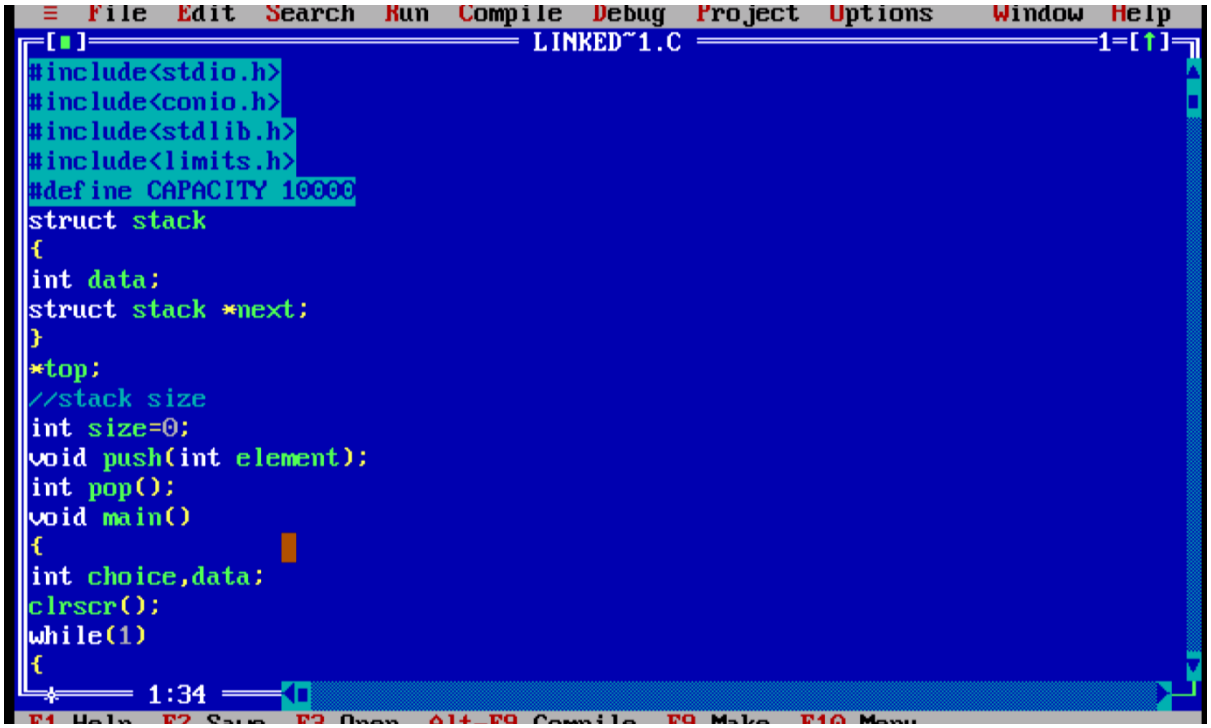
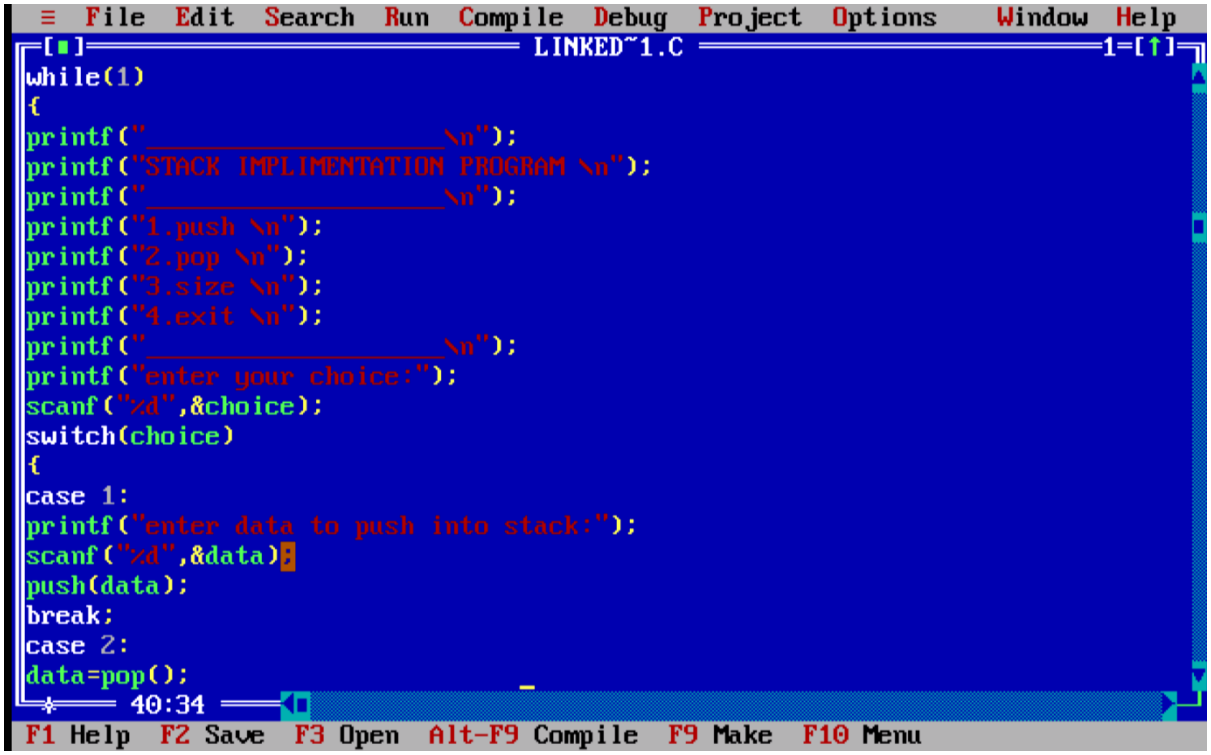
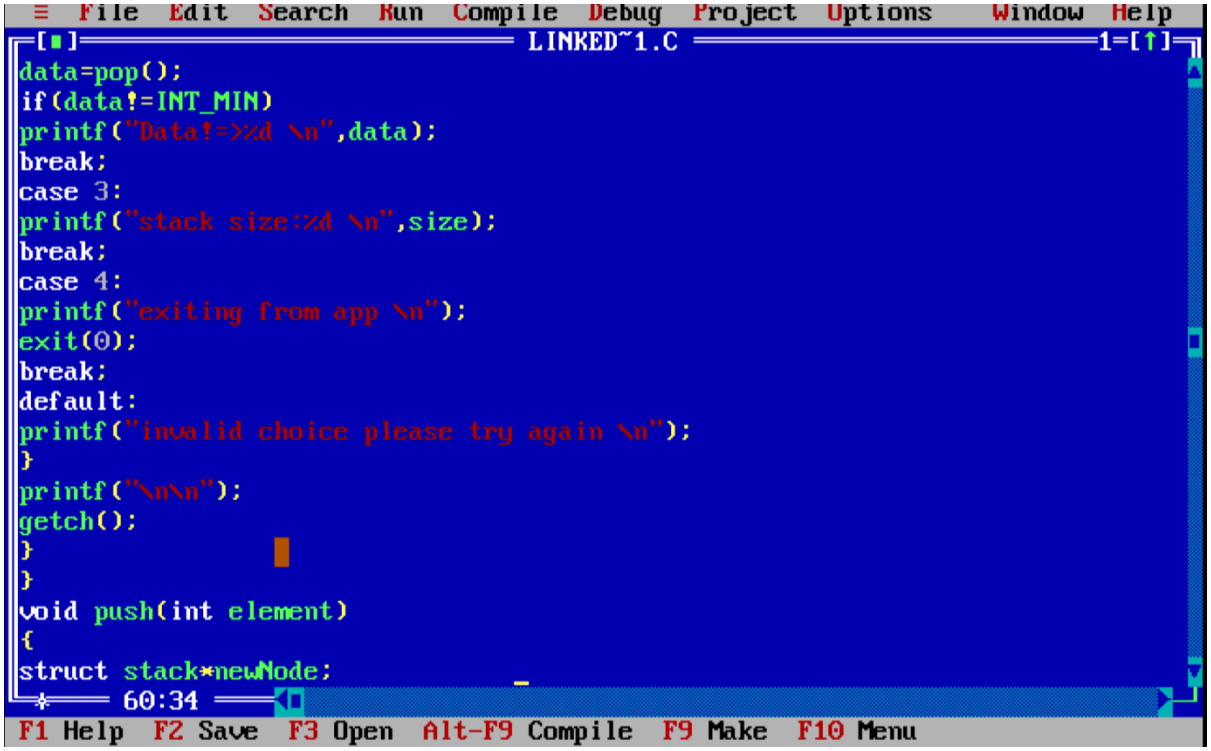
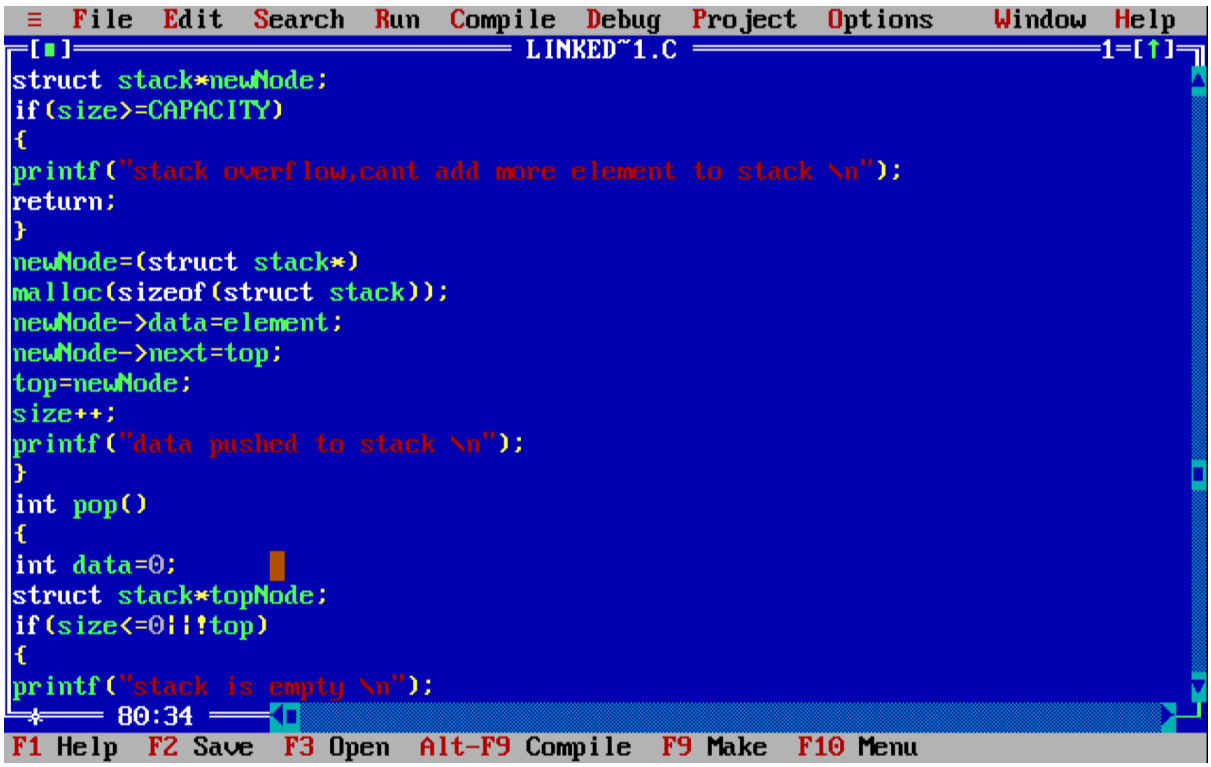
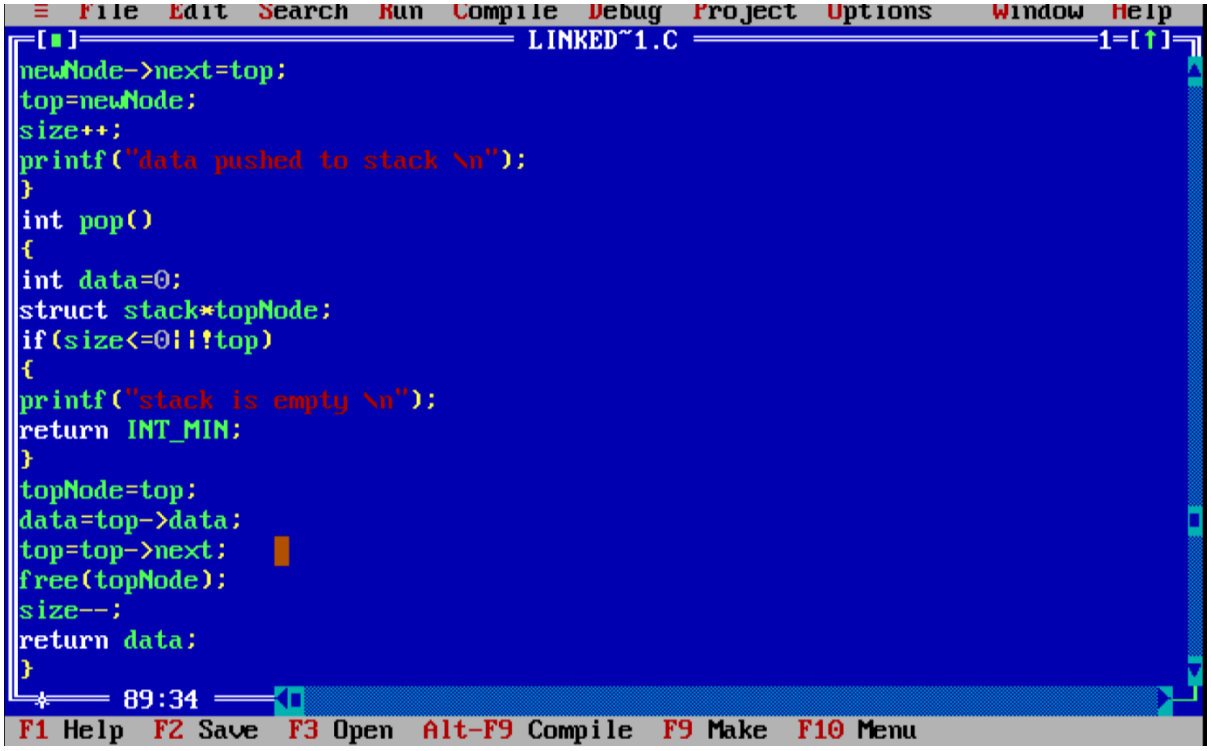
1.IMPLEMENT LINKED STACK





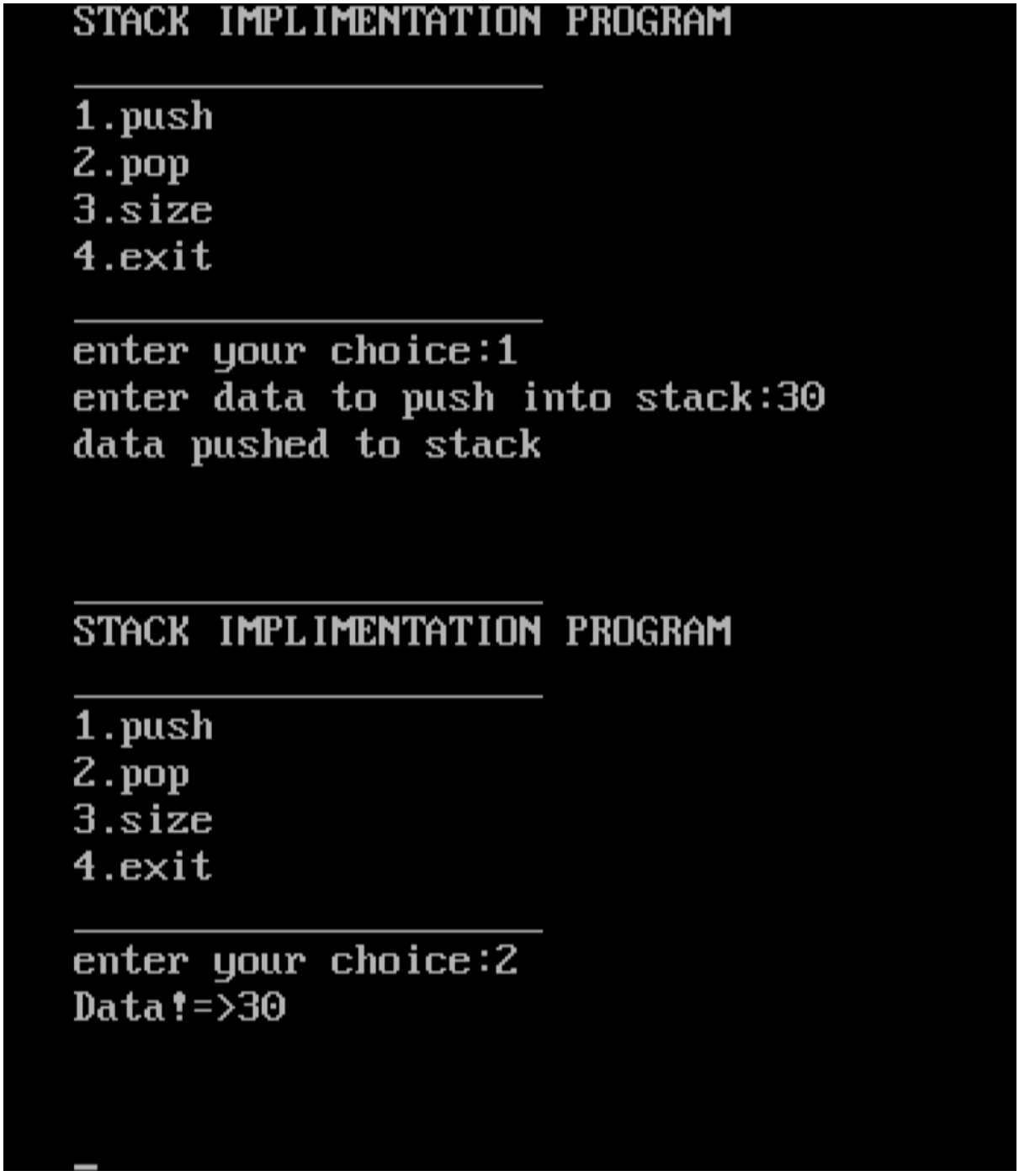

File Edit Search Run Compile Debug Project Options Window Help
LINKED~1.C
data=pop();
if(data!=INT_MIN)
printf("Data!=>%d \n",data);
break;
case 3:
printf("stack size:%d \n",size);
break;
