**ASSIGNMENT 1**

**AIM:-**

To create ADT that implement the "set" concept.

1. Add (newElement) -Place a value into the set
2. Remove (element)
3. Contains (element) Return true if element is in collection
4. Size () Return number of values in collection
5. Intersection of two sets
6. Union of two sets
7. Difference between two sets
8. Subset

**CODE:**

#include<iostream>

#include<stdlib.h>

using namespace std;

void create(int set[])

{

int n;

cout<<"\n enter the size of set : ";

cin>>n;

cout<<"\n enter the elements in the set : ";

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

cin>>set[i];

set[0]=n;

}

bool member(int set[],int num)

{

for(int i=1;i<=set[0];i++)

if(set[i]==num)

return true;

return false;

}

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

{

for(int i=1;i<=set2[0];i++)

{

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

{

set3[0]++;

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

}

}

}

void union1(int set1[],int set2[],int set4[])

{

for(int i=0;i<=set1[0];i++)

set4[i]=set1[i];

for(int i=1;i<=set2[0];i++)

{

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

{

set4[0]++;

set4[set4[0]]=set2[i];

}

}

}

void difference1(int set1[],int set2[],int set5[])

{

for(int i=1;i<=set1[0];i++)

{

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

{

set5[0]++;

set5[set5[0]]=set1[i];

}

; }

}

void contains(int set[])

{

int num;

cout<<"\n enter the element to be searched ";

cin>>num;

if((member(set,num))== true)

cout<<"\n element is present ";

else

cout<<"\n element is not present ";

}

bool subset(int seta[],int setb[])

{

for(int i=1;i<=setb[0];i++)

{

if((member(seta,setb[i]))==true)

continue;

else

return false;

}

return true;

}

void remove(int set[])

{

int pos;

cout<<"\n enter the position from which you want to remove the element : ";

cin>>pos;

if(pos<=set[0])

{

if(pos<set[0])

{

for(int i=pos;i<=set[0];i++)

{

set[i]=set[i+1];

}

set[0]--;

}

else if(pos==set[0])

{

set[0]--;

}

}

else

{

cout<<"\n entered position exceeds the size of the set " ;

}

}

void size(int set[])

{

cout<<set[0];

}

void display(int set[])

{

cout<<"\n size : "<<set[0]<<"\t";

for(int i=1;i<=set[0];i++)

{

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

}

}

int main()

{

int set1[10];

cout<<"\n FOR SET 1 ";

create(set1);

int set2[10];

cout<<"\n FOR SET 2 ";

create(set2);

int ch,c;

char choice;

do{

cout<<"\n\n -------------- OPERATION MENU ---------------- ";

cout<<"\n 1 for INTERSECTION ";

cout<<"\n 2 for UNION ";

cout<<"\n 3 for DIFFERENCE ";

cout<<"\n 4 for CONTAINS( if element is present in set or not)";

cout<<"\n 5 for SUBSET";

cout<<"\n 6 for REMOVE";

cout<<"\n 7 for SIZE";

cout<<"\n 8 for DISPLAY";

cout<<"\n 9 for EXIT";

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

cin>>ch;

switch(ch)

{

case 1:

{

int set3[1];

set3[0]=0;

cout<<"\n the intersection of two sets : \t";

intersection(set1,set2,set3);

display(set3);

break;

}

case 2:

{

int set4[set1[0]+1];

set4[0]=0;

cout<<"\n the union of two sets \t";

union1(set1,set2,set4);

display(set4);

break;

}

case 3:

{

int set5[1];

set5[0]=0;

cout<<"\n the difference of two sets \t";

difference1(set1,set2,set5);

display(set5);

break;

}

case 4:

{

cout<<"\n enter 1 for searching in set1 and 2 for searching in set2 ";

cin>>c;

switch(c)

{

case 1: contains(set1); break;

case 2: contains(set2); break;

default: cout<<"\n wrong choice entered ";

}

break;

}

case 5:

{

label:

cout<<"\n enter 1 for checking if set1 is subset of set2 else enter 2 ";

cin>>c;

switch(c)

{

case 1:

{

if(subset(set2,set1)==true )

cout<<"\n set1 is subset of set2";

else

cout<<"\n set1 is not a subset of set2";

break;

}

case 2:

{

if(subset(set1,set2)==true )

cout<<"\n set2 is subset of set1";

else

cout<<"\n set2 is not a subset of set1";

break;

}

default:

cout<<"\n wrong choice entered ";

goto label;

}

break;

}

case 6:

