Experiment No : **1**

Aim : Program for Maximum and minimum in an array

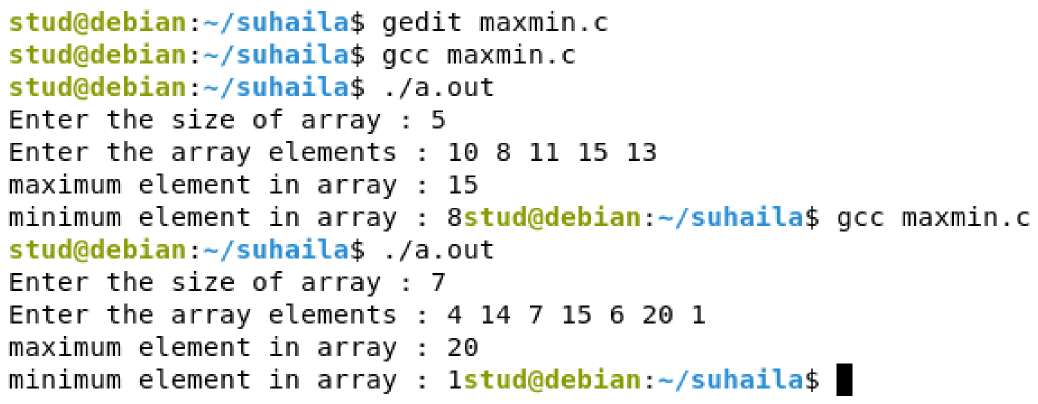
Concept : Array

Description : An array is defined as the collection of similar type of data items stored at contiguous memory locations. Array is the simplest data structure where each data element can be randomly accessed by using its index number.

Program code :

```
#include<stdio.h>
int main()
{
int n,i,j,k,max,min;
int arr[30];
printf("Enter the size of array : ");
scanf("%d",&n);
printf("Enter the array elements : ");
for(i=0;i<n;i++)
{
scanf("%d",&arr[i]);
}
max=arr[0];
for(j=0;j<n;j++)
{
if(arr[j]>max)
{
max=arr[j];
}
}
min=arr[0];
for(k=0;k<n;k++)
{
if(arr[k]<min)
{
min=arr[k];
}
}
printf("maximum element in array : %d\n",max);
printf("minimum element in array : %d",min);
return 0;
}
```

Output :



```
stud@debian:~/suhaila$ gedit maxmin.c
stud@debian:~/suhaila$ gcc maxmin.c
stud@debian:~/suhaila$ ./a.out
Enter the size of array : 5
Enter the array elements : 10 8 11 15 13
maximum element in array : 15
minimum element in array : 8stud@debian:~/suhaila$ gcc maxmin.c
stud@debian:~/suhaila$ ./a.out
Enter the size of array : 7
Enter the array elements : 4 14 7 15 6 20 1
maximum element in array : 20
minimum element in array : 1stud@debian:~/suhaila$ 
```

Experiment No : **2**

Aim : Merge two sorted arrays and store in a third array

Concept : Array

Description : An array is defined as the collection of similar type of data items stored at contiguous memory locations. Array is the simplest data structure where each data element can be randomly accessed by using its index number.

Program code :

```
#include<stdio.h>
int main()
{
int s1,s2,s3,i,k;
int arr1[50],arr2[50];
int arr3[100];
printf("Enter the size of first array : ");
scanf("%d",&s1);
printf("Enter the elements of first array in ascending order : ");
for(i=0;i<s1;i++)
{
scanf("%d",&arr1[i]);
arr3[i]=arr1[i];
}
printf("Enter the size of second array : ");
scanf("%d",&s2);
printf("Enter the elements of second array in ascending order : ");
k=i;
for(i=0;i<s2;i++)
{
scanf("%d",&arr2[i]);
arr3[k]=arr2[i];
k++;
}
s3=s1+s2;
printf("\n merged array is : ");
for(i=0;i<s3;i++)
{
printf("%d\t",arr3[i]);
}
return 0;
}
```

Output :

```
stud@debian:~/suhaila$ gcc sorting.c
stud@debian:~/suhaila$ ./a.out
Enter the size of first array : 5
Enter the elements of first array in ascending order : 2 3 4 5 6
Enter the size of second array : 4
Enter the elements of second array in ascending order : 10 20 25 30

 merged array is : 2	3	4	5	6	10	20	25	30
```

Experiment No : **3**

Aim : Program to implement stack using array

Concept : Stack using Array

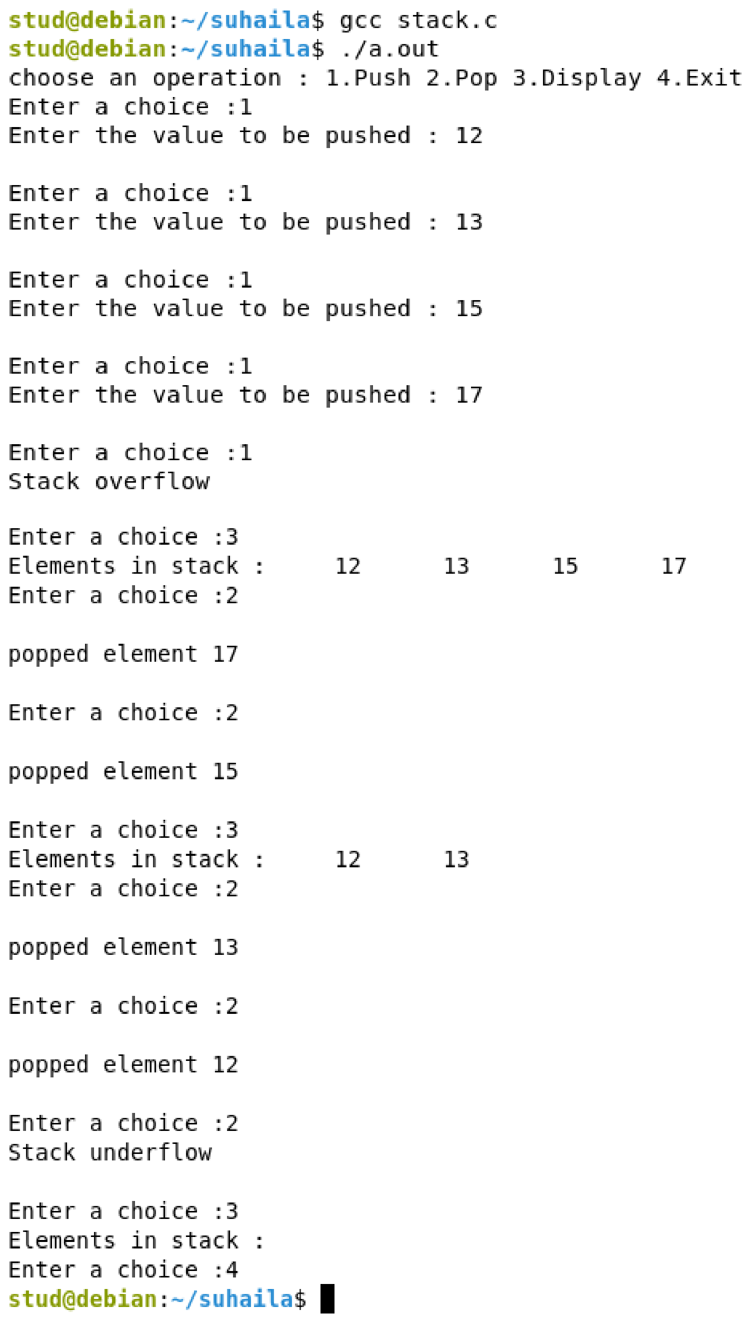
Description : A stack is a linear data structure that follows LIFO (Last In First Out) principle. Two important operations performed on stack is Push (insertion of element at the top of stack) and Pop (deletion of element from the top of stack). Here we will see stack implementation using array.

Program code :

```
#include<stdio.h>
#include<stdlib.h>
int stack[4];
int top = -1;
void push()
{
int p;
if(top==3)
{
printf("Stack overflow \n ");
}
else
{
printf("Enter the value to be pushed : ");
scanf("%d",&p);
top=top+1;
stack[top]=p;
}
}
void pop()
{
if(top==-1)
{
printf("Stack underflow\n");
}
else
{
printf("\npopped element %d\n",stack[top]);
top=top-1;
}
}
```

```
void display()
{
printf("Elements in stack : \t");
for(int i=0;i<=top;i++)
{
printf("%d\t",stack[i]);
}
}
int main()
{
int ch;
printf("choose an operation : 1.Push 2.Pop 3.Display 4.Exit");
while(1)
{
printf("\nEnter a choice :");
scanf("%d",&ch);
switch(ch)
{
case 1: push();
break;
case 2: pop();
break;
case 3:display();
break;
case 4:exit(0);
break;
default: printf("invalid value");
break;
}
}
return 0;
}
```

Output :



```
stud@debian:~/suhaila$ gcc stack.c
stud@debian:~/suhaila$ ./a.out
choose an operation : 1.Push 2.Pop 3.Display 4.Exit
Enter a choice :1
Enter the value to be pushed : 12

Enter a choice :1
Enter the value to be pushed : 13

Enter a choice :1
Enter the value to be pushed : 15

Enter a choice :1
Enter the value to be pushed : 17

Enter a choice :1
Stack overflow

Enter a choice :3
Elements in stack :     12      13      15      17
Enter a choice :2

popped element 17

Enter a choice :2

popped element 15

Enter a choice :3
Elements in stack :     12      13
Enter a choice :2

popped element 13

Enter a choice :2

popped element 12

Enter a choice :2
Stack underflow

Enter a choice :3
Elements in stack :
Enter a choice :4
stud@debian:~/suhaila$ 
```

