**PROGRAM-01**

**AIM:** Menu driven program to perform operations on an array.

(insert,delete,display,search,sort)

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

int choice,j,i=0,a[50],b[10],s;

void main()

{

printf("\nEnter the size of array :");

scanf("%d",&s);

printf("Enter the elements of array:");

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

{

scanf("%d",&a[i]);

}

menu();

getch();

}

menu()

{

printf("\n1. Insert");

printf("\n2. Delete");

printf("\n3. Sort");

printf("\n4. Search");

printf("\n5. Display");

printf("\n6. Exit");

printf("\nEnter your choice :");

scanf("%d",&choice);

switch(choice)

{

case 1: insert();

break;

case 2:

delete();

break;

case 3: sort();

break;

case 4: search();

break;

case 5 : display();

break;

case 6 :

exit(1);

}

}

search()

{ int ele,loc;

printf("\nEnter Element to be searched : ");

scanf("%d",&ele);

printf("Elements Found at :");

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

{ if(ele==a[i])

printf("%d ",i+1);

}

getch();

menu();

}

sort()

{

printf("\nCurrent Array : ");

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

printf("\t%d",a[i]);

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

{

for(j=i+1;j<s;j++)

if (a[i]>a[j])

{ a[i]=a[i]+a[j];

a[j]=a[i]-a[j];

a[i]=a[i]-a[j];

}

}

printf("\nSorted Array : ");

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

printf("\t%d",a[i]);

getch();

menu();

}

insert()

{

int loc,ele;

printf("\nEnter the location : ");

scanf("%d",&loc);

printf("Enter the element to be inserted : ");

scanf("%d",&ele);

for(i=s;i>loc-1;i--)

a[i]=a[i-1];

a[loc-1]=ele;

printf("\nElement Inserted");

printf("\nNew array after insertion : ");

s++;

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

printf("\t%d",a[i]);

getch();

menu();

}

delete()

{ int loc;

printf("\nEnter location : ");

scanf("%d",&loc);

for(i=loc-1;i<s-1;i++)

{

a[i]=a[i+1];

}

a[s-1]=NULL;

printf("\nElement Deleted");

printf("\nNew array after deletion : ");

s--;

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

printf("\t%d ",a[i]);

getch();

menu();

}

display()

{

int i;

if(s==0)

{

printf("No elements to display!");

return;

}

printf("Array elements are :");

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

{

printf("%d\t",a[i]);

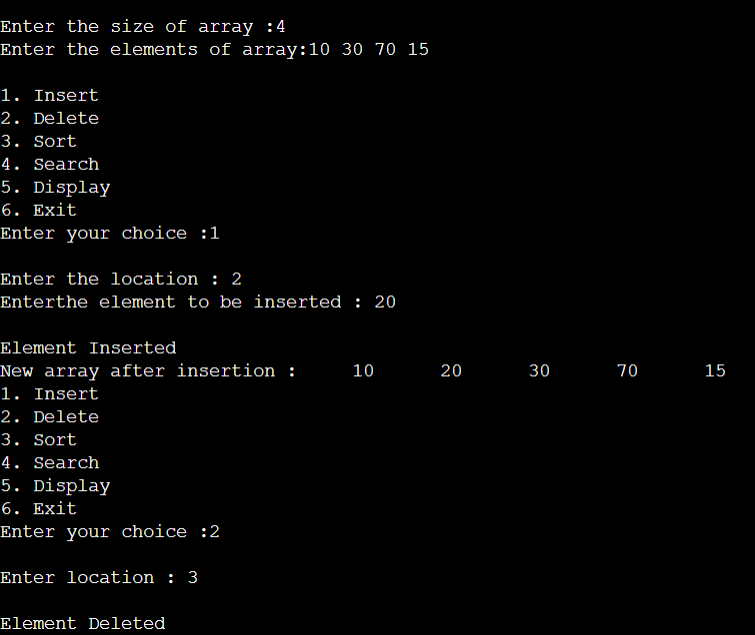
}

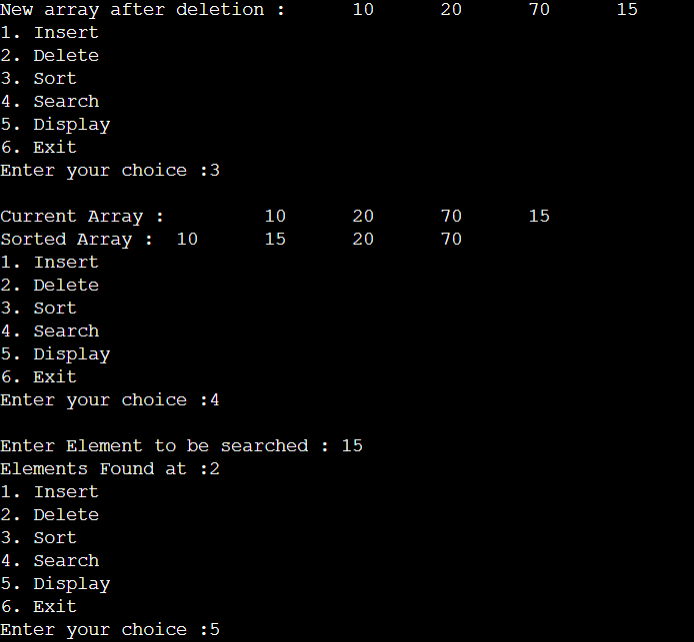
getch();

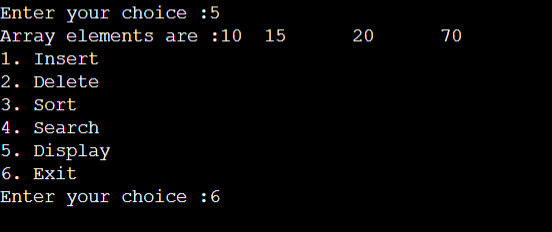
menu();

}

**OUTPUT:**

****

****

****

**PROGRAM-02**

**AIM:** Merge two sorted array into a third array.

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int a[50],b[50],c[50],m,n,i,j,k;

printf("Enter the size of 1st array:");

scanf("%d",&m);

printf("Enter the elements of 1st array in sorted order:");

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

scanf("%d",&a[i]);

printf("Enter the size of 2nd array:");

scanf("%d",&n);

printf("Enter the elements of 2nd array in sorted order:");

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

scanf("%d",&b[i]);

i=0,j=0;

while(i<m && j<n)

{

if(a[i]<b[j])

{

c[k]=a[i];

i++;

}

else

{

c[k]=b[j];

j++;

}

k++;

}

if(i>=m)

{

while(j<n)

{

c[k]=b[j];

j++;

k++;

}

}

if(j>=n)

{

while(i<m)

{

c[k]=a[i];

i++;

k++;

}

}

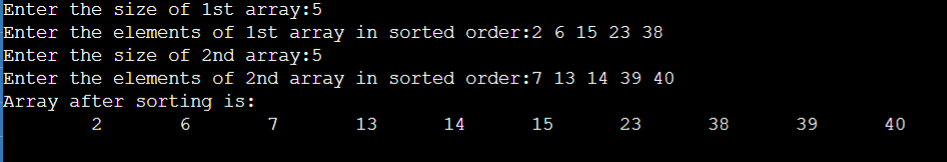
printf("Array after sorting is:\n");

