head = newNode;
}
}
void displayList()
{
struct node *temp;
if(head!=NULL)
```

```
{
temp = head;
while(temp != NULL)
{
printf("%d\t", temp->data);
temp = temp->next;
}
}
}
```

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

```
//****************************************************************************************************
This Program is Created By Hrishav Sharma(211B138)
/****************************************************************************************************/
#include <iostream>
using namespace std;
class SinglyLinkedListNode {
   public:
      int data;
      SinglyLinkedListNode *next;

      SinglyLinkedListNode(int node_data) {
         this->data = node_data;
         this->next = nullptr;
      }
};

class SinglyLinkedList {
   public:
      SinglyLinkedListNode *head;
      SinglyLinkedListNode *tail;

      SinglyLinkedList() {
         this->head = nullptr;
         this->tail = nullptr;
      }

      void insert_node(int node_data) {
         SinglyLinkedListNode* node = new SinglyLinkedListNode(node_data);

         if (!this->head) {
            this->head = node;
         } else {
            this->tail->next = node;
         }

         this->tail = node;
```

```
        }
};

void print_singly_linked_list(SinglyLinkedListNode* node, string sep, ofstream& fout) {
    while (node) {
        fout << node->data;

        node = node->next;

        if (node) {
            fout << sep;
        }
    }
}

void free_singly_linked_list(SinglyLinkedListNode* node) {
    while (node) {
        SinglyLinkedListNode* temp = node;
        node = node->next;

        free(temp);
    }
}


SinglyLinkedListNode* insertNodeAtPosition(SinglyLinkedListNode* head, int data, int
position) {
SinglyLinkedListNode *temp=new SinglyLinkedListNode(data);
SinglyLinkedListNode *nh= head;
for(int i=0;i<position-1;i++)
{
    nh=nh->next;
}
SinglyLinkedListNode *t;
t=nh->next ;
nh->next=temp;
temp->next=t;
return head;
}
int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    SinglyLinkedList* llist = new SinglyLinkedList();

    int llist_count;
    cin >> llist_count;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    for (int i = 0; i < llist_count; i++) {
        int llist_item;
        cin >> llist_item;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        llist->insert_node(llist_item);
    }

    int data;
    cin >> data;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    int position;
    cin >> position;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    SinglyLinkedListNode* llist_head = insertNodeAtPosition(llist->head, data, position);

    print_singly_linked_list(llist_head, " ", fout);
    fout << "\n";

    free_singly_linked_list(llist_head);

    fout.close();

    return 0;
}
```

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

//**

This Program is Created by Hrishav Sharma(211B138)

/**

```
#include<stdio.h>
#include<stdlib.h>

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

void printList(struct node* head_ref)
{
    struct node* head_ref = (struct node*)malloc(sizeof(struct node));
    if(head_ref == NULL)
    printf("The list is empty");

    while(head_ref!=NULL)
    {
        printf("%d\n",head_ref->data);
        head_ref = head_ref->next;
    }
}

void insert_beg(struct node **head_ref,int new_data)
{
    struct node* new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = new_data;
    new_node->next = *head_ref;
    *head_ref = new_node;
}

void delete(struct node **head_ref,int position)
{
    int i=1;
    if(*head_ref == NULL)
    return;

    struct node *tails,*temp = *head_ref;
    if(position == 0)
    {

        *head_ref = temp->next;
        free(temp);
        return;
    }

    while(temp->next!=NULL)
    {
        tails = temp->next;
        temp = temp->next;

        if(i == position)
        {
            tails->next = temp->next;
            free(temp);
            return;
        }

        i++;
    }

}

int main()
{
    struct node *head = NULL;
    insert_beg(&head,36);
    insert_beg(&head,35);
    insert_beg(&head,34);
    insert_beg(&head,33);

    printList(head);
    int position;
    printf("Enter the position of the node u wanna delete\n");
    scanf("%d",&position);

    delete(&head,position);
    printf("\n");
    printList(head);
}
```

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

```
//*****************************************************************************************************
This Program is Created By Hrishav Sharma(211b138)
*****************************************************************************************************//
#include <stdio.h>
#include <stdlib.h>
struct node {
int data; //Data part
struct node *next; //Address part
}*head;
void createList(int n);
void reverseList();
void displayList();
int main()
{
int n, choice;
printf("Enter the total number of nodes: ");
scanf("%d", &n);
createList(n);
printf("\nData in the list \n");
displayList();
printf("\nPress 1 to reverse the order of singly linked list\n");
scanf("%d", &choice);
if(choice == 1)
{
reverseList();
}
printf("\nData in the list\n");
displayList();
return 0;
}
void createList(int n)
{
struct node *newNode, *temp;
int data, i;
if(n <= 0)
25
{
printf("List size must be greater than zero.\n");
```

```
return;
}
head = (struct node *)malloc(sizeof(struct node));
if(head == NULL)
{
printf("Unable to allocate memory.");
}
else
{
printf("Enter the data of node 1: ");
scanf("%d", &data);
head->data = data;
head->next = NULL;
temp = head;
for(i=2; i<=n; i++)
{
newNode = (struct node *)malloc(sizeof(struct node));
if(newNode == NULL)
{
printf("Unable to allocate memory.");
break;
}
else
{
printf("Enter the data of node %d: ", i);
scanf("%d", &data);
newNode->data = data;
newNode->next = NULL;
temp->next = newNode;
temp = temp->next;
}
}
printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");
}
}
void reverseList()
{
struct node *prevNode, *curNode;
if(head != NULL)
{
prevNode = head;
curNode = head->next;
head = head->next;
prevNode->next = NULL;
while(head != NULL)
{
head = head->next;
curNode->next = prevNode;
26
prevNode = curNode;
curNode = head;
}
```

```
head = prevNode;
}
}
void displayList()
{
struct node *temp;
if(head == NULL)
{
printf("List is empty.");
}
else
{
temp = head;
while(temp != NULL)
{
printf("Data = %d\n", temp->data);
temp = temp->next;
}
}
}
```

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 Created By Hrishav Sharma(211B138)
//***************************************************************************************************


#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");
      }

      printf("\n\n\n\n\n");
   }

   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;
        printf("DATA IN THE LIST:\n");

        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;
      }
```

```
        else
        {
           printf("Error, Invalid position\n");
        }
    }
}
```