Experiment No : **4**

Aim : Program to implement queue using array

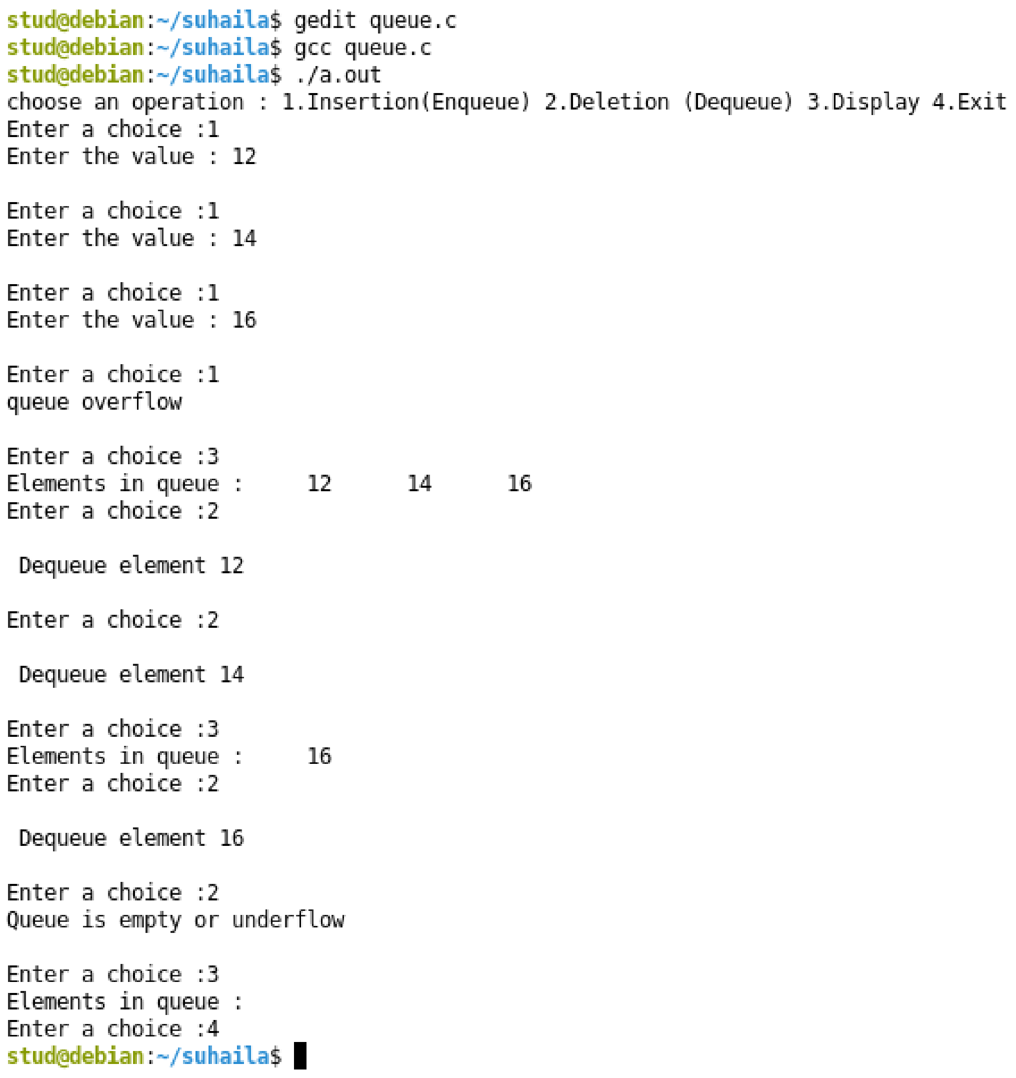
Concept : Queue using Array

Description : A queue is a linear data structure that follows FIFO (First In First Out) principle. Two important operations performed on queue is Enqueue (insertion of element at the rear of queue) and Dequeue (deletion of element from the front of queue). Here we will see queue implementation using array.

Program code :

```
#include<stdio.h>
#include<stdlib.h>
int queue[3];
int front=-1;
int rear=-1;
void enqueue()
{
int p;
if(rear==2)
{
printf("queue overflow \n ");
}
else
{
if(front==-1)
{
front=front+1;
}
printf("Enter the value : ");
scanf("%d",&p);
rear=rear+1;
queue[rear]=p;
}
}
void dequeue()
{
if(front==-1 || front>rear)
{
printf("Queue is empty or underflow\n");
}
else
{
printf("\n Dequeue element %d\n",queue[front]);
front=front+1;
}
}
void display()
{
printf("Elements in queue : \t");
for(int i=front;i<=rear;i++)
{
printf("%d\t",queue[i]);
}
}
int main()
{
int ch;
printf("choose an operation : 1.Insertion (Enqueue) 2.Deletion (Dequeue) 3.Display 4.Exit");
while(1)
{
printf("\nEnter a choice :");
scanf("%d",&ch);
switch(ch)
{
case 1: enqueue();
break;
case 2: dequeue();
break;
case 3: display();
break;
case 4: exit(0);
break;
default: printf("invalid value");
break;
}
}
return 0;
}
```

Output :



```
stud@debian:~/suhaila$ gedit queue.c
stud@debian:~/suhaila$ gcc queue.c
stud@debian:~/suhaila$ ./a.out
choose an operation : 1.Insertion(Enqueue) 2.Deletion (Dequeue) 3.Display 4.Exit
Enter a choice :1
Enter the value : 12

Enter a choice :1
Enter the value : 14

Enter a choice :1
Enter the value : 16

Enter a choice :1
queue overflow

Enter a choice :3
Elements in queue :     12      14      16
Enter a choice :2

 Dequeue element 12

Enter a choice :2

 Dequeue element 14

Enter a choice :3
Elements in queue :     16
Enter a choice :2

 Dequeue element 16

Enter a choice :2
Queue is empty or underflow

Enter a choice :3
Elements in queue :
Enter a choice :4
stud@debian:~/suhaila$
```

Experiment No : **5**

Aim : Program for Circular Queue

Concept : Circular Queue

Description : A circular queue is the extended version of a regular queue where the last element is connected to the first element. Thus, forming a circle-like structure. In a normal queue, after a bit of insertion and deletion, there will be non-usable empty space. Circular queue solves this major limitation of the normal queue.

Program code :

```
#include<stdio.h>
#include<stdlib.h>
#define max 4
int queue[max];
int front=-1;
int rear=-1;
void enqueue()
{
int p;
if((front==0 && rear==max-1) || rear==front-1)
{
printf("circular queue overflow or full\n ");
}
else
{
if(front==-1)
{
front=front+1;
}
printf("Enter the value : ");
scanf("%d",&p);
rear=(rear+1)%max;
queue[rear]=p;
}
}
void dequeue()
{
if(front==-1)
{
printf("Queue is empty or underflow\n");
}
```

```
else
{
printf("Dequeue element %d\n",queue[front]);
if(front==rear)
{
front=rear=-1;
}
else
{
front=(front+1)%max;
}
}
}
void display ()
{
int i,j;
if (front ==-1 && rear==-1)
printf("Queue is underflow\n");
if(front>rear)
{
for(i=front; i<max; i++)
printf("%d \t", queue[i]);
for(j=0;j<=rear; j++)
printf("%d \t", queue[j]);
}
else
{
for (i=front;i<=rear;i++)
printf("%d \t", queue[i]);
}
printf("\n");
}
void main()
{
int ch;
printf("choose an operation : 1.Insertion (Enqueue) 2.Deletion (Dequeue) 3.Display 4.Exit");
while(1)
{
printf("\nEnter a choice :");
scanf("%d",&ch);
switch(ch)
{
case 1: enqueue();
break;
case 2: dequeue();
break;
case 3: display();
break;
case 4: exit(0);
break;
default: printf("invalid value");
break;
}
}
}
```

Output :

```
stud@debian:~/suhaila$ gedit circular.c
stud@debian:~/suhaila$ gcc circular.c
stud@debian:~/suhaila$ ./a.out
choose an operation : 1.Insertion(Enqueue) 2.Deletion(Dequeue) 3.Display 4.Exit
Enter a choice :1
Enter the value : 2

Enter a choice :1
Enter the value : 5

Enter a choice :1
Enter the value : 9

Enter a choice :1
Enter the value : 6

Enter a choice :1
circular queue overflow or full

Enter a choice :3
2       5       9       6

Enter a choice :2
Dequeue element 2

Enter a choice :2
Dequeue element 5

Enter a choice :3
9       6

Enter a choice :1
Enter the value : 12

Enter a choice :3
9       6       12

Enter a choice :2
Dequeue element 9

Enter a choice :2
Dequeue element 6

Enter a choice :2
Dequeue element 12

Enter a choice :2
Queue is empty or underflow

Enter a choice :3
Queue is underflow
0

Enter a choice :1
Enter the value : 12

Enter a choice :3
12

Enter a choice :4
stud@debian:~/suhaila$ 
```

Experiment No : **6**

Aim : Program to implement set operations using bitstring

Concept : Bit String

Description : A Bit String (bitset) is a data structure to implement a set in computer memory. The number of bits is equal to the number of elements in the universal set. A given set is then represented by a bit string in which the bits corresponding to the elements of that set are 1 and all other bits are 0. We can perform certain operations on bitset such as union, intersection and compliment.