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

printf("\t%d",c[k]);

}

**OUTPUT:**

****

**PROGRAM-03 AIM:** Stack Implementation using Arrays.

**SOURCE CODE:**

#include<stdlib.h>

#include<stdio.h>

void Push(int ele);

void Pop();

void Display();

void Peek();

int stack[50],top=-1,n;

void main()

{

int stack[50],i,n,ch,ele;

printf("Enter the size of the stack:");

scanf("%d",&n);

while(1)

{

printf("\nMenu:\n 1.Push\n 2.Display\n 3.Pop\n 4.Peek\n 5.Exit \n");

printf("Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("Enter the element to be inserted:");

scanf("%d",&ele);

Push(ele);

break;

case 2: Display();

break;

case 3: Pop();

break;

case 4: Peek();

break;

case 5: exit(0);

default:printf("Wrong selection!!");

}

}

}

void Display()

{

int i=0;

if(top==-1)

printf("Stack is empty!");

else

{

printf("The stack elements are:");

for(i=top;i>-1;i--)

{

printf("\t%d",stack[i]);

}

}

}

void Push(int ele)

{

if(top!=50)

{

top=top+1;

stack[top]=ele;

printf("Insertion successfull!!");

}

else

printf("Can't insert,stack is full!!");

}

void Pop()

{

int ele;

if(top==-1)

printf("Coudn't retrieve data,stack is empty!!\n");

else

{

ele=stack[top];

printf("Deleted:%d",ele);

top=top-1;

}

}

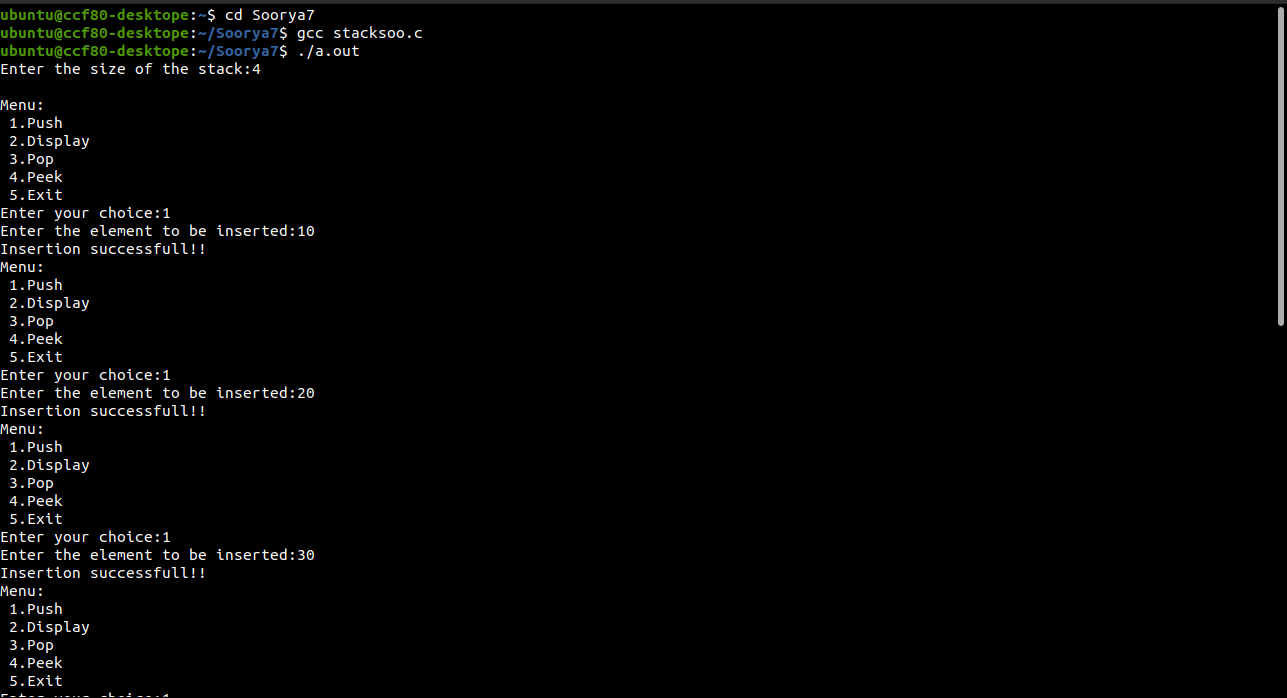
void Peek()

{

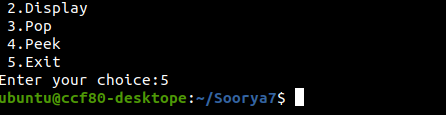
printf("The top most element of the stack is:\t%d",stack[top]);

}

**OUTPUT:**







**PROGRAM-04**

**AIM:** QueueImplementation using arrays**.**

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

void enq(int val);

void deq();

void display();

void peek();

int queue[50],front=-1,rear=-1;

void main()

{

int val,ch;

while(1)

{

printf("\n Menu:\n 1.Enqueue\n 2.Dequeue\n 3.Display\n 4.Peek\n 5.Exit\n");

printf("Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:printf("Enter the element to insert:");

scanf("%d",&val);

enq(val);

break;

case 2: deq();

break;

case 3: display();

break;

case 4: peek();

break;

case 5:exit(0);

default:printf("Wrong selection!!");

}

}

}

void enq(int val)

{

if(rear!=50)

{

front=0;

rear++;

queue[rear]=val;

printf("Insertion successfull!!\n");

}

else

printf("Queue is full! Imsertion not possible!");

}

void deq()

{

if(front==rear)

printf("Queue is empty.Deletion is not possible.!\n");

else

{

printf("Deleted:%d",queue[front]);

front++;

if(front==rear)

front=rear=-1;

}

}

void display()

{

if(rear==-1)

printf("Queue is empty!!");

else

{

int i;

printf("Queue elements are:");

for(i=front;i<=rear;i++)

printf("\t%d",queue[i]);

}

}

void peek()

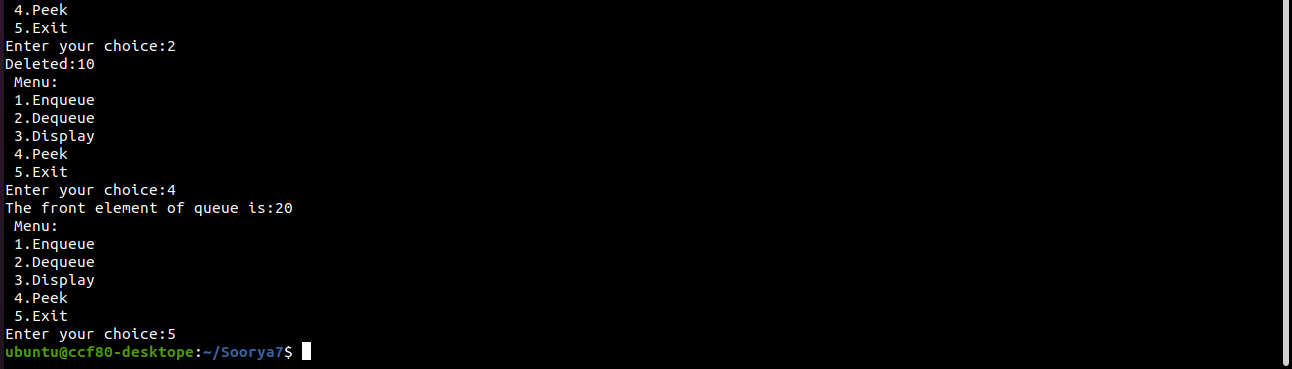
{

printf("The front element of queue is:%d",queue[front]);