case 4:
printf("exiting from app \n");
exit(0);
break;
default:
printf("invalid choice please try again \n");
}
printf("\n\n");
getch();
}
}
void push(int element)
{
struct stack*newNode;
60:34
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
LINKED~1.C
struct stack*newNode;
if(size>=CAPACITY)
{
printf("stack overflow,cant add more element to stack \n");
return;
}
newNode=(struct stack*)
malloc(sizeof(struct stack));
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");
80:34
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
LINKED~1.C
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;
}
89:34
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
_____________________________
STACK IMPLIMENTATION PROGRAM
_____________________________
1.push
2.pop
3.size
4.exit
_____________________________
enter your choice:1
enter data to push into stack:10
data pushed to stack



_____________________________
STACK IMPLIMENTATION PROGRAM
_____________________________
1.push
2.pop
3.size
4.exit
_____________________________
enter your choice:1
```

```
_____________________
STACK IMPLIMENTATION PROGRAM
_____________________
1.push
2.pop
3.size
4.exit
_____________________
enter your choice:1
enter data to push into stack:20
data pushed to stack



_____________________
STACK IMPLIMENTATION PROGRAM
_____________________
1.push
2.pop
3.size
4.exit
_____________________
enter your choice:1
enter data to push into stack:30
```

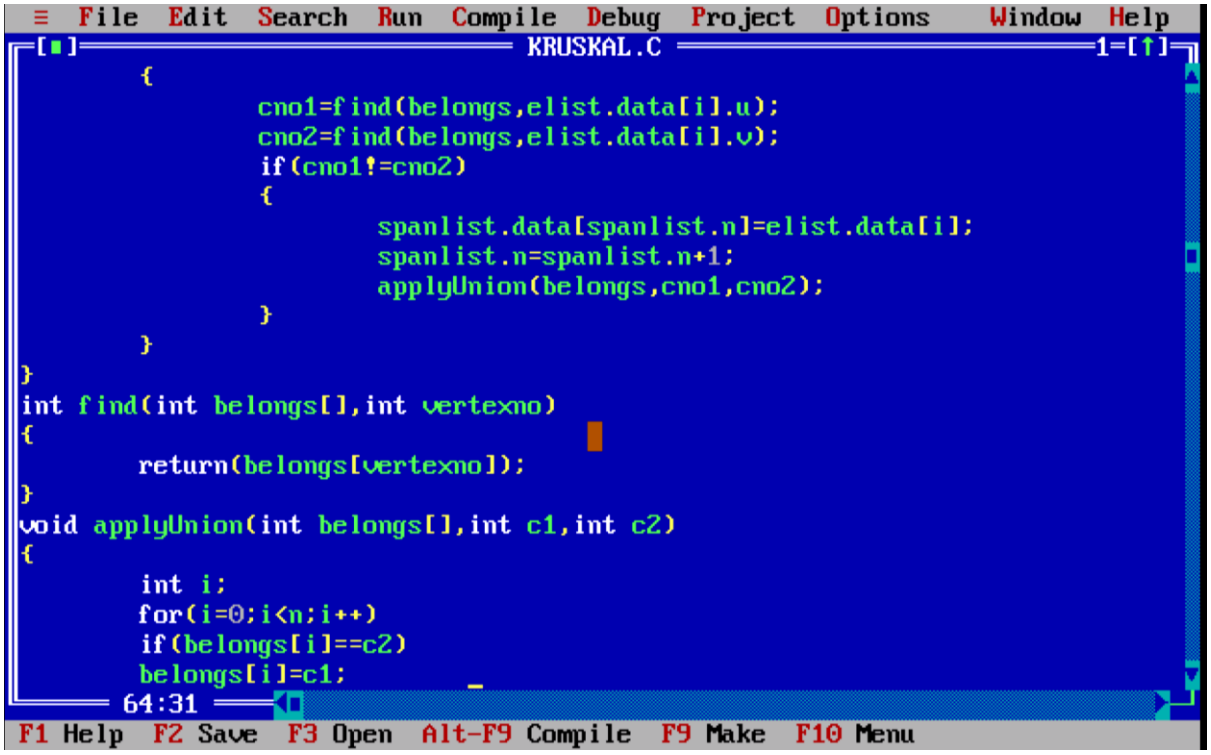

STACK IMPLIMENTATION PROGRAM
1.push
2.pop
3.size
4.exit
enter your choice:1
enter data to push into stack:30
data pushed to stack
STACK IMPLIMENTATION PROGRAM
1.push
2.pop
3.size
4.exit
enter your choice:2
Data!=>30

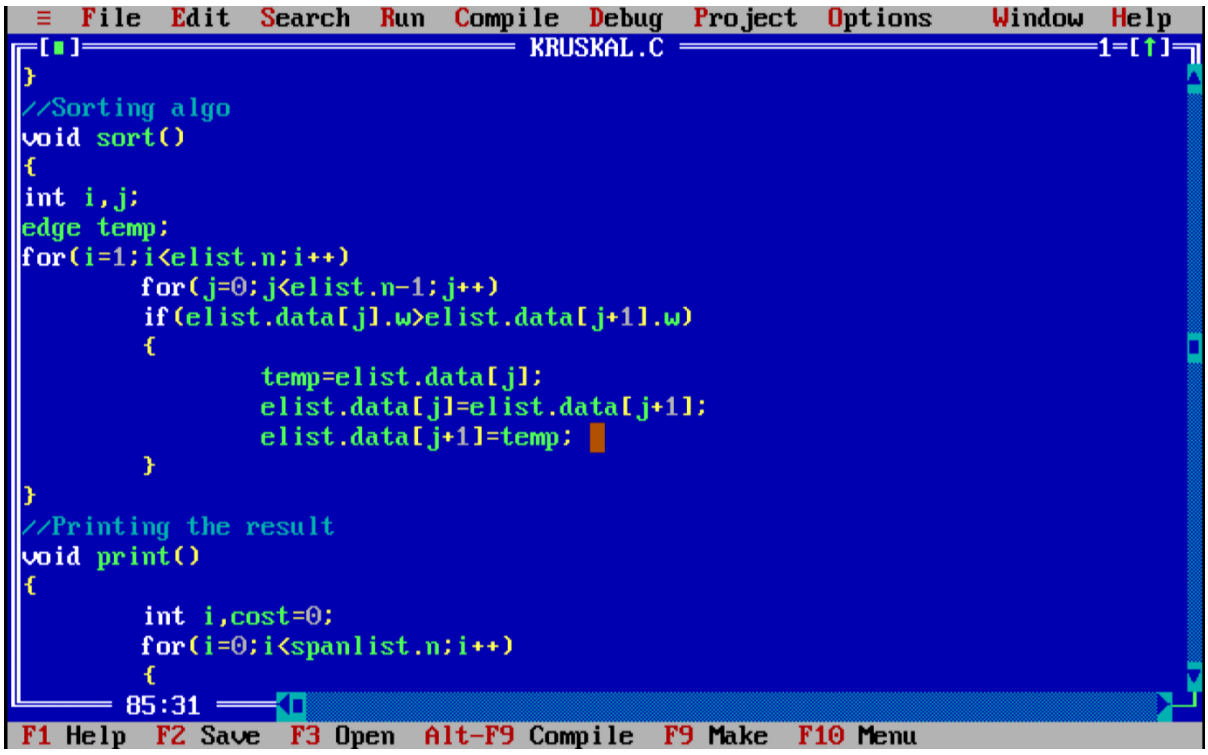
```
enter your choice:2
Data!=>30



_____________________________
STACK IMPLIMENTATION PROGRAM
_____________________________
1.push