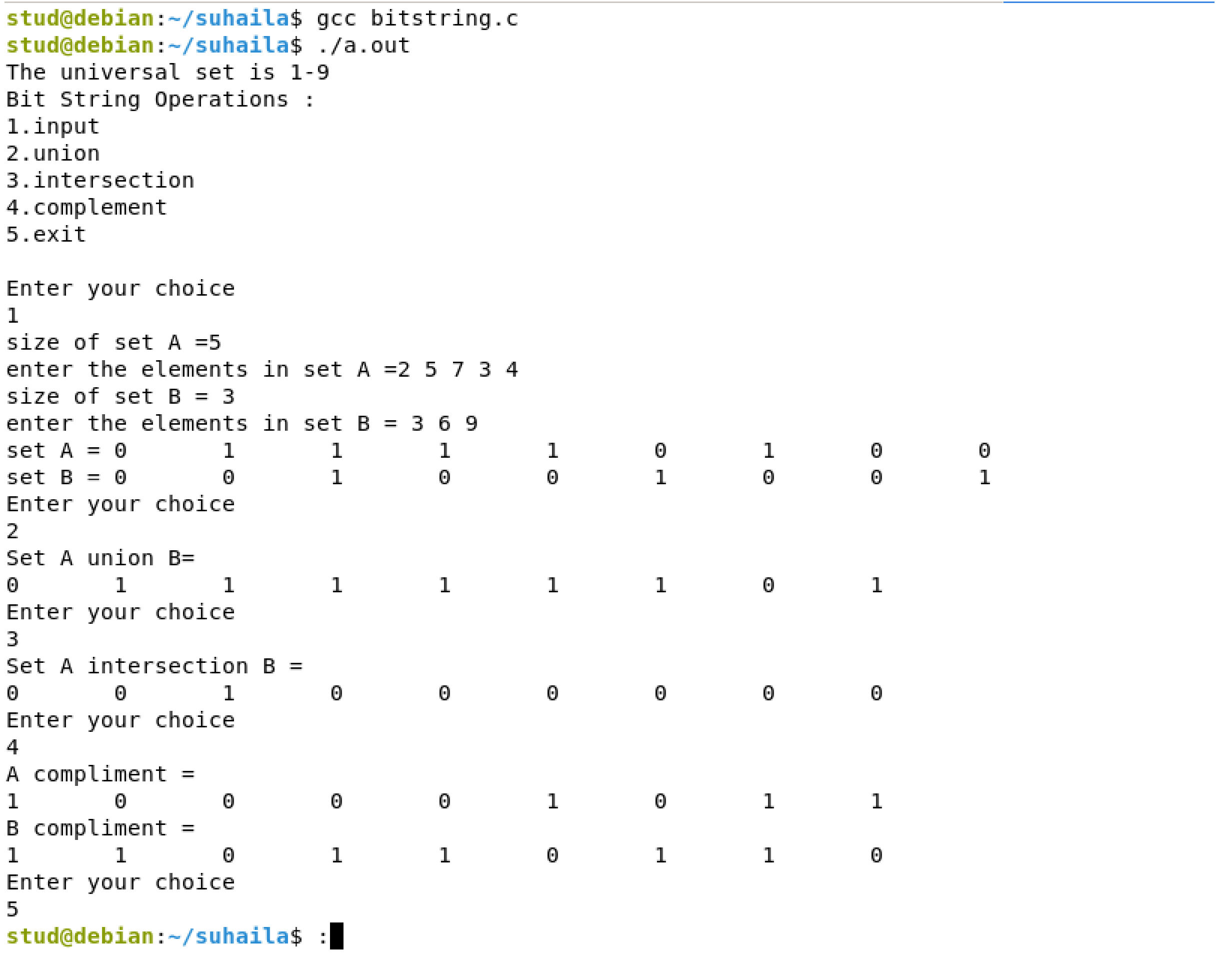
Program code :

```
#include<stdio.h>
#include<stdlib.h>
int u[10],a[10],b[10],i;
void input()
{
int x,sizea;
printf("size of set A =");
scanf("%d",&sizea);
printf("enter the elements in set A =");
for (i=0;i<sizea;i++)
{
scanf("%d",&x);
a[x]=1;
}
int y,sizeb;
printf("size of set B = ");
scanf("%d",&sizeb);
printf("enter the elements in set B = ");
for (i=0;i<sizeb;i++)
{
scanf("%d",&y);
b[y]=1;
}
printf("set A = ");
for (i=1;i<10;i++)
{
printf("%d\t",a[i]);
}
printf("\n");
printf("set B = ");
for (i=1;i<10;i++)
```

```
{
printf("%d\t",b[i]);
}
}
void setunion(int a[],int b[])
{
printf("Set A union B= \n");
for (i=1;i<10;i++)
{
if((a[i]||b[i]))
     printf("1\t");
else
     printf("0\t");
}
}
void setinter(int a[],int b[])
{
printf("Set A intersection B = \n");
for (i=1;i<10;i++)
{
if((a[i]&&b[i]))
   printf("1\t");
else
   printf("0\t");
}
}
void setcomp(int a[],int b[])
{
printf("A compliment = \n");
for (i=1;i<10;i++)
{
if(a[i])
   printf("0\t");
else
   printf("1\t");
}
printf("\n");
printf("B compliment = \n");
for (i=1;i<10;i++)
{
if(b[i])
   printf("0\t");
else
   printf("1\t");
}
```

```
}
int main()
{
int ch;
printf("The universal set is 1-9\n");
printf("Bit String Operations :");
printf("\n1.input\n2.union \n3.intersection \n4.complement \n5.exit\n");
while(1)
{
printf("\nEnter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: input();
break;
case 2: setunion(a,b);
break;
case 3: setinter(a,b);
break;
case 4: setcomp(a,b);
break;
case 5: exit(0);
break;
default: printf("invalid choice");
break;
}
}
return 0;
}
```

Output :



```
stud@debian:~/suhaila$ gcc bitstring.c
stud@debian:~/suhaila$ ./a.out
The universal set is 1-9
Bit String Operations :
1.input
2.union
3.intersection
4.complement
5.exit

Enter your choice
1
size of set A =5
enter the elements in set A =2 5 7 3 4
size of set B = 3
enter the elements in set B = 3 6 9
set A = 0       1       1       1       1       0       1       0       0
set B = 0       0       1       0       0       1       0       0       1
Enter your choice
2
Set A union B=
0       1       1       1       1       1       1       0       1
Enter your choice
3
Set A intersection B =
0       0       1       0       0       0       0       0       0
Enter your choice
4
A compliment =
1       0       0       0       0       1       0       1       1
B compliment =
1       1       0       1       1       0       1       1       0
Enter your choice
5
stud@debian:~/suhaila$ :
```

Experiment No : **7**

Aim : Program to implement stack using Singly Linked List

Concept : Stack using Singly Linked List

Description : Singly Linked List is as collection of objects called nodes that are randomly stored in the memory. It can be traversed only in one direction that is from head to tail node. We can implement stack using Singly linked list. Two important operations performed on stack is Push (insertion of element at the end of list) and Pop (deletion of element from the end of list).

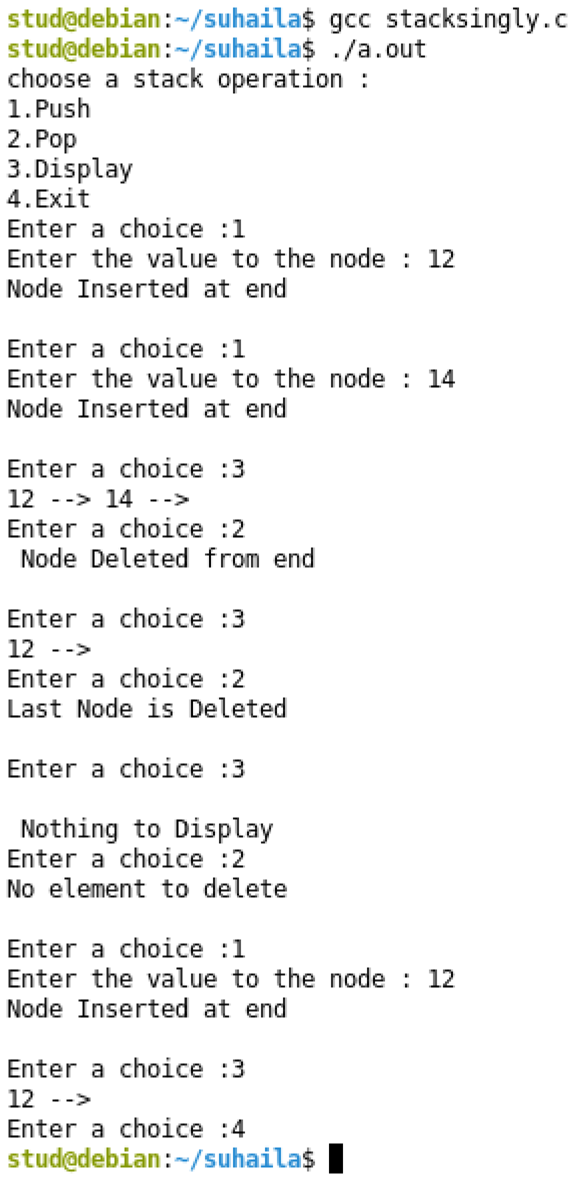
Program code :