}

**OUTPUT:**



****

**PROGRAM-05**

**AIM:** Circular queue Implementation using Arrays.

**SOURCE CODE:**

#include<stdlib.h>

void cenqueue();

void cdequeue();

void display();

int isEmpty();

int isFull();

int rear=-1;

int front=-1;

int queue[100],n,x;

void main()

{

int ch;

printf("Enter the size of the queue:");

scanf("%d",&n);

do

{

printf("\nCIRCULAR QUEUE OPERATIONS\n1-Enqueue\n2-Dequeue\n3-Display\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:cenqueue();

break;

case 2:cdequeue();

break;

case 3:display();

break;

default:printf("Invalid choice\n");

break;

}

}while(ch<4);

}

void cenqueue()

{

if(isFull())

{

printf("Queue is full");

return ;

}

else

{

if(front==-1)

front=0;

printf("Enter the element to be inserted:");

scanf("%d",&x);

rear=(rear+1)%n;

queue[rear]=x;

printf("Element inserted successfully!");

}

}

void cdequeue()

{

if(isEmpty())

{

printf("Queue is empty");

return;

}

else

{

x=queue[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else

{

front=(front+1)%n;

}

printf("Element deleted is %d",x);

}

}

void display()

{

int i;

if (isEmpty())

printf(" \n Empty Queue\n");

else {

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

printf("%d ", queue[i]);

}

printf("%d ", queue[i]);

}

}

int isEmpty()

{

if(front==-1)

return 1;

else

return 0;

}

int isFull()

{

if((front==0 && rear==n-1) || (front==rear+1))

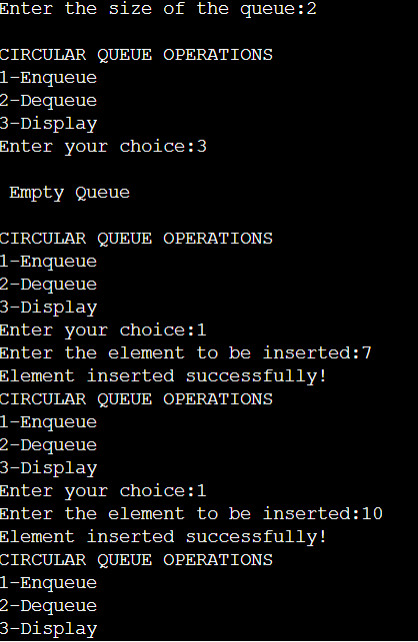
return 1;

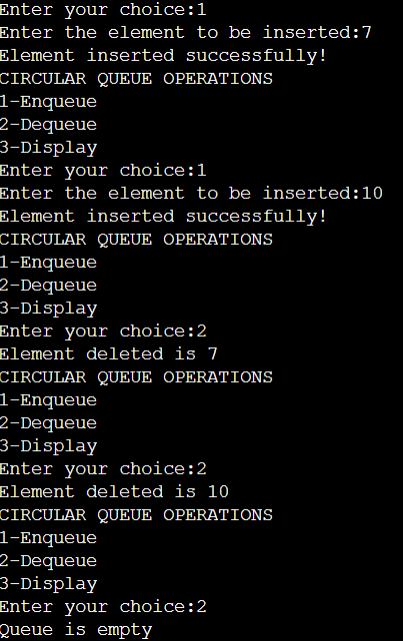
else

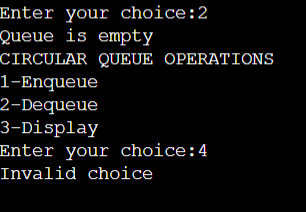
return 0;

}

**OUTPUT:**

****

****

****

**PROGRAM-06**

**AIM:** Operations on Singly Linked List.

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

void display();

int in\_b();

void in\_e();

void in\_bw();

void del\_b();

void del\_e();

void del\_bw();

struct node

{

int data;

struct node\* link;

};

typedef struct node node1;

node1 \* start = NULL;

node1 \*create()

{

node1 \*nptr=((node1\*)malloc(sizeof(node1)));

if(nptr==NULL)

{

printf("Memory Overflow");

return 0;

}

else

return nptr;

}

void main()

{

int x;

printf("Singly linked list\n");

printf("MENU\n");

printf("1.Display.\n2.Inserting a node at begining.\n3.Inserting a node at end.\n4.Inserting a node in between two nodes.\n5.Deleting a node at the begining of the list.\n6.Deleting a node at the end of the list.\n7.Deleting a node in between two nodes.\n8.Exit.\n");

while(1)

{

printf("\nEnter your choice: ");

scanf("%d",&x);

switch(x)

{

case 1:

display();

break;

case 2:

in\_b();

break;

case 3:

in\_e();

break;

case 4:

in\_bw();

break;

case 5:

del\_b();

break;

case 6:

del\_e();

break;

case 7:

del\_bw();

break;

case 8:

exit(0);

}

}

}

void display()

{

struct node \*ptr=start;

if(ptr==NULL)

{

printf("List is empty\n");

}

printf("Nodes in the linked list are: ");

while (ptr!=NULL)

{

printf("%d \t",ptr -> data);

ptr=ptr->link;

}

}

int in\_b()

{

int val;

node1 \*nptr=create();

printf("Enter the element to be inserted:");

scanf("%d",&val);

nptr -> data=val;

if(start==NULL)

{

start=nptr;

nptr->link=NULL;

}

else

{

nptr->link=start;

start=nptr;

}

printf("Insertion successfull.!!");

}

void in\_e()

{

node1 \*temp;

node1 \*nptr=create();

int val;

printf("Enter the element:");

scanf("%d",&val);

nptr->data=val;

nptr->link=NULL;

temp=start;

while(temp->link!=NULL)

{

temp=temp->link;

}

temp->link=nptr;

printf("Insertion succesfull.!!");

}

void in\_bw()

{

node1 \*temp;

node1 \*nptr=create();

int val,pos,i;

printf("Enter the element and position to be inserted:");

scanf("%d %d",&val,&pos);

nptr->data=val;

nptr->link=NULL;

temp=start;

for(i=1;i<(pos-1);i++)

temp=temp->link;

nptr->link=temp->link;

temp->link=nptr;

printf("Insertion Successfull.!!");

}

void del\_b()

