**Exp 1 dataase sort**

#include <iostream>

using namespace std;

#include<string.h>

int const size=3;

struct student{

int rno;

char name[20];

float SGPA;

};

void accept(struct student list[size]);

void display(struct student list [80]);

void displayTop(struct student list[80]);

void bubbleSort(struct student list[size]);

void insertSort(struct student list[size]);

void quickSort(struct student list[size],int,int);

void search(struct student list[size] );

void binarysearch(struct student list[size]);

main()

{

int ch, i;

struct student data[20];

accept (data);

cout<<"\n 1:Bubble Sort";

cout<<"\n 2:Insertion Sort";

cout<<"\n 3:Quick Sort";

cout<<"\n 4:Search";

cout<<"\n 5:Binary Search";

cout<<"\n Select your choice:";

cin>>ch;

switch(ch)

{

case 1:

bubbleSort(data);

display(data);

break;

case 2:

insertSort(data);

display(data);

break;

case 3:

quickSort(data,0,size-1);

displayTop(data);

break;

case 4:

search(data);

break;

case 5:

binarysearch(data);

break;

default:

cout<<"Invalid choice....";

}

}

void accept(struct student list[size])

{

int i;

for (i=0;i<size;i++)

{

cout<<"Enter rollno,name & SGPA:";

cin>>list[i].rno>>list[i].name>>list[i].SGPA;

}

}

void display(struct student list[80])

{

int i;

cout<<"\n Roll no \t Name \t SGPA \n";

for(i=0;i<size;i++)

{

cout<<"\n"<<list[i].rno<<"\t"<<list[i].name<<"\t"<<list[i].SGPA;

}

}

void displayTop(struct student list[80])

{

int i;

cout<<"\n\nRollno\tName\tSGPA\n";

for(i=0;i<3;i++)

{

cout<<"\n"<<list[i].rno<<"\t"<<list[i].name<<"\t"<<list[i].SGPA;

}

}

void bubbleSort(struct student list[size])

{

int i,j;

struct student temp;

for(i=0;i<size-1;i++)

{

for(j=0;j<(size-1-i);j++)

{

if(list[j].rno>list[j+1].rno)

{

temp=list[j];

list[j]=list[j+1];

list[j+1]=temp;

}

}

}

}

void insertSort(struct student list[size])

{

int k,j;

struct student temp;

for(k=1;k<size;k++)

{

temp=list[k];

j=k-1;

while(strcmp(list[j].name,temp.name)>0&&j>=0)

{

list[j+1]=list[j];

--j;

}

list[j+1]=temp;

}

}

void quickSort(struct student list[size],int first,int last)

{

int pivot,i,j;

struct student temp;

if(first<last)

{

pivot=first;

i=first;

j=last;

while(i<j)

{

while(list[i].SGPA>=list[pivot].SGPA&&i<last)

i++;

while(list[j].SGPA<list[pivot].SGPA)

j--;

if(i<j)

{

temp=list[i];

list[i]=list[j];

list[j]=temp;

}

}

temp=list[pivot];

list[pivot]=list[j];

list[j]=temp;

quickSort(list,first,j-1);

quickSort(list,j+1,last);

}

}

void search(struct student list[size])

{

float SGPA;

int i;

cout<<"\n Enter SGPA";

cin>>SGPA;

cout<<"\n Rollno \t Name \t SGPA \n";

for(int i=0;i<size;i++)

{

if(SGPA==list[i].SGPA)

cout<<"\n"<<list[i].rno<<"\t"<<list[i].name<<"\t"<<list[i].SGPA;

}

}

void binarysearch(struct student list[size])

{

int k, lower, upper, mid;

char search[80];

cout<<"\n Enter name of the students you want to search";

cin>>search;

lower=0;

upper=size-1;

mid=(lower+upper)/2;

while(lower<=upper)

{

if(strcmp(list[mid].name,search)<0)

lower=mid+1;

else if(strcmp(list[mid].name,search)==0)

{

cout<<"\n"<<list[mid].rno<<"\t"<<list[mid].name<<"\t"<<list[mid].SGPA;

break;

}

else

upper=mid-1;

mid=(lower+upper)/2;

}

if(lower>upper)

cout<<search<<"not found in the list";