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *top = NULL,*new,*h,*e;
void create()
{
int v;
new=(struct node*)malloc(sizeof(struct node));
if(new==NULL)
   printf("\n Memory Full");
else
{
printf("Enter the value to the node : ");
scanf("%d",&v);
new->data=v;
new->next=NULL;
}
}
void push()
{
create();
if(top==NULL)
   top=new;
else
{
h=top;
```

```
while(h->next!=NULL)
{
h=h->next;
}
h->next=new;
}
printf("Node Inserted at end \n");
}
void pop()
{
if(top==NULL)
{
printf("No element to delete \n");
}
else if(top->next==NULL)
{
h=top;
top=NULL;
free(h);
printf("Last Node is Deleted \n");
}
else
{
h=top;
while(h->next!=NULL)
{
e=h;
h=h->next;
}
e->next=NULL;
free(h);
printf(" Node Deleted from end \n");
}
}
void display()
{
if(top==NULL)
{
printf("\n Nothing to Display");
}
else
{
h=top;
while(h!=NULL)
{
```

```
printf("%d --> ",h->data);
h=h->next;
}
}
}
//main function
int main()
{
int ch;
printf("choose a stack operation : \n1.Push \n2.Pop \n3.Display \n4.Exit");
while(1)
{
printf("\nEnter a choice :");
scanf("%d",&ch);
switch(ch)
{
case 1: push();
break;
case 2: pop();
break;
case 3: display();
break;
case 4: exit(0);
break;
default : printf("invalid value");
break;
}
}
return 0;
}
```

Output :



```
stud@debian:~/suhaila$ gcc stacksingly.c
stud@debian:~/suhaila$ ./a.out
choose a stack operation :
1.Push
2.Pop
3.Display
4.Exit
Enter a choice :1
Enter the value to the node : 12
Node Inserted at end

Enter a choice :1
Enter the value to the node : 14
Node Inserted at end

Enter a choice :3
12 --> 14 -->
Enter a choice :2
 Node Deleted from end

Enter a choice :3
12 -->
Enter a choice :2
Last Node is Deleted

Enter a choice :3

 Nothing to Display
Enter a choice :2
No element to delete

Enter a choice :1
Enter the value to the node : 12
Node Inserted at end

Enter a choice :3
12 -->
Enter a choice :4
stud@debian:~/suhaila$
```

Experiment No : **8**

Aim : Program to implement queue using Singly Linked List

Concept : Queue using Singly Linked List

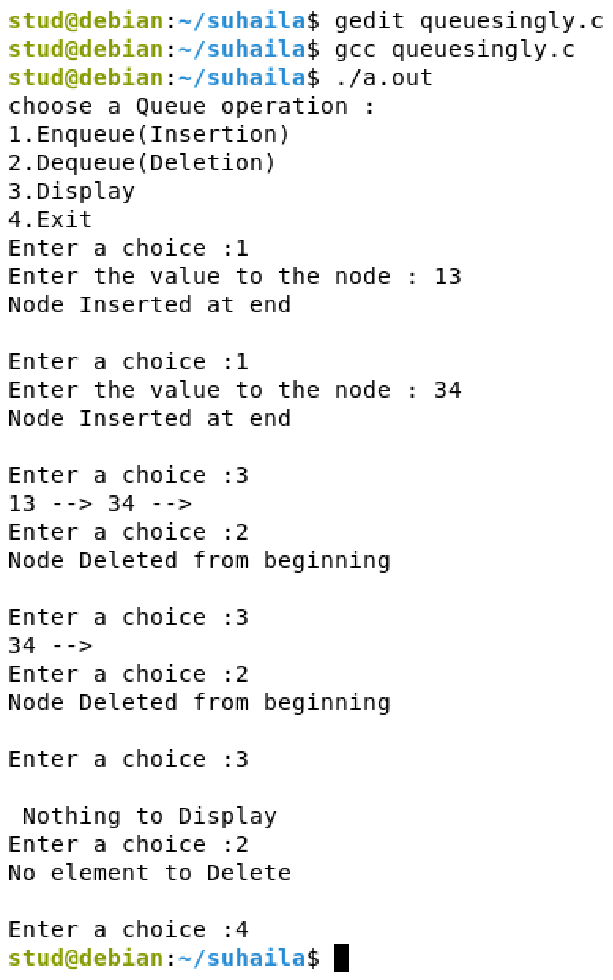
Description : Singly Linked List is as collection of objects called nodes that are randomly stored in the memory. It can be traversed only in one direction that is from head to tail node. We can implement queue using Singly linked list. Two important operations performed on queue is Enqueue (insertion of element at the end of queue) and Dequeue (deletion of element from the beginning of queue).

Program code :

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head = NULL,*new,*h,*t;
void create()
{
int v;
new=(struct node*)malloc(sizeof(struct node));
if(new==NULL)
   printf("\n Memory Full");
else
{
printf("Enter the value to the node : ");
scanf("%d",&v);
new->data=v;
new->next=NULL;
}
}
void enqueue()
{
create();
if(head==NULL)
   head=new;
else
{
h=head;
while(h->next!=NULL)
```

```
{
h=h->next;
}
h->next=new;
}
printf("Node Inserted at end \n");
}
void dequeue()
{
if(head==NULL)
   printf("No element to Delete \n ");
else
{
t=head;
head=head->next;
free(t);
printf("Node Deleted from beginning \n");
}
}
void display()
{
if(head==NULL)
   printf("\n Nothing to Display");
else
{
h=head;
while(h!=NULL)
{
printf("%d --> ",h->data);
h=h->next;
}
}
}
int main()
{
int ch;
printf("choose a Queue operation : \n1.Enqueue(Insertion) \n2.Dequeue(Deletion) \n3.Display
\n4.Exit");
while(1)
{
printf("\nEnter a choice :");
scanf("%d",&ch);
switch(ch)
{
case 1: enqueue();
break;
case 2: dequeue();
break;
case 3: display();
break;
case 4: exit(0);
break;
default : printf("invalid value");
break;
}
}
return 0;
}
```

Output :



```
stud@debian:~/suhaila$ gedit queuesingly.c
stud@debian:~/suhaila$ gcc queuesingly.c
stud@debian:~/suhaila$ ./a.out
choose a Queue operation :
1.Enqueue(Insertion)
2.Dequeue(Deletion)
3.Display
4.Exit
Enter a choice :1
Enter the value to the node : 13
Node Inserted at end

Enter a choice :1
Enter the value to the node : 34
Node Inserted at end

Enter a choice :3
13 --> 34 -->
Enter a choice :2
Node Deleted from beginning

Enter a choice :3
34 -->
Enter a choice :2
Node Deleted from beginning

Enter a choice :3

 Nothing to Display
Enter a choice :2
No element to Delete

Enter a choice :4
stud@debian:~/suhaila$ 
```

Experiment No : **9**

Aim : Program to implement Doubly Linked List

Concept : Doubly Linked List

Description : Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. In a singly linked list, we could traverse only in one direction. Doubly linked list overcome this limitation of singly linked list.

Program code :

```
#include<stdio.h>
#include<stdlib.h>
int count=0;
struct node
{
struct node *prev;
int data;
struct node *next;
};
struct node *head = NULL,*new,*h;
void create()
{
int v;
new=(struct node*)malloc(sizeof(struct node));
if(new==NULL)
   printf("\n Memory Full");
else
{
printf("Enter the value to the node : ");
scanf("%d",&v);
new->data=v;
new->prev=NULL;
new->next=NULL;
}
}
void ins_beg()
{
create();
if(head==NULL)
   head=new;
else
{
```

