1)Write a C program to sort an integer array

**PROGRAM**

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j,a,n,number[30];

printf("Enter the value of n :");

scanf("%d", &n);

printf("Enter the number\n");

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

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

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

{

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

{

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

{

a= number[i];

number[i]=number[j];

number[j]=a;

}

}

}

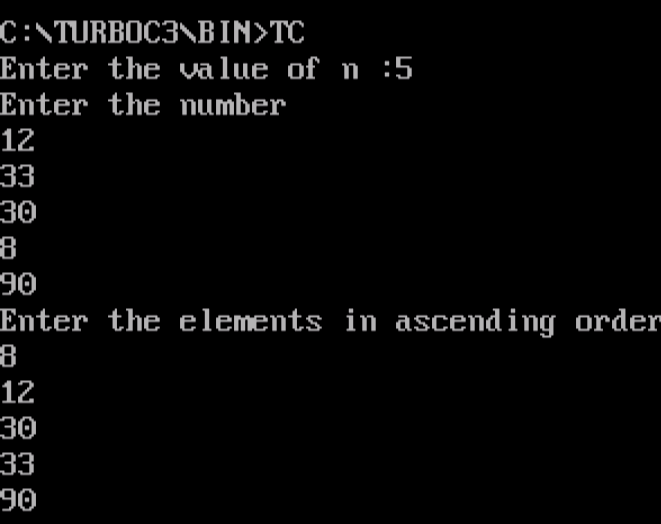
printf("Enter the elements in ascending order\n");

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

printf("%d\n", number[i]);

getch();}

**OUTPUT**

****

2)Write a Program to merge two sorted arrays

**PROGRAM**

|  |
| --- |
|  |

#include<stdio.h>

#include<conio.h>

void main()

{

int array1[50], array2[50], array3[100], m, n, i, j, k=0;

clrscr();

printf("\n Enter the size of array :");

scanf("%d", &m);

printf("\n Ente the sorted elements of first array\n");

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

{

scanf("%d", &array1[i]);

}

printf("\n Enter size of second array ");

scanf("%d", &n);

printf("\n Enter sorted elements of second array\n");

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

{

scanf("%d", &array2[i]);

}

i=0;

j=0;

while (i<m&&j<n)

{

if(array1[i]<array2[j])

{

array3[k]=array1[i];

i++;

}

else

{

array3[k]=array2[j];

j++;

}

k++;

}

if(i>=m)

{

while(j<n)

{

array3[k]=array2[j];

j++;

k++;

}

}

if(j>=n)

{

while(i<m)

{

array3[k]=array1[i];

i++;

k++;

}

}

printf("\n after merging :\n");

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

{

printf("\n%d",array3[i]);

}

getch();

}

OUTPUT

****

3)write a program to implement stack operations.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define MAXSIZE 5

struct stack

{

int stk[MAXSIZE];

int top;

};

typedef struct stack STACK;

STACK s;

void push(void);

int pop(void);

void display(void);

void main ()

{

int choice;

int option = 1;

clrscr();

s.top = -1;

printf ("STACK OPERATION\n");

while (option)

{

printf ("------------------------------------------\n");

printf (" 1 --> PUSH \n");

printf (" 2 --> POP \n");

printf (" 3 --> DISPLAY \n");

printf (" 4 --> EXIT \n");

printf ("------------------------------------------\n");

printf ("Enter your choice\n");

scanf ("%d", &choice);

switch (choice)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

return;

}

fflush (stdin);

printf ("Do you want to continue(0 or 1)?\n");

scanf ("%d", &option);

}

}

/\* Function to add an element to the stack \*/

void push ()

{

int num;

if (s.top == (MAXSIZE - 1))

{

printf ("Stack is Full\n");

return;

}

else

{

printf ("Enter the element to be pushed\n");

scanf ("%d", &num);

s.top = s.top + 1;

s.stk[s.top] = num;

}

return;

}

/\* Function to delete an element from the stack \*/

int pop ()

{

int num;

if (s.top == - 1)

{

printf ("Stack is Empty\n");

return (s.top);

}

else

{

num = s.stk[s.top];

printf ("poped element is = %d\n", s.stk[s.top]);

s.top = s.top - 1;

}

return(num);

}

/\* Function to display the status of the stack \*/

void display ()

{

int i;

if (s.top == -1)

{

printf ("Stack is empty\n");

return;

}

else

{

printf ("\n The status of the stack is \n");

for (i = s.top; i >= 0; i--)

{

printf ("%d\n", s.stk[i]);

}

}

printf ("\n");