}

OUTPUT:-

Enter rollno,name & SGPA:25

Aadarsh

8.77

Enter rollno,name & SGPA:26

Ritesh

8.25

Enter rollno,name & SGPA:27

Bharat

8.95

1:Bubble Sort

2:Insertion Sort

3:Quick Sort

4:Search

5:Binary Search

Select your choice:1

Roll no Name SGPA

25 Aadarsh 8.77

26 Ritesh 8.25

27 Bharat 8.95

**Exp 2 stack**

**Input:**

**#include<iostream>**

**#include<ctype.h>**

**#include<string.h>**

**using namespace std;**

**class Stack**

**{**

**//Structure for Expression**

**struct Stk**

**{**

**float Operator;**

**Stk \*Next;**

**Stk(){ Next=NULL;}**

**};**

**Stk \*Top;**

**public:**

**Stack(){Top=NULL;}**

**int Empty();**

**void Push(float Opr);**

**float Pop();**

**};**

**int Stack::Empty()**

**{**

**if(Top==NULL)**

**return 1;**

**return 0;**

**}**

**void Stack::Push(float Opr)**

**{**

**Stk \*Node;**

**Node=new Stk;**

**Node->Operator=Opr;**

**Node->Next=Top;**

**Top=Node;**

**}**

**float Stack::Pop()**

**{**

**Stk \*Temp=Top;**

**float Opr;**

**Top=Top->Next;**

**Opr=Temp->Operator;**

**delete Temp;**

**return Opr;**

**}**

**//Stack class End**

**//Function return Operater Priority**

**int Priority(char Op)**

**{**

**if(Op=='^')**

**return 2;**

**if(Op=='+' || Op=='-')**

**return 0;**

**else return 1;**

**}**

**//Return the result of given operation**

**float Operation(char Op,float A,float B)**

**{**

**int I=0;**

**float P=1;**

**if(Op=='\*') P=A\*B;**

**else if(Op=='/') P=A/B;**

**else if(Op=='+') P=A+B;**

**else if(Op=='-') P=A-B;**

**else while(I++<B) P=P\*A;**

**return P;**

**}**

**void infixTOpostfix(char str[20])**

**{**

**char Opr,post[20];**

**int i,j=0;**

**Stack S;**

**for(i=0;str[i]!='\0';i++)**

**{**

**if(isalnum(str[i])) post[j++]=str[i];**

**else**

**{**

**if(str[i]== ')')**

**{**

**Opr=S.Pop();**

**while(Opr!='(')**

**{ post[j++]=Opr; Opr=S.Pop(); }//while**

**}**

**else { if(str[i]=='(');**

**else while(!S.Empty())**

**{**

**Opr=S.Pop();**

**if(Opr!='('&&Priority(Opr)>= Priority(str[i]))**

**post[j++]=Opr;**

**else**

**{S.Push(Opr);**

**break;}**

**}//while**

**S.Push(str[i]);**

**}**

**}**

**}//for**

**while(!S.Empty())**

**post[j++]=S.Pop();**

**post[j]='\0';**

**cout<<post;**

**}**

**void infixTOprefix(char str[20])**

**{**

**char Opr,pre[20];**

**int i,j=0;**

**Stack S;**

**for(i=strlen(str)-1;i>=0;i--)**

**{**

**if(isalnum(str[i])) pre[j++]=str[i];**

**else**

**{**

**if(str[i]== '(')**

**{**

**Opr=S.Pop();**

**while(Opr!=')')**

**{ pre[j++]=Opr; Opr=S.Pop(); }//while**

**}**

**else { if(str[i]==')');**

**else while(!S.Empty())**

**{**

**Opr=S.Pop();**

**if(Opr!=')'&&Priority(Opr)>Priority(str[i]))**

**pre[j++]=Opr;**

**else**

**{S.Push(Opr);**

**break;}**

**}//while**

**S.Push(str[i]);**

**}**

**}**

**}//for**

**while(!S.Empty())**

**pre[j++]=S.Pop();**

**pre[j]='\0';**

**for(j--;j>=0;j--)**

**cout<<pre[j];**

**}**

**float Postfix\_Evaluation(char String[20])**

**{**

**int I=0;**

**float Operand1,Operand2,Result;**

**Stack S;**

**while(String[I]!='\0')**

**{**

**if(String[I]>='0' &&String[I]<='9')**

**S.Push(String[I]-48);**

**else**

**{**

**Operand2=S.Pop();**

**Operand1=S.Pop();**

**Result=Operation(String[I],Operand1,Operand2);**

**S.Push(Result);**

**}**

**I++;**

**}**

**return S.Pop();**

**}**

**//PreFix Expression Evaluation**

**float Prefix\_Evaluation(char String[20])**

**{**

**int I=strlen(String)-1;**

**float Operand1,Operand2,Result;**

**Stack S;**

**while(I>=0)**

**{**

**if(String[I]>='0' &&String[I]<='9')**

**S.Push(String[I]-48);**

**else**

**{**

**Operand1=S.Pop();**

**Operand2=S.Pop();**

**Result=Operation(String[I],Operand1,Operand2);**

**S.Push(Result);**

**}**

**I--;**

**}**

**return S.Pop();**

**}**

**int main()**

**{**

**int Choice;**

**char Expression[25],Answer;**

**do**

**{**

**cout<<"\n1:Infix to Prefix\n2:Infix to Postfix\n3:Postfix**

**Evaluation\n4:Prefix Evaluation";**

**cout<<"\nEnter your Choice: ";**

**cin>>Choice;**

**switch(Choice)**

**{**

**case 1:**

**cout<<"\nEnter infix Expression";**

**cin>>Expression;**

**infixTOprefix(Expression);**

**break;**

**case 2:**

**cout<<"\nEnter infix Expression";**

**cin>>Expression;**

**infixTOpostfix(Expression);**

**break;**

**case 3:**

**cout<<"\nEnter Postfix Expression";**

**cin>>Expression;**

**cout<<"\nEvaluated Result :"**

**<<Postfix\_Evaluation(Expression);**

**break;**

**case 4:**

**cout<<"\nEnter Prefix Expression";**

**cin>>Expression;**

**cout<<"\nEvaluated Result "**

**<<Prefix\_Evaluation(Expression);**

**break;**

**}**

**cout<<"\nContinue(y/n)...";**

**cin>>Answer;**

**}while(Answer=='y'||Answer=='Y');**

**return 0;**

**}**

**Output:**

**Exp3 circular queue**

Circular Queue

#include <iostream>

#define SIZE 5 /\* Size of Circular Queue \*/

using namespace std;

class Queue {

private:

int items[SIZE], front, rear;

public:

Queue() {

front = -1;

rear = -1;

}

// Check if the queue is full

bool isFull() {

if (front == 0 && rear == SIZE - 1) {

return true;

}

if (front == rear + 1) {

return true;

}

return false;

}

// Check if the queue is empty

bool isEmpty() {

if (front == -1)

return true;

else

return false;

}

// Adding an element

void enQueue() {

int element;

if (isFull()) {

cout << "Queue is full";

} else {

if (front == -1) front = 0;

rear = (rear + 1) % SIZE;

cout<<"Enter the element to be inserted: ";

cin>>element;

items[rear] = element;

cout << endl

<< "Inserted " << element << endl;

}

}

// Removing an element

int deQueue() {

int element;

if (isEmpty()) {

cout << "Queue is empty" << endl;

return (-1);

} else {

element = items[front];

if (front == rear) {

front = -1;

rear = -1;

}

// Q has only one element,

// so we reset the queue after deleting it.

else {

front = (front + 1) % SIZE;

}

return (element);

}

}

void display() {

// Function to display status of Circular Queue

int i;

if (isEmpty()) {

cout << endl

<< "Empty Queue" << endl;

} else {

cout << "Front -> " << front;

cout << endl

<< "Items -> ";

for (i = front; i != rear; i = (i + 1) % SIZE)

cout << items[i];

cout << items[i];

cout << endl

<< "Rear -> " << rear;

}

}

};

int main() {

Queue q;

// Fails because front = -1

q.deQueue();

q.enQueue();

q.enQueue();

q.enQueue();

q.enQueue();

q.enQueue();

// Fails to enqueue because front == 0 && rear == SIZE - 1

q.enQueue();

q.display();

int elem = q.deQueue();

if (elem != -1)

cout << endl

<< "Deleted Element is " << elem;

q.display();

q.enQueue();

q.display();

// Fails to enqueue because front == rear + 1

q.enQueue();

return 0;