{

label2:

cout<<"\n enter 1 for removing element from set 1 and 2 for removal from set 2 ";

cin>>c;

switch(c)

{

case 1:

{

remove(set1);

break;

}

case 2:

{

remove(set2);

break;

}

default:

cout<<"\n wrong choice entered ";

goto label2;

}

break;

}

case 7:

{

cout<<"\n size of set 1 : ";

size(set1);

cout<<"\n size of set 2 : ";

size(set2);

break;

}

case 8:

{

display(set1);

display(set2);

break;

}

case 9:

exit(0);

default:

cout<<"\n wrong choice entered ";

}

cout<<"\n want to continue with the operation ?(y/n) :";

cin>>choice;

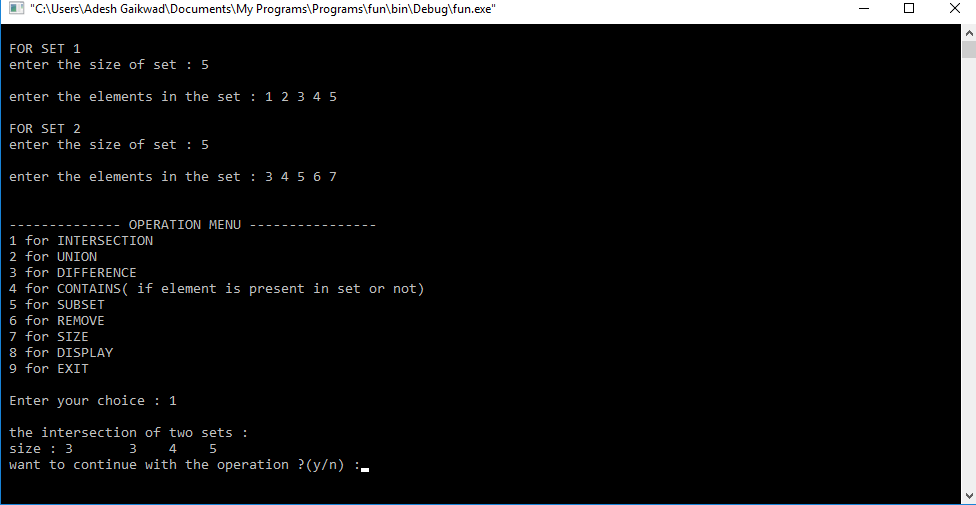
}while((choice=='y')||(choice=='Y'));

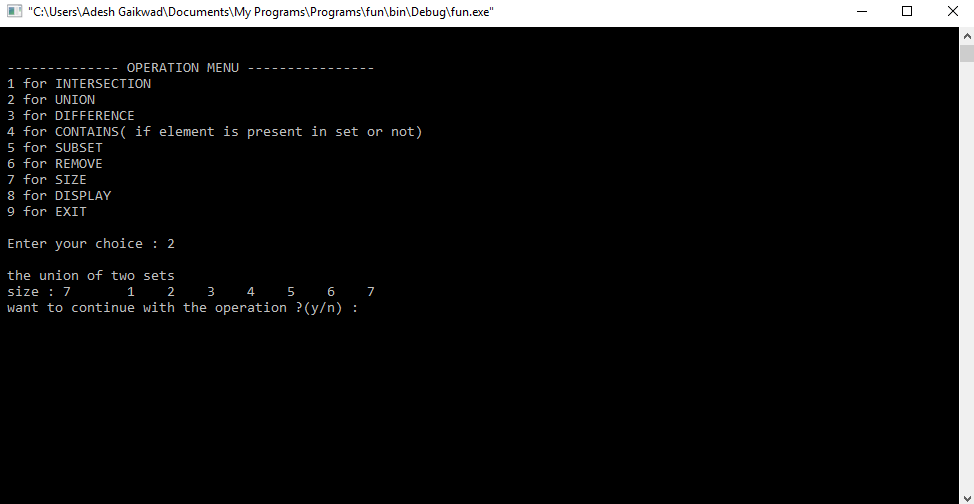
return 0;