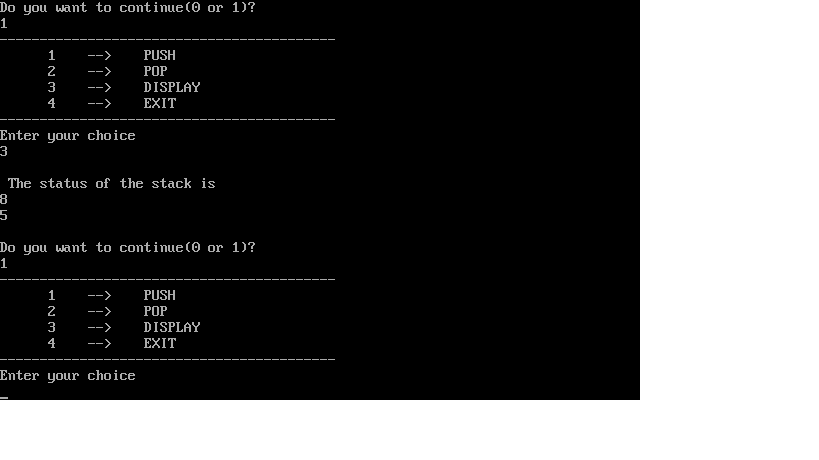
}

|  |
| --- |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**OUTPUT**







4)Write a program to implement Circular Queue.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define MAX 5

int cqueue\_arr[MAX];

int front=-1;

int rear=-1;

void insert(int item)

{

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

{

printf("queue overflow\n");

return;

}

else if(front == -1)

{

front=0;

rear=0;

}

else

{

if(rear==MAX-1)

{

rear=0;

}

else

{

rear=rear+1;

}

}

cqueue\_arr[rear]=item;

}

void deletion()

{

if(front==-1)

{

printf("queue underflow\n");

return;

}

printf("element deleted from queue is : %d ",cqueue\_arr[front]);

if(front==rear)

{

front=rear=-1;

}

else

{

if(front==MAX-1)

front=0;

else

front+=1;

}

}

void display()

{

int front\_pos=front,rear\_pos=rear;

if(front==-1)

{

printf("empty queue\n");

return;

}

printf("queue elements are : ");

if(front\_pos<=rear\_pos)

{

while(front\_pos<=rear\_pos)

{

printf("%d",cqueue\_arr[front\_pos]);

front\_pos++;

}

}

else

{

while(front\_pos<=MAX-1)

{

printf("%d",cqueue\_arr[front\_pos]);

front\_pos++;

}

front\_pos=0;

while(front\_pos<=rear\_pos)

{

printf("%d",cqueue\_arr[front\_pos]);

front\_pos++;

}

}

printf("\n");

}

void main()

{

int choice,item;

clrscr();

do

{

printf("\n1.insert\n2.delete\n3.display\n4.quit\n Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("input the elements to queue :");

scanf("%d",&item);

insert(item);

break;

case 2:

deletion();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("invalid input !!");

}

}while(choice!=4);

getch();

}

|  |
| --- |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**OUTPUT**





5)Write a program to implement linked stack.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<limits.h>

#define CAPACITY 1000

struct stack

{

int data;

struct stack \*next;

} \*top;

int size = 0;

void push(int element);

int pop();

void main()

{

int choice, data;

clrscr();

while(1)

{

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

printf(" Stack Implementation Program \n");

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

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Size\n");

printf("4. Exit\n");

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

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter data to push into stack: ");

scanf("%d", &data);

push(data);

break;

case 2:

data = pop();

if (data != INT\_MIN)

printf("Data => %d\n", data);

break;

case 3:

printf("Stack size: %d\n", size);

break;

case 4:

printf("Exiting from app...\n");

exit(0);

break;

default:

printf("Invalid choice, Please try again.\n");

}

printf("\n\n");

}

}

void push(int element)

{

struct stack \* newNode = (struct stack \*) malloc(sizeof(struct stack));

if(size >= CAPACITY)

{

printf("Stack Overflow, can't add more element to stack.\n");

return;

}

newNode->data = element;

newNode->next = top;

top = newNode;

size++;

printf("Data pushed to stack.\n");

}

int pop()

{

int data = 0;

struct stack \* topNode;

if (size <=0 || !top)

{

printf("Stack is empty.\n");

return INT\_MIN;

}

topNode = top;

data = top->data;

top = top->next;

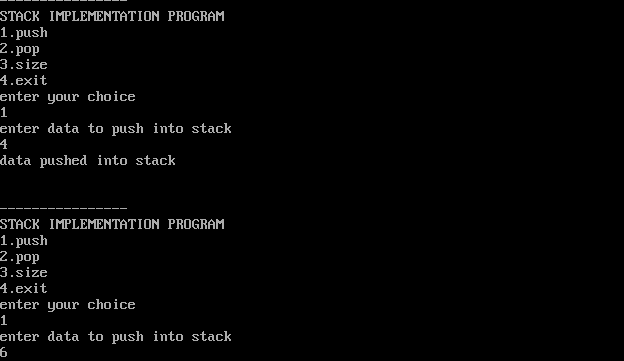
free(topNode);