2.pop
3.size
4.exit
_____________________________
enter your choice:3
stack size:2



_____________________________
STACK IMPLIMENTATION PROGRAM
_____________________________
1.push
2.pop
3.size
4.exit
_____________________________
enter your choice:4_
```

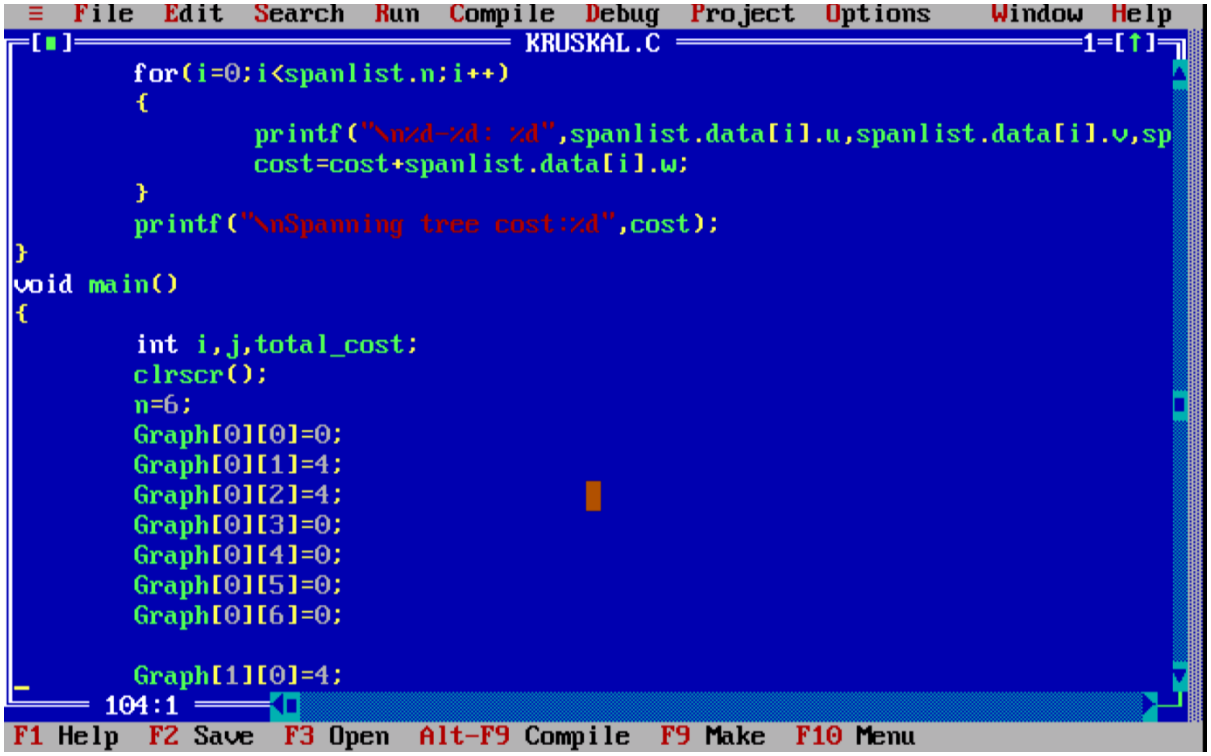
2.IMPLEMENT KRUSKAL ALGORITHM

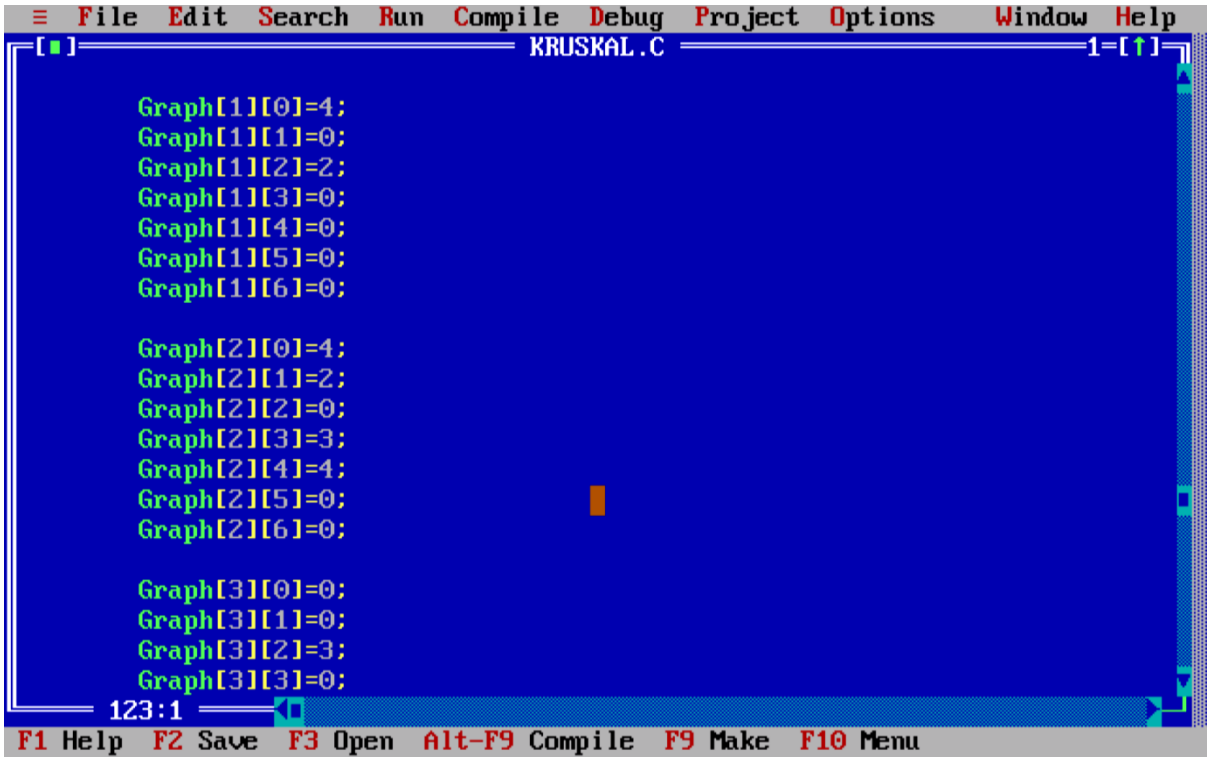
```
File  Edit  Search  Run  Compile  Debug  Project  Options  Window  Help
