**Exp 1 searching and sortng**

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

**using namespace std;**

**struct name**

**{**

**char firstname[10];**

**char lastname[12];**

**};**

**struct profile**

**{**

**name student\_name;**

**float SGPA;**

**int Roll\_no;**

**};**

**class student\_data**

**{**

**private :**

**struct profile std\_db[15]={"Abhishek", "Jadhav", 9.89, 23232, "Abhijeet", "Ingle", 9.8, 23231, "Ronal", "Dhodia", 9.54, 23222, "Pratik", "Gabhale", 9.12, 23224, "Ayush", "Gala", 9.65, 23228, "Kunal", "Kachare", 9.5, 23235, "Mihir", "Inamdar", 9.01, 23230, "Sanika", "Joshi", 9.2, 23234, "Atharva", "Kinikar", 9.8, 23241, "Mrunal", "Kashid", 9.45, 23238, "Kalyani", "Nilpankar", 8.97, 23237, "Tanmay", "Goregaonkar", 8.85, 23227, "Shruti", "Jagdale", 9.74, 23233, "Sandesh", "Dholas", 8.86, 23223, "Akshay", "K", 8.98, 23242};**

**int n = 15;**

**public :**

**void view(int n)**

**{**

**cout<<"\n\n Student        Name                 SGPA        Roll\n   No.                                           No.\n--------------------------------------------------------------\n";**

**for (int i = 0; i <= n-1; i++)**

**{**

**cout<<"    "<<i+1<<".        "<<std\_db[i].student\_name.firstname<<" "<<std\_db[i].student\_name.lastname;**

**cout<<"   ->>-  "<<std\_db[i].SGPA<<"   ->>-  "<<std\_db[i].Roll\_no<<"\n\n";**

**}**

**}**

**void add()**

**{**

**cout<<"\n Enter no. of Student details to be added (max 15) : ";**

**int n;**

**cin>>n;**

**cout<<"\n ------------ No. of Students to added : "<<n<<" -----------\n      Filling data such as Name, SGPA, Roll no.\n ------------------------------------------------------";**

**for ( int i = 2; i < 2+n; i++)**

**{**

**cout<<"\n\n# Details of Student no. "<<i+1<<" - \n\n                             FirstName Surname\n  > Enter Name of Student : ";**

**cin>>std\_db[i].student\_name.firstname>>std\_db[i].student\_name.lastname;**

**cout<<"\n  > Enter Student SGPA : ";**

**cin>>std\_db[i].SGPA;**

**cout<<"\n  > Enter Student Roll no. : ";**

**cin>>std\_db[i].Roll\_no;**

**cout<<" --------------------------------------------------------------";**

**}**

**cout<<"---------------- Addition Done Successfully ! ----------------";**

**student\_data :: view(15);**

**}**

**void search\_sgpa(float r)  //linear search on SGPA**

**{**

**cout<<"\n\n< Searching in SGPA using Linear Search...>\n-----------------------------------------------";**

**int t = 1;**

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

**{**

**if(std\_db[i].SGPA == r)**

**{**

**cout<<"\n\n#"<<t<<" Details of Student with SGPA "<<std\_db[i].SGPA<<" -";**

**cout<<"\n\n Student        Name                 SGPA        Roll\n    No.                                          No.\n--------------------------------------------------------------\n";**

