**Assignment 1**

**Aim:** To create ADT that implement the "set" concept. a. Add (newElement) -Place a value into the set b. Remove (element) c. Contains (element) Return true if element is in collection d. Size () Return number of values in collection e. Intersection of two sets f. Union of two sets g. Difference between two sets h.

**Code:**

#include<iostream>

using namespace std;

void create(int set[],int n);

void display(int set[]);

void intersection(int set1[],int set2[],int set3[]);

void unions(int set1[],int set2[],int set3[]);

void diff(int set1[],int set2[],int set3[]);

int member(int set[],int x);

int subset(int set[],int sset[]);

int member(int set[],int n,int x);

#define max 30

int main()

{

int set1[max],set2[max],set\_union[max],set\_int[max],set\_diff[max],set\_s[max];

int c;

do

{

cout<<"---------------------------------MENU----------------------------------"<<endl;

cout<<"1] Create\n2] Display\n3] Intersection\n4] Union\n5] A-B\n6] B-A\n7] Check subset\n8] Exit\n";

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

cout<<"Enter your choice: ";

cin>>c;

switch(c)

{

case 1:{

int s1,s2;

cout<<"Enter number elements of set A: ";

cin>>s1;

create(set1,s1);

cout<<"Enter number elements of set B: ";

cin>>s2;

create(set2,s2);

}

break;

case 2:{

cout<<"The elements of set A are: ";

display(set1);

cout<<"The elements of set B are: ";

display(set2);

}

break;

case 3:{

intersection(set1,set2,set\_int);

cout<<"The intersection of A and B is: ";

display(set\_int);

}

break;

case 4:{

unions(set1,set2,set\_union);

cout<<"The union of A and B is: ";

display(set\_union);

}

break;

case 5:{

diff(set1,set2,set\_diff);

cout<<"The difference between A and B is: ";

display(set\_diff);

}

break;

case 6:{

diff(set2,set1,set\_diff);

cout<<"The difference between B and A is: ";

display(set\_diff);

}

break;

case 7:{

int n;

cout<<"Enter number elements of subset: ";

cin>>n;

create(set\_s,n);

if(subset(set1,set\_s) && subset(set2,set\_s))

cout<<"The given set is a subset of both sets\n";

else if(subset(set2,set\_s))

cout<<"The given set is a subset of set B\n";

else if(subset(set1,set\_s))

cout<<"The given set is a subset of set A\n";

else

cout<<"The given set is not a subset of any set\n";

}

break;

}

}while(c!=8);

return 0;

}

void create(int set[],int n)

{

set[0]=0;

for(int i=1;i<=n;i++)

{

cin>>set[i];

}

set[0]=n;

}

void display(int set[])

{

int n=set[0];

for(int i=1;i<=n;i++)

{

cout<<set[i]<<"\t";

}

cout<<endl;

}

int member(int set[],int x)

{

int n=set[0];

for(int i=1;i<=n;i++)

{

if(set[i]==x)

return 1;

}

return 0;

}

void intersection(int set1[],int set2[],int set3[])

{

set3[0]=0;

int n=set1[0];

for(int i=1;i<=n;i++)

{

if(member(set2,set1[i]))

{

set3[0]++;

set3[set3[0]]=set1[i];

}

}

}

void unions(int set1[],int set2[],int set3[])

{

int n=set1[0];

set3[0]=n;

for(int i=1;i<=n;i++)

{

set3[i]=set1[i];

}

n=set2[0];

for(int i=1;i<=n;i++)

{

if(!member(set3,set2[i]))

{

set3[0]++;

set3[set3[0]]=set2[i];

}

}

}

void diff(int set1[],int set2[],int set3[])

{

set3[0]=0;

int n=set1[0];

for(int i=1;i<=n;i++)

{

if(!member(set2,set1[i]))

{

set3[0]++;

set3[set3[0]]=set1[i];

}

}

}

int subset(int set[],int sset[])

{

int n=sset[0];

int flag=0;

for(int i=1;i<=n;i++)

{

if(!member(set,sset[i]))

{

flag=1;

break;

}

}

if(flag==1)

return 0;

else

return 1;

}

**Assignment 2**

**Aim:** Construct a threaded binary search tree by inserting values in the given order and traverse it in inorder traversal using threads.