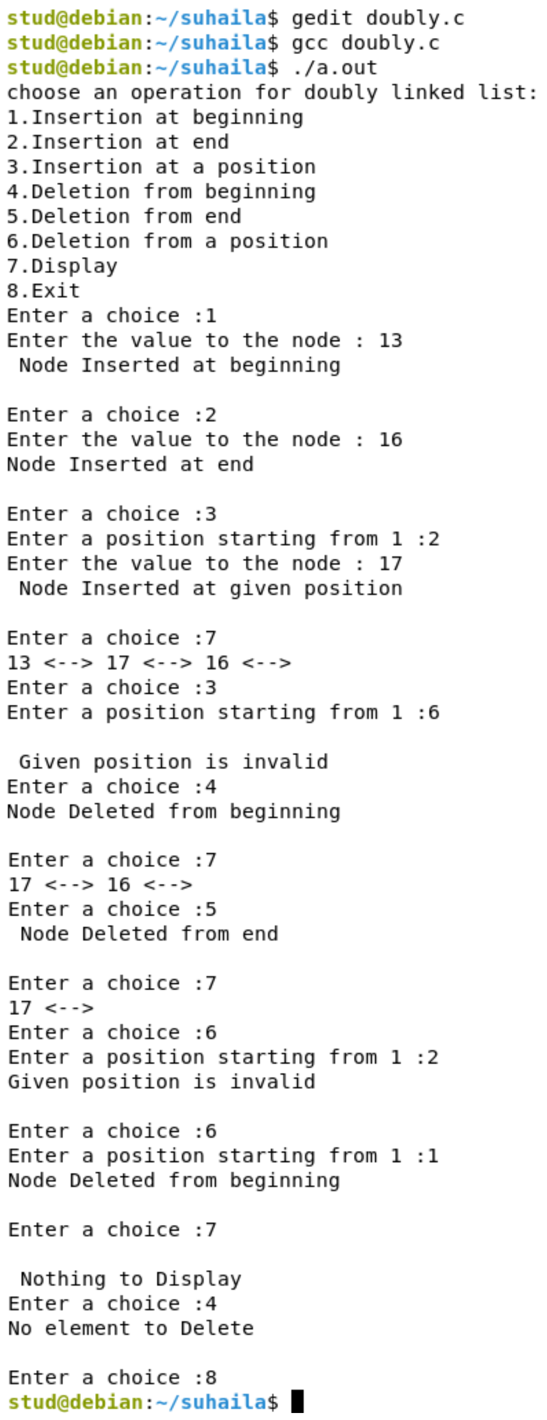
```
new->next=head;
new->prev=new;
head=new;
}
printf(" Node Inserted at beginning \n");
count++;
}
void ins_end()
{
create();
if(head==NULL)
   head=new;
else
{
h=head;
while(h->next!=NULL)
{
h=h->next;
}
h->next=new;
new->prev=h;
}
printf("Node Inserted at end \n");
count++;
}
void ins_pos()
{
int pos;
printf("Enter a position starting from 1 :");
scanf("%d",&pos);
if(pos==1)
   ins_beg();
else if(pos<=count+1)
{
create();
h=head;
for(int l=1;l<pos-1;l++)
{
h=h->next;
}
new->prev=h;
new->next=h->next;
h->next->prev=new;
h->next=new;
printf(" Node Inserted at given position \n");
```

```
count++;
}
else
  printf("\n Given position is invalid ");
}
void del_beg()
{
if(head==NULL)
   printf("No element to Delete \n ");
else if(head->next==NULL)
{
h=head;
head=NULL;
free(h);
count--;
printf("Node Deleted from beginning \n");
}
else
{
h=head;
head=head->next;
head->prev=NULL;
free(h);
printf("Node Deleted from beginning \n");
count--;
}
}
void del_end()
{
if(head==NULL)
   printf("No element to delete \n");
else if(head->next==NULL)
{
h=head;
head=NULL;
free(h);
count--;
printf("Node Deleted from end \n");
}
else
{
h=head;
while(h->next!=NULL)
{
h=h->next;
```

```
}
h->prev->next=NULL;
free(h);
count--;
printf(" Node Deleted from end \n");
}
}
void del_pos()
{
int pos;
printf("Enter a position starting from 1 :");
scanf("%d",&pos);
if(head==NULL)
     printf("No element to delete \n");
else if(pos==1)
     del_beg();
else if (pos<=count)
{
h=head;
for(int i=1;i<=pos-1;i++)
{
h=h->next;
}
h->prev->next=h->next;
h->next->prev=h->prev;
free(h);
count--;
printf(" Node Deleted from given position \n");
}
else if (pos==count)
     del_end();
else
    printf("Given position is invalid \n");
}
void display()
{
if(head==NULL)
    printf("\n Nothing to Display");
else
{
h=head;
while(h!=NULL)
{
printf("%d <--> ",h->data);
h=h->next;
```

```
}
}
}
int main()
{
int ch;
printf("choose an operation for doubly linked list: \n1.Insertion at beginning \n2.Insertion at end\n3.Insertion at a position \n4.Deletion from beginning \n5.Deletion from end\n6.Deletion from a position \n7.Display \n8.Exit");
while(1)
{
printf("\nEnter a choice :");
scanf("%d",&ch);
switch(ch)
{
case 1: ins_beg();
break;
case 2: ins_end();
break;
case 3: ins_pos();
break;
case 4: del_beg();
break;
case 5: del_end();
break;
case 6: del_pos();
break;
case 7:display();
break;
case 8:exit(0);
break;
case 9: printf("invalid value");
break;
}
}
return 0;
}
```

Output :



```
stud@debian:~/suhaila$ gedit doubly.c
stud@debian:~/suhaila$ gcc doubly.c
stud@debian:~/suhaila$ ./a.out
choose an operation for doubly linked list:
1.Insertion at beginning
2.Insertion at end
3.Insertion at a position
4.Deletion from beginning
5.Deletion from end
6.Deletion from a position
7.Display
8.Exit
Enter a choice :1
Enter the value to the node : 13
 Node Inserted at beginning

Enter a choice :2
Enter the value to the node : 16
Node Inserted at end

Enter a choice :3
Enter a position starting from 1 :2
Enter the value to the node : 17
 Node Inserted at given position

Enter a choice :7
13 <--> 17 <--> 16 <-->
Enter a choice :3
Enter a position starting from 1 :6

 Given position is invalid
Enter a choice :4
Node Deleted from beginning

Enter a choice :7
17 <--> 16 <-->
Enter a choice :5
 Node Deleted from end

Enter a choice :7
17 <-->
Enter a choice :6
Enter a position starting from 1 :2
Given position is invalid

Enter a choice :6
Enter a position starting from 1 :1
Node Deleted from beginning

Enter a choice :7

 Nothing to Display
Enter a choice :4
No element to Delete

Enter a choice :8
stud@debian:~/suhaila$ 
```

Experiment No : **10**

Aim : Menu driven program to implement BST operations

Concept : Binary Search Tree

Description : A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.

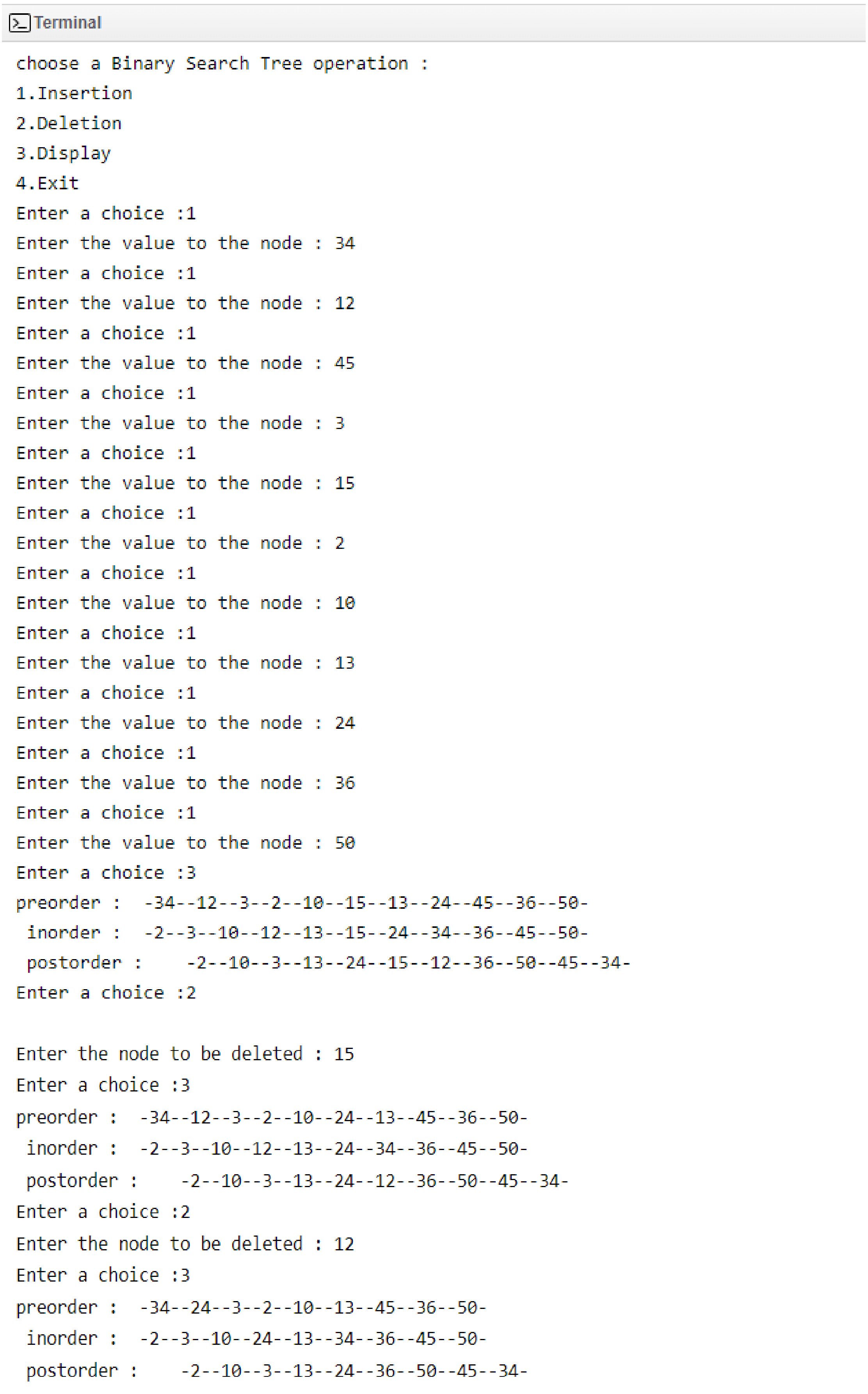
Program code :