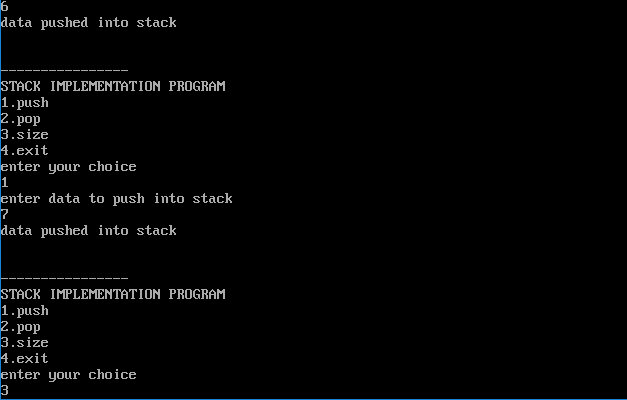
size--;

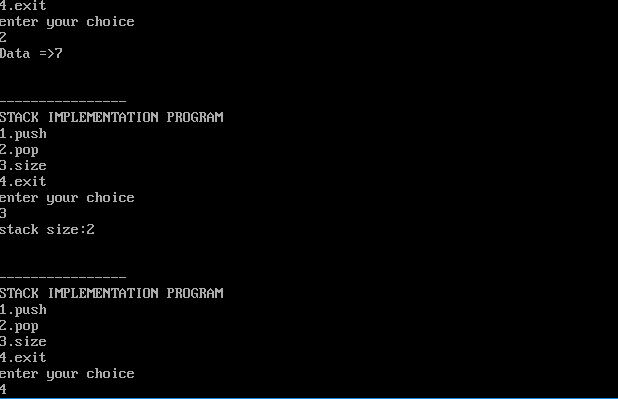
return data;

}

**OUTPUT**

****





6)Write a program to implement doubly linked list.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

void main()

{

int choice=0;

clrscr();

while(choice!=9)

{

printf("\n\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\n");

printf("\nChoose one option from the following list...\n");

printf("\n=============================================\n");

printf("\n1.Insert at the beginning\n2.Insert at the last\n3.Insert at any random location\n4.Delete from beginning\n5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

printf("\nEnter your choice:\n");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

insertion\_beginning();

break;

case 2:

insertion\_last();

break;

case 3:

insertion\_specified();

break;

case 4:

deletion\_beginning();

break;

case 5:

deletion\_last();

break;

case 6:

deletion\_specified();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

getch();

}

void insertion\_beginning()

{

struct node \*ptr;

int item;

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

if(ptr==NULL)

{

printf("\nOverflow");

}

else

{

printf("\nEnter item value:");

scanf("%d",&item);

if(head==NULL)

{

ptr->next=NULL;

ptr->prev=NULL;

ptr->data=item;

head=ptr;

}

else

{

ptr->data=item;

ptr->prev=NULL;

ptr->next=head;

head->prev=ptr;

head=ptr;

}

printf("\nNode inserted\n");

}

}

void insertion\_last()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr==NULL)

{

printf("\nOverflow\n");

}

else

{

printf("Enter the value:");

scanf("%d",&item);

ptr->data=item;

if(head==NULL)

{

ptr->next=NULL;

ptr->prev=NULL;

head=ptr;

}

else

{

temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=ptr;

ptr->prev=temp;

ptr->next=NULL;

}

}

printf("\nNode inserted\n");

}

void insertion\_specified()

{

struct node \*ptr,\*temp;

int item,loc,i;

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

if(ptr==NULL)

{

printf("\nOverflow");

}

else

{

temp=head;

printf("Enter the location:");

scanf("%d",&loc);

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

{

temp=temp->next;

if(temp==NULL)

{

printf("\n There are less than %d elements",loc);

return;

}

}

printf("Enter value:");

scanf("%d",&item);

ptr->data=item;

ptr->next=temp->next;

ptr->prev=temp;

temp->next=ptr;

temp->next->prev=ptr;

printf("\nNode inserted\n");

}

}

void deletion\_beginning()

{

struct node \*ptr;

if(head==NULL)

{

printf("\nUnderflow");

}

else if(head->next==NULL)

{

head=NULL;

free(head);

printf("\nNode deleted\n");

}

else

{

ptr=head;

head=head->next;

head->prev=NULL;

free(ptr);

printf("\nNode deleted\n");

}

}

void deletion\_last()

{

struct node \*ptr;

if(head==NULL)