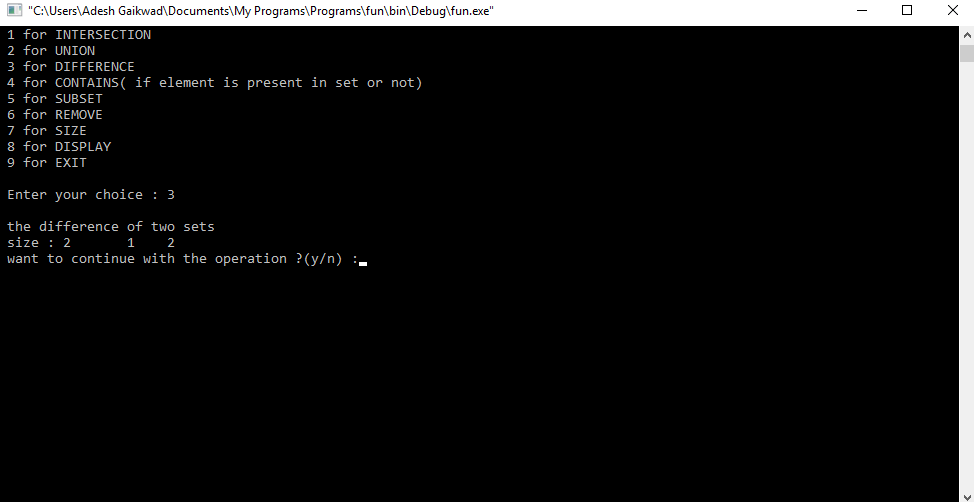
}

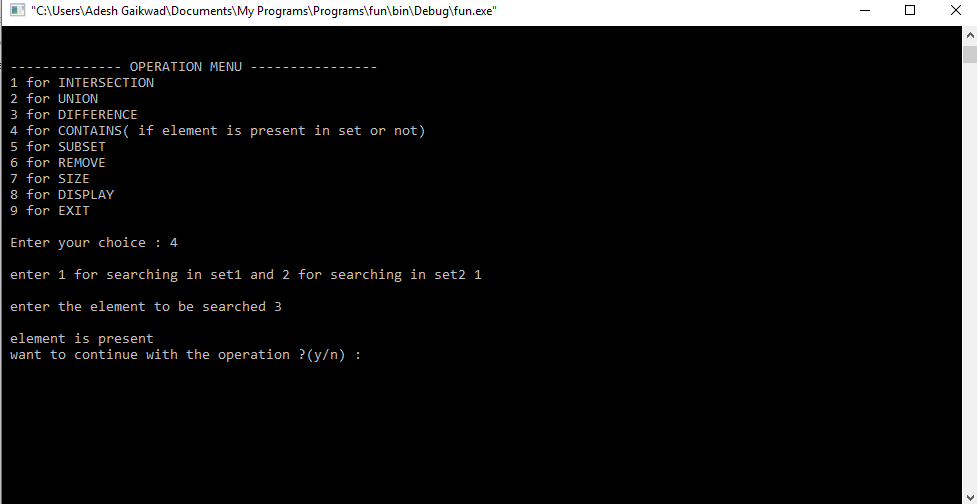
**OUTPUT:**











**ASSIGNMENT NO :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>

#include<stdlib.h>

using namespace std;

class treenode

{

int data;

int lf,rf;

treenode \*left,\*right;

treenode \*root,\*head;

public:

treenode\* nn(int);

void create();

void in();

}tree;

treenode\* treenode::nn(int z)

{

treenode \*n;

n=new treenode;

n->left=NULL;

n->right=NULL;

n->lf=1;

n->rf=1;

n->data=z;

return n;

}

void treenode::create()

{

int x;

char ch;

treenode \*n,\*temp;

if(head == NULL)

{

head=nn(-999);

head->right=head;

head->lf=head->rf=0;

}

if(root == NULL)

{

cout<<"Enter the root node's data\n";

cin>>x;

root=nn(x);

root->left=head;

root->right=head;

head->left=root;

}

do

{

cout<<"Enter the new node's data\n";

cin>>x;

n=nn(x);

temp=root;

while(temp)

{

if(n->data < temp->data)

{

if(temp->lf == 1)

{

n->left=temp->left;

temp->left=n;

temp->lf=0;

n->right=temp;

break;

}

else

{

temp=temp->left;

}

}

else

{

if(temp->rf == 1)

{

n->right=temp->right;

temp->right=n;

temp->rf=0;

n->left=temp;

break;

}

else

{

temp=temp->right;

}

}

}

cout<<"More nodes?\n";

cin>>ch;

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

}

void treenode::in()

{

treenode \*temp;

temp=root;

while(temp->lf == 0)

{

temp=temp->left;

}

while(temp != head)

{

cout<<""<<temp->data<<"\t";

if(temp->rf ==1)

{

temp=temp->right;

}

else

{

temp=temp->right;

while(temp->lf == 0)

{

temp=temp->left;

}

}

}

}

int main()

{

tree.create();

tree.in();

ret**urn 0;**

**}**

**OUTPUT:**

Enter the root node's data

56

Enter the new node's data

34

More nodes?