**cout<<"    "<<i+1<<".        "<<std\_db[i].student\_name.firstname<<" "<<std\_db[i].student\_name.lastname;**

**cout<<"   ->>-  "<<std\_db[i].SGPA<<"   ->>-  "<<std\_db[i].Roll\_no;**

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

**t++;**

**continue;**

**}**

**}**

**if(t == 0)**

**{**

**cout<<"\n Student '"<<r<<"' not found or Input correct SGPA";**

**}**

**}**

**void modify()**

**{**

**cout<<"\n Enter Student no. which is to be modified : ";**

**int m;**

**cin>>m;**

**int m\_i = m - 1;**

**for (int i = 0; i < 15; i++)**

**{**

**if ( i == m\_i)**

**{**

**char stopModify = 'Y';**

**while (stopModify == 'Y')**

**{**

**cout<<"\n# Modify - \t1. Name\t\t2. SGPA\t\t3. Student Roll no.\n  Choice (1/2/3): ";**

**int choice;**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:**

**cout<<"                             FirstName Surname\n  > Enter Name of Student : ";**

**cin>>std\_db[i].student\_name.firstname>>std\_db[i].student\_name.lastname;**

**cout<<" -------- Changes Done Successfully ! --------";**

**student\_data :: view(15);**

**break;**

**case 2:**

**cout<<"\n  > Enter Student SGPA : ";**

**cin>>std\_db[i].SGPA;**

**cout<<" -------- Changes Done Successfully ! --------";**

**student\_data :: view(15);**

**break;**

**case 3:**

**cout<<"\n  > Enter Student Roll\_no : ";**

**cin>>std\_db[i].Roll\_no;**

**cout<<" -------- Changes Done Successfully ! --------";**

**student\_data :: view(15);**

**break;**

**default:**

**cout<<" Invalid choice !\n -------------------------------------------";**

**}**

**cout<<"\nDo you want to continue modification (Y/N) ? : ";**

**cin>>stopModify;**

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

**}**

**break;**

**}**

**else**

**{**

**cout<<"\n Student no. is out of range or Invalid input";**

**}**

**}**

**}**

**void sort\_names(int n)  //insertion sort for sorting names**

**{**

**for (int k=n-1; k>0; k--)**

**{**

**struct profile temp = std\_db[k];**

**int j = k-1;**

**while (j >= 0 && strcmp(temp.student\_name.firstname, std\_db[j].student\_name.firstname) < 0)  //compares both the strings character by character**

**{**

**std\_db[j+1] = std\_db[j];**

**j = j-1;**

**}**

**std\_db[j+1] = temp;**

**}**

**}**

**void sort\_SGPA(int l, int k)  //sorting SGPA using quick sorting**

**{**

**int r = k-1;**

**if (l>=r) return;**

**int i=l;**

**int j=r+1;**

**struct profile prec;**

**int p = std\_db[l].SGPA; //Select pivot element**

**prec = std\_db[l]; //temporarily storing pivot record prec**

**while(1)**

**{**

**do{ i++; } while (std\_db[i].SGPA < p && i <= r);**

**do{ j--; } while (std\_db[j].SGPA > p && j >= l);**

**if(i >= j) break;**

**struct profile temp;**

**temp = std\_db[j];**

**std\_db[j] = std\_db[i];**

**std\_db[i] = temp;**

**}**

**std\_db[l] = std\_db[j];**

**std\_db[j] = prec;**

**sort\_SGPA(l,j); //left list**

**sort\_SGPA(j+1,r); //right list**

**student\_data :: view(10);**

**}**

**void sort\_RollNo()  //sorting roll no. in ascending order using bubble sort**

**{**

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

**{**

**for (int j = 0; j < n-1; j++)**

**{**

**if((std\_db[j].Roll\_no) < (std\_db[j+1].Roll\_no)) //Swapping**

**{**

**struct profile temp;**

**temp = std\_db[j];**

**std\_db[j] = std\_db[j+1];**

**std\_db[j+1] = temp;**

**}**

**}**

**}**

**student\_data :: view(n);**

**}**

**void search\_name()**

**{**

**cout<<"\n Enter student name to be searched : ";**

**char search[10];**

**cin>>search;**

**cout<<"\n< Searching name using Binary Search...>";**

**int lower = 0, upper, mid;**

**upper = n - 1;**

**mid = (lower + upper)/2;**

**student\_data :: sort\_names(n);**

**while (lower <= upper)**

**{**

**if(strcmp(std\_db[mid].student\_name.firstname, search)<0)**

**{**

**lower = mid + 1;**

**}**

**else if(strcmp(std\_db[mid].student\_name.firstname, search)==0)**

**{**

**cout<<"\n\n# Details of Student with name "<<std\_db[mid].student\_name.firstname<<" -";**

**cout<<"\n\n Student        Name                 SGPA        Roll\n    No.                                           No.\n--------------------------------------------------------------\n";**

**cout<<"    1.        "<<std\_db[mid].student\_name.firstname<<" "<<std\_db[mid].student\_name.lastname;**

**cout<<"   ->>-  "<<std\_db[mid].SGPA<<"   ->>-  "<<std\_db[mid].Roll\_no;**

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

**break;**

**}**

**else**

**{**

**upper = mid - 1;**

**mid = (lower + upper)/2;**

**}**

**}**

**if(lower > upper)**

**{**

**cout<<"\n Student '"<<search<<"' details not found or Input correct name";**

**}**

**}**

**};**

**int main()**

**{**

**struct profile std\_db[15]={"Abhishek", "Jadhav", 9.89, 23232, "Abhijeet", "Ingle", 9.8, 23231, "Ronal", "Dhodia", 9.54, 23222, "Pratik", "Gabhale", 9.12, 23224, "Ayush", "Gala", 9.65, 23228, "Kunal", "Kachare", 9.5, 23235, "Mihir", "Inamdar", 9.01, 23230, "Sanika", "Joshi", 9.2, 23234, "Atharva", "Kinikar", 9.8, 23241, "Mrunal", "Kashid", 9.45, 23238, "Kalyani", "Nilpankar", 8.97, 23237, "Tanmay", "Goregaonkar", 8.85, 23227, "Shruti", "Jagdale", 9.74, 23233, "Sandesh", "Dholas", 8.86, 23223, "Akshay", "K", 8.98, 23242};**