{

printf("\nUnderflow");

}

else if(head->next==NULL)

{

head=NULL;

free(head);

printf("\nNode deleted\n");

}

else

{

ptr=head;

if(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->prev->next=NULL;

free(ptr);

printf("\nNode Deleted");

}

}

void deletion\_specified()

{

struct node \*ptr,\*temp;

int val;

printf("\nEnter the data after which the node is to be deleted:");

scanf("%d",&val);

ptr=head;

while(ptr->data!=val)

ptr=ptr->next;

if(ptr->next==NULL)

{

printf("\nCan't delete\n");

}

else if(ptr->next->next==NULL)

{

ptr->next=NULL;

}

else

{

temp=ptr->next;

ptr->next=temp->next;

temp->next->prev=ptr;

free(temp);

printf("\nNode deleted\n");

}

}

void display()

{

struct node \*ptr;

printf("\nPrinting values...\n");

ptr=head;

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag;

ptr=head;

if(ptr==NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while(ptr!=NULL)

{

if(ptr->data==item)

{

printf("\nItem found at location %d",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr=ptr->next;

}

if(flag==1)

{

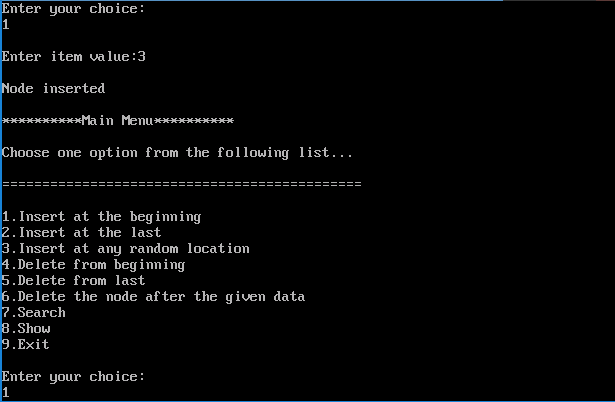
printf("\nItem not found\n");