Y

Enter the new node's data

55

More nodes?

y

Enter the new node's data

89

More nodes?

y

Enter the new node's data

100

More nodes?

n

34 55 56 89 100

**Assignment No: 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.

**Program:**

#include<iostream>

using namespace std;

int min(int dis[],int n,int vis[])

{

int m,i,j;

m=32767;

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

{

if(dis[i]<=m && vis[i]==0)

{

m=dis[i];

j=i;

}

}

return j;

}

int main()

{

int n,i,j;

cout<<"Enter the number of vertices\n";

cin>>n;

int arr[n][n];

cout<<"Enter the adjecancy matrix\n";

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

{

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

{

cin>>arr[i][j];

}

}

cout<<"Entered matrix is:\n";

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

{

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

{

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

}

cout<<"\n";

}

int count=1;

int vis[n],dis[n];

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

{

vis[i]=0;

dis[i]=32767;

}

dis[0]=0;

while(count<=n)

{

int u=min(dis,n,vis);

vis[u]=1;

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

{

if(arr[u][j]!=0 && dis[j]> dis[u]+arr[u][j])

{

dis[j]=dis[u]+arr[u][j];

}

}

count++;

}

cout<<"After using Dijkstra's algorithm:\n";

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

{

cout<<""<<i<<" : "<<dis[i]<<"\n";

}

return 0;

}

**OUTPUT:**

Enter the number of vertices

9

Enter the adjecancy matrix

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 14 0 0 0

0 0 0 9 0 10 0 0 0

0 0 4 14 10 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

Entered matrix is:

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 14 0 0 0

0 0 0 9 0 10 0 0 0

0 0 4 14 10 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

After using Dijkstra's algorithm:

0 : 0

1 : 4

2 : 12

3 : 19

4 : 21

5 : 11

6 : 9

7 : 8

8 : 14

**Assignment 4**

**Aim:**

For a weighted graph G, find the minimum spanning tree using Prims Algorithm.

**Code:**

#include<iostream>

using namespace std;

int min(int key[],int n,int mst[])

{

int i,m=32767,h;//IMPORTANT

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

{

if(key[i]<=m && mst[i]==0)

{

m=key[i];

h=i;

}

}

cout<<h;

return h;

}

int main()

{

int n,i,j,u,v;

cout<<"Enter the number of vertices\n";

cin>>n;

int arr[n][n];

cout<<"Enter the Adjacency matrix with weights\n";

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

{

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

{

cin>>arr[i][j];

}

}

cout<<"The entered graph is:\n";

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

{

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

{

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

}

cout<<"\n";

}

int mst[n],parent[n];

int key[n];

int count=1;

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

{

mst[i]=0;

parent[i]=0;

key[i]=32767;

}

key[0]=0;

while(count<=n)

{

u=min(key,n,mst);

mst[u]=1;

cout<<""<<parent[u]<<"-"<<u<<" "<<key[u]<<"\n";

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

{

if(arr[u][i]!=0 && arr[u][i]<key[i])

{

parent[i]=u;

key[i]=arr[u][i];

}

}

count++;

}

return 0;

}

**OUTPUT:**

Enter the number of vertices

9

Enter the Adjacency matrix with weights

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 1 0 0 0

0 0 0 9 0 11 0 0 0

0 0 4 1 11 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

The entered graph is:

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 1 0 0 0

0 0 0 9 0 11 0 0 0

0 0 4 1 11 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

00-0 0

10-1 4

70-7 8

67-6 1

56-5 2

35-3 1

25-2 4

82-8 2

43-4 9

**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;

#include<vector>

#include<algorithm>

int par(int m,int parent[],int v)

{

if(m==parent[m])

return m;

else

return par(parent[m],parent,v);

}

int main()

{

int n,i,v;

pair<int,int>p1;

pair<int,pair<int,int> >p2;

vector< pair<int,pair<int,int> > >g;

cout<<"Enter the total number of sides\n";

cin>>n;

cout<<"Enter the sides(first enter the distance)\n";

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

{

cin>>p2.first>>p2.second.first>>p2.second.second;

g.push\_back(p2);

}

sort(g.begin(),g.end());

cout<<"After sorting:\n";

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

{

cout<<""<<g[i].first<<" "<<g[i].second.first<<" "<<g[i].second.second<<"\n";

}

cout<<"Enter the number of vertices of the graph\n";

cin>>v;

int parent[v];

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

{

parent[i]=i;