**student\_data std;**

**cout<<"--------------------------------------------------------------------\n\t\t----- SE IT Student Database -----\n--------------------------------------------------------------------\n";**

**std.view(15);**

**char stopApp;**

**stopApp = 'Y';**

**while(stopApp == 'Y')**

**{**

**cout<<"\n\nSelect action from following : \n";**

**cout<<"> 1. VIEW RECORDS\t> 2. ADD RECORDS\t> 3. MODIFY RECORD\t> 4. SEARCH SGPA\t> 5. SORT NAME\t\t> 6. SORT SGPA(Toppers)\t\t> 7. SORT ROLL NO.\t\t> 8. SEARCH NAME\t> 9. EXIT\n\t\t\t\t\t\t\t\t\tEnter choice (1/2/3/4/5/6/7/8/9): ";**

**int choice;**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:**

**std.view(15);**

**break;**

**case 2:**

**std.add();**

**break;**

**case 3:**

**std.modify();**

**break;**

**case 4:**

**cout<<"\n Enter Student SGPA to be searched : ";**

**float r;**

**cin>>r;**

**std.search\_sgpa(r);**

**break;**

**case 5:**

**cout<<"\n< Sorting name alphabetically using Insertion Sort...>";**

**std.sort\_names(15);**

**std.view(15);**

**break;**

**case 6:**

**cout<<"\n< Sorting top 10 SGPA using Quick Sort...>";**

**std.sort\_SGPA(0, 10);**

**break;**

**case 7:**

**cout<<"\n< Sorting Roll No. wise using Bubble Sort...>";**

**std.sort\_RollNo();**

**break;**

**case 8:**

**std.search\_name();**

**break;**

**case 9:**

**cout<<" >Exited successful<\n --| END OF CODE |--";**

**return 0;**

**default :**

**cout<<"\n Invalid choice !";**

**}**

**cout<<"\n\nDo you want to continue (Y/N) ? : ";**

**cin>>stopApp;**

**if (stopApp == 'N' | 'n')**

**{**

**cout<<" ----------------| END OF CODE |----------------\n";**

**}**

**}**

**return 0;**

**}**

**b)**

#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\n 2:Infix to Postfix\n 3:Postfix Evaluation\n 4: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;

}

b)

#include<iostream>

#include<string>

using namespace std;

#define size 3

struct patient\_name

{

    string firstname;

    string lastname;

};

struct patient

{

    patient\_name name;

    int age;

    string disease;

};

class patientQ

{

    private:

        int front, rear;

        struct patient s[size];

    public:

        patientQ()

        {

            front = -1;

            rear = -1;

        };

        bool isQueueFull()

        {

            if ((rear == size-1) || ((front == rear+1)))

            {

                return true;

            }

            // else if (front == (rear + 1))

            // {

            //     return true;

            // }

            else

            {

                return false;

            }

        }

        void enqueue(string fname, string lname, string d, int ag)

        {

            // if (rear == front)

            // {

            //     cout<<"\n\n \_\_\_\_\_ Currently queue is Overflowed ! \_\_\_\_\_";

            //     return;

            // }

            if (front == -1)

            {

                front = 0;

                rear = 0;  //addition

            }

            int temp = (rear+1)%size;

            s[rear].name.firstname = fname;

            s[rear].name.lastname = lname;

            s[rear].disease = d;

            s[rear].age = ag;

            // rear = temp;

        }

        bool isQueueEmpty()

        {

            if (front == -1)

            {

                return true;

            }

            else if (front == -1 && rear == -1)

            {

                return true;

            }

            else

            {

                return false;

            }

        }

        string dequeue()

        {

            {

                struct patient garbage;

                garbage = s[(size - rear)%size];

                if (front == rear)

                {

                    front = -1;

                    rear = -1;

                }

                else

                {

                    front = (front + 1)%size;

                }

                return (garbage.name.firstname);

            }

        }

        void view(int sz)

        {

            cout<<"\n\n Patient        Name              Age        Disease\n   No.                                           \n--------------------------------------------------------------\n";

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

            {

                cout<<"    "<<(i+1)<<".        "<<s[i].name.firstname<<" "<<s[i].name.lastname;

                cout<<"   ->>-  "<<s[i].age<<"   ->>-  "<<s[i].disease<<"\n\n";

            }

        }

};

int main()

{

    cout<<"-------------------------------------------------------------\n\t----- Standing in the Hospital Queue  -----\n-------------------------------------------------------------\n\n\t\t   /\_\_Garbage Values\_\_/";

    patientQ p;

    p.view(size);

    char stopApp[0];

    stopApp[0] = 'Y';

    int i = 0;

    while(true)