**Code:**

#include <iostream>

using namespace std;

class tnode

{

int data;

int l,r;

tnode \*lt,\*rt;

public:

tnode \*create(int item)

{

tnode \*nn=new tnode;

nn->data=item;

nn->lt=nn->rt=NULL;

nn->l=nn->r=1;

return nn;

}

tnode \*insert\_r(tnode \*r,int item)

{

tnode \*rp=r;

tnode \*p=NULL;

while(rp!=NULL)

{

p=rp;

if(item<rp->data)

{

if(rp->l==0)

rp=rp->lt;

else

break;

}

else if(item>rp->data)

{

if(rp->r==0)

rp=rp->rt;

else

break;

}

}

tnode \*nn;

nn=nn->create(item);

if(p==NULL)

{

r=nn;

}

else if(item<p->data)

{

nn->lt=p->lt;

nn->rt=p;

p->lt=nn;

p->l=0;

}

else if(item>p->data)

{

nn->rt=p->rt;

nn->lt=p;

p->rt=nn;

p->r=0;

}

return r;

}

void inorder(tnode \*r)

{

tnode \*temp=r;

while(temp->l==0)

{

temp=temp->lt;

}

while(temp!=NULL)

{

cout<<temp->data<<endl;

if(temp->r==1)

temp=temp->rt;

else

{

temp=temp->rt;

while(temp->l==0)

{

temp=temp->lt;

}

}

}

}

void reverse\_inorder(tnode \*r)

{

tnode \*temp=r;

while(temp->r==0)

{

temp=temp->rt;

}

while(temp!=NULL)

{

cout<<temp->data<<endl;

if(temp->l==1)

temp=temp->lt;

else

{

temp=temp->lt;

while(temp->r==0)

{

temp=temp->rt;

}

}

}

}

};

tnode \*root=NULL;

int main()

{

char r;

do

{

int flag=0;

root=NULL;

char op;

do

{

if(flag==0)

{

int n;

cout<<"Enter the number of elements you wish to enter: ";

cin>>n;

int a[n];

cout<<"Start entering the elements\n";

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

{

cin>>a[i];

root=root->insert\_r(root,a[i]);

}

flag=1;

}

int c;

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

cout<<"1] Insert\n2] Display ascending\n3] Display descending";

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

cout<<"Enter your choice: ";

cin>>c;

switch(c)

{

case 1: {

int item;

cout<<"Enter data you wish to enter: ";

cin>>item;

root=root->insert\_r(root, item);

}

break;

case 2: {

root->inorder(root);

}

break;

case 3: {

root->reverse\_inorder(root);

}

break;

default:cout<<"Invalid\n";

}

cout<<"Do you wish to continue?(y/n): ";

cin>>op;

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

cout<<"Test pass?(y/n): ";

cin>>r;

}while(r=='n' || r=='N');

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

cout<<"\* Thank You! \*\n";

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

return 0;

}

**Assignment 3**

**Aim:** There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representations used.

**Code:**

#include<iostream>

#define MAX 10

using namespace std;

class airport

{

string city[MAX];

int distance[10][10];

public :

int n;

airport();

void read\_city();

void show\_graph();

};

airport::airport()

{

n=0;

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

{

for(int j=0;j<MAX;j++)

distance[i][j]=0;

}

}

void airport::read\_city()

{

int k;

cout<<"\nEnter the no. of cities: " ;

cin>>n;

cout<<"Enter city name:\n";

for(int k=0;k<n;k++)

{

cout<<k+1<<"] ";

cin>>city[k];