}

int count=0;

i=0;

cout<<"\nOutput list:\n";

while(count!=(v-1))

{

// cout<<"Side is:"<<g[i].first<<" "<<g[i].second.first<<" "<<g[i].second.second<<"\n";

int p1,p2,uu,vv;

uu=g[i].second.first;

vv=g[i].second.second;

p1=par(uu,parent,v);

p2=par(vv,parent,v);

// cout<<"Parent of"<<uu<<" is:"<<p1<<", parent of "<<vv<<" is:"<<p2<<"\n";

if(p1!=p2)

{

cout<<"Distance:"<<g[i].first<<" Edge -> "<<g[i].second.first<<" "<<g[i].second.second<<"\n";

count++;

parent[p1]=p2;

}

i++;

}

return 0;

}

**OUTPUT:**

Enter the total number of sides

14

Enter the sides(first enter the distance)

4 0 1

8 1 2

7 2 3

9 3 4

10 4 5

2 5 6

1 6 7

8 0 7

11 1 7

7 7 8

2 2 8

6 6 8

4 2 5

14 2 5

After sorting:

1 6 7

2 2 8

2 5 6

4 0 1

4 2 5

6 6 8

7 2 3

7 7 8

8 0 7

8 1 2

9 3 4

10 4 5

11 1 7

14 2 5

Enter the number of vertices of the graph

9

Output list:

Distance:1 Edge -> 6 7

Distance:2 Edge -> 2 8

Distance:2 Edge -> 5 6

Distance:4 Edge -> 0 1

Distance:4 Edge -> 2 5

Distance:7 Edge -> 2 3

Distance:8 Edge -> 0 7

Distance:9 Edge -> 3 4

**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 heap

{

int \*arr,size;

public:

heap(int x)

{

arr=new int(x);

size=x;

}

void create();

void heapify(int,int);

void heap\_s();

};

void heap::create()

{

int i;

cout<<"Enter the heap members\n";

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

{

cin>>arr[i];

}

cout<<"Entered heap is:\n";

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

{

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

}

}

void heap::heapify(int size,int i)

{

int l,r,largest;

largest=i;

l=2\*i+1;

r=2\*i+2;

if(l<size && arr[l]>arr[largest])

largest=l;

if(r<size && arr[r]>arr[largest])

largest=r;

if(largest!=i)

{

int temp=arr[i];

arr[i]=arr[largest];

arr[largest]=temp;

heapify(size,largest);

}

}

void heap::heap\_s()

{

int i;

for(i=size/2;i>=0;i--)

{

heapify(size,i);

}

cout<<"\nAfter heapifying to a max heap:\n";

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

{

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

}

for(i=size-1;i>=0;i--)

{

int temp=arr[0];

arr[0]=arr[i];

arr[i]=temp;

heapify(i,0);

}

cout<<"\nAfter using a heap sort:\n";

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

{

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

}

}

int main()

{

int n;

cout<<"Enter the heap's size\n";

cin>>n;

heap h(n);

h.create();

h.heap\_s();

return 0;

}

**OUTPUT:**

Enter the heap's size

6

Enter the heap members

12 3 4 56 2 22

Entered heap is:

12 3 4 56 2 22

After heapifying to a max heap:

56 12 22 3 2 4

After using a heap sort:

2 3 4 12 22 56

**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>

#include<stdlib.h>

using namespace std;

int size=10;

void display(int hash[])

{

int i;

cout<<"HASH TABLE:\n";

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

{

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

}

}

int main()

{

int hash[size],val,i;

char ch;

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

{

hash[i]=-111;

}

do

{

cout<<"Enter the value\n";

cin>>val;

int m=val%size;

if(hash[m] == -111)

{

hash[m]=val;

display(hash);

}

else

{

cout<<"Collision\n";

cout<<""<<hash[m]<<"\n";

if(hash[m]%10!=m)

{

int h=hash[m];

hash[m]=val;

val=h;

}

int x=1+(val%(size-1));

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

{

int y=(val+i\*x);

int z=y%size;

if(hash[z]==-111)

{

hash[z]=val;

display(hash);

break;

}

}

}

cout<<"\nWant to add more values?\n";

cin>>ch;

int ss=0;

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

{

if(hash[i]!= -111)

ss++;

}

if(ss==10)

{

cout<<"Hast table is full\n";

exit(1);

}

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

display(hash);

return 0;

}

**OUTPUT:**

Enter the value

11

HASH TABLE:

-111 11 -111 -111 -111 -111 -111 -111 -111 -111

Want to add more values?