{

node1 \*temp;

if(start==NULL)

printf("List is empty!!");

else

{

temp=start;

start=start->link;

free(temp);

}

printf("Deleted!");

}

void del\_e()

{

node1 \*temp,\*prev;

temp=start;

while(temp->link!=NULL)

{

prev=temp;

temp=temp->link;

}

prev->link=NULL;

free(temp);

printf("Deleted.!!");

}

void del\_bw()

{

node1 \*temp,\*prev;

int i,pos;

printf("Enter the position of the node to be deleted:");

scanf("%d",&pos);

temp=start;

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

{

prev=temp;

temp=temp->link;

}

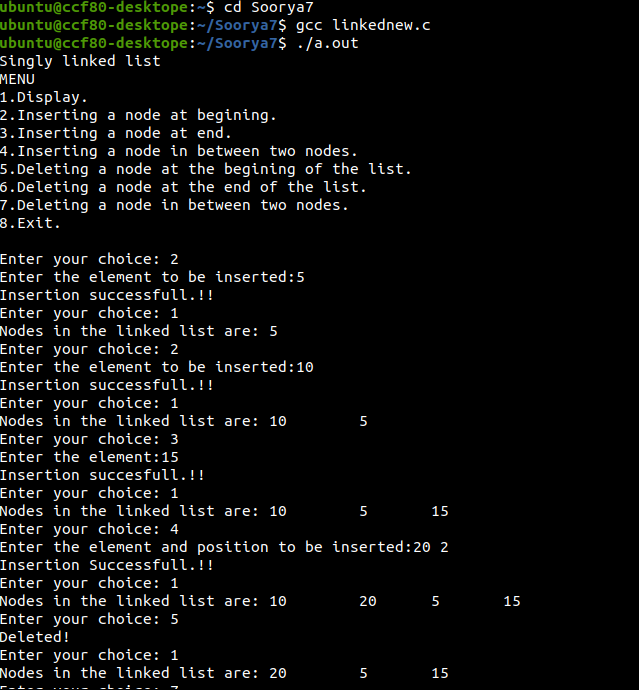
prev->link=temp->link;

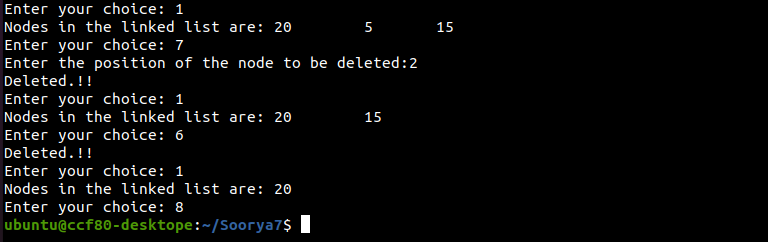
free(temp);

printf("Deleted.!!");

}

**OUTPUT:**

****



**PROGRAM-07**

**AIM:** Binary Search Tree operations.

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

struct btnode

{

int value;

struct btnode \*l;

struct btnode \*r;

}\*root = NULL, \*temp = NULL, \*t2, \*t1;

void delete1();

void insert();

void delete();

void inorder(struct btnode \*t);

void create();

void search(struct btnode \*t);

void search1(struct btnode \*t,int data);

void main()

{

int ch;

printf("\nOPERATIONS ---");

printf("\n1 - Insert an element into tree\n");

printf("2 - Delete an element from the tree\n");

printf("3 - Inorder Traversal\n");

printf("4 - Exit\n");

while(1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

inorder(root);

break;

case 4:

exit(0);

default :

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

void insert()

{

create();

if (root == NULL)

root = temp;

else

search(root);

}

void create()

{

int data;

printf("Enter data of node to be inserted : ");

scanf("%d", &data);

temp = (struct btnode\*)malloc(1\*sizeof(struct btnode));

temp->value = data;

temp->l = temp->r = NULL;

}

void search(struct btnode \*t)

{

if ((temp->value > t->value) && (t->r != NULL))

else if ((temp->value > t->value) && (t->r == NULL))

t->r = temp;

else if ((temp->value < t->value) && (t->l != NULL))

else if ((temp->value < t->value) && (t->l == NULL))

t->l = temp;

void inorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

if (t->l != NULL)

inorder(t->l);

printf("%d -> ", t->value);

if (t->r != NULL)

inorder(t->r);

}

void delete()

{

int data;

if (root == NULL)

{

printf("No elements in a tree to delete");

return;

}

printf("Enter the data to be deleted : ");

scanf("%d", &data);

search1(root, data);

}

void search1(struct btnode \*t, int data)

{

if ((data>t->value))

{

t1 = t;

search1(t->r, data);

}

else if ((data < t->value))

{

t1 = t;

search1(t->l, data);

}

else if ((data==t->value))

{

delete1(t);

}

}

void delete1(struct btnode \*t)

{

int k;

if ((t->l == NULL) && (t->r == NULL))

{

if (t1->l == t)

{

t1->l = NULL;

}

else

{

t1->r = NULL;

}

t = NULL;

free(t);

return;

}

else if ((t->r == NULL))

{

if (t1 == t)

{

root = t->l;

t1 = root;

}

else if (t1->l == t)

{

t1->l = t->l;

}

else

{

t1->r = t->l;

}

t = NULL;

free(t);

return;

}

else if (t->l == NULL)

{

if (t1 == t)

{

root = t->r;

t1 = root;

}

else if (t1->r == t)

t1->r = t->r;

else

t1->l = t->r;

t == NULL;

free(t);

return;

}

else if ((t->l != NULL) && (t->r != NULL))

{

k = smallest(t->r);

search1(root, k);

t->value = k;

}

}

int smallest(struct btnode \*t)

{

if (t->l != NULL)

{

return(smallest(t->l));

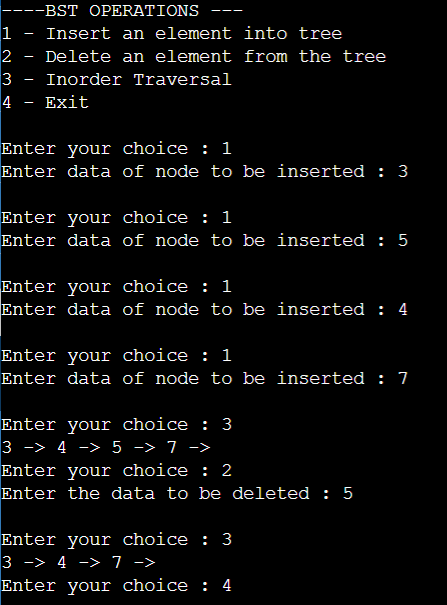
}

else

return (t->value);

}

**OUTPUT:**

****

**PROGRAM-08**

**AIM:** Operations onDoubly Linked List.

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

void display();

int in\_b();

void in\_e();

void in\_bw();

void dispf();

void dispb();

void del\_b();

void del\_e();

void del\_bw();

struct link

{

int data;

struct link \*prev;

struct link \*next;

};

typedef struct link node;

node \*start,\*head,\*tail=NULL;

node \*create()