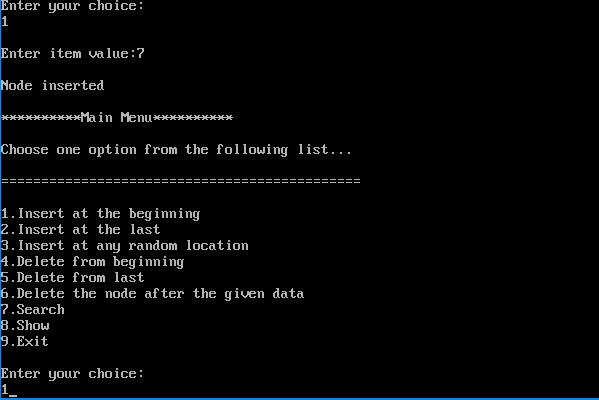
}

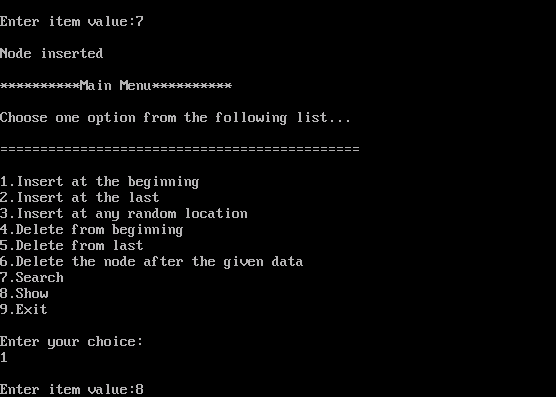
}

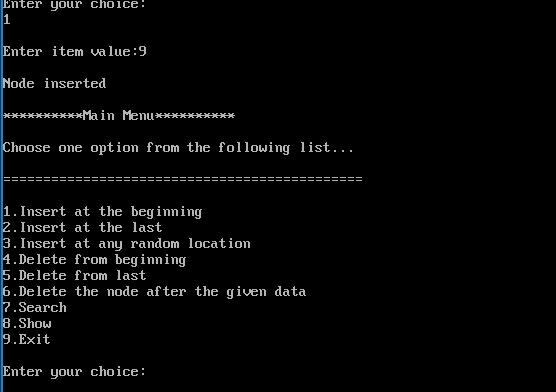
}

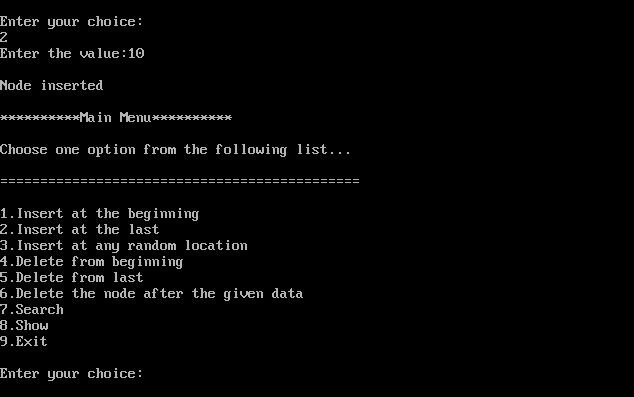
**OUTPUT**

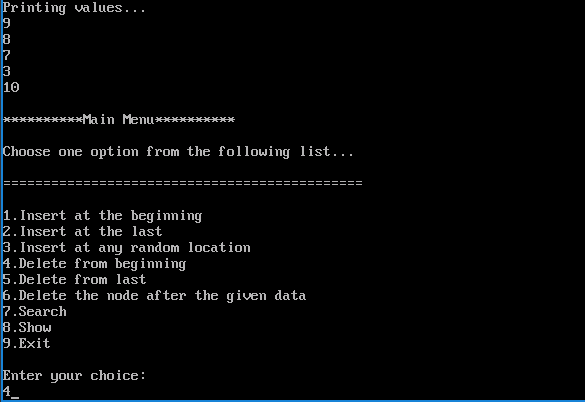
****

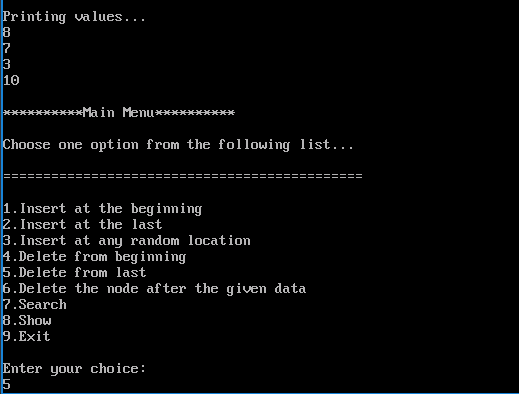
****

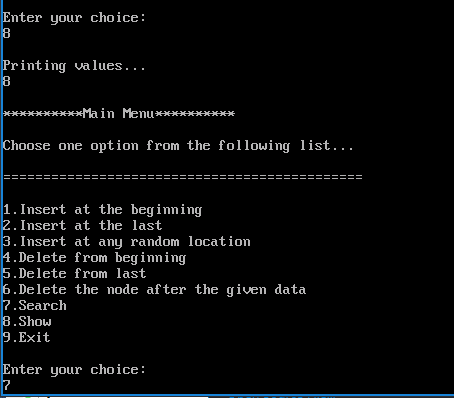
****

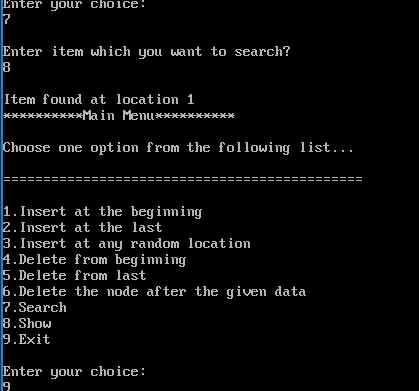
****

****

****

****

****

****

7)Write a program to implement set operations.

**PROGRAM**

#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};

void main()

{

int ch,wish;

do

{

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

printf("1.Input\n2.Union\n3.Intersection\n4.Difference\n");

printf("Enter choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1:input();

break;

case 2:setunion();

break;

case 3:intersection();

break;

case 4:difference();

break;

}

printf("\nDo you wish to continue ?(1/0)\n");

scanf("%d",&wish);

}

while(wish==1);

}

void input()

{

int n,x,i;

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

scanf("%d",&n);

printf("Ente elements\n");

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

{

scanf("%d",&x);

a[x]=1;

}

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

scanf("%d",&n);

printf("enter elements\n");

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

{

scanf("%d",&x);

b[x-1]=1;

}

printf("\n 1st set\n");

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

{

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

}

printf("\n 2nd set\n");

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<0;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]);