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   struct node *lchild;
   int data;
   struct node *rchild;
};
struct node *root = NULL,*new,*r=NULL,*rt,*t1,*tp;
void create()
{
   int v;
   new=(struct node*)malloc(sizeof(struct node));
   printf("Enter the value to the node : ");
   scanf("%d",&v);
   new->data=v;
   new->lchild=NULL;
   new->rchild=NULL;
}
void search(struct node *rt)
{
   if(new->data < rt->data && rt->lchild==NULL)
      rt->lchild=new;
   else if(new->data > rt->data && rt->rchild==NULL)
      rt->rchild=new;
   else if(new->data < rt->data && rt->lchild!=NULL)
      search(rt->lchild);
   else if(new->data > rt->data && rt->rchild!=NULL)
      search(rt->rchild);
   else if(new->data ==rt->data)
      printf("\nSame value has been entered twice!");
   else
      printf("\nInvalid Entry");
}
void insert()
{
   create();
   if(root==NULL)
      root=new;
   else
      search(root);
}
void deletenode(struct node *rt)
{
   if(r==NULL)
      r=rt;
   if(rt->lchild==NULL && rt->rchild==NULL) {
      if(rt==r->lchild)
         r->lchild=NULL;
      else if(rt==r->rchild)
         r->rchild=NULL;
      else
         printf("\n Invalid Value");
     if(root == rt)
         root = NULL;
      free(rt);
   }
   else if(rt->lchild!=NULL && rt->rchild== NULL) {
    if(rt==r) {
        root=rt->lchild;
    } else if(rt==r->lchild) {
        r->lchild=rt->lchild;
    } else if(rt==r->rchild) {
        r->rchild=rt->rchild;
    } else
        printf("\n Invalid Value");
    free(rt);
}
else if(rt->lchild==NULL && rt->rchild!= NULL) {
    if(rt==r) {
        root=rt->rchild;
    } else if(rt==r->lchild) {
        r->lchild=rt->rchild;
    } else if(rt==r->rchild) {
        r->rchild=rt->rchild;
    } else
        printf("\n Invalid Value");
    free(rt);
}
else if(rt->lchild!=NULL && rt->rchild!=NULL) {
    t1=rt->rchild;
    if(t1->rchild !=NULL) {
        tp=rt;
        while(t1->lchild!=NULL) {
            tp=t1;
            t1=t1->lchild;
        }
        rt->data=t1->data;
        if(t1->rchild !=NULL)
            tp ->rchild = t1 -> rchild;
        if(tp != rt)
```

```
            tp ->lchild = NULL;
          free(t1);
        }
      else {
          rt ->data = t1 ->data;
          if(t1 -> lchild != NULL)
              rt -> rchild = t1 -> lchild;
           else
              rt -> rchild =NULL;
          free(t1);
        }
    }
      else
        printf("\n Invalid Value ");
}
void dsearch(struct node *rt,int dval)
{
    if(rt->data==dval) {
        deletenode(rt);
    } else if(dval < rt->data && rt->lchild!=NULL) {
        r=rt;
        dsearch(rt->lchild,dval );
    } else if(dval > rt->data && rt->rchild!=NULL) {
        r=rt;
        dsearch(rt->rchild,dval);
    } else
        printf("\nValue not found !");
}
void delete()
{
    int dval;
    printf("Enter the node to be deleted : ");
    scanf("%d",&dval);
```

```
   if(root==NULL)
      printf("\n The BST is Empty");
   else
      dsearch(root,dval);
}
void preorder(struct node *root)
{
   if(root!=NULL) {
      printf("-%d-",root->data);
      preorder(root->lchild);
      preorder(root->rchild);
   }
}
void inorder(struct node *root)
{
   if(root!=NULL) {
      inorder(root->lchild);
      printf("-%d-",root->data);
      inorder(root->rchild);
   }
}
void postorder(struct node *root)
{
   if(root!=NULL) {
      postorder(root->lchild);
      postorder(root->rchild);
      printf("-%d-",root->data);
   }
}
void display()
{
   printf("\n preorder : \t");
   preorder(root);
   printf("\n inorder : \t");
   inorder(root);
   printf("\n postorder : \t");
   postorder(root);
}
int main()
{
   int ch;
   printf("choose a Binary Search Tree operation : \n1.Insertion \n2.Deletion \n3.Display
\n4.Exit");
   while(1) {
      printf("\nEnter a choice :");
      scanf("%d",&ch);
      switch(ch) {
      case 1:
         insert();
         break;
      case 2:
         delete();
         break;
      case 3:
         display();
         break;
      case 4:
         exit(0);
         break;
      default :
         printf("invalid value");
         break;
      }
   }
   return 0;
}
```

Output :


Terminal

```
choose a Binary Search Tree operation :
1.Insertion
2.Deletion
3.Display
4.Exit
Enter a choice :1
Enter the value to the node : 34
Enter a choice :1
Enter the value to the node : 12
Enter a choice :1
Enter the value to the node : 45
Enter a choice :1
Enter the value to the node : 3
Enter a choice :1
Enter the value to the node : 15
Enter a choice :1
Enter the value to the node : 2
Enter a choice :1
Enter the value to the node : 10
Enter a choice :1
Enter the value to the node : 13
Enter a choice :1
Enter the value to the node : 24
Enter a choice :1
Enter the value to the node : 36
Enter a choice :1
Enter the value to the node : 50
Enter a choice :3
preorder :  -34--12--3--2--10--15--13--24--45--36--50-
 inorder :  -2--3--10--12--13--15--24--34--36--45--50-
 postorder :    -2--10--3--13--24--15--12--36--50--45--34-
Enter a choice :2

Enter the node to be deleted : 15
Enter a choice :3
preorder :  -34--12--3--2--10--24--13--45--36--50-
 inorder :  -2--3--10--12--13--24--34--36--45--50-
 postorder :    -2--10--3--13--24--12--36--50--45--34-
Enter a choice :2
Enter the node to be deleted : 12
Enter a choice :3
preorder :  -34--24--3--2--10--13--45--36--50-
 inorder :  -2--3--10--24--13--34--36--45--50-
 postorder :    -2--10--3--13--24--36--50--45--34-
```

```
Enter a choice :2
Enter the node to be deleted : 34
Enter a choice :3
preorder :  -36--24--3--2--10--13--45--50-
 inorder :  -2--3--10--24--13--36--45--50-
 postorder :    -2--10--3--13--24--50--45--36-
Enter a choice :2
Enter the node to be deleted : 10
Enter a choice :3
preorder :  -36--24--3--2--13--45--50-
 inorder :  -2--3--24--13--36--45--50-
 postorder :    -2--3--13--24--50--45--36-
Enter a choice :2
Enter the node to be deleted : 24
Enter a choice :2
Enter the node to be deleted : 3
Enter a choice :3

preorder :  -36--13--2--45--50-
inorder :   -2--13--36--45--50-
 postorder :    -2--13--50--45--36-
Enter a choice :2
Enter the node to be deleted : 45
Enter a choice :3
preorder :  -36--13--2--50-
 inorder :  -2--13--36--50-
 postorder :    -2--13--50--36-
Enter a choice :2
Enter the node to be deleted : 13
Enter a choice :3
preorder :  -36--2--50-
 inorder :  -2--36--50-
 postorder :    -2--50--36-
Enter a choice :1
Enter the value to the node : 4
Enter a choice :3
preorder :  -36--2--4--50-
 inorder :  -2--4--36--50-
 postorder :    -4--2--50--36-
Enter a choice :4
```

Experiment No : **11**

Aim : Program to implement BFS

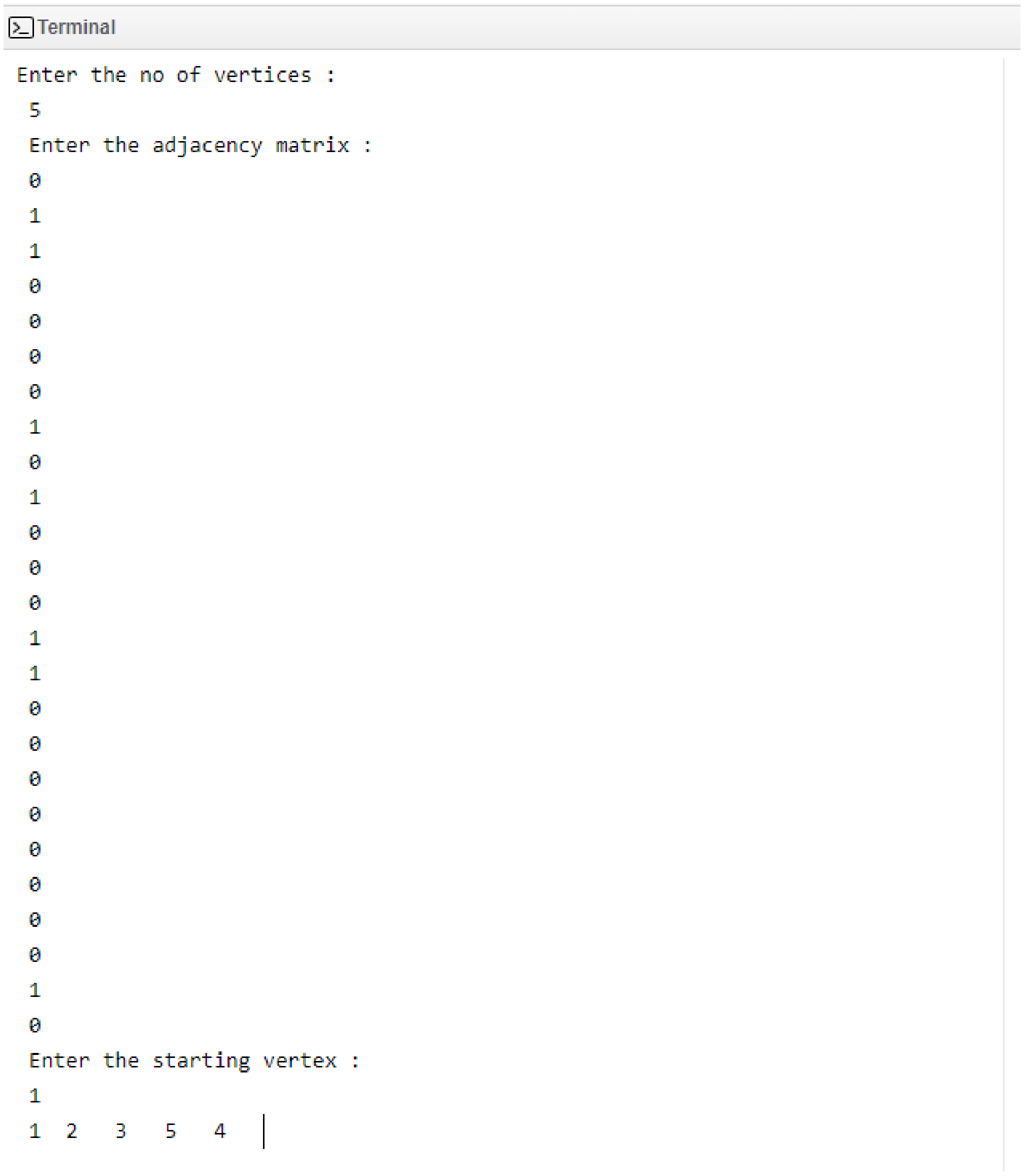
Concept : Breadth First Search

Description : Breadth-First Search**,** is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue.

Program code :