    {

        cout<<"\n\nSelect action from following : \n";

        cout<<" > 1. VIEW RECORDS\t> 2. ENQUEUE RECORDS\t> 3. DEQUEUE RECORD\t> 4. EXIT\n\t\t\tEnter choice (1/2/3/4): ";

        int choice;

        cin>>choice;

        (choice == 2)? i += 1 : 0;

        switch(choice)

        {

            case 1:

                p.view(size);

                break;

            case 2:

                if (p.isQueueFull() == false)

                {

                    string firstname, lastname, disease;

                    int age;

                    cout<<"\nPatient details -\n    "<<i<<".        Enter Patient Name : ";

                    cin>>firstname;

                    cin>>lastname;

                    cout<<"              Enter Age : ";

                    cin>>age;

                    cout<<"              Enter Disease : ";

                    cin>>disease;

                    p.enqueue(firstname, lastname, disease, age);

                    p.view(size);

                    break;

                }

                else

                {

                    cout<<"\n\n \_\_\_\_\_\_ Currently queue is Full ! \_\_\_\_\_\_\n    Please dequeue to get a empty spot\n";

                    break;

                }

            case 3:

                if (p.isQueueEmpty() == true)

                {

                    cout<<"\n\n \_\_\_\_\_ Currently queue is Empty ! \_\_\_\_\_";

                    break;

                }

                else

                {

                    cout<<"\n----------------- Last Patient '"<<p.dequeue()<<"' Discharged ! -----------------\n";

                    i--;

                    break;

                }

            case 4:

                cout<<" >Exited Hospital<\n --| END OF CODE |--";

                break;

            default :

                cout<<"\n Invalid choice !";

                break;

        }

        if (choice == 4)

        {

            break;

        }

        cout<<"\nDo you want to continue (Y/N) ? : ";

        cin>>stopApp[0];

        if (stopApp[0] == 'N' | stopApp[0] == 'n')

        {

            cout<<"\n\t ----------------| END OF CODE |----------------";

        }

    }

}

**Exp 4: Expression tree**

**#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:

Exp: 5 BST

#include<iostream>

using namespace std;

typedef struct TreeNode

{

    int data;

    struct TreeNode \*left;

    struct TreeNode \*right;

}TreeNode;

typedef struct Node

{

   struct TreeNode \*TreeNode;

   struct Node \*next;

}Node;

class Queue

{

   Node \*front,\*rear;

public:

    Queue()

    {

       front=NULL;

       rear=NULL;

    }

    int isEmpty()

    {

       return (front==NULL)?1:0;

    }

    void enque(TreeNode \*t)

    {

        Node \*p;

        p=new Node();

        p->TreeNode = t;

        p->next=NULL;

        if(front==NULL)

        {

         front=p;

         rear=p;

        }

        else

        {

         rear->next = p;

         rear = rear->next;

        }

    }

    TreeNode\* deque()

    {

        Node \*p;

        TreeNode \*temp;

        p = front;

        temp = front->TreeNode;

        if(front==rear)

        {

            front=NULL;

            rear=NULL;

        }

        else

        {

            front = front->next;

        }

        delete p;

        return temp;

    }

};

class Tree

{

    TreeNode \*t;

public:

    Tree()

    {

        t=NULL;

    }

    TreeNode \*insert(int x)

    {

        TreeNode \*p,\*q,\*r;

        p = new TreeNode();

        p->data=x;

        p->left=NULL;

        p->right=NULL;

        if(t==NULL)

            return p;

        q=t;

        r=t;

        while(r!=NULL)

        {

            q=r;

            if(x < r->data)

                r=r->left;

            else

                r=r->right;

        }

        if(x<q->data)

            q->left=p;

        else

            q->right=p;

        return t;

    }

    TreeNode \*create()

    {

        int n,i,key;

        cout<<"\nEnter the number of Nodes : ";

        cin>>n;

        for(i=0;i<n;i++)

        {

            cout<<"\nEnter the data : ";

            cin>>key;

            t=insert(key);

        }

        return t;

    }

    void inorder(TreeNode \*t)

    {

        if(t!=NULL)

        {

            inorder(t->left);

            cout<<"\t"<<t->data;

            inorder(t->right);

        }

    }

    TreeNode\* search(int key)

    {

        TreeNode \*s=t;

        while(s!=NULL)

        {

            if(s->data==key)

                return t;

            else if(s->data<key)

                s=s->right;

            else

                s=s->left;

        }

        return NULL;

    }

    TreeNode \*getMin(TreeNode \*r)

    {

        while(r->left != NULL)

        {

            r=r->left;

        }

        return r;

    }

    TreeNode \*del(TreeNode \*t,int key)

    {

        TreeNode \*temp;

        if(t==NULL)

        {

            return NULL;

        }

        if(key<t->data)