{

node \*nptr=((node\*)malloc(sizeof(node)));

if(nptr==NULL)

{

printf("Memory Overflow");

return 0;

}

else

return nptr;

}

void main()

{

int x;

printf("Singly linked list\n");

printf("MENU\n");

printf("1.Display.\n2.Inserting a node at begining.\n3.Insert a node at the end.\n4.Inserting in between two nodes.\n5.Display forward.\n6.Display backward.\n7.Deletion at begining\n8.Deletion at end\n9.Deletion in between\n");

while(1)

{

printf("\nEnter your choice: ");

scanf("%d",&x);

switch(x)

{

case 1:

display();

break;

case 2:

in\_b();

break;

case 3:

in\_e();

break;

case 4:

in\_bw();

break;

case 5:

dispf();

break;

case 6:

dispb();

break;

case 7:

del\_b();

break;

case 8:

del\_e();

break;

case 9:

del\_bw();

break;

default:

exit(0);

}

}

}

void display()

{

node \*current=start;

if(current==NULL)

{

printf("List is empty\n");

return;

}

else

{

printf("Nodes of doubly linked list: \n");

while(current!=NULL)

{

printf("%d\t",current->data);

current=current->next;

}

}

}

int in\_b()

{

node \*nptr=create();

int e;

if(nptr==NULL)

{

printf("Memory Overflow");

}

else

{

printf("Enter the item:");

scanf("%d",&e);

nptr->data=e;

if(start==NULL)

{

start=nptr;

nptr->next=NULL;

nptr->prev=NULL;

}

else

{

nptr->next=start;

nptr->prev=NULL;

start->prev=nptr;

start=nptr;

}

}

printf("Element inserted successfully!!");

}

void in\_e()

{

node \*nptr=create();

node \*temp=start;

int e;

if(nptr==NULL)

printf("Memory Overflow");

else

{

printf("Enter the item:");

scanf("%d",&e);

nptr->data=e;

if(temp==NULL)

{

start=nptr;

nptr->next=NULL;

nptr->prev=NULL;

}

else

{

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=nptr;

nptr->prev=temp;

nptr->next=NULL;

}

}

printf("Element inserted succesfully.!!");

}

void in\_bw()

{

node \*nptr=create();

node \*temp=start;

node \*ptr;

int pos,val,c=1;

printf("Enter the position to be inserted");

scanf("%d",&pos);

printf("Enter the element:");

scanf("%d",&val);

nptr->data=val;

nptr->next=NULL;

while(temp!=NULL)

{

if(c==pos)

{

nptr->next=temp;

temp->prev=nptr;

nptr->prev=ptr;

ptr->next=nptr;

break;

}

ptr=temp;

temp=temp->next;

c++;

}

printf("Element inserted successfully.!");

}

void dispf()

{

node \*temp;

temp=start;

if(temp==NULL)

printf("Empty List!");

else

{

printf("Linkded list is:\n");

while(temp!=NULL)

{

printf("%d\t",temp->data);

temp=temp->next;

}

}

}

void dispb()

{

int count;

node \*temp;

temp=start;

if(temp==NULL)

printf("Empty List!.");

else

{

printf("Linked list is:\n");

while(temp->next!=NULL)

{

count=1;

temp=temp->next;

}

while(temp!=NULL)

{

printf("%d\t",temp->data);

temp=temp->prev;

}

}

}

void del\_b()

{

node \*nptr;

if(start==NULL)

printf("Empty list.!!");

else

{

nptr=start;

printf("Element deleted is:%d",nptr->data);

start=nptr->next;

}

nptr->next->prev=NULL;

free(nptr);

}

void del\_e()

{

node \*nptr,\*temp,\*ptr;

int c;

if(start==NULL)

printf("Empty List.!!");

else

{

temp=start;

while(temp->next!=NULL)

{

ptr=temp;

temp=temp->next;

c++;

}

}

printf("Element deleted is:%d",temp->data);

ptr->next=NULL;

free(temp);

if(c==1)

start=NULL;

}

void del\_bw()

{

int c=1,pos;

node \*ptr,\*temp;

printf("Enter the position:");

scanf("%d",&pos);

temp=start;

while(temp!=NULL)

{

if(c==pos)

{

ptr->next=temp->next;

printf("Element deleted is:%d",temp->data);

temp->next->prev=ptr;

free(temp);

}

ptr=temp;

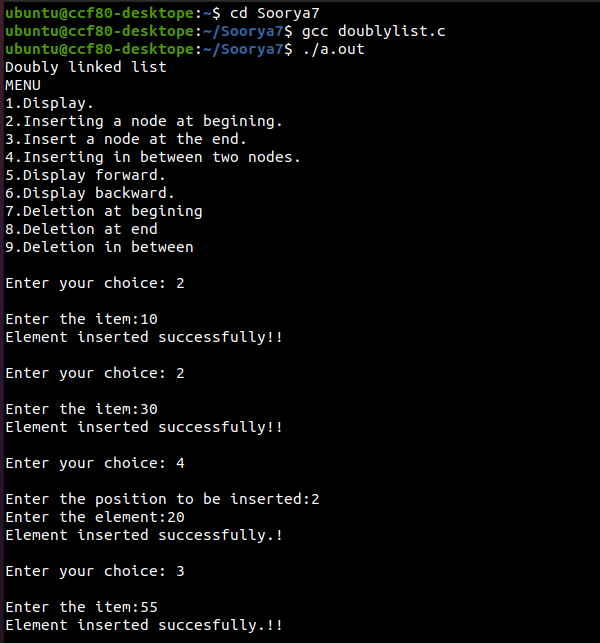
temp=temp->next;

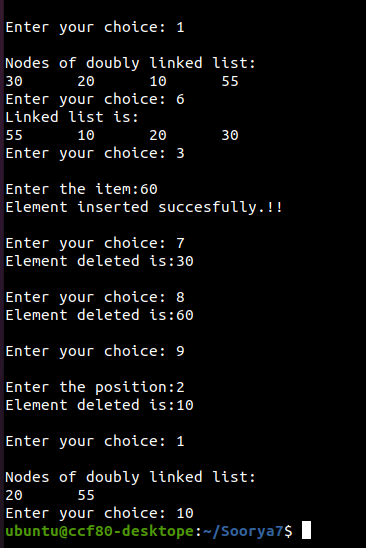
c++;

}

}

**OUTPUT:**

****

****

**PROGRAM-09**

**AIM:** Operations on Sets using bit string implementation.

**SOURCE CODE:**

#include <stdio.h>

void input();

void output();

void setunion();

void intersection();

void difference();

int a[]={0,0,0,0,0,0,0,0,0},b[]={0,0,0,0,0,0,0,0,0} ;

int main()

{

int ch,wish;

while(1)

{

printf("\n\_MENU\_");

printf("\n1.Input\n2.Union\n3.Intersection\n4.Difference\n");

printf("Enter choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:input();

break;

case 2:setunion();

break;

case 3:intersection();

break;

case 4:difference();

break;

default :printf("Wrong input");

exit(0);

}

}

}

void input()