y

Enter the value

32

HASH TABLE:

-111 11 32 -111 -111 -111 -111 -111 -111 -111

Want to add more values?

y

Enter the value

64

HASH TABLE:

-111 11 32 -111 64 -111 -111 -111 -111 -111

Want to add more values?

y

Enter the value

74

Collision

64

HASH TABLE:

-111 11 32 -111 64 -111 -111 74 -111 -111

Want to add more values?

y

Enter the value

66

HASH TABLE:

-111 11 32 -111 64 -111 66 74 -111 -111

Want to add more values?

y

Enter the value

97

Collision

74

HASH TABLE:

74 11 32 -111 64 -111 66 97 -111 -111

Want to add more values?

n

HASH TABLE:

74 11 32 -111 64 -111 66 97 -111 -111

**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 maintain data

**C++ Code:**

#include<iostream>

#include<fstream>

using namespace std;

class student

{

int roll\_num;

char div;

string name;

string address;

public:

void getdata()

{

cout<<"\n Enter the Roll Number";

cin>>roll\_num;

cout<<"\n Enter the division ";

cin>>div;

cout<<"\n Enter the Name";

fflush(stdin);

getline(cin,name);

cout<<"\n Enter the Address";

fflush(stdin);

getline(cin,address);

}

void putdata(int n)

{

student st[n];

ifstream infile;

infile.open("student.dat",ios::binary|ios::in);

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

{

infile.read((char \*)&st[i],sizeof(st[i]));

cout<<"\n Roll Number: "<<st[i].roll\_num;

cout<<"\n Division: "<<st[i].div;

fflush(stdin);

cout<<"\n Name: "<<st[i].name;

fflush(stdin);

cout<<"\n Address: "<<st[i].address;

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

}

infile.close();

}

void search\_(int n)

{

student st[n];

ifstream infile;

cout<<"\n Enter the Roll Number to be searched";

int r;

cin>>r;

infile.open("student.dat",ios::in|ios::binary);

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

{

infile.read((char \*)&st[i],sizeof(st[i]));

if(st[i].roll\_num==r)

{

cout<<"\n Found";

cout<<"\n Details: "<<endl;

cout<<"\n Roll Number: "<<st[i].roll\_num;

cout<<"\n Division: "<<st[i].div;

fflush(stdin);

cout<<"\n Name: "<<st[i].name;

fflush(stdin);

cout<<"\n Address: "<<st[i].address;

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

infile.close();

return;

}

}

cout<<"\n Not Found";

infile.close();

}

void del(int n)

{

student st[n];

int r;

cout<<"\n Enter the roll number to be deleted ";

cin>>r;

ifstream infile;

ofstream outfile;

infile.open("student.dat",ios::binary|ios::in);

outfile.open("temp.dat",ios::binary|ios::out);

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

{

infile.read((char \*)&st[i],sizeof(st[i]));

if(st[i].roll\_num==r)

{

continue;

}

else

{

outfile.write((char \*)&st[i],sizeof(st[i]));

}

}

outfile.close();

infile.close();

remove("student.dat");

int re=rename("temp.dat","student.dat");

}

};

int main()

{

int n;

cout<<"\n Enter the Number of Students";

cin>>n;

student s[n];

ofstream outfile;

outfile.open("student.dat",ios::out|ios::binary);

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

{cout<<"\n Enter the Number of Students";

s[i].getdata();

outfile.write((char \*)&s[i],sizeof(s[i]));

}

outfile.close();

int c;

student d;

do

{

cout<<"\n 1.Search";

cout<<"\n 2.Delete";

cout<<"\n 3.Display";

cout<<"\n 4.Exit";

cout<<"\n Enter Your Choice";

cin>>c;

switch(c)

{

case 1:d.search\_(n);break;

case 2:d.del(n);n=n-1;break;

case 3:d.putdata(n);break;

case 4:break;

}

}

while(c!=4);

}

**Output:**

Enter the Number of Students3

Enter the details of Student 1

Enter the Roll Number34

Enter the division A

Enter the NameRam J

Enter the AddressPune 14

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Enter the details of Student 2

Enter the Roll Number89

Enter the division B

Enter the NameSham K

Enter the AddressKondhwa

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Enter the details of Student 3

Enter the Roll Number56

Enter the division A

Enter the NameDavid M

Enter the AddressPune

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.Search

2.Delete

3.Display

4.Exit

Enter Your Choice1

Enter the Roll Number to be searched89

Found

Details:

Roll Number: 89

Division: B

Name: Sham K

Address: Kondhwa

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **ASSIGNMENT 09**

**Aim**:

Department maintains a employee information. The file contains 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 main the data.