        {

            t->left=del(t->left,key);

            return t;

        }

        if(key>t->data)

        {

            t->right=del(t->right,key);

            return t;

        }

        //element found

        //no child

        if(t->left==NULL&t->right==NULL)

        {

            temp=t;

            delete temp;

            return NULL;

        }

        //one child

        if(t->left!=NULL&&t->right==NULL)

        {

            temp=t;

            t=t->left;

            delete temp;

            return t;

        }

        if(t->left==NULL&&t->right!=NULL)

        {

            temp=t;

            t=t->right;

            delete temp;

            return t;

        }

        //both child present

        temp=getMin(t->right);

        t->data=temp->data;

        t->right=del(t->right,temp->data);

        return t;

    }

    TreeNode \*mirror(TreeNode \*t)

    {

        TreeNode \*temp;

        if(t==NULL)

        {

            return NULL;

        }

        temp=t->left;

        t->left=mirror(t->right);

        t->right=mirror(temp);

        return t;

    }

    void level\_wise()

    {

        TreeNode \*t1;

        Queue q1;

        if(t==NULL)

            return;

        q1.enque(t);

        cout<<"\n "<<t->data;

        while(q1.isEmpty()!=1)

        {

            cout<<"\n";

            Queue q2;

            while(q1.isEmpty()!=1)

            {

                t1=q1.deque();

                if(t1->left!=NULL)

                {

                    q2.enque(t1->left);

                    cout<<" "<<t1->left->data;

                }

                if(t1->right!=NULL)

                {

                    q2.enque(t1->right);

                    cout<<" "<<t1->right->data;

                }

            }

            q1=q2;

        }

    }

};

int main()

{

    int choice,key;

    TreeNode \*root,\*result;

    Tree t;

    do

    {

        cout<<"\n=======================================";

        cout<<"\nMENU";

        cout<<"\n=======================================";

        cout<<"\n1.Create\n2.Insert\n3.Display\n4.Search\n5.Delete\n6.Mirror image\n7.Display Level wise\n8.Minimum\n9.Exit";

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

        cin>>choice;

        switch(choice)

        {

        case 1:root = t.create();break;

        case 2:

            cout<<"\nEnter the number to Insert : ";

            cin>>key;

            root=t.insert(key);

            break;

        case 3:

            cout<<"\n-----------------------------------------------";

            cout<<"\nBINARY TREE :-";

            t.inorder(root);

            cout<<"\n-----------------------------------------------";

            break;

        case 4:

            cout<<"\n--------------------------";

            cout<<"\nEnter the Node to search :";

            cin>>key;

            result = t.search(key);

            if(result==NULL)

            {

                cout<<"\nNODE "<<key<<" NOT FOUND";

            }

            else

            {

                cout<<"\nNODE "<<key<<" IS FOUND";

            }

            cout<<"\n--------------------------";

            break;

        case 5:

            cout<<"\n--------------------------";

            cout<<"\nEnter the Node to delete :";

            cin>>key;

            result = t.del(root,key);

            root=result;

            cout<<"\nNODE DELETED !!";

            cout<<"\n--------------------------";

            break;

        case 6:

            cout<<"\n--------------------------";

            root=t.mirror(root);

            cout<<"\nMirror image of the binary Tree :";

            t.inorder(root);

            cout<<"\n--------------------------";

            break;

        case 7:

            cout<<"\n--------------------------";

            cout<<"\nLevel Wise Display :";

            cout<<"\n--------------------------";

            t.level\_wise();

            cout<<"\n--------------------------";

            break;

        case 8:

            result = t.getMin(root);

            cout<<"\n--------------------------";

            cout<<"\nMINIMUM = "<<result->data;

            cout<<"\n--------------------------";

            break;

        case 9:return 0;

        default:cout<<"\nInvalid Choice !!";