{

int n,x,i;

printf("Enter size of the 1st set:");

scanf("%d",&n);

printf("Enter elements:");

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

{ scanf("%d",&x);

a[x-1]=1;

}

printf("Enter size of the 2nd set:");

scanf("%d",&n);

printf("Enter elements:");

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

{ scanf("%d",&x);

b[x-1]=1;

}

printf("\n1st set:");

for(i=0;i<9;i++)

{

printf("%d",a[i]);

}

printf("\n2nd set:");

for(i=0;i<9;i++)

{

printf("%d",b[i]);

}

}

void output(int c[])

{

int i;

printf("\n Set is");

for(i=0;i<9;i++)

{

if (c[i]!=0)

printf(" %d ",i+1);

}

}

void setunion()

{

int i,c[10];

for(i=0;i<9;i++)

{

if (a[i]!=b[i])

c[i]=1;

else c[i]=a[i];

}

for(i=0;i<9;i++)

{

printf("%d",c[i]);

}

output(c);

}

void intersection()

{

int i,c[10];

for(i=0;i<9;i++)

{

if (a[i]==b[i])

c[i]=a[i];

else c[i]=0;

}

for(i=0;i<9;i++)

{

printf("%d",c[i]);

}

output(c);

}

void difference()

{

int i,c[10];

for(i=0;i<9;i++)

{

if (a[i]==1 && b[i]==0)

c[i]=1;

else c[i]=0;

}

for(i=0;i<9;i++)

{

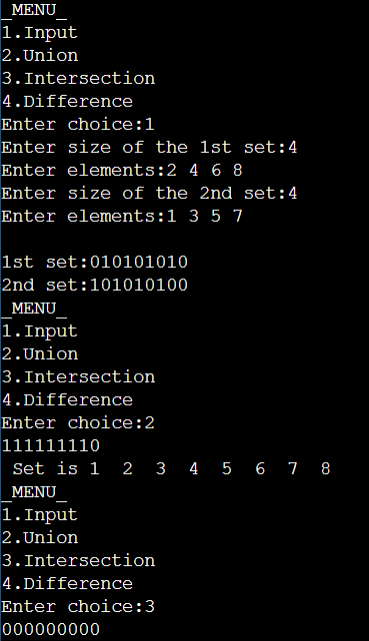
printf("%d",c[i]);

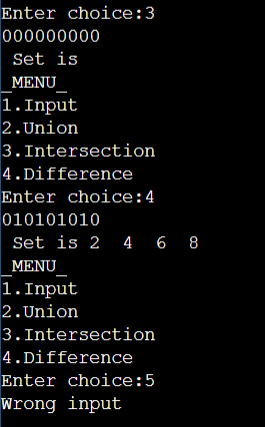
}

output(c);

}

**OUTPUT:**

****

****

**PROGRAM-10**

**AIM:** Disjoint sets operation.

**SOURCE CODE:**

#include <stdio.h>

struct DisjSet

{

int parent[10];

int rank[10];

int n;

}dis;

void makeSet()

{

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

{

dis.parent[i] = i;

dis.rank[i]=0;

}

}

void displaySet()

{

printf("\nParent Array\n");

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

{

printf("%d ",dis.parent[i]);

}

printf("\nRank Array\n");

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

{

printf("%d ",dis.rank[i]);

}

printf("\n");

}

int find(int x)

{

if (dis.parent[x] != x) {

dis.parent[x] = find(dis.parent[x]);

}

return dis.parent[x];

}

void Union(int x, int y)

{

int xset = find(x);

int yset = find(y);

if (xset == yset)

return;

if (dis.rank[xset] <dis.rank[yset])

{

dis.parent[xset] = yset;

dis.rank[xset]=-1;

}

else if (dis.rank[xset] >dis.rank[yset])

{

dis.parent[yset] = xset;

dis.rank[yset]=-1;

}

else

{

dis.parent[yset] = xset;

dis.rank[xset] = dis.rank[xset] + 1;

dis.rank[yset]=-1;

}

}

int main()

{

int n,x,y;

printf("How many elements ?");

scanf("%d",&dis.n);

makeSet();

int ch,wish;

do

{

printf("\n\_\_\_\_MENU\_\_\_\_\n");

printf("1. Union \n2.Find\n3.Display\n");

printf("Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("Enter elements to perform union:");

scanf("%d %d",&x,&y);

Union(x, y);

break;

case 2: printf("Enter elements to check if connected components:");

scanf("%d %d",&x,&y);

if (find(x) == find(y))

printf("Connected components\n") ;

else

printf("Not onnected components \n") ;

break;

case 3: displaySet();

break;

}

printf("Do you wish to continue ?(1/0):");

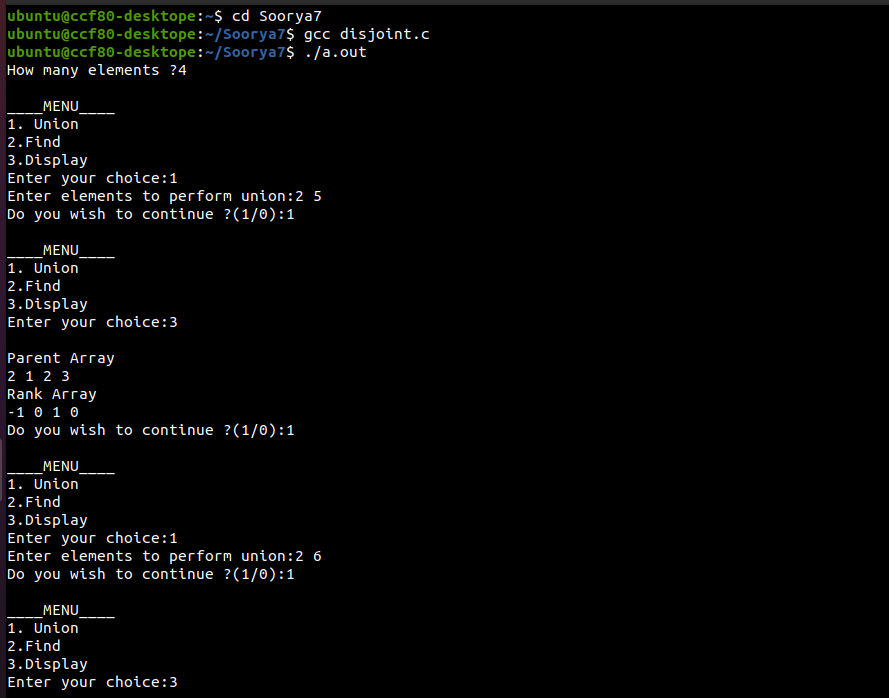
scanf("%d",&wish);