}

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

{

for(int j=i+1 ; j<n ; j++)

{

cout<<"\nEnter Distance between "<<city[i]<<" to "<<city[j]<<": ";

cin>>distance[i][j];

distance[j][i]=distance[i][j];

}

}

}

void airport::show\_graph()

{

cout<<"\t";

for(int k=0;k<n;k++)

{

cout<<city[k]<<"\t";

}

cout<<endl;

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

{

cout<<city[i]<<"\t";

for(int j=0;j<n;j++)

{

cout<<distance[i][j]<<"\t";

}

cout<<endl;

}

}

int main()

{

airport obj;

obj.read\_city();

obj.show\_graph();

}

**Assignment 4**

**Aim:** For a weighted graph G, find the minimum spanning tree using Prims algorithm

**Code:**

#include <iostream>

using namespace std;

class graph

{

int a[100][100];

int v;

public:

void insert\_edge(int n1,int n2,int wt)

{

if(n1-1>=v||n2-1>=v)

cout<<"Vertex request out of range\n";

else

{

a[n1-1][n2-1]=wt;

a[n2-1][n1-1]=wt;

}

}

void display()

{

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

{

for(int j=0;j<v;j++)

{

cout<<a[i][j]<<"\t";

}

cout<<endl;

}

}

void update\_v(int n)

{

v=n;

}

void prims(int src)

{

int sp[v],dist[v],visited[v],parent[v],c=0;

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

{

visited[i]=0;

dist[i]=9999;

}

dist[src-1]=0;

parent[src-1]=-1;

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

{

int min=9999,min\_ind;

for(int j=0;j<v;j++)

{

if(!visited[j] && dist[j]<min )

{

min=dist[j];

min\_ind=j;

}

}

int U=min\_ind;

visited[U]=1;

sp[c]=U;

c++;

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

{

if(!visited[V] && a[U][V] && a[U][V]<dist[V] && dist[U]!=9999)

{

parent[V]=U;

dist[V]=a[U][V];

}

}

}

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

{

cout<<sp[i]+1<<" link from "<<parent[i]+1<<endl;

}

cout<<endl;

}

};

int main()

{

char r;

do

{

graph g;

char op;

int v;

cout<<"Enter number of vertices: ";

cin>>v;

g.update\_v(v);

do

{

int c;

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

cout<<"1] Insert edge\n2] Increase number of vertices\n3] Display matrix\n4] Find shortest path\n";

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout<<"Enter your choice: ";

cin>>c;

switch(c)

{

case 1: {

int n1,n2,wt;

cout<<"Enter the nodes between which there is an edge\n";

cin>>n1>>n2;

cout<<"Enter weight: ";

cin>>wt;

g.insert\_edge(n1,n2,wt);

}

break;

case 2: {

int n;

cout<<"Enter the number by which you wish to increase the vertices: ";

cin>>n;

v+=n;

g.update\_v(v);

}

break;

case 3: {

g.display();

}

break;

case 4: {

int src,dst;

cout<<"Source: ";

cin>>src;

g.prims(src);

}

break;

default:cout<<"Error 404.....page not found\n";

}

cout<<"Do you wish to continue(y/n): ";

cin>>op;

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

cout<<"Test pass(y/n): ";

cin>>r;

}while(r=='n' || r=='N');

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

cout<<"\* Thank You! \*\n";

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

return 0;

}

**Assignment 5**

**Aim:** You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures

**Code:**

#include <iostream>

using namespace std;

const int MAX=10;

class edge

{

friend class graph;

friend class edgelist;

int u,v,wt;

public:

edge()

{}

edge(int x,int y, int w)

{

u=x;

v=y;

wt=w;

}

};

class edgelist

{

friend class graph;

edge data[MAX];

int n;

public:

edgelist()

{

n=0;

}

void sort()

{

edge temp;

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

{

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

{

if(data[j].wt>data[j].wt)