}

**Exp 4:**

**#include <iostream>**

**using namespace std;**

**struct n {**

**char d;**

**n \*l;**

**n \*r;**

**};**

**char pf[50];**

**int top = -1;**

**n \*a[50];**

**int r(char inputch) {**

**if (inputch == '+' || inputch == '-' || inputch == '\*' || inputch== '/')**

**return (-1);**

**else if (inputch >= 'A' || inputch <= 'Z')**

**return (1);**

**else if (inputch >= 'a' || inputch <= 'z')**

**return (1);**

**else**

**return (-100);**

**}**

**void push(n \*tree) {**

**top++;**

**a[top] = tree;**

**}**

**n \*pop() {**

**top--;**

**return (a[top + 1]);**

**}**

**void construct\_expression\_tree(char \*suffix) {**

**char s;**

**n \*newl, \*p1, \*p2;**

**int flag;**

**s = suffix[0];**

**for (int i = 1; s != 0; i++) {**

**flag = r(s);**

**if (flag == 1) {**

**newl = new n;**

**newl->d = s;**

**newl->l = NULL;**

**newl->r = NULL;**

**push(newl);**

**} else {**

**p1 = pop();**

**p2 = pop();**

**newl = new n;**

**newl->d = s;**

**newl->l = p2;**

**newl->r = p1;**

**push(newl);**

**}**

**s = suffix[i];**

**}**

**}**

**void preOrder(n \*tree) {**

**if (tree != NULL) {**

**cout << tree->d;**

**preOrder(tree->l);**

**preOrder(tree->r);**

**}**

**}**

**void inOrder(n \*tree) {**

**if (tree != NULL) {**

**inOrder(tree->l);**

**cout << tree->d;**

**inOrder(tree->r);**

**}**

**}**

**void postOrder(n \*tree) {**

**if (tree != NULL) {**

**postOrder(tree->l);**

**postOrder(tree->r);**

**cout << tree->d;**

**}**

**}**

**int main(int argc, char \*\*argv) {**

**cout << "Enter Postfix Expression : ";**

**cin >> pf;**

**construct\_expression\_tree(pf);**

**cout << "In-Order Traversal : \n";**

**inOrder(a[0]);**

**cout << "\nPre-Order Traversal : \n";**

**preOrder(a[0]);**

**cout << "\nPost-Order Traversal : \n";**

**postOrder(a[0]);**

**return 0;**

**}**

**4 2 Source code:** #include <cstdlib> #include <iostream> using namespace std; typedef struct node

{

char data;

struct node\*left; struct node\*right;

} node;

typedef struct stacknode

{

node\* data;

struct stacknode\*next;

} stacknode;

class stack

{

stacknode \*top; public:

stack()

{

top = NULL;

}

node\*topp()

{

return top->data;

}

int isempty()

{

if (top == NULL)

{

return 1;

}

return 0;

}

void push(node\*a)

{

stacknode \*p;

p = new stacknode(); p->data = a;

p->next = top; top = p;

}

node\*pop()

{

stacknode \*p; node\* x;

x = top->data; p = top;

top = top->next; return x;

}

};

node\* create\_pre(char prefix[10]); node\* create\_post(char postfix[20]); void inorder\_non\_recursive(node\*t); void inorder(node\*p);

void preorder(node\*p); void postorder(node\*p);

void preorder\_non\_recursive(node\*t); void postorder\_non\_recursive(node\*t);

node\*create\_post(char postfix[10])

{

node \*p; stack s;

for (int i = 0; postfix[i] != '\0'; i++)

{

char token = postfix[i]; if (isalnum(token))

{

}

else

{

}

p = new node(); p->data = token; p->left = NULL;

p->right = NULL; s.push(p);

p = new node(); p->data = token;

p->right = s.pop(); p->left = s.pop();

s.push(p);

}

return s.pop();

}

node\* create\_pre(char prefix[10])

{

node \*p; stack s; int i;

for (i = 0; prefix[i] != '\0'; i++) {

}

i = i - 1;

for (; i>0; i--)

{

char token = prefix[i]; if (isalnum(token))

{

p = new node(); p->data = token; p->left = NULL;

p->right = NULL; s.push(p);

}

else

{

p = new node();

p->data = token; p->left = s.pop();

p->right = s.pop(); s.push(p);

}

}

return s.pop();

}

void inorder(node \*p)

{

if (p != NULL)

{

inorder(p->left); cout<<p->data; inorder(p->right);

}

}

void preorder(node \*p)

{

if (p != NULL)

{

cout<<p->data; preorder(p->left); preorder(p->right);

}

}

void postorder(node \*p)

{

if (p != NULL)

{

postorder(p->left); postorder(p->right); cout<<p->data;

}

}

void inorder\_non\_recursive(node \*t)

{

stack s;

while (t != NULL)

{

s.push(t); t = t->left;

}

while (s.isempty() != 1)

{

t = s.pop(); cout<<t->data; t = t->right;

while (t != NULL)

{

s.push(t); t = t->left;

}

}

}

void preorder\_non\_recursive(node \*t)

{

stack s;

while (t != NULL)

{

cout<<t->data; s.push(t);

t = t->left;

}

while (s.isempty() != 1)

{

t = s.pop(); t = t->right;

while (t != NULL)

{

cout<<t->data; s.push(t);

t = t->left;

}

}

}

void postorder\_non\_recursive(node \*t)

{

stack s, s1; node \*t1;

while (t != NULL)

{

s.push(t); s1.push(NULL); t = t->left;

}

while (s.isempty() != 1)

{

t = s.pop();

t1 = s1.pop();

if (t1 == NULL)

{

s.push(t); s1.push((node \*) 1); t = t->right;

while (t != NULL)

{

s.push(t); s1.push(NULL); t = t->left;

}

}

else

{

cout<<t->data;

}

}

}

int main()

{

node \*r = NULL, \*r1;

char postfix[10], prefix[10]; int x;

int ch, choice; do

{

cout<<" \tTREE OPERATIONS"<<endl;

cout<<"1) Construct tree from postfix/prefix expression "<<endl; cout<<"2) Inorder Traversal of Tree"<<endl;

cout<<"3) Preorder traversal"<<endl; cout<<"4) Postorder traversal"<<endl; cout<<"5) EXIT"<<endl; cout<<"Enter your choice : "; cin>>ch;

cout<<endl; switch (ch)