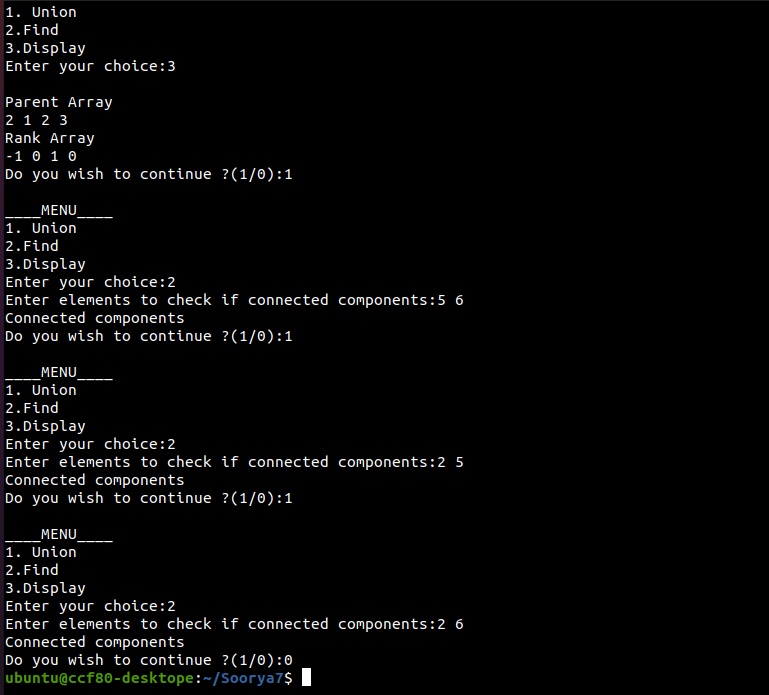
}while(wish==1);

return 0;

}

**OUTPUT:**

****

****

**PROGRAM-11**

**AIM :** Operations onLinked stack.

**SOURCE CODE:**

#include<stdlib.h>

#include<stdio.h>

void Push();

void Pop();

void Display();

struct node

{

int val;

struct node \*next;

};

struct node \*head;

void main()

{

int ch=0;

printf("Stack Implementation using linked list");

while(ch!=4)

{

printf("\nMENU\n\n1.Push\n2.Pop\n3.Display\n");

printf("Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:Push();

break;

case 2:Pop();

break;

case 3:Display();

break;

default:printf("WRONG INPUT");

exit(0);

}

}

}

void Push()

{

int val;

struct node \*ptr=(struct node\*)malloc(sizeof(struct node));

if(ptr==NULL)

printf("Can't Push element!!");

else

{

printf("Enter the value to be inserted:");

scanf("%d",&val);

if(head==NULL)

{

ptr->val=val;

ptr->next=NULL;

head=ptr;

}

else

{

ptr->val=val;

ptr->next=head;

head=ptr;

}

printf("Item Pushed!!");

}

}

void Pop()

{

int item;

struct node \*ptr;

if(head==NULL)

printf("Underflow!!");

else

{

item=head->val;

ptr=head;

head=head->next;

free(ptr);

}

printf("Item Popped!!");

}

void Display()

{

int i;

struct node\*ptr;

ptr=head;

if(ptr==NULL)

printf("Stack is empty!!");

else

{

printf("Stack elements are:\n");

while(ptr!=NULL)

{

printf("%d\t",ptr->val);

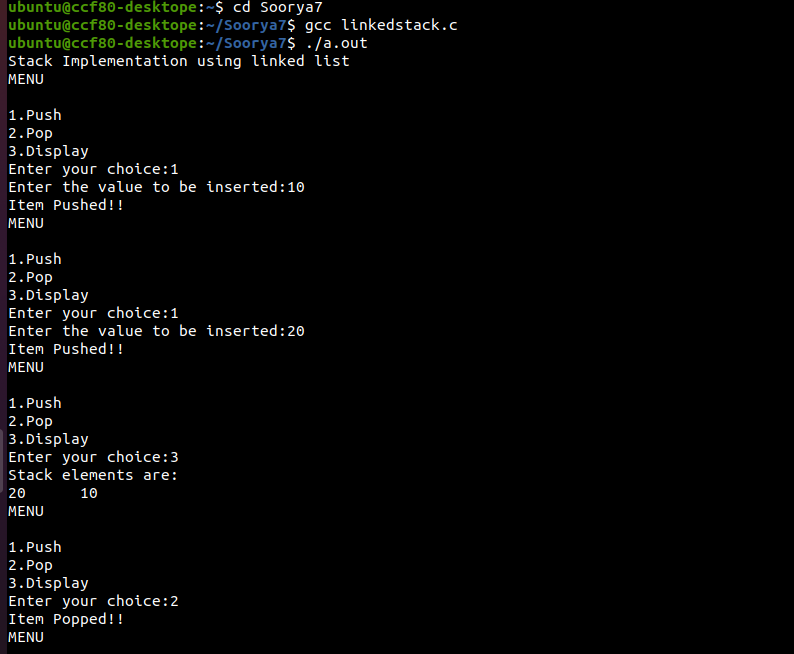
ptr=ptr->next;

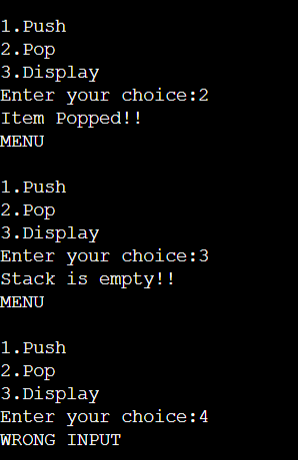
}

}

}

**OUTPUT:**

****

****

**PROGRAM-12**

**AIM :** Breadth First Search &Depth First Search.

**SOURCE CODE:**

#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];

int delete();

void add(int item);

void bfs(int s,int n);

void dfs(int s,int n);

void push(int item);

int pop();

void main()

{

int n,i,s,ch,j;

char c,dummy;

printf("Enter the number of vertices:");

scanf("%d",&n);

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

{

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

{

printf("ENTER 1 if %d has a node with %d else 0:",i,j);

scanf("%d",&a[i][j]);

}

}

printf("The adjacency matrix is\n");

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

{

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

{

printf(" %d",a[i][j]);

}

printf("\n");

}

do

{

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

vis[i]=0;

printf("\nMENU");

printf("\n1.B.F.S");

printf("\n2.D.F.S");

printf("\nEnter your choice:");

scanf("%d",&ch);

printf("Enter the source vertex:");

scanf("%d",&s);

switch(ch)

{

case 1:bfs(s,n);

break;

case 2:dfs(s,n);

break;

}

printf("Do you want to continue(Y/N) ? ");

scanf("%c",&dummy);

scanf("%c",&c);

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

}

void bfs(int s,int n)

{

int p,i;

add(s);

vis[s]=1;

p=delete();

if(p!=0)

printf(" %d",p);

while(p!=0)

{

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

if((a[p][i]!=0)&&(vis[i]==0))

{

add(i);

vis[i]=1;

}

p=delete();

if(p!=0)

printf(" %d ",p);

}

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

if(vis[i]==0)

bfs(i,n);

}

void add(int item)

{

if(rear==19)

printf("QUEUE FULL");

else

{

if(rear==-1)

{

q[++rear]=item;

front++;

}

else

q[++rear]=item;

}

}

int delete()

{

int k;

if((front>rear)||(front==-1))

return(0);

else

{

k=q[front++];

return(k);