**Code:**

#include<iostream>

#include<fstream>

using namespace std;

typedef struct emp\_records

{

int emp\_id;

string name;

long int sal;

}rec;

typedef struct index\_file

{

int id;

int pos;

}index;

class employee

{

rec record;

index index\_record;

public:

void getdata()

{

ofstream seqfile;

ofstream indexfile;

int i=0;

indexfile.open("index.dat",ios::out|ios::binary);

seqfile.open("seq.dat",ios::out|ios::binary);

char ch;

do

{

cout<<"\n Enter the Employee ID"<<endl;

cin>>record.emp\_id;

cout<<"\n Enter the Name"<<endl;

cin>>record.name;

cout<<"\n Enter the Salary"<<endl;

cin>>record.sal;

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

index\_record.id=record.emp\_id;

index\_record.pos=i;

indexfile.write((char\*)&index\_record,sizeof(index\_record));

i++;

cout<<"\n Continue? (y/n)"<<endl;

cin>>ch;

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

indexfile.close();

seqfile.close();

}

void display()

{

ifstream seqfile;

ifstream indexfile;

seqfile.open("seq.dat",ios::in|ios::binary);

indexfile.open("index.dat",ios::in|ios::binary);

int i=0;

while(indexfile.read((char\*)&index\_record,sizeof(index\_record)))

{

i=index\_record.pos\*sizeof(record);

seqfile.seekg(i,ios::beg);

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

if(record.emp\_id!=-1)

{

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

cout<<"\n EMP ID: "<<record.emp\_id;

cout<<"\n Salary: "<<record.sal;

cout<<endl;

}

}

seqfile.close();

indexfile.close();

}

int Search()

{

ifstream seqfile;

ifstream indexfile;

int pos=-1;

int offset;

int x;

cout<<"\n Enter the ID";

cin>>x;

indexfile.open("index.dat",ios::in|ios::binary);

seqfile.open("seq.dat",ios::in|ios::binary);

while(indexfile.read((char\*)&index\_record,sizeof(index\_record)))

{

if(index\_record.id==x)

{

pos=index\_record.pos;

break;

}

}

if(pos==-1)

{

cout<<"\n Not Present in the File";

return 0;

}

// cout<<pos<<endl;

offset=pos\*sizeof(record);

seqfile.seekg(offset,ios::beg);

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

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

{

cout<<"\n Data was deleted"<<endl;

}

else

{

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

cout<<"\n EMP ID: "<<record.emp\_id;

cout<<"\n Salary: "<<record.sal;

cout<<endl;

return(offset);

}

}

void del()

{

int offset=Search();

cout<<offset<<endl;

fstream seqfile;

seqfile.open("seq.dat",ios::binary|ios::in|ios::out);

seqfile.seekg(offset,ios::beg);

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

cout<<offset<<endl;

record.emp\_id=-1;

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

cout<<"\n EMP ID: "<<record.emp\_id;

cout<<"\n Salary: "<<record.sal;

cout<<endl;

//offset=offset-(sizeof(record));

//cout<<offset<<endl;

seqfile.seekp(offset,ios::beg);

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

cout<<"\n Deleted Successfully"<<endl;

seqfile.close();

}

};

int main()

{

employee e;

e.getdata();

e.display();

e.Search();

e.del();

e.display();

}

**Output:**

Enter the Employee ID

1

Enter the Name

abc

Enter the Salary

10000

Continue? (y/n)

y

Enter the Employee ID

2

Enter the Name

dhfg

Enter the Salary

2384734

Continue? (y/n)

y

Enter the Employee ID

3

Enter the Name

weqq

Enter the Salary

56000

Continue? (y/n)

n

Name: abc

EMP ID: 1

Salary: 10000

Name: dhfg

EMP ID: 2

Salary: 2384734

Name: weqq

EMP ID: 3

Salary: 56000

Enter the ID3

Name: weqq

EMP ID: 3

Salary: 56000

Enter the ID2

Name: dhfg

EMP ID: 2

Salary: 2384734

32

32

Name: dhfg

EMP ID: -1

Salary: 2384734

Deleted Successfully

Name: abc

EMP ID: 1

Salary: 10000

Name: weqq

EMP ID: 3

Salary: 56000

**PROJECT:**

**Code:**

//SMART AND EFFECTIVE PARKING SOLUTION

#include<iostream>

#include<vector>

#include<stdlib.h>

using namespace std;

class node

{

int data;

node \*next;

node \*prev;

public:

node\* nn(int);

void create1(int uu,node\*\* ptr);

void create(int,int,node\*\* ptr);

void display(node\*\* ptr,int);