```
#include<stdio.h>
int n,s,adj[10][10],queue[10];
int visited[10]= {0,0,0,0,0,0,0,0,0,0};
int front=-1,rear=-1,item;
void enqueue(int item)
{
   if(rear==9)
      printf("Queue if Full \n");
   else {
      if(rear==-1) {
         front=rear=0;
         queue[rear]=item;
      } else {
         rear=rear+1;
         queue[rear]=item;
      }
   }
}
int dequeue()
{
   int k;
   if((front>rear) || (front==-1))
      return(0);
   else {
        k=queue[front];
        front++;
        return(k);
    }
}
void bfs(int s,int n)
{
    int p;
    enqueue(s);
    visited[s]=1;
    p=dequeue();
    if(p!=0) {
        printf("%d \t",p);
    }
    while(p!=0) {
        for(int i=1; i<=n; i++) {
            if(adj[p][i]==1 && visited[i]==0) {
                enqueue(i);
                visited[i]=1;
            }
        }
        p=dequeue();
        if(p!=0) {
            printf("%d \t",p);
        }
    }
}
int main()
{
    printf("Enter the no of vertices : \n ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix : \n ");
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) {
            scanf("%d",&adj[i][j]);
        }
    }
    printf("Enter the starting vertex : \n ");
    scanf("%d",&s);
    bfs(s,n);
    return 0;
}
```

Output :



Terminal

```
Enter the no of vertices :
 5
 Enter the adjacency matrix :
 0
 1
 1
 0
 0
 0
 0
 1
 0
 1
 0
 0
 0
 1
 1
 0
 0
 0
 0
 0
 0
 0
 0
 1
 0
 Enter the starting vertex :
 1
 1  2   3   5   4
```

Experiment No : **12**

Aim : Program to implement DFS

Concept : Depth First Search

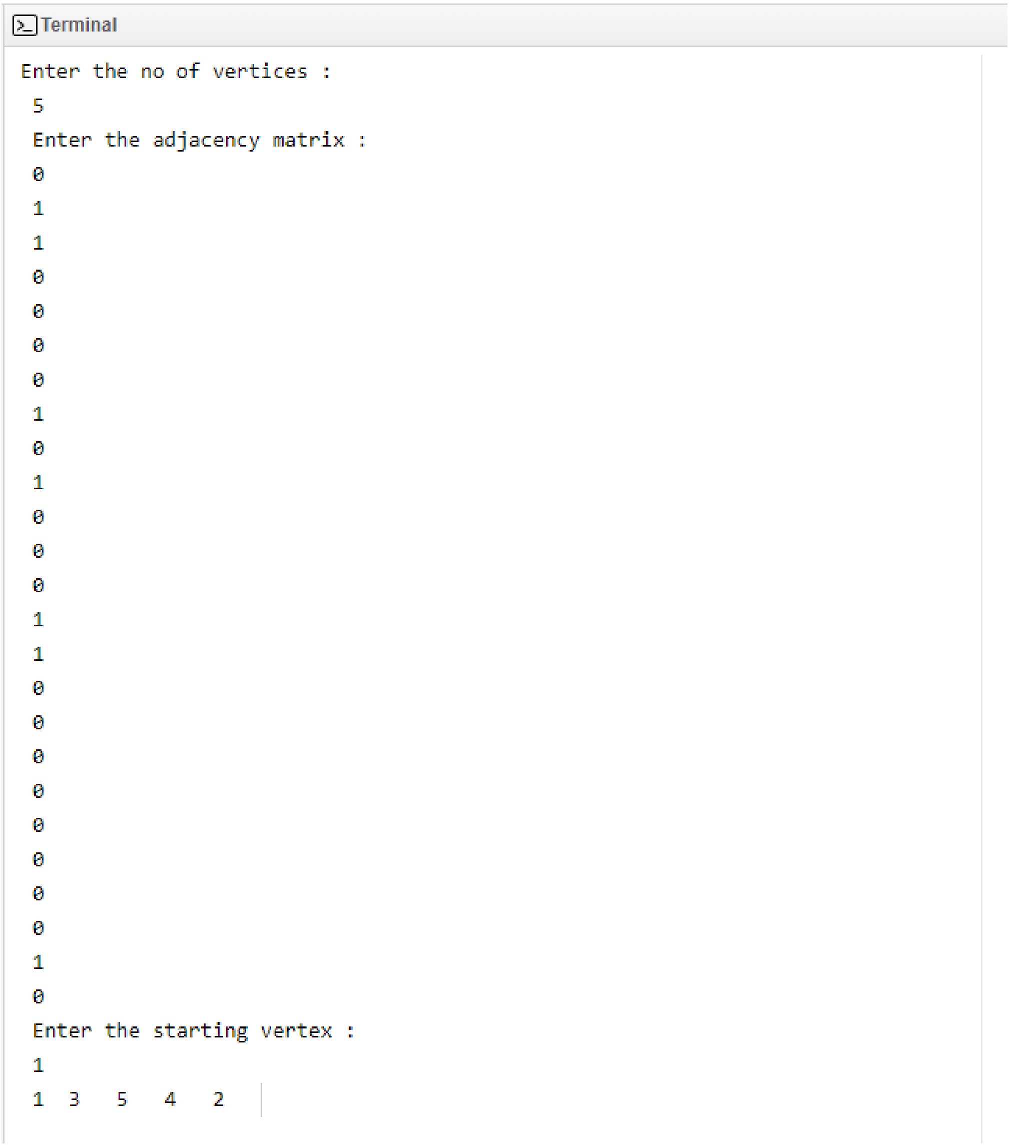
Description : Depth First Search**,** is an edge-based technique. It uses the Stack data structure and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices' then visited vertices are popped.

Program code :

```
#include<stdio.h>
int n,s,adj[10][10],stack[10];
int visited[10]= {0,0,0,0,0,0,0,0,0,0};
int top=-1;
int item;
void push(int item)
{
   if(top==9)
      printf("Stack if Full \n");
   else {
      if(top==-1) {
         top=0;
         stack[top]=item;
      } else {
         top=top+1;
         stack[top]=item;
      }
   }
}
int pop()
{
   int k;
   if(top==-1)
      return(0);
```

```
    else {
        k=stack[top];
        top--;
        return(k);
    }
}
void dfs(int s,int n)
{
    int p;
    push(s);
    visited[s]=1;
    p=pop();
    if(p!=0) {
        printf("%d \t",p);
    }
    while(p!=0) {
        for(int i=1; i<=n; i++) {
            if(adj[p][i]==1 && visited[i]==0) {
                push(i);
                visited[i]=1;
            }
        }
        p=pop();
        if(p!=0) {
            printf("%d \t",p);
        }
    }
}
int main()
{
    printf("Enter the no of vertices : \n ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix : \n ");
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) {
            scanf("%d",&adj[i][j]);
        }
    }
    printf("Enter the starting vertex : \n ");
    scanf("%d",&s);
    dfs(s,n);
    return 0;
}
```

Output :


Terminal

