**Assignment No -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.Subset**

**Code -**

#include<iostream>

#include <cstdlib>

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()

{

system("COLOR FC");

system("cls");

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

int c;

do

{

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

cout<<"\n1.Create\n2.Display\n3.Intersection\n4.Union\n5.A-B\n6.B-A\n7.Check subset\n8.Exit\n";

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

cout<<"\nEnter your choice - ";

cin>>c;

switch(c)

{

case 1:{

int s1,s2;

cout<<"\nEnter number elements of set A -";

cin>>s1;

create(set1,s1);

cout<<"\nEnter number elements of set B - ";

cin>>s2;

create(set2,s2);

}

break;

case 2:{

cout<<"\nThe elements of set A are - ";

display(set1);

cout<<"\nThe elements of set B are - ";

display(set2);

}

break;

case 3:{

intersection(set1,set2,set\_int);

cout<<"\nThe intersection of A and B is - ";

display(set\_int);

}

break;

case 4:{

unions(set1,set2,set\_union);

cout<<"\nThe union of A and B is - ";

display(set\_union);

}

break;

case 5:{

diff(set1,set2,set\_diff);

cout<<"\nThe difference between A and B is - ";

display(set\_diff);

}

break;

case 6:{

diff(set2,set1,set\_diff);

cout<<"\nThe difference between B and A is -";

display(set\_diff);

}

break;

case 7:{

int n;

cout<<"\nEnter number elements of subset - ";

cin>>n;

create(set\_s,n);

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

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

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

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

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

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

else

cout<<"\nThe 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 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 <cstdlib>**

**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;//current node**

**tnode \*p=NULL;//parent node of current**

**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()**

**{**

**system("COLOR FC");**

**system("cls");**

**char r;**

**do**

**{**

**int flag=0;**

**root=NULL;**

**char op;**

**do**

**{**

**if(flag==0)**

**{**

**int n;**

**cout<<"\nEnter the total number of Elements - ";**

**cin>>n;**

**int a[n];**

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

**cout<<"\n1. Insert\n2. Display Ascending\n3. Display Descending\n";**

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

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

**cin>>c;**

**switch(c)**

**{**

**case 1: {**

**int item;**

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

**cout<<"\nEnter the Data - ";**

**cin>>item;**

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

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

**}**

**break;**

**case 2: {**

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

**root->inorder(root);**

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

**}**

**break;**

**case 3: {**

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

**root->reverse\_inorder(root);**

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

**}**

**break;**

**default:cout<<"\nInvalid\n";**

**}**

**cout<<"\nDo you want to continue - ";**

**cin>>op;**

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

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

**return 0;**

**}**

**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.

**Code –**

**#include<iostream>**

**#include<stdlib.h>**

**#include<string.h>**

**using namespace std;**

**struct node**

**{ string vertex;**

**int time;**

**node \*next;**

**};**

**class adjmatlist**

**{ int m[10][10],n,i,j; char ch; string v[20]; node \*head[20]; node \*temp=NULL;**

**public:**

**adjmatlist()**

**{ for(i=0;i<20;i++)**

**{ head[i]=NULL; }**

**}**

**void getgraph();**

**void adjlist();**

**void displaym();**

**void displaya();**

**};**

**void adjmatlist::getgraph()**

**{**

**cout<<"\n enter no. of cities(max. 20)";**

**cin>>n;**

**cout<<"\n enter name of cities";**

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

**cin>>v[i];**

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

**{**

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

**{ cout<<"\n if path is present between city "<<v[i]<<" and "<<v[j]<<" then press enter y otherwise n";**

**cin>>ch;**

**if(ch=='y')**

**{**

**cout<<"\n enter time required to reach city "<<v[j]<<" from "<<v[i]<<" in minutes";**

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

**}**

**else if(ch=='n')**

**{ m[i][j]=0; }**

**else**

**{ cout<<"\n unknown entry"; }**

**}**

**}**

**adjlist();**

**}**

**void adjmatlist::adjlist()**

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

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

**{ node \*p=new(struct node);**

**p->next=NULL;**

**p->vertex=v[i];**

**head[i]=p; cout<<"\n"<<head[i]->vertex;**

**}**

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

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

**{**

**if(m[i][j]!=0)**

**{**

**node \*p=new(struct node);**

**p->vertex=v[j];**

**p->time=m[i][j];**

**p->next=NULL;**

**if(head[i]->next==NULL)**

**{ head[i]->next=p; }**

**else**

**{ temp=head[i];**

**while(temp->next!=NULL)**

**{ temp=temp->next; }**

**temp->next=p;**

**}**

**}**

**}**

**}**

**}**

**void adjmatlist::displaym()**

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

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

**{ cout<<"\t"<<v[j]; }**

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

**{ cout<<"\n "<<v[i];**

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

**{ cout<<"\t"<<m[i][j];**

**}**

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

**}**

**}**

**void adjmatlist::displaya()**

**{**

**cout<<"\n adjacency list is";**

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

**{**

**if(head[i]==NULL)**

**{ cout<<"\n adjacency list not present"; break; }**

**else**

**{**

**cout<<"\n"<<head[i]->vertex;**

**temp=head[i]->next;**

**while(temp!=NULL)**

**{ cout<<"-> "<<temp->vertex;**

**temp=temp->next; }**

**}**

**}**

**cout<<"\n path and time required to reach cities is";**

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

**{**

**if(head[i]==NULL)**

**{ cout<<"\n adjacency list not present"; break; }**

**else**

**{**

**temp=head[i]->next;**

**while(temp!=NULL)**

**{ cout<<"\n"<<head[i]->vertex;**

**cout<<"-> "<<temp->vertex<<"\n [time required: "<<temp->time<<" min ]";**

**temp=temp->next; }**

**}**

**}**

**}**

**int main()**

**{ int m;**

**adjmatlist a;**

**while(1)**

**{**

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

**cout<<"\n\n Enter the choice";**

**cout<<"\n 1.Enter graph";**

**cout<<"\n 2.display adjacency matrix for cities";**

**cout<<"\n 3.display adjacency list for cities";**

**cout<<"\n 4.exit";**

**cin>>m;**

**switch(m)**

**{ case 1: a.getgraph();**

**break;**

**case 2: a.displaym();**

**break;**

**case 3: a.displaya();**

**break;**

**case 4: exit(0);**

**default: cout<<"\n unknown choice";**

**}**

**}**

**return 0;**

**}**

**Assignment No – 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 No - 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;**

**class graph**

**{**

**int a[100][100];**

**int v;**

**public:**

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

**{**

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

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

**else**

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

**}**

**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;**

**}**

**};**

**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\n";**

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

**cout<<"Enter your choice: ";**

**cin>>c;**

**switch(c)**

**{**

**case 1: {**

**int n1,n2;**

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

**cin>>n1>>n2;**

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

**}**

**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;**

**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');**

**return 0;**

**}**