        }

    }while(choice!=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:**

**Prisms algo**

**#include<iostream>**

**using namespace std;**

**// Number of vertices in the graph**

**const int V=6;**

**// Function to find the vertex with minimum key value**

**int min\_Key(int key[], bool visited[])**

**{**

**int min = 999, min\_index; // 999 represents an Infinite value**

**for (int v = 0; v < V; v++) {**

**if (visited[v] == false && key[v] < min) {**

**// vertex should not be visited**

**min = key[v];**

**min\_index = v;**

**}**

**}**

**return min\_index;**

**}**

**// Function to print the final MST stored in parent[]**

**int print\_MST(int parent[], int cost[V][V])**

**{**

**int minCost=0;**

**cout<<"Edge \tWeight\n";**

**for (int i = 1; i< V; i++) {**

**cout<<parent[i]<<" - "<<i<<" \t"<<cost[i][parent[i]]<<" \n";**

**minCost+=cost[i][parent[i]];**

**}**

**cout<<"Total cost is"<<minCost;**

**}**

**// Function to find the MST using adjacency cost matrix representation**

**void find\_MST(int cost[V][V])**

**{**

**int parent[V], key[V];**

**bool visited[V];**

**// Initialize all the arrays**

**for (int i = 0; i< V; i++) {**

**key[i] = 999; // 99 represents an Infinite value**

**visited[i] = false;**

**parent[i]=-1;**

**}**

**key[0] = 0; // Include first vertex in MST by setting its key vaue to 0.**

**parent[0] = -1; // First node is always root of MST**

**// The MST will have maximum V-1 vertices**

**for (int x = 0; x < V - 1; x++)**

**{**

**// Finding the minimum key vertex from the**

**//set of vertices not yet included in MST**

**int u = min\_Key(key, visited);**

**visited[u] = true; // Add the minimum key vertex to the MST**

**// Update key and parent arrays**

**for (int v = 0; v < V; v++)**

**{**

**// cost[u][v] is non zero only for adjacent vertices of u**

**// visited[v] is false for vertices not yet included in MST**

**// key[] gets updated only if cost[u][v] is smaller than key[v]**

**if (cost[u][v]!=0 && visited[v] == false && cost[u][v] < key[v])**

**{**

**parent[v] = u;**

**key[v] = cost[u][v];**

**}**

**}**

**}**

**// print the final MST**

**print\_MST(parent, cost);**

**}**

**// main function**

**int main()**

**{**

**int cost[V][V];**

**cout<<"Enter the vertices for a graph with 6 vetices";**

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

**{**

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

**{**

**cin>>cost[i][j];**

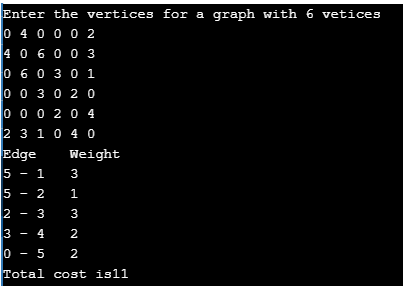
**}**

**}**

**find\_MST(cost);**

**return 0;**

**}**

Figure 2

**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**

**9.**

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

**using namespace std;**

**//Heapify function to maintain heap property.**

**void Heapify(int A[], int n, int i)**

**{**

**int large = i;**

**int left = 2 \* i + 1, right = 2 \* i + 2;**

**if (left < n && A[large] < A[left])**

**large = left;**

**if (right < n && A[large] < A[right])**

**large = right;**

**// large contains the index of largest value from all three**

**if (large == i)**

**return;**

**swap(A[large], A[i]);**

**Heapify(A, n, large);**

**}**

**//Function to build a Heap from array.**

**void buildHeap(int A[], int n)**

**{**

**// internal nodes are from n/2-1 to 0**

**for (int i = n / 2 - 1; i >= 0; --i)**

**Heapify(A, n, i);**

**}**

**//Function to sort an array using Heap Sort.**

**void heapSort(int A[], int n)**

**{**

**buildHeap(A, n);**

**for (int i = n - 1; i > 0; --i)**

**{**

**swap(A[0], A[i]);**

**Heapify(A, i, 0);**

**}**

**}**

**int main()**

**{**

**cout<<"\nEnter size of array (>7): ";**

**int size;**

**againSize :**

**{**

**cin>>size;**

**if(size < 8)**

**{**

**cout<<"\nSize of array is smaller, input size should be greater than '7'";**

**goto againSize;**

**}**

**}**

**cout<<"\nEnter "<<size<<" no. of elements of array : ";**

**int array[size];**

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

**{**

**cin>>array[i];**

**}**

**cout<<"\nArray befor Heap Sort :";**

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

**{**

**cout<<" "<<array[i];**

**}**

**cout<<"\nArray after Heap Sort :";**

**heapSort(&array[0], size);**

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

**{**

**cout<<" "<<array[i];**

**}**

**}**

**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**

#include <bits/stdc++.h>

using namespace std;

struct DoB

{

    int day, month, year;

};

struct StudentForm

{

    int roll\_no;

    string stud\_name, stud\_address;

    DoB stud\_DoB;

    float stud\_percent;

};

class DatabaseFile

{

    StudentForm stud\_rec;  //single declaration for simplified usage of struct StudentForm

    public :

        void CreateAFile();

        void AddNewRecord();

        void DisplayFileContent();

        void SearchRecord();

        void ModifyRecord();

        void DeleteRecord();

};

void DatabaseFile :: CreateAFile()

{

    ifstream file\_exist("StudentData.dat");

    if(file\_exist)  //validity to check if file needs to be created again or exist already

    {

        cout<<"\nFile 'StudentData.dat' already exist in this directory !\nFile contents are...";

        DatabaseFile :: DisplayFileContent();

        return;