{

temp=data[j];

data[j]=data[j+1];

data[j+1]=temp;

}

}

}

}

void print()

{

cout<<n<<endl;

int cost=0;

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

{

cout<<"\n"<<i+1<<" "<<data[i].u<<" --> "<<data[i].v<<" = "<<data[i].wt;

cost=cost+data[i].wt;

}

cout<<"\nThe minimum cost of the minimum spanning tree is "<<cost<<endl;

}

};

class graph

{

int g[MAX][MAX];

int v;

public:

graph()

{

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

for(int j=0;j<v;j++)

g[i][j]=0;

}

void insert\_edge(int n1,int n2,int wt)

{

if(n1-1>=v||n2-1>=v)

cout<<"Vertex request out of range\n";

else

{

g[n1-1][n2-1]=wt;

g[n2-1][n1-1]=wt;

}

}

void display()

{

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

{

for(int j=0;j<v;j++)

{

cout<<g[i][j]<<"\t";

}

cout<<endl;

}

}

void update\_v(int n)

{

v=n;

}

void krushkal(edgelist mst)

{

edgelist list;

int belongs[v];

int c1,c2;

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

{

for(int j=0;j<v;j++)

{

if(g[i][j]!=0)

{

list.data[list.n]=edge(i,j,g[i][j]);

list.n++;

}

}

}

list.sort();

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

belongs[i]=i;

for(int i=0;i<list.n;i++)

{

c1=find(belongs,list.data[i].u);

c2=find(belongs,list.data[i].v);

if(c1!=c2)

{

mst.data[mst.n]=list.data[i];

mst.n++;

uni(belongs,c1,c2);

}

}

mst.print();

}

int find(int belongs[],int x)

{

return belongs[x];

}

void uni(int belongs[],int c1,int c2)

{

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

{

if(belongs[i]==c2)

belongs[i]=c1;

}

}

};

int main()

{

char r;

do

{

graph g;

char op;

int v;

cout<<"Enter number of vertices: ";

cin>>v;

g.update\_v(v);

do

{

int c;

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

cout<<"1] Insert edge\n2] Increase number of vertices\n3] Display matrix\n4] MST by krushkal's\n";

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout<<"Enter your choice: ";

cin>>c;

switch(c)

{

case 1: {

int n1,n2,wt;

cout<<"Enter the nodes between which there is an edge\n";

cin>>n1>>n2;

cout<<"Enter weight: ";

cin>>wt;

g.insert\_edge(n1,n2,wt);

}

break;

case 2: {

int n;

cout<<"Enter the number by which you wish to increase the vertices: ";

cin>>n;

v+=n;

g.update\_v(v);

}

break;

case 3: {

g.display();

}

break;

case 4: {

edgelist mst;

g.krushkal(mst);

}

break;

default:cout<<"Error 404.....page not found\n";

}

cout<<"Do you wish to continue(y/n): ";

cin>>op;

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

cout<<"Test pass(y/n): ";

cin>>r;

}while(r=='n' || r=='N');

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

cout<<"\* Thank You! \*\n";

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

return 0;

}

**Assignment 6**

**Aim:** Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject using heap data structure.

**Code:**

#include<iostream>

using namespace std;

class HeapSort

{

int heap[20],heap1[20],x,n1,i;

public:

HeapSort()

{

heap[0]=0; heap1[0]=0;

}

void getdata();

void insertMax(int heap[],int);

void buildMax(int heap[],int);

void insertMin(int heap1[],int);

void buildMin(int heap1[],int);

void minmax();

};

void HeapSort::getdata()

{

cout<<"\nEnter no. of students -";

cin>>n1;

cout<<"\nEnter marks - ";

for(i=0;i<n1;i++)

{ cin>>x;

insertMax(heap,x);

insertMin(heap1,x);

}

}

void HeapSort::insertMax(int heap[20],int x)