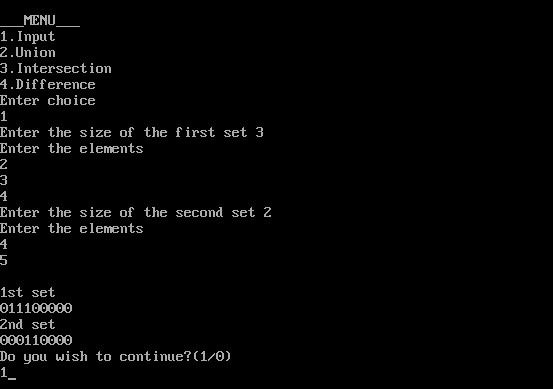
}

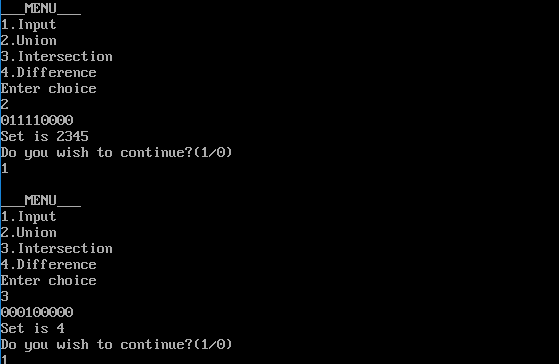
output(c);

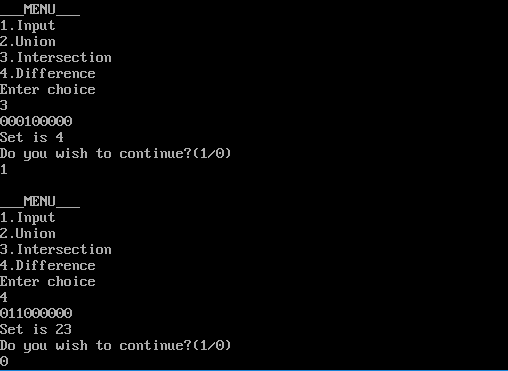
getch();

}

**OUTPUT**

****

****

****

8)Write a program to implement disjoint set operations.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

struct DisjSet

{

int parent[10];

int rank[10];

int n;

}

dis;

void makeSet()

{

int i;

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

{

dis.parent[i]=i;

dis.rank[i]=0;

}

}

void displaySet()

{

int i;

printf("\nParent Array\n");

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

{

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

}

printf("\nRank Array\n");

for(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 x,y,n,ch,wish;

clrscr();

printf("How many elements?");

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

makeSet();

do

{

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

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

printf("Enter choice:\n");

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 connected components");

break;

case 3:

displaySet();

break;

}

printf("\nDo you wish to continue?(1/0)\n");

scanf("%d",&wish);

}

while(wish==1);

getch();

return 0;

}

**OUTPUT**

****

9)Write a program to implement Graph traversal operations and find the strongly connected components.

**PROGRAM**

#include<conio.h>

#include<stdio.h>

struct node

{

int vertex;

struct node \*next;

};

int v,e;

struct node \*\*adj,\*\*adj1;

int \*que,\*visited,\*ft;

int f=-1,r=-1,t=0;

void dfs();

void dfsvisit(int);

void dfs1();

void dfsvisit1(int);

void adjlistRep(struct node \*\*adj,int s,int en)

{

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

ne->vertex=en;

ne->next=adj[s];;

adj[s]=ne;

}

void main()

{

int s,i,en;

struct node \*ptr;

clrscr();

printf("Enter no of vertices:");

scanf("%d",&v);

adj= (struct node \*\*)malloc((v+1)\*sizeof(struct node \*));

adj1= (struct node \*\*)malloc((v+1)\*sizeof(struct node \*));

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

adj[i]=adj1[i]=NULL;

printf("Enter no of edges:");

scanf("%d",&e);

printf("Enter the edges\n");

printf("Start edge:\n");

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

{

scanf("%d%d",&s,&en);

adjlistRep(adj,s,en);

adjlistRep(adj1,en,s);

}

dfs();

dfs1();

getch();

}

void dfs()

{

int i;

visited=(int\*)malloc((v+1)\*sizeof(int));

ft=(int\*)malloc((v+1)\*sizeof(int));

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

visited[i]=0;

printf("\ndfs\n");

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

if (visited[i]==0)

dfsvisit(i);

}

void dfsvisit(int u)

{

int w;

struct node \*ptr;

visited[u]=1;

printf(" %d ",u);

ptr=adj[u];

while(ptr!=NULL)

{

w=ptr->vertex;

if(visited[w]==0)

dfsvisit(w);

ptr=ptr->next;

}

t++;

ft[u]=t;

}

void dfsvisit1(int u)

{

int w;

struct node \*ptr;

visited[u]=1;

printf(" %d ",u);

ptr=adj1[u];

while(ptr!=NULL)

{

w=ptr->vertex;

if(visited[w]==0)

dfsvisit1(w);

ptr=ptr->next;

}

}

int visitedAll()

{

int i,flag=1;

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

if (visited[i]==0)

{

flag=0;

break;

}

return flag;

}

void dfs1()

{

int i,max=0,ver;

printf("\n components:\n");

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

visited[i]=0;

while(!visitedAll())

{

max=0;

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

{

if (visited[i]==0 && ft[i]>max)