    }

    fstream file("StudentData.dat", ios::out | ios::binary);  //open file in write mode to create new database

    cout<<"\nNew file StudentData.dat created..\n";

    AddNewRecord();  //add initial record

    file.close();

}

void DatabaseFile :: AddNewRecord()

{

    cout<<"\nFill student details :\nRoll No. : ";

    cin>>stud\_rec.roll\_no;

    cin.ignore();

    fstream write\_file("StudentData.dat", ios::app | ios::binary); //open file in append mode

    fstream read\_file("StudentData.dat", ios::in | ios::binary); //open file in read with different pointer

    StudentForm s1;

    read\_file.read((char\*)&s1, sizeof(s1));

    int flag\_interating\_through\_file = 1;

    while(!read\_file.eof())  //read file till end for roll no. validity check

    {

        if(s1.roll\_no == stud\_rec.roll\_no)

        {

            cout<<"\nStudent with roll no. "<<stud\_rec.roll\_no<<" already exist !\n";

            flag\_interating\_through\_file = 0;

            break;

        }

        else

        {

            read\_file.read((char\*)&s1, sizeof(s1));  //read next record

        }

    }

    if(flag\_interating\_through\_file == 1)  //if roll no. is unique then only write to the file

    {

        cout<<"\nName : ";

        getline(cin, stud\_rec.stud\_name);

        cout<<"\nAddress : ";

        getline(cin, stud\_rec.stud\_address);

        cout<<"\nDate of birth(DD/MM/YYYY) : ";

        string date;

        getline(cin, date);

        stud\_rec.stud\_DoB.day = stoi(date.substr(0, 2));

        stud\_rec.stud\_DoB.month = stoi(date.substr(3, 2));

        stud\_rec.stud\_DoB.year = stoi(date.substr(6, 4));

        cout<<"\nPercentage marks : ";

        cin>>stud\_rec.stud\_percent;

        write\_file.write((char\*)&stud\_rec, sizeof(stud\_rec));  //write to the file

        cout<<"\nRecord added successfully to the file !\n";

    }

    read\_file.close();

    return;

}

void DatabaseFile :: DisplayFileContent()

{

    fstream read\_file;

    read\_file.open("StudentData.dat", ios::in | ios::binary);

    if(!read\_file)  //file not found

    {

        cout<<"\nFile Not Found !\nCheck local directory, or try creating the file first from Menu option\n";

        return;

    }

    else

    {

        read\_file.read((char\*)&stud\_rec, sizeof(stud\_rec));  //read first record

        cout<<"\nRoll No.\tName\t\tDoB\t\tPercentage\tAddress\n---------------------------------------------------------------------------";

        while(!read\_file.eof())  //read file till  end of file

        {

            cout<<"\n"<<stud\_rec.roll\_no<<"\t\t"<<stud\_rec.stud\_name<<"\t"<<stud\_rec.stud\_DoB.day<<"/"<<stud\_rec.stud\_DoB.month<<"/"<<stud\_rec.stud\_DoB.year<<"\t"<<stud\_rec.stud\_percent<<"\t\t"<<stud\_rec.stud\_address;

            read\_file.read((char\*)&stud\_rec, sizeof(stud\_rec));  //read next record

        }

    }

    read\_file.close();

}

void DatabaseFile :: SearchRecord()

{

    fstream read\_file("StudentData.dat", ios::in | ios::binary);  //open file in reading mode

    if(!read\_file)

    {

        cout<<"\nFile Not Found !\nCheck local directory, or try creating the file first from Menu option\n";

        return;

    }

    else

    {

        bool flag = false;

        cout<<"\nEnter roll no. to be searched : ";

        int key;

        cin>>key;

        read\_file.read((char\*)&stud\_rec, sizeof(stud\_rec)); //read first record

        while(!read\_file.eof())  //search till end of file

        {

            if(stud\_rec.roll\_no == key)  //if key is equal to correct roll number

            {

                cout<<"\nRecord found !\n";

                flag = true;  //set flag

                cout<<"\nRoll No.\tName\t\tDoB\t\tPercentage\tAddress\n---------------------------------------------------------------------------";

                cout<<"\n"<<stud\_rec.roll\_no<<"\t\t"<<stud\_rec.stud\_name<<"\t"<<stud\_rec.stud\_DoB.day<<"/"<<stud\_rec.stud\_DoB.month<<"/"<<stud\_rec.stud\_DoB.year<<"\t"<<stud\_rec.stud\_percent<<"\t\t"<<stud\_rec.stud\_address;  //display student details

                break;

            }

            else

            {

                read\_file.read((char\*)&stud\_rec, sizeof(stud\_rec)); //read next record

            }

        }

        if(!flag)  //if roll no. is not found

            cout<<"\nRecord not found !\n";

    }

    read\_file.close();