{

int n;

n=heap[0];

heap[n+1]=x;

heap[0]=n+1;

buildMax(heap,n+1);

}

void HeapSort::buildMax(int heap[20],int i)

{

int temp;

while(i>1&&heap[i]>heap[i/2])

{

temp=heap[i];

heap[i]=heap[i/2];

heap[i/2]=temp;

i=i/2;

}

}

void HeapSort::insertMin(int heap1[20],int x)

{

int n;

n=heap1[0];

heap1[n+1]=x;

heap1[0]=n+1;

buildMin(heap1,n+1);

}

void HeapSort::buildMin(int heap1[20],int i)

{

int temp1;

while(i>1&&heap1[i]<heap1[i/2])

{

temp1=heap1[i];

heap1[i]=heap1[i/2];

heap1[i/2]=temp1;

i=i/2;

}

}

void HeapSort::minmax()

{

cout<<"\nMaximum Marks -\n\n ";

for(i=0;i<=n1;i++)

{

cout<<"\n"<<heap[i];

}

cout<<"\nMin Marks - \n\n";

for(i=0;i<=n1;i++)

{

cout<<"\n"<<heap1[i];

}

}

int main()

{

HeapSort h;

h.getdata();

h.minmax();

return 0;

}

**Assignment 7**

**Aim:** Insert the keys into a hash table of length m using open addressing using double hashing with h(k)=1+(k mod(m-1)).

**Code:**

#include <iostream>

using namespace std;

class dr

{

int n=10;

int arr[100][3];

int c;

public:

dr()

{

cout<<"Table of size "<<n<<" created\n";

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

{

arr[i][0]=0;

arr[i][1]=-1;

arr[i][2]=-1;

}

c=0;

}

void add(int,int);

int find\_key(int);

void display();

void update\_val(int,int);

};

void dr::add(int key,int value)

{

int new\_hash\_addr1,new\_hash\_addr2,main\_hash\_addr=-1,j=0;

if(this->find\_key(key)!=-1)

{

cout<<"Key already exists\n";

return;

}

if(c==(n-1))

{

cout<<"Table full, request denied\n";

}

new\_hash\_addr1=(key)%n;

new\_hash\_addr1=1+(key%(n-1));

if(arr[new\_hash\_addr1][1]==-1)

{

arr[new\_hash\_addr1][0]=key;

arr[new\_hash\_addr1][1]=value;

}

else if(arr[new\_hash\_addr2][1]==-1)

{

arr[new\_hash\_addr2][0]=key;

arr[new\_hash\_addr2][1]=value;

}

else

{

while(arr[new\_hash\_addr2][2]!=-1)

{

main\_hash\_addr=new\_hash\_addr2;

new\_hash\_addr2=arr[main\_hash\_addr][2];

}

main\_hash\_addr=new\_hash\_addr2;

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

{

new\_hash\_addr2=(main\_hash\_addr+i)%n;

if(arr[new\_hash\_addr2][1]==-1)

{

arr[new\_hash\_addr2][0]=key;

arr[new\_hash\_addr2][1]=value;

arr[main\_hash\_addr][2]=new\_hash\_addr2;

c++;

break;

}

}

}

}

void dr::display()

{

cout<<"Key\t\tValue\t\tChain\n";

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

{

cout<<arr[i][0]<<"\t\t"<<arr[i][1]<<"\t\t"<<arr[i][2]<<endl;

}

}

int dr::find\_key(int key)

{

int search\_addr=key%n,f=0;

while(arr[search\_addr][0]!=key && arr[search\_addr][2]!=-1)

{

search\_addr=arr[search\_addr][2];

}

if(arr[search\_addr][0]==key)

{

return arr[search\_addr][1];

}

else if(arr[search\_addr][2]==-1)

{

return -1;

}

}

int main()

{

char r;

do

{

char op;

dr table;

int c;

do

{

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

cout<<"1] Insert value\n2] Display\n";