{

ver=i;

max=ft[i];

}

}

printf("{");

dfsvisit1(ver);

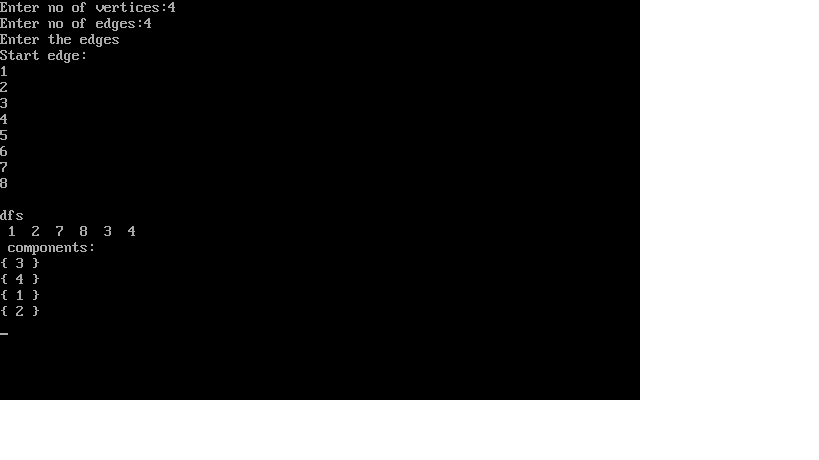
printf("}\n");

}

}

|  |
| --- |
|  |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**OUTPUT**

****

10)Write a program to implement Prim’s algorithm.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

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

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{

clrscr();

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

scanf("%d",&n);

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

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;

}

visited[1]=1;

printf("\n");

while(ne < n)

{

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

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

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

if(visited[i]!=0)

{

min=cost[i][j];

a=u=v;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

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

}

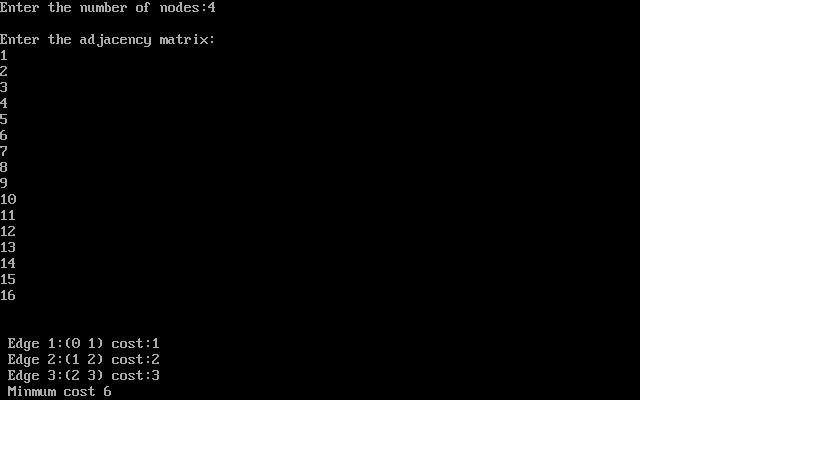
printf("\n Minmum cost %d",mincost);

getch();

}

|  |
| --- |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**OUTPUT**



11)Write a program to implement Kruskal’s algorithm.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define MAX 30

typedef struct edge

{

int u,v,w;

}edge;

typedef struct edge\_list

{

edge data[MAX];

int n;

}edge\_list;

edge\_list elist;

int Graph[MAX][MAX],n;

edge\_list spanlist;

void kruskalAlgo();

int find(int belongs[],int vertexno);

void applyUnion(int belongs[],int c1,int c2);

void sort();

void print();

void kruskalAlgo()

{

int belongs[MAX],i,j,cno1,cno2;

elist.n=0;

printf("elements of graph are\n");

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

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

{

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

{

elist.data[elist.n].u=i;

elist.data[elist.n].v=j;

elist.data[elist.n].w=Graph[i][j];

elist.n++;

}

}

sort();

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

belongs[i]=i;

spanlist.n=0;

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

{

cno1=find(belongs,elist.data[i].u);

cno2=find(belongs,elist.data[i].v);

if(cno1!=cno2)

{

spanlist.data[spanlist.n]=elist.data[i];

spanlist.n=spanlist.n+1;

applyUnion(belongs,cno1,cno2);

}

}

}

int find(int belongs[],int vertexno)

{

return(belongs[vertexno]);

}

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

{

int i;

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

if(belongs[i]==c2)

belongs[i]=c1;

}

void sort()

{

int i,j;

edge temp;

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

for(j=0;j<elist.n-1;j++)

if(elist.data[j].w > elist.data[j+1].w)

{

temp=elist.data[j];