}

}

void dfs(int s,int n)

{

int i,k;

push(s);

vis[s]=1;

k=pop();

if(k!=0)

printf(" %d ",k);

while(k!=0)

{

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

if((a[k][i]!=0)&&(vis[i]==0))

{

push(i);

vis[i]=1;

}

k=pop();

if(k!=0)

printf(" %d ",k);

}

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

if(vis[i]==0)

dfs(i,n);

}

void push(int item)

{

if(top==19)

printf("Stack overflow ");

else

stack[++top]=item;

}

int pop()

{

int k;

if(top==-1)

return(0);

else

{

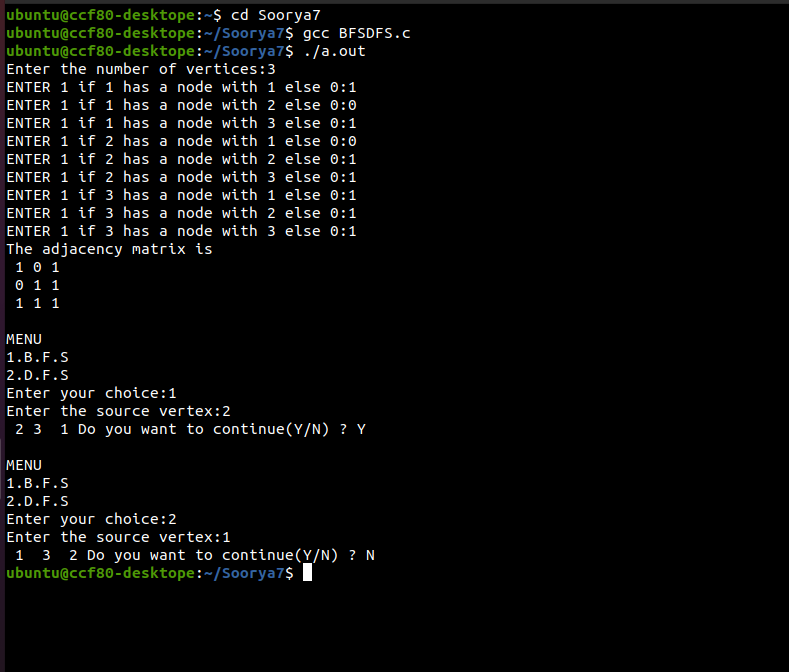
k=stack[top--];

return(k);

}

}

**OUTPUT:**

****

**PROGRAM-13**

**AIM :** Minimum cost spanning tree using Prim’s Algorithm.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

int n,cost[10][10];

void prim();

void main()

{

int i,j;

printf("\nEnter the number of vertices:");

scanf("%d",&n);

printf("\nEnter the costs of edges in matrix form:");

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

{

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

{

scanf("%d",&cost[i][j]);

}

}

printf("\n The matrix is:\n");

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

{

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

{

printf("%d ",cost[i][j]);

}

printf("\n\n");

}

prim();

getch();

}

void prim()

{

int i,j,k,l,x,nr[10],temp,mincost=0,tree[10][3];

temp=cost[0][0];

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

{

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

{

if(temp>cost[i][j])

{

temp=cost[i][j];

k=i;

l=j;

}

}

}

tree[0][0]=k;

tree[0][1]=l;

tree[0][2]=temp;

mincost=temp;

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

{

if(cost[i][k]<cost[i][l])

nr[i]=k;

else

nr[i]=l;

}

nr[k]=100;

nr[l]=100;

temp=101;

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

{

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

{

if(nr[j]!=100&&cost[j][nr[j]]<temp)

{

temp=cost[j][nr[j]];

x=j;

}

}

tree[i][0]=x;

tree[i][1]=nr[x];

tree[i][2]=cost[x][nr[x]];

mincost=mincost+cost[x][nr[x]];

int p;

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

{

if(nr[p]!=100&&cost[p][nr[p]]>cost[p][x])

nr[p]=x;

}

nr[x]=100;

temp=101;

}

printf("\nThe minimum cost spanning tree is:\n");

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

{

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

{

printf("%d ",tree[i][j]);

}

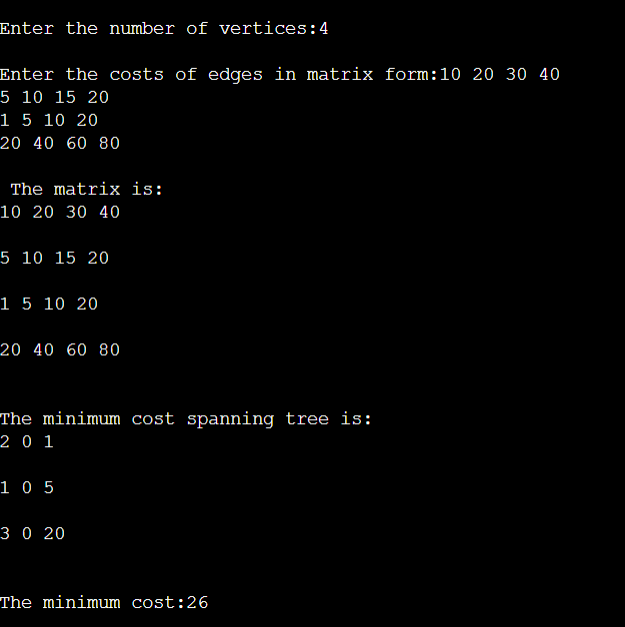
printf("\n\n");

}

printf("\nThe minimum cost:%d",mincost);

}

**OUTPUT:**

****

**PROGRAM-14**

**AIM :** Minimum cost spanning tree using Kruskal’s Algorithm.

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void main()

{

printf("\nImplementation of Kruskal's Algorithm");

printf("\nEnter the no. of vertices:");

scanf("%d",&n);

printf("\nEnter the cost adjacency matrix:");

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

{

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

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

}

printf("The edges of Minimum Cost Spanning Tree are\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

{

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

{

if(cost[i][j] < min)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v))

{

printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

cost[a][b]=cost[b][a]=999;

}

printf("\nMinimum cost = %d\n",mincost);

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

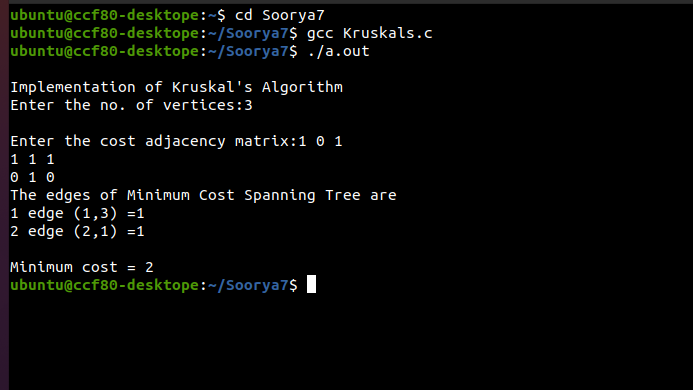
return 1;

}

return 0;

}

**OUTPUT:**

****