{

case 1:

cout<<"1) Postfix expression"<<endl; cout<<"2) Prefix expression"<<endl; cout<<"ENTER YOUR CHOICE :";

cin>>choice; cout<<endl;

if (choice == 1)

{

cout<<"Enter postfix expression : "<<endl; cin>>postfix;

r = create\_post(postfix);

}

else

{

cout<<"Enter prefix expression : "<<endl; cin>>prefix;

r = create\_pre(prefix);

}

cout<<endl;

cout<<"Tree created successfully"<<endl; cout<<endl;

break; case 2:

cout<<"Inorder Traversal of Tree with recursion : "; inorder(r);

cout<<endl;

cout<<"Inorder Traversal of Tree without recursion : "; inorder\_non\_recursive(r);

cout<<endl; cout<<endl; break;

case 3:

cout<<"Preorder traversal with recursion of tree "; preorder(r);

cout<<endl;

cout<<"Preorder traversal without recursion : "; preorder\_non\_recursive(r);

cout<<endl; cout<<endl; break;

case 4:

cout<<"Postorder traversal with recursion of tree : "; postorder(r);

cout<<endl;

cout<<"Postorder traversal without recursion of tree : "; postorder\_non\_recursive(r);

cout<<endl; cout<<endl; break;

}

}

while (ch != 5); return 0;

}

Ouput:

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 1

1. Postfix expression
2. Prefix expression ENTER YOUR CHOICE :1 Enter postfix expression : abc\*+

Tree created successfully TREE OPERATIONS

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 2

Inorder Traversal of Tree with recursion : a+b\*c

Inorder Traversal of Tree without recursion : a+b\*c

TREE OPERATIONS

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 3

Preorder traversal with recursion of tree +a\*bc

Preorder traversal without recursion : +a\*bc

TREE OPERATIONS

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 4

Postorder traversal with recursion of tree : abc\*+

Postorder traversal without recursion of tree : abc\*+

TREE OPERATIONS

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 1

1. Postfix expression
2. Prefix expression ENTER YOUR CHOICE :2 Enter prefix expression :

\*-xy

Tree created successfully TREE OPERATIONS

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 2

Inorder Traversal of Tree with recursion : x-y

Inorder Traversal of Tree without recursion : x-y

TREE OPERATIONS

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 3

Preorder traversal with recursion of tree -xy

Preorder traversal without recursion : -xy

TREE OPERATIONS

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 4

Postorder traversal with recursion of tree : xy-

Postorder traversal without recursion of tree : xy-

TREE OPERATIONS

1. Construct tree from postfix/prefix expression
2. Inorder Traversal of Tree
3. Preorder traversal
4. Postorder traversal
5. EXIT

Enter your choice : 5

Exp: 5

#include<iostream> using namespace std; class BST

{

class Node

{ public:

int Data;

Node \*Left,\*Right; Node(){ Left=Right=NULL; }

} \*Root; public:

BST(){Root=NULL;}

int isEmpty(){ return Root==NULL;} void Create();

void Insert(int); //Binary Search Tree Creation int Search(int);//Search

void inOrder(Node\*);//Display

void preOrder(Node\*);//Dsiplay Leaf Nodes int Depth();//BFS

void mirror(); //Display mirror image

void Levelwise();//Breath First Search Levelwise Display void Delete(int); //Delete Node

Node\* return\_Root(){ return Root;}

};

//create tree

void BST::Create()

{

int Num; char ans; do

{

cout<<"Enter Number"; cin>>Num;

Insert(Num); Levelwise();

cout<<"\nAdd More...(Y/N)"; cin>>ans;

}while(ans=='y' || ans=='Y');

}

//Binary Search Tree Creation void BST::Insert(int Num)

{

Node \*nNode,\*Temp; nNode=new Node; nNode->Data=Num; if(Root==NULL)

Root=nNode; else

{

Temp=Root; while(1)

{

if(nNode->Data>Temp->Data)

{if(Temp->Right==NULL)

{Temp->Right=nNode;break;} else Temp=Temp->Right;

}

else if(nNode->Data<Temp->Data)

{

if(Temp->Left==NULL)

{Temp->Left=nNode;break;} else Temp=Temp->Left;

}

else { cout<<"\nNumber already present";break;}

}//while

}

}

//Display

void BST::inOrder(Node \*Temp)

{

if(Temp!=NULL)

{

inOrder(Temp->Left); cout<<" "<<Temp->Data; inOrder(Temp->Right);

}

}

int BST::Search(int Num)

{

Node \*Tree=Root; while(Tree!=NULL)

{

if(Num>Tree->Data) Tree=Tree->Right;

else if(Num<Tree->Data) Tree=Tree->Left;

else return 1;

}

return 0;

}

//Display Leaf Nodes

void BST::preOrder(Node \*Temp)

{

if(Temp!=NULL)

{

if(Temp->Left==NULL && Temp->Right==NULL)

cout<<" "<<Temp->Data; preOrder(Temp->Left); preOrder(Temp->Right);

}

}

//Find Depth of Tree int BST ::Depth()

{ Node \*Queue[20],\*Temp=Root; int Front=0,Rear=1,Level=0;

if(isEmpty()) return 0; else {

do {

Queue[0]=Temp; Queue[1]=NULL;

Temp=Queue[Front++];

if(Temp==NULL)

{ Queue[++Rear]=NULL;

++Level; }

else

{

if(Temp->Left!=NULL) Queue[++Rear]=Temp->Left;

if(Temp->Right!=NULL) Queue[++Rear]=Temp->Right;

}

}while(Front<Rear);

}

return Level+1;

}

//Display Mirror image void BST::mirror()

{

Node \*Queue[20],\*Temp=Root; int Front=0,Rear=1;

if(isEmpty())cout<<"\nEmpty Tree"; else

{ Queue[0]=Temp; Queue[1]=NULL;

do {

Temp=Queue[Front++];

if(Temp==NULL)

{ Queue[++Rear]=NULL; cout<<"\n";

else{cout<<" "<<Temp->Data; if(Temp->Right!=NULL)

Queue[++Rear]=Temp->Right;

if(Temp->Left!=NULL) Queue[++Rear]=Temp->Left;

}

}while(Front<Rear);

}

}

//Display Levelwise void BST ::Levelwise()