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout<<"Enter your choice: ";

cin>>c;

switch(c)

{

case 1: {

int key,val;

cout<<"Enter key: ";

cin>>key;

cout<<"Enter value: ";

cin>>val;

table.add(key,val);

}

break;

case 2: table.display();

break;

default:cout<<"Invalid\n";

}

cout<<"\nDo you wish to go again? ";

cin>>op;

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

cout << "Test pass?(y/n): " << endl;

cin>>r;

}while(r=='n' || r=='N');

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

cout<<"\* Thank You! \*\n";

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

return 0;

}

**Assignment 8**

**Aim:** Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

**Code:**

#include <iostream>

#include<fstream>

#include<cstring>

#include<iomanip>

#include <cstdlib>

using namespace std;

const int MAX=30;

class Student

{

int rollno;

char name[20];

char div;

public:

Student()

{

strcpy(name,"");

rollno=div=0;

}

Student(int rollno,char name[MAX],int year,char div,char address[MAX])

{

strcpy(this->name,name);

this->rollno=rollno;

this->div=div;

}

int getRollNo()

{

return rollno;

}

void displayRecord()

{

cout<<endl<<setw(5)<<rollno<<setw(20)<<name<<setw(5)<<setw(5)<<div<<setw(10);

}

};

class FileOperations

{

fstream file;

public:

FileOperations(char\* filename)

{

file.open(filename,ios::in|ios::out|ios::ate|ios::binary);

}

void insertRecord(int rollno, char name[MAX],int year, char div,char city[MAX])

{

Student s1(rollno,name,year,div,city);

file.seekp(0,ios::end);

file.write((char \*)&s1,sizeof(Student));

}

void displayRecord(int rollNo)

{

Student s1;

file.seekg(0,ios::beg);

bool flag=false;

while(file.read((char\*)&s1,sizeof(Student)))

{

if(s1.getRollNo()==rollNo)

{

s1.displayRecord();

flag=true;

break;

}

}

if(flag==false)

{

cout<<"\nRecord of "<<rollNo<<"is not present.";

}

}

void deleteRecord(int rollno)

{

ofstream outFile("new.dat",ios::binary);

file.seekg(0,ios::beg);

bool flag=false;

Student s1;

while(file.read((char \*)&s1, sizeof(Student)))

{

if(s1.getRollNo()==rollno)

{

flag=true;

continue;

}

outFile.write((char \*)&s1, sizeof(Student));

}

if(!flag)

{

cout<<"\nERROR...... Record is not present.";

}

file.close();

outFile.close();

remove("student.dat");

rename("new.dat","student.dat");

file.open("student.dat",ios::in|ios::out|ios::ate|ios::binary);

}

};

int main()

{

system("COLOR FC");

system("cls");

ofstream newFile("student.dat",ios::app|ios::binary);

newFile.close();

FileOperations file((char\*)"student.dat");

int rollNo,year,choice=0;

char div;

char name[MAX],address[MAX];

while(choice!=4)

{

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

cout<<"\n1.Add Record\n2.Display Record of specific person\n3.Delete a record \n";

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

cout<<"\nEnter your choice - \n";

cin>>choice;

switch(choice)

{

case 1 :

{

cout<<"\nEnter RollNo & Name - \n";

cin>>rollNo>>name;

cout<<"\nEnter Division - \n";

cin>>div;

cout<<"\nEnter address - \n";

cin>>address;

file.insertRecord(rollNo,name,year,div,address);

cout<<"\nRecord Inserted - Sucess";

break;

}

case 2 :

{

cout<<"\nEnter Roll Number -";

cin>>rollNo;

file.displayRecord(rollNo);

break;

}

case 3:

{

cout<<"\nEnter RollNo -";

cin>>rollNo;

file.deleteRecord(rollNo);

break;

}

}

}

return 0;