```
Enter the no of vertices :
 5
 Enter the adjacency matrix :
 0
 1
 1
 0
 0
 0
 0
 1
 0
 1
 0
 0
 0
 1
 1
 0
 0
 0
 0
 0
 0
 0
 0
 1
 0
 Enter the starting vertex :
 1
 1  3   5   4   2
```

Experiment No : **13**

Aim : Program to implement Prim's algorithm

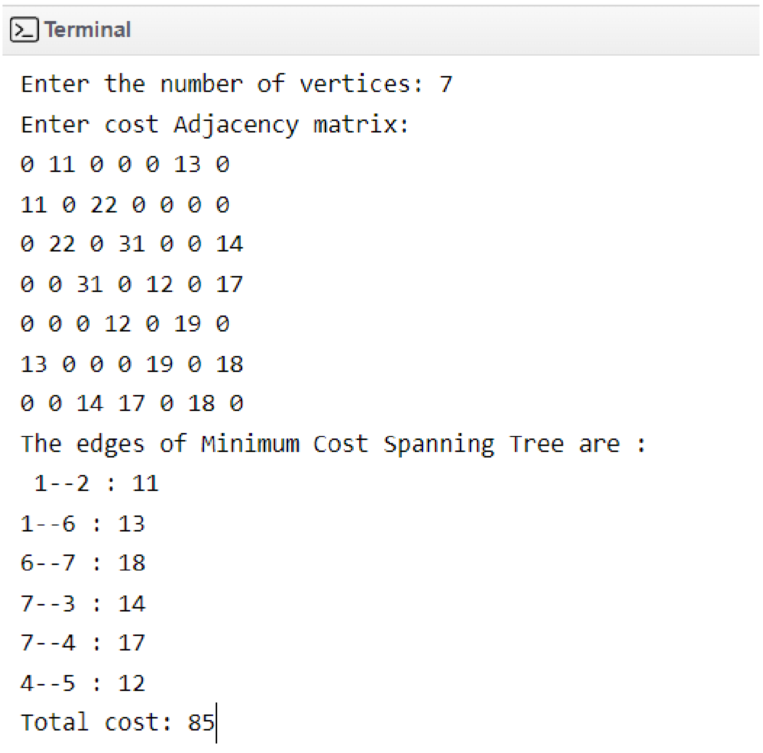
Concept : Minimum cost Spanning Tree using Prim's Algorithm

Description : Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. It finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized. It starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

Program code :

```
#include<stdio.h>
#define INF 999
int cost[10][10],visited[10]= {0,0,0,0,0,0,0,0,0,0},min;
int n,no_edges=0,total_cost=0;
int main()
{
   printf("Enter the number of vertices: ");
   scanf("%d",&n);
   printf("Enter cost Adjacency matrix:\n");
   for(int i=1; i<=n; i++) {
      for(int j=1; j<=n; j++) {
         scanf("%d",&cost[i][j]);
         if(cost[i][j]==0) {
            cost[i][j]=INF;
         }
      }
   }
   visited[1]=1;
   printf("The edges of Minimum Cost Spanning Tree are :\n ");
   while(no_edges < n-1) {
      min=INF;
      int a=0;
      int b=0;
      for(int i=1; i<=n; i++) {
            if(visited[i]==1) {
                for(int j=1; j<=n; j++) {
                    if(visited[j]==0 && cost[i][j]!=INF) {
                        if(cost[i][j]<min) {
                            min=cost[i][j];
                            a=i;
                            b=j;
                        }
                    }
                }
            }
        }
        no_edges++;
        visited[b]=1;
        printf("%d--%d : %d\n",a,b,min);
        total_cost=total_cost+min;
    }
    printf("Total cost: %d",total_cost);
}
```

Output :



Terminal

```
Enter the number of vertices: 7
Enter cost Adjacency matrix:
0 11 0 0 0 13 0
11 0 22 0 0 0 0
0 22 0 31 0 0 14
0 0 31 0 12 0 17
0 0 0 12 0 19 0
13 0 0 0 19 0 18
0 0 14 17 0 18 0
The edges of Minimum Cost Spanning Tree are :
 1--2 : 11
1--6 : 13
6--7 : 18
7--3 : 14
7--4 : 17
4--5 : 12
Total cost: 85
```

Experiment No : **14**

Aim : Program to implement Kruskal's algorithm

Concept : Minimum cost Spanning Tree using Kruskal's algorithm

Description : Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. It finds the subset of edges by using which we can traverse every vertex of the graph. It follows the greedy approach that finds an optimum solution at every stage instead of focusing on a global optimum. In this we start from edges with the lowest weight and keep adding the edges until the goal is reached.

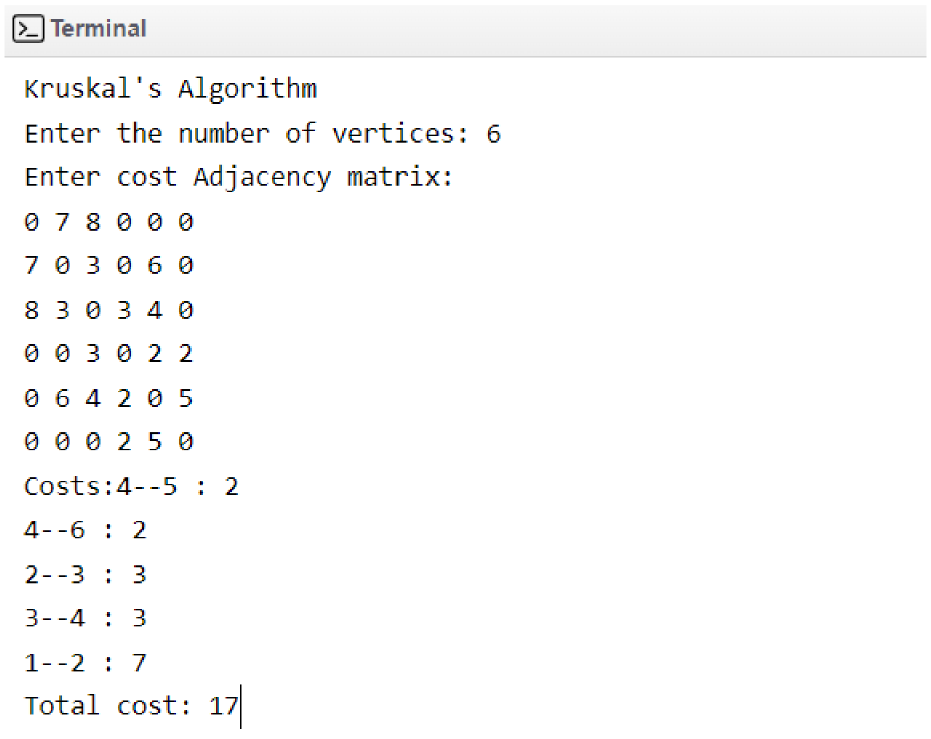
Program code :

```
#include<stdio.h>
#define INF 999
int stack[10];
int top = -1;
int pass = 0;
int cost[10][10];
int adj[10][10];
int visited[10]= {0,0,0,0,0,0,0,0,0,0};
int visited2[10] = {0,0,0,0,0,0,0,0,0,0};
int n,no_edges=0,total_cost=0,min,a,b;

void push(int item);
int pop();
int dft(int s, int ccp);
void kruskal(int s);

int main()
{
   printf("Kruskal's Algorithm \n");
   printf("Enter the number of vertices: ");
   scanf("%d",&n);
   printf("Enter cost Adjacency matrix:\n");
   for(int i=1; i<=n; i++) {
       for(int j=1; j<=n; j++) {
           scanf("%d",&cost[i][j]);
           if(cost[i][j]==0) {
               cost[i][j]=INF;
           }
           adj[i][j] = 0;
       }
   }
   kruskal(1);
```

```
}
void kruskal(int s)
{
   visited[s] = 1;
   printf("\nCosts:");
   while(no_edges < n - 1) {
      min = INF;
      a=0;
      b=0;
      for(int i = 1; i <= n; i++) {
         for(int j = 1; j <= n; j++) {
            pass = 0;
            if(cost[i][j] < min) {
               pass = dft(i, j);
               if(pass != 1) {
                  min = cost[i][j];
                  a = i;
                  b = j;
               }
               for(int i = 1; i <= n; i++) {
                  visited2[i] = 0;
               }
               while(top > -1) {
                  stack[top] = 0;
                  top--;
               }
            }
         }
      }
      printf("%d--%d : %d\n",a,b,min);
      total_cost=total_cost+min;
      visited[a]=1;
      visited[b]=1;
      adj[a][b] = adj[b][a] = 1;
      cost[a][b]=cost[b][a]=INF;
      no_edges++;
   }
   printf("Total cost: %d",total_cost);
}


void push(int item)
{
   if(top == 9) {
      printf("Stack Overflow");
```

```
    } else {
        top++;
        stack[top] = item;
    }
}

int pop()
{
    int val;
    if(top == -1) {
        return(0);
    } else {
        val = stack[top];
        top--;
        return(val);
    }
}

int dft(int s, int ccp)
{
    int p;
    push(s);
    visited2[s] = 1;
    p = pop();
    if(p != 0) {
        if(p == ccp) {
            return 1;
        }
    }
    while(p != 0) {
        for(int i = 1; i <= n; i++) {
            if((adj[p][i] == 1) && (visited2[i] == 0)) {
                push(i);
                visited2[i] = 1;
            }
        }
        p = pop();
        if(p != 0) {
            if(p == ccp) {
                return 1;
            }
        }
    }
    return 0;
}
```

Output :



```
Kruskal's Algorithm
Enter the number of vertices: 6
Enter cost Adjacency matrix:
0 7 8 0 0 0
7 0 3 0 6 0
8 3 0 3 4 0
0 0 3 0 2 2
0 6 4 2 0 5
0 0 0 2 5 0
Costs:4--5 : 2
4--6 : 2
2--3 : 3
3--4 : 3
1--2 : 7
Total cost: 17
```