{ Node \*Queue[20],\*Temp=Root;

int Front=0,Rear=1;

if(isEmpty())cout<<"\nEmpty Tree"; else {

do {

Queue[0]=Temp; Queue[1]=NULL;

Temp=Queue[Front++];

if(Temp==NULL)

{ Queue[++Rear]=NULL; cout<<"\n";

else{cout<<" "<<Temp->Data; if(Temp->Left!=NULL)

Queue[++Rear]=Temp->Left; if(Temp->Right!=NULL)

Queue[++Rear]=Temp->Right;

}

}while(Front<Rear);

}

}

//Delete Node

void BST::Delete(int Num)

{

Node \*Stack[10],\*pTemp=Root,\*Temp=Root,\*RTemp,\*pRTemp; char Flag;

while(Temp->Data!=Num &&Temp!=NULL)

{

if(Num>Temp->Data)

{ pTemp=Temp; Flag='R'; Temp=Temp->Right;

}

else

{

pTemp=Temp; Flag='L'; Temp=Temp->Left;

}

}//while if(Temp!=NULL)

{

if(Temp->Left==NULL && Temp->Right==NULL)//leaf node

{

if(Root==Temp) Root=NULL; if(Flag=='R') pTemp->Right=NULL; else pTemp->Left=NULL;

delete Temp;

}

else // delete Node with Left and Right children if(Temp->Left!=NULL && Temp->Right!=NULL)

{

pRTemp=RTemp=Temp->Right; while(RTemp->Left!=NULL)

{

pRTemp=RTemp; RTemp=RTemp->Left; }

if(pRTemp!=RTemp)

pRTemp=RTemp; RTemp=RTemp->Left;

pRTemp->Left=RTemp->Right; else Temp->Right= RTemp->Right; Temp->Data=RTemp->Data; delete RTemp;

} //if else

{

if(Temp->Right!=NULL)//with Right child

{

if(Root==Temp) Root=Root->Right;

else if(Flag=='L') pTemp->Left=Temp->Right; else pTemp->Right=Temp->Right;

else if(Temp->Left!=NULL) //with Left child

{

if(Root==Temp) Root=Root->Left;

else if(Flag=='L') pTemp->Left=Temp->Left; else pTemp->Right=Temp->Left;

}

delete Temp;

}

}

else

cout<<"\nNode is not present";

}//Function end int main()

{ BST B;

int D,ch,Num; cout<<"\nCreate Tree: \n"; B.Create();

do

{

cout<<"\n 1: Insert 2: Display Tree 3:Search 4:Display Leaf Nodes";

cout<<"\n5:Depth of Tree 6: Display Mirror image 7 :Display Levelwise 8: Delete Node 9: Exit"; cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter Number";

cin>>Num; B.Insert(Num); B.Levelwise();

break; case 2:

cout<<"\n Inorder Display"; B.inOrder(B.return\_Root()); break;

case 3: cout<<"\nEnter Data to search"; cin>>D;

if(B.Search(D)==1) cout<<"\n Node Present";

else cout<<"\nNode Not Present "; break;

case 4:

cout<<"\nLeaf Nodes"; B.preOrder(B.return\_Root()); break;

case 5:

D=B.Depth(); if(D==0)cout<<"\nEmpty Tree"; else cout<<"\n Depth of Tree "<<D; break;

case 6:

cout<<"\nMirror\n"; B.mirror();

break; case 7:

cout<<"\n Levelwise Display\n"; B.Levelwise();

break; case 8:

cout<<"\nEnter Data to Delete"; cin>>D;

B.Delete(D);

cout<<"\n Inorder Display"; B.inOrder(B.return\_Root()); break;

}

}while(ch<9);

}

Output: Create Tree:

Enter Number5 5

Add More...(Y/N)y Enter Number2

5

2

Add More...(Y/N)y Enter Number2

Number already present 5 2

Add More...(Y/N)y Enter Number4

5

2

4

Add More...(Y/N)y Enter Number6

5

2 6

4

Add More...(Y/N)y Enter Number1

5

2 6

1 4

Add More...(Y/N)n

1: Insert 2: Display Tree 3:Search 4:Display Leaf Nodes

5:Depth of Tree 6: Display Mirror image 7 :Display Levelwise 8: Delete Node 9: Exit2

Inorder Display 1 2 4 5 6

1: Insert 2: Display Tree 3:Search 4:Display Leaf Nodes

5:Depth of Tree 6: Display Mirror image 7 :Display Levelwise 8: Delete Node 9: Exit3 Enter Data to search6

Node Present

1: Insert 2: Display Tree 3:Search 4:Display Leaf Nodes

5:Depth of Tree 6: Display Mirror image 7 :Display Levelwise 8: Delete Node 9: Exit4

Leaf Nodes 1 4 6

1: Insert 2: Display Tree 3:Search 4:Display Leaf Nodes

5:Depth of Tree 6: Display Mirror image 7 :Display Levelwise 8: Delete Node 9: Exit5

Depth of Tree 3

1: Insert 2: Display Tree 3:Search 4:Display Leaf Nodes

5:Depth of Tree 6: Display Mirror image 7 :Display Levelwise 8: Delete Node 9: Exit6

Mirror 5

6 2

4 1

1: Insert 2: Display Tree 3:Search 4:Display Leaf Nodes

5:Depth of Tree 6: Display Mirror image 7 :Display Levelwise 8: Delete Node 9: Exit9

**Exp 6: threded binary tree**

#include<iostream> using namespace std; class Ttree

{ public:

char Data;

int LFlag,RFlag; Ttree \*Left,\*Right;

Ttree(char c='\0'){Data=c;LFlag=RFlag=1;Left=Right=NULL;}

};

class Threaded\_Tree

{

Ttree \*Head; public:

Threaded\_Tree(){Head=new Ttree; Head->Right=Head;} void Create(char[]);

void PreTrav(); void InTrav();

};

void Threaded\_Tree ::Create(char Estr[25])

{

Ttree \*Stk[20], \*Temp; int Top=-1, I=0; while(Estr[I]!='\0')

{

Ttree \*Node=new Ttree(Estr[I]); Node->Left=Node->Right=Head; if(isalnum(Estr[I]));

else

{

Node->Right=Temp=Stk[Top--]; while(Temp->Left!=Head)

Temp=Temp->Left;

Temp->Left=Node; Node->RFlag=0;

Node->Left=Temp=Stk[Top--]; while(Temp->Right!=Head)