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

elist.data[j+1]=temp;

}

}

void print()

{

int i,cost=0;

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

{

printf("\n%d %d %d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);

cost=cost+spanlist.data[i].w;

}

printf("\nspanning tree cost %d",cost);

}

void main()

{

int i,j,total\_cost;

n=6;

Graph[0][0]=0;

Graph[0][1]=4;

Graph[0][2]=4;

Graph[0][3]=0;

Graph[0][4]=0;

Graph[0][5]=0;

Graph[0][6]=0;

Graph[1][0]=4;

Graph[1][1]=0;

Graph[1][2]=2;

Graph[1][3]=0;

Graph[1][4]=0;

Graph[1][5]=0;

Graph[1][6]=0;

Graph[2][0]=4;

Graph[2][1]=2;

Graph[2][2]=0;

Graph[2][3]=3;

Graph[2][4]=4;

Graph[2][5]=0;

Graph[2][6]=0;

Graph[3][0]=0;

Graph[3][1]=0;

Graph[3][2]=3;

Graph[3][3]=0;

Graph[3][4]=3;

Graph[3][5]=0;

Graph[3][6]=0;

Graph[4][0]=0;

Graph[4][1]=0;

Graph[4][2]=4;

Graph[4][3]=3;

Graph[4][4]=0;

Graph[4][5]=0;

Graph[4][6]=0;

Graph[5][0]=0;

Graph[5][1]=0;

Graph[5][2]=2;

Graph[5][3]=0;

Graph[5][4]=3;

Graph[5][5]=0;

Graph[5][6]=0;

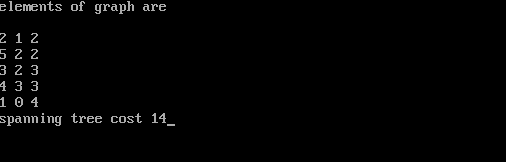
kruskalAlgo();

print();

getch();

}

**OUTPUT**

****

12)Write a program to implement Binary search tree.

**PROGRAM**

#include<stdio.h>

#include<conio.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 preorder(struct btnode \*t);

void postorder(struct btnode \*t);

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

int smallest(struct btnode \*t);

int largest(struct btnode \*t);

int flag=1;

void main()

{

int ch;

printf("\nOPERATION");

printf("\n1.insert\n");

printf("\n2.delete\n");

printf("\n3.inorder traversal\n");

printf("\n4.preorder traversal\n");

printf("\n5.postorder traversal\n");

printf("\n6.exit\n");

while(1)

{

printf("\n enter your choice");

scanf("%d",&ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

inorder(root);

break;

case 4:

preorder(root);

break;

case 5:

postorder(root);

break;

case 6:

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,l;

printf("enter data to be inserted");

scanf("%d",&data);

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

temp->value=data;

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

}

void search(struct btnode \*t)

{

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

search(t->r);

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

t->r=temp;

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

search(t->l);

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);

t1=root;

t2=root;

search1(root,data);

}

void preorder(struct btnode \*t)

{

if(root==NULL)

{

printf("no elements in tree to display");

return;

}

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

if(t->l!=NULL)

preorder(t->l);

if(t->r!=NULL)

preorder(t->r);

}

void postorder(struct btnode \*t)

{

if(root==NULL)

{

printf("no elements in tree to display");

return;

}

if(t->l!=NULL)

postorder(t->l);

if(t->r!=NULL)

postorder(t->r);

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

}

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))

{

t2=root;

if(t->r!=NULL)

{

k=smallest(t->r);

flag=1;

}

else

{

k=largest(t->l);

flag=2;

}

search1(root,k);

t->value=k;

}

}

int smallest(struct btnode \*t)

{

t2=t;

if(t->l!=NULL)

{

t2=t;

return(smallest(t->l));

}

else

return(t->value);

}

int largest(struct btnode \*t)

{

if(t->r!=NULL)

{

t2=t;

return(largest(t->r));

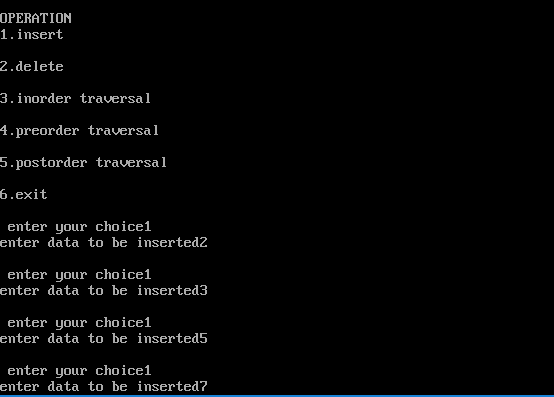
}

else

return(t->value);

}

**OUTPUT**

****