}

**Assignment 9**

**Aim:** Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

**Code:**

#include<iostream>

#include<fstream>

using namespace std;

typedef struct EMP\_REC

{

char name[10];

int emp\_id;

int salary;

}Rec;

typedef struct INDEX\_REC

{

int emp\_id;

int position;

}Ind\_Rec;

class Employee

{

Rec Records;

Ind\_Rec Ind\_Records;

public:

void Create();

void Display();

void Search();

};

void Employee::Create()

{

char ch='y';

ofstream seqfile;

ofstream indexfile;

int i=0;

indexfile.open("IND.DAT",ios::out|ios::binary);

seqfile.open("EMP.DAT",ios::out|ios::binary);

do

{

cout<<"\n Enter Name: ";

cin>>Records.name;

cout<<"\n Enter Emp\_ID: ";

cin>>Records.emp\_id;

cout<<"\n Enter Salary: ";

cin>>Records.salary;

cout<<Records.name<<" "<<Records.emp\_id<<" "<<Records.salary;

seqfile.write((char\*)&Records,sizeof(Records));

Ind\_Records.emp\_id=Records.emp\_id;

Ind\_Records.position=i;

indexfile.write((char\*)&Ind\_Records,sizeof(Ind\_Records));

cout<<Ind\_Records.emp\_id<<" "<<Ind\_Records.position;

i++;

cout<<"\nDo you want to add more records?";

cin>>ch;

}while(ch=='y');

seqfile.close();

indexfile.close();

}

void Employee::Display()

{

ifstream seqfile;

ifstream indexfile;

seqfile.open("EMP.DAT",ios::in|ios::binary);

indexfile.open("IND.DAT",ios::in|ios::binary);

cout<<"\n The Contents of file are ..."<<endl;

int i=0;

while(indexfile.read((char \*)&Ind\_Records,sizeof(Ind\_Records)))

{

cout<<"insidewhile";

i=Ind\_Records.position\*sizeof(Rec);//getting pos from index file

cout<<"i:"<<i<<endl;

seqfile.seekg(i,ios::beg);//seeking record of that pos from seq.file

seqfile.read((char \*)&Records,sizeof(Records));//reading record

if(Records.emp\_id!=-1)//if rec. is not deleted logically

{ //then display it

cout<<"\nName: "<<Records.name<<flush;

cout<<"\nEmp\_ID: "<<Records.emp\_id;

cout<<"\nSalary: "<<Records.salary;

cout<<"\n";

}

}

seqfile.close();

indexfile.close();

}

void Employee::Search()

{

fstream seqfile;

fstream indexfile;

int id,pos,offset;

cout<<"\n Enter the Emp\_ID for searching the record ";

cin>>id;

indexfile.open("IND.DAT",ios::in|ios::binary);

pos=-1;

//reading index file to obtain the index of desired record

while(indexfile.read((char \*)&Ind\_Records,sizeof(Ind\_Records)))

{

if(id==Ind\_Records.emp\_id)//desired record found

{

pos=Ind\_Records.position;//seeking the position

break;

}

}

if(pos==-1)

{

cout<<"\n Record is not present in the file";

return;

}

//calculate offset using position obtained from ind. file

offset=pos\*sizeof(Records);

seqfile.open("EMP.DAT",ios::in|ios::binary);

//seeking the record from seq. file using calculated offset

seqfile.seekg(offset,ios::beg);//seeking for reading purpose

seqfile.read((char \*)&Records,sizeof(Records));

if(Records.emp\_id==-1)

{

cout<<"\n Record is not present in the file";

return;

}

else //emp\_id=desired record’s id

{

cout<<"\n The Record is present in the file and it is...";

cout<<"\n Name: "<<Records.name;

cout<<"\n Emp\_ID: "<<Records.emp\_id;

cout<<"\n Salary: "<<Records.salary;

}

seqfile.close();

indexfile.close();

}

int main()

{

Employee e;

e.Create();

e.Display();

e.Search();

return 0;

}