Temp=Temp->Right; Temp->Right=Node; Node->LFlag=0;

}

Stk[++Top]=Node; I++;

}

Head->Left=Stk[Top--]; }

void Threaded\_Tree::InTrav()

{

Ttree \*Temp=Head->Left; do

{

while(!Temp->LFlag) Temp=Temp->Left; cout<<Temp->Data; Temp=Temp->Right; cout<<Temp->Data; Temp=Temp->Right;

}while(Temp!=Head);

}

void Threaded\_Tree::PreTrav()

{

Ttree \*Temp=Head->Left; do

{

while(!Temp->LFlag)

{

cout<<Temp->Data; Temp=Temp->Left;

}

cout<<Temp->Data; Temp=Temp->Right; Temp=Temp->Right;

}while(Temp!=Head);

}

int main()

{

Threaded\_Tree B; char Estr[25];

cout<<"Enter Postfix Expression"; cin>>Estr;

B.Create(Estr);

cout<<" \n \n "; cout<<" \nPreorder : \n";

B.PreTrav();

cout<<" \n \n "; cout<<" \n Inorder : \n";

B.InTrav();

cout<<" \n \n ";

}

Output:

Enter Postfix Expression31+

- Preorder :

+31

-

Inorder :

3+1

**7 kruskal Input:**

**#include<iostream>**

**#include<string.h>**

**using namespace std;**

**class Graph**

**{**

**char Vnames[10][10];**

**int cost[10][10],no;**

**public:**

**Graph();**

**void creat\_graph();**

**void display();**

**int Position(char[]);**

**void kru();**

**void prims();**

**};**

**Graph::Graph()**

**{**

**no=0;**

**for(int i=0;i<10;i++)**

**for(int j=0;j<10;j++)**

**{**

**if(i==j)**

**cost[i][j]=0;**

**else**

**cost[i][j]=999;**

**}**

**}**

**void Graph::creat\_graph()**

**{**

**char ans,Start[10],End[10];**

**int wt,i,j;**

**cout<<"Enter number of nodes:";**

**cin>>no;**

**cout<<"\n Enter vertex name:";**

**for(i=0;i<no;i++)**

**cin>>Vnames[i];**

**do**

**{**

**cout<<"\nEnter Start and end point of edge:";**

**cin>>Start>>End;**

**cout<<"Enter weight:";**

**cin>>wt;**

**i=Position(Start);**

**j=Position(End);**

**cost[j][i]=cost[i][j]=wt;**

**cout<<"\nMore Edges: ";**

**cin>>ans;**

**}while(ans=='y' || ans=='Y');**

**}**

**void Graph::display()**

**{**

**int i,j;**

**cout<<"\nAdjecancy Matrix\n\t";**

**for(i=0;i<no;i++)**

**cout<<"\t"<<Vnames[i];**

**for(i=0;i<no;i++)**

**{**

**cout<<"\n\t"<<Vnames[i];**

**for(j=0;j<no;j++)**

**cout<<"\t"<<cost[i][j];**

**}**

**}**

**int Graph::Position(char S[10])**

**{**

**int i;**

**for(i=0;i<10;i++)**

**if(strcmp(Vnames[i],S)==0)**

**break;**

**return i;**

**}**

**void Graph::kru()**

**{**

**int i,j,v[10],x,y,min\_cost=0,min,gr=1,flag=0,temp,d;**

**for(i=0;i<no;i++)**

**v[i]=0;**

**cout<<"\n node1\tnode2\tweight";**

**while(flag==0)**

**{**

**min=999;**

**for(i=0;i<no;i++)**

**{ for(j=0;j<no;j++)**

**{ if(i!=j && cost[i][j]<min)**

**{**

**min=cost[i][j];**

**x=i;**

**y=j;**

**}**

**}**

**}**

**if(v[x]==0 && v[y]==0)**

**{ v[x]=gr;v[y]=gr; gr++; }**

**else if(v[x]==0 && v[y]!=0)**

**v[x]=v[y];**

**else if(v[y]==0 && v[x]!=0)**

**v[y]=v[x];**

**else if(v[y]!=0 && v[x]!=0)**

**{**

**d=v[y];**

**for(i=0;i<no;i++)**

**if(v[i]==d)**

**v[i]=v[x];**

**}**

**cost[x][y]=cost[y][x]=999;**

**cout<<"\n"<<Vnames[x]<<"\t"<<Vnames[y]<<"==>\t"<<min;**

**min\_cost+=min;**

**temp=v[0]; flag=1;**

**for(i=0;i<no;i++)**

**{**

**if(temp!=v[i])**

**{ flag=0; break;}**

**}**

**}**

**cout<<"\nminimum path is of value "<<min\_cost;**

**}**

**void Graph::prims()**

**{**

**int c=1,b,i,j,x,y,min\_cost=0,min,v[10]={0};**

**char start[10]="\0";**

**cout<<"\nfrom which City you want to start:";**

**cin>>start;**

**b=Position(start);**

**v[b]=1;**

**cout<<"\n City1\tCity2\tDistance";**

**while(c<no)**

**{**

**min=999;**

**for(i=0;i<no;i++)**

**{**

**if(v[i])**

**{**

**for(j=0;j<no;j++)**

**{**

**if(cost[i][j]<min && v[j]==0)**

**{**

**min=cost[i][j];**

**x=i;y=j;**

**}**

**}**

**}**

**}**

**cout<<"\n"<< Vnames [x]<<"\t"<< Vnames [y]<<"\t"<<min;**

**min\_cost+=min;**

**cost[x][y]=cost[y][x]=999;**

**v[y]=1;**

**c++;**

**}**

**cout<<"\nMinimum Total cost"<<min\_cost;**

**}**

**main()**

**{**

**Graph G,G1;**

**G1.creat\_graph();**

**G1.display();**

**G1.prims();**

**G.creat\_graph();**

**G.display();**

**G.kru();**

**}**

**Output:**

**Exp 8 : shortest path**

**#include <iostream>**

**#include<string.h>**

**#include<iomanip>**

**using namespace std;**

**class graph**

**{**

**char Vnames[10][10];**

**int nodes,cost[10][10];**

**public:**

**graph();**

**int Position(char S[10]);**

**void creat\_graph();**

**void display();**

**void Dijkstra();**

**};**

**graph::graph()**

**{**

**nodes=0;**

**for(int i=0;i<10;i++)**

**for(int j=0;j<10;j++)**

**{**

**if(i==j)**

**cost[i][j]=0;**

**else**

**cost[i][j]=999;**

**}**

**}**

**void