}nnn;

node \*temp,\*start=NULL,\*n,\*temp1;

node\* node::nn(int x)

{

n=new node;

n->data=x;

n->next=NULL;

n->prev=NULL;

return n;

}

void node::create1(int uu,node\*\* ptr)

{

if(ptr[uu]==NULL)

{

n=nn(uu);

ptr[uu]=n;

}

}

void node::create(int uu,int m,node\*\* ptr)

{

//cout<<"u being passed:"<<uu<<"\n";

temp=ptr[uu];

n=nn(m);

while(temp->next != NULL)

{

temp=temp->next;

}

temp->next=n;

n->prev=temp;

}

void node::display(node\*\* ptr,int j)

{

temp=ptr[j];

while(temp->next != NULL)

{

temp=temp->next;

}

cout<<"The route is:\n";

while(temp->prev != NULL)

{

cout<<"p"<<temp->data<<"\t";

temp=temp->prev;

}

}

int min(int dis[],int n,int vis[])

{

int m,i,j;

m=32767;

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

{

if(dis[i]<=m && vis[i]==0)

{

m=dis[i];

j=i;

}

}

return j;

}

int main()

{

int n,i,j;

vector<int>v;

vector<int>::iterator it;

vector<int>v1;

vector<int>::iterator it1;

cout<<"Enter the number of vertices\n";

cin>>n;

node \*ptr[n];

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

{

ptr[i]=NULL;

}

int arr[n][n];

cout<<"Enter the adjecancy matrix\n";

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

{

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

{

cin>>arr[i][j];

}

}

int count=1;

int vis[n],dis[n],parent[n];

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

{

vis[i]=0;

dis[i]=32767;

parent[i]=0;

}

dis[0]=0;

parent[0]=-1;

while(count<=n)

{

v.clear();

int u=min(dis,n,vis);

vis[u]=1;

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

{

if(arr[u][j]!=0 && dis[j]> dis[u]+arr[u][j])

{

dis[j]=dis[u]+arr[u][j];

parent[j]=u;

}

}

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

cout<<""<<u<<": \t";

nnn.create1(u,ptr);

int m=u;

while(parent[u]!=-1)

{

nnn.create(m,u,ptr);

v.push\_back(u);

u=parent[u];

}

nnn.create(m,0,ptr);

v.push\_back(0);

it=v.end();

it--;

while(it!=(v.begin()))

{

cout<<"p"<<\*it<<" -\t";

it--;

}

cout<<"p"<<\*it<<"\n";

count++;

}

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

{

vis[i]=0;

}

int z=0;

while(1)

{

int mm=32767;

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

{

if(dis[i]<mm && vis[i]==0)

{

mm=dis[i];

j=i;

}

}

vis[j]=1;

char ch;

if(z<n-1)

{

z++;

cout<<"The nearest parking space is:"<<j<<"\n";

cout<<"Is the parking slot really vacant?\n";

cin>>ch;

}

else

{

cout<<"SORRY! parking is full\n";

exit(1);

}

if(ch=='y' || ch=='Y')

{

cout<<"Parking space allocated\n";

cout<<"Parking slot is:p"<<j<<"\n";

nnn.display(ptr,j);

exit(1);

}

else

{

continue;

}

}

return 0;

}

**Output:**

Enter the number of vertices

9

Enter the adjecancy matrix

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 14 0 0 0

0 0 0 9 0 10 0 0 0

0 0 4 14 10 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

-------------------------------------------------------

0: p0

-------------------------------------------------------

1: p0 - p1

-------------------------------------------------------

7: p0 - p7

-------------------------------------------------------

6: p0 - p7 - p6

-------------------------------------------------------

5: p0 - p7 - p6 - p5

-------------------------------------------------------

2: p0 - p1 - p2

-------------------------------------------------------

8: p0 - p1 - p2 - p8

-------------------------------------------------------

3: p0 - p1 - p2 - p3

-------------------------------------------------------

4: p0 - p7 - p6 - p5 - p4

The nearest parking space is:1

Is the parking slot really vacant?

N

The nearest parking space is:7

Is the parking slot really vacant?

n

The nearest parking space is:6

Is the parking slot really vacant?

n

The nearest parking space is:5

Is the parking slot really vacant?

y

Parking space allocated

Parking slot is:p5

The route is:

p0 p7 p6 p5