}

void DatabaseFile :: ModifyRecord()

{

    cout<<"\nEnter roll no. of the student : ";

    int rno, flag = 0;

    cin>>rno;

    fstream read\_file("StudentData.dat", ios::in | ios::binary);  //open file in read mode

    fstream newFile("temp.dat", ios::out | ios::binary);  //create new temp file for temporary data storage

    if(!read\_file)

    {

        cout<<"\nFile Not Found !\nCheck local directory, or try creating the file first from Menu option\n";

        return;

    }

    else

    {

        read\_file.read((char\*)&stud\_rec, sizeof(stud\_rec));

        while(!read\_file.eof())

        {

            if(stud\_rec.roll\_no != rno)  //if current record is not that to be updated

            {

                newFile.write((char\*)&stud\_rec, sizeof(stud\_rec));  //write to the temp file

            }

            else  //if current record is that ts to be updated

            {

                cout<<"\nRecord Found !\n";

                flag = 1;

                cout<<"\nFill student details :\nRoll No. : "<<rno;

                stud\_rec.roll\_no = rno;

                cin.ignore();

                cout<<"\nName : ";

                getline(cin, stud\_rec.stud\_name);

                cout<<"\nAddress : ";

                getline(cin, stud\_rec.stud\_address);

                cout<<"\nDate of birth(DD/MM/YYYY) : ";

                string date;

                getline(cin, date);

                stud\_rec.stud\_DoB.day = stoi(date.substr(0, 2));

                stud\_rec.stud\_DoB.month = stoi(date.substr(3, 2));

                stud\_rec.stud\_DoB.year = stoi(date.substr(6, 4));

                cout<<"\nPercentage marks : ";

                cin>>stud\_rec.stud\_percent;

                newFile.write((char\*)&stud\_rec, sizeof(stud\_rec));  //write to the file

                cout<<"\nRecord updated successfully to the file !\n";  //write updated record to the temp file

            }

            read\_file.read((char\*)&stud\_rec, sizeof(stud\_rec));  //read next record from the

        }

        read\_file.close();

        newFile.close();

        remove("StudentData.dat");

        rename("temp.dat", "StudentData.dat");

        if(flag == 0)  //flag remained same then

        {

            cout<<"\nRecord with roll no. "<<rno<<" is not found in the file 'StudentData.dat'\n";

        }

    }

}

void DatabaseFile :: DeleteRecord()

{

    cout<<"\nEnter roll no. of the student : ";

    int rno, flag = 0;

    cin>>rno;

    fstream read\_file("StudentData.dat", ios::in | ios::binary);  //open file in read mode

    fstream newFile("temp.dat", ios::out | ios::binary);  //create new temp file for temporary data storage

    if(!read\_file)

    {

        cout<<"\nFile Not Found !\nCheck local directory, or try creating the file first from Menu option\n";

        return;

    }

    else

    {

        read\_file.read((char\*)&stud\_rec, sizeof(stud\_rec));

        while(!read\_file.eof())

        {

            if(stud\_rec.roll\_no != rno)  //if current record is not that to be updated

            {

                newFile.write((char\*)&stud\_rec, sizeof(stud\_rec));  //write to the temp file

            }

            else  //if current record is that ts to be updated

            {

                cout<<"\nRecord Found and Deleted!\n";

                flag = 1;

            }

            read\_file.read((char\*)&stud\_rec, sizeof(stud\_rec));  //read next record from the

        }

        read\_file.close();

        newFile.close();

        remove("StudentData.dat");

        rename("temp.dat", "StudentData.dat");

        if(flag == 0)  //flag remained same then

        {

            cout<<"\nRecord with roll no. "<<rno<<" is not found in the file 'StudentData.dat'\n";

        }

    }

}

int main()

{

    DatabaseFile fileOp;

    int ch;

    do

    {

        cout<<"\n================================================================MENUS================================================================\n\nChoose Operation -\n1.Create File \t2.Add Record\t3.View Records  \t4.Search Record \t5.Modify Record \t6.Delete Record \t7.Exit\n\t\t\t\t\t \t Choose Operation : ";

        cin>>ch;

        cout<<"\n====================================================================================================================================="<<endl;

        switch(ch)

        {

            case 1:

                fileOp.CreateAFile();

                break;

            case 2:

                fileOp.AddNewRecord();

                break;

            case 3:

                fileOp.DisplayFileContent();

                break;

            case 4:

                fileOp.SearchRecord();

                break;

            case 5:

                fileOp.ModifyRecord();

                break;

            case 6:

                fileOp.DeleteRecord();

                break;

            case 7:

                cout<<"\n--- Program Terminated ---\n";

                return 0;

            default:

                cout<<"\nEnter Valid Operation\n";

                break;

        }

    }while(true);

    return 0;

}