graph::creat\_graph()**

**{**

**char ans,Start[10],End[10];**

**int wt,i,j;**

**cout<<"Enter number of nodes";**

**cin>>nodes;**

**cout<<"\n Enter vertex name:";**

**for(i=0;i<nodes;i++)**

**{ cin>>Vnames[i];**

**}**

**do**

**{**

**cout<<"\nEnter Start and end point of edge";**

**cin>>Start>>End;**

**cout<<"Enter weight";**

**cin>>wt;**

**i=Position(Start);**

**j=Position(End);**

**cost[j][i]=cost[i][j]=wt;**

**cout<<"\nMore Edges ";**

**cin>>ans;**

**}while(ans=='y' || ans=='Y');**

**}**

**void graph::display()**

**{**

**int i,j;**

**cout<<"\nAdjecancy Matrix\n\t";**

**for(i=0;i<nodes;i++)**

**cout<<"\t"<<Vnames[i];**

**for(i=0;i<nodes;i++)**

**{**

**cout<<"\n\t"<<Vnames[i];**

**for(j=0;j<nodes;j++)**

**cout<<"\t"<<cost[i][j];**

**}**

**}**

**int graph::Position(char S[10])**

**{**

**int i;**

**for(i=0;i<10;i++)**

**if(strcmp(Vnames[i],S)==0)**

**break;**

**return i;**

**}**

**void graph::Dijkstra()**

**{**

**int x,dis[10],visit[10],flag[10]={0};**

**int i,j,v,sor,min,mnode,k;**

**char Start[10];**

**cout<<"\nSingle Source & multiple destinations Algo";**

**cout<<"\nEnter Source: ";**

**cin>>Start;**

**sor=Position(Start);**

**flag[sor]=1; //init.**

**dis[sor]=0;**

**for(v=0;v<nodes;v++) // initial distance matrix**

**{ dis[v]=cost[sor][v];}**

**visit[0]=sor;**

**cout<<"\nShortest path matrics\nNode\t\t Weight\n\t ";**

**for(i=0;i<nodes;cout<<setw(3)<<i,i++);**

**i=0;**

**cout<<"\n";**

**for(x=0;x<=i;x++)**

**cout<<setw(3)<<visit[x];**

**for(x=i+1;x<nodes;x++)**

**cout<<" -";**

**cout<<" : ";**

**for(x=0;x<nodes;x++)**

**cout<<setw(3)<<dis[x];**

**/\*main loop \*/**

**for(i=1;i<nodes;i++)**

**{ min=999;**

**for(k=0;k<nodes;k++)**

**{**

**if(flag[k]==0 && dis[k] < min) //find min. distance**

**{min=dis[k];mnode=k;}**

**}**

**flag[mnode]=1;**

**visit[i]=mnode;**

**for(j=0;j<nodes;j++)**

**{ if(flag[j]==0 && cost[mnode][j]!=999)**

**{ if(dis[j]>dis[mnode]+cost[mnode][j])**

**dis[j]=dis[mnode]+cost[mnode][j];**

**}**

**}**

**cout<<"\n";**

**for(x=0;x<=i;x++)**

**cout<<setw(3)<<visit[x];**

**for(x=i+1;x<nodes;x++)**

**cout<<" -";**

**cout<<" : ";**

**for(x=0;x<nodes;x++)**

**cout<<setw(3)<<dis[x];**

**}**

**}**

**int main()**

**{**

**graph ShortestPath;**

**ShortestPath.creat\_graph();**

**ShortestPath.display();**

**ShortestPath.Dijkstra();**

**return 0;**

**}**

**Output:**

**Enter number of nodes 4**

**Enter vertex name: A B C D**

**Enter Start and end point of edge A B**

**Enter weight12**

**More Edges y**

**Enter Start and end point of edgeB C**

**Enter weight35**

**More Edges y**

**Enter Start and end point of edgeC D**

**Enter weight11**

**More Edges y**

**Enter Start and end point of edgeA D**

**Enter weight67**

**More Edges n**

**Adjecancy Matrix**

**A B C D**

**A 0 12 999 67**

**B 12 0 35 999**

**C 999 35 0 11**

**D 67 999 11 0**

**Single Source & multiple destinations Algo**

**Enter Source: D**

**Shortest path matrics**

**Node Weight**

**0 1 2 3**

**3 - - - : 67999 11 0**

**3 2 - - : 67 46 11 0**

**3 2 1 - : 58 46 11 0**

**3 2 1 0 : 58 46 11 0**

**Exp 9: SOURCE CODE: heap sort**

**#include <iostream>**

**using namespace std;**

**class Heap**

**{**

**int H[20];**

**public:**

**Heap(){H[0]=0;}**

**void insert(int);**

**void Sort();**

**};**

**void Heap::insert(int Num)**

**{**

**int i,T;**

**i=++H[0];**

**H[i]=Num;**

**while(i/2>0)**

**{ if(H[i] > H[i/2])**

**{ T=H[i];H[i]=H[i/2];H[i/2]=T;}**

**i=i/2;**

**}**

**cout<<endl<<H[0]<<" :";**

**for(i=1;i<=H[0];i++) cout<<" "<<H[i];**

**}**

**void Heap::Sort()**

**{ cout<<"\nSort";**

**int i=H[0],j,k,T;**

**while(i>1)**

**{**

**T=H[1];H[1]=H[i];H[i]=T;**

**i--;**

**k=j=1;**

**while(j<=k)**

**{ //cout<<"\nL"<<H[j\*2]<<" R"<<H[j\*2+1];**

**if((j\*2+1)<=i)**

**{ if(H[j\*2]>H[j\*2+1]) k=j\*2;//cout<<"if";}**

**else k=j\*2+1;}//cout<<"else";}}**

**else if((j\*2)<=i) k=j\*2 ;//cout<<"else2";}**

**//cout<<"\nj:"<<j;**

**if(j!=k && H[j]<H[k])**

**{ T=H[k];H[k]=H[j];H[j]=T;**

**j=k; }**

**else break;**

**cout<<endl<<H[0]<<" :";**

**for(int l=1;l<=i;l++) cout<<" "<<H[l];**

**}**

**}**

**cout<<"\nSorted Data\n";**

**cout<<endl<<H[0]<<" :";**

**for(i=1;i<=H[0];i++) cout<<" "<<H[i];**

**}**

**int main()**

**{**

**Heap Hp;**

**int no,Num;**

**cout<<"\n How many numbers you want to insert";**

**cin>>no;**

**for(int i=0;i<no;i++)**

**{**

**cout<<"\nEnter Number:";**

**cin>>Num;**

**Hp.insert(Num);**

**}**

**Hp.Sort();**

**return 1;**

**}**

**Output:**

**How many numbers you want to insert 5**

**Enter Number:23**

**1 : 23**

**Enter Number:112**

**2 : 112 23**

**Enter Number:34**

**3 : 112 23 34**

**Enter Number:123**

**4 : 123 112 34 23**

**Enter Number:54**

**5 : 123 112 34 23 54**

**Sort**

**5 : 112 54 34 23**

**5 : 54 23 34**

**Sorted Data**

**5 : 23 34 54 112 123**

**Exp 10: file handling**

**#include <iostream>**

**#include<fstream>**

**using namespace std;**

**class student**

**{ int roll,div;**

**char name[10],address[10];**

**public:**

**student(){ name[0]=address[0]='\0';div=roll=-1;}**

**void getdata(); void putdata();**

**int return\_Roll(){ return roll;} };**

**void student::getdata()**

**{ cout<<"\nEnter Student Data";**

**cout<<"\nName: ";cin>>name;**

**cout<<"\nAddress: ";cin>>address;**

**cout<<"\nRoll Number: ";cin>>roll;**

**cout<<"\nDiv(1/2): ";cin>>div; }**

**void student::putdata()**

**{**

**cout<<"\n"<<div<<"\t"<<roll<<"\t"<<name<<"\t"<<address;**

**}**

**class seq**

**{**

**char fname[10];**

**public:**

**void create();**

**void display();**

**void Add();**

**void Remove(int);**

**void Modify(int);**

**void search(int);**

**};**

**void seq::create()**

**{**

**ofstream fp;**

**student s;**

**cout<<"Enter file name=";**

**cin>>fname;**

**fp.open(fname);**

**s.getdata();**

**fp.write(reinterpret\_cast<char\*>(&s),sizeof(s));**

**fp.close();**

**}**

**void seq::display()**

**{**

**ifstream fp;**

**student s;**

**fp.open(fname);**

**fp.read(reinterpret\_cast<char\*>(&s),sizeof(s));**

**while(!fp.eof())**

**{**

**s.putdata();**

**fp.read(reinterpret\_cast<char\*>(&s),sizeof(s));**

**}**

**fp.close();**

**}**

**void seq::Add()**

**{**

**ofstream fp;**

**student s;**

**fp.open(fname,ios::app);**

**s.getdata();**

**fp.write(reinterpret\_cast<char\*>(&s),sizeof(s));**

**fp.close();**

**}**

**void seq::Remove(int key)**

**{ ifstream f1;**

**ofstream f2;**

**student s; int Flag=0;**

**f1.open(fname);**

**f2.open("Temp.Txt");**

**f1.read(reinterpret\_cast<char\*>(&s),sizeof(s));**

**while(!f1.eof())**

**{**

**if(key==s.return\_Roll())**

**Flag=1;**

**else**

**f2.write(reinterpret\_cast<char\*>(&s),sizeof(s));**

**f1.read(reinterpret\_cast<char\*>(&s),sizeof(s));**

**}**

**f1.close(); f2.close();**

**remove(fname);**

**rename("Temp.Txt",fname);**

**if(Flag==0) cout<<"Record does not present ";**

**else cout<<"Record deleted successfully";**

**}**

**void seq::Modify(int key)**

**{**

**ifstream f1;**

**ofstream f2; student s;**

**f1.open(fname);**

**f2.open("Temp.Txt");**

**f1.read(reinterpret\_cast<char\*>(&s),sizeof(s));**

**while(!f1.eof())**

**{**

**if(key==s.return\_Roll())**

**{**

**cout<<"Enter data to modify";**

**s.getdata();**

**}**

**f2.write(reinterpret\_cast<char\*>(&s),sizeof(s));**

**f1.read(reinterpret\_cast<char\*>(&s),sizeof(s));**

**}**

**f1.close(); f2.close();**

**remove(fname);**

**rename("Temp.Txt",fname);**

**}**

**void seq::search(int key)**

**{**

**ifstream fp;**

**student s; int flag=0;**

**fp.open(fname);**

**fp.read(reinterpret\_cast<char\*>(&s),sizeof(s));**

**while(!fp.eof())**

**{**

**if(key==s.return\_Roll())**

**{ flag=1;**

**s.putdata();**

**break;**

**}**

**fp.read(reinterpret\_cast<char\*>(&s),sizeof(s));**

**}**

**if(flag==0) cout<<"Record does not present ";**

**fp.close();**

**}**

**int main()**

**{**

**seq ob; int ch,key;**

**do**

**{**

**cout<<"1: Create Database\n2: Display\n3: Add a record\n4: Delete \n5: Modify ";**

**cout<<"Enter your choice: "; cin>>ch;**

**switch(ch)**

**{**

**case 1:**

**ob.create(); break;**

**case 2:**

**ob.display(); break;**

**case 3:**

**ob.Add(); break;**

**case 4:**

**cout<<"\nEnter Roll No to delete"; cin>>key;**

**ob.Remove(key);**

**break;**

**case 5:**

**cout<<"\nEnter Roll No to Modify"; cin>> key;**

**ob.Modify(key);**

**break;**

**}**

**}while(ch<6);**

**}**

**OUTPUT:**

**/tmp/gG3oT6zayL.o**

**1: Create Database**

**2: Display**

**3: Add a record**

**4: Delete**

**5: Modify Enter your choice: 1**

**Enter file name=SCOE**

**Enter Student Data**

**Name: M**

**Address: PUNE**

**Roll Number: 02**

**Div(1/2): 2**

**1: Create Database**

**2: Display**

**3: Add a record**

**4: Delete**

**5: Modify Enter your choice: 3**

**Enter Student Data**

**Name: MANAV**

**Address: PUNE**

**Roll Number: 04**

**Div(1/2): 2**

**1: Create Database**

**2: Display**

**3: Add a record**

**4: Delete**

**5: Modify Enter your choice: 2**