                              KRUSKAL.C                              1=[↑]
#include<stdio.h>
#include<conio.h>
#define MAX 30
typedef struct edge
{
        int u,v,w;
}edge;
typedef struct edge_list
{
        edge data[MAX];
        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();
                              19:31
F1 Help  F2 Save  F3 Open  Alt-F9 Compile  F9 Make  F10 Menu
```

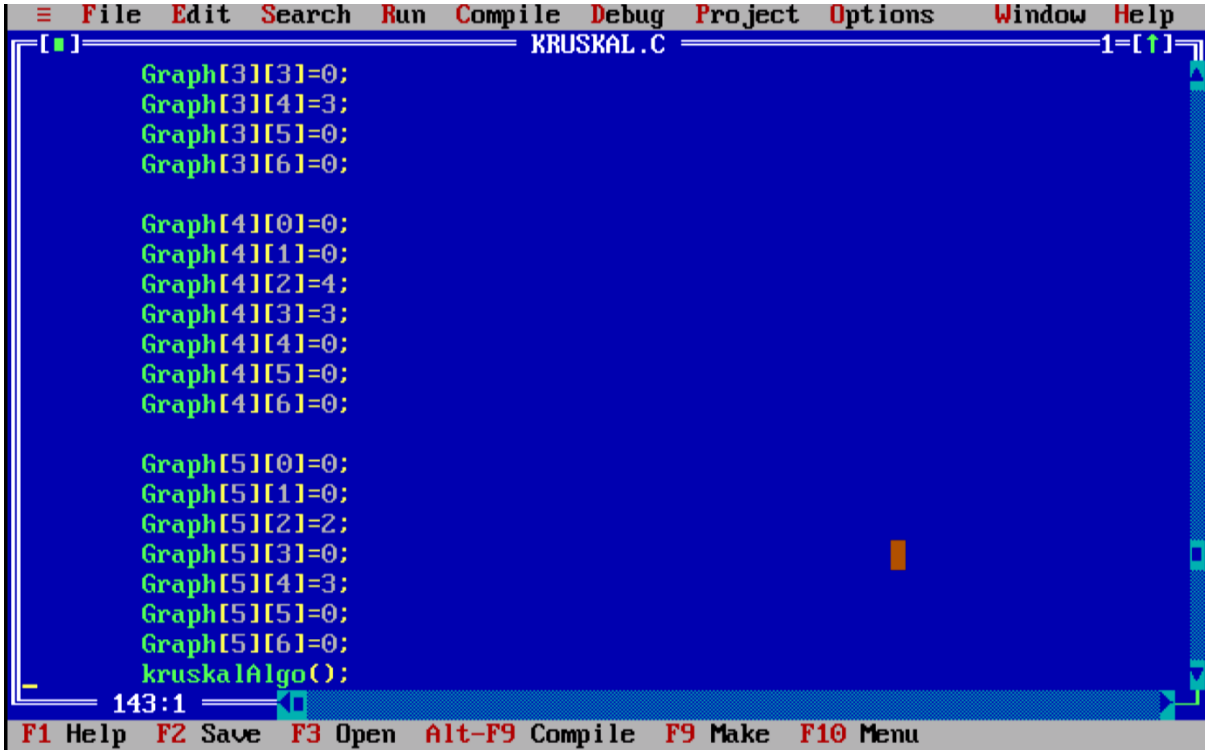
```
                              KRUSKAL.C                              1=[↑]
void kruskalAlgo()
{
        int belongs[MAX],i,j,cno1,cno2;
        elist.n=0;
        printf("Elements of the graph are\n");
        for(i=1;i<n;i++)
        for(j=0;j<i;j++)
        {
                if(Graph[i][j]!=0)
                {
                        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++)
                              43:31
F1 Help  F2 Save  F3 Open  Alt-F9 Compile  F9 Make  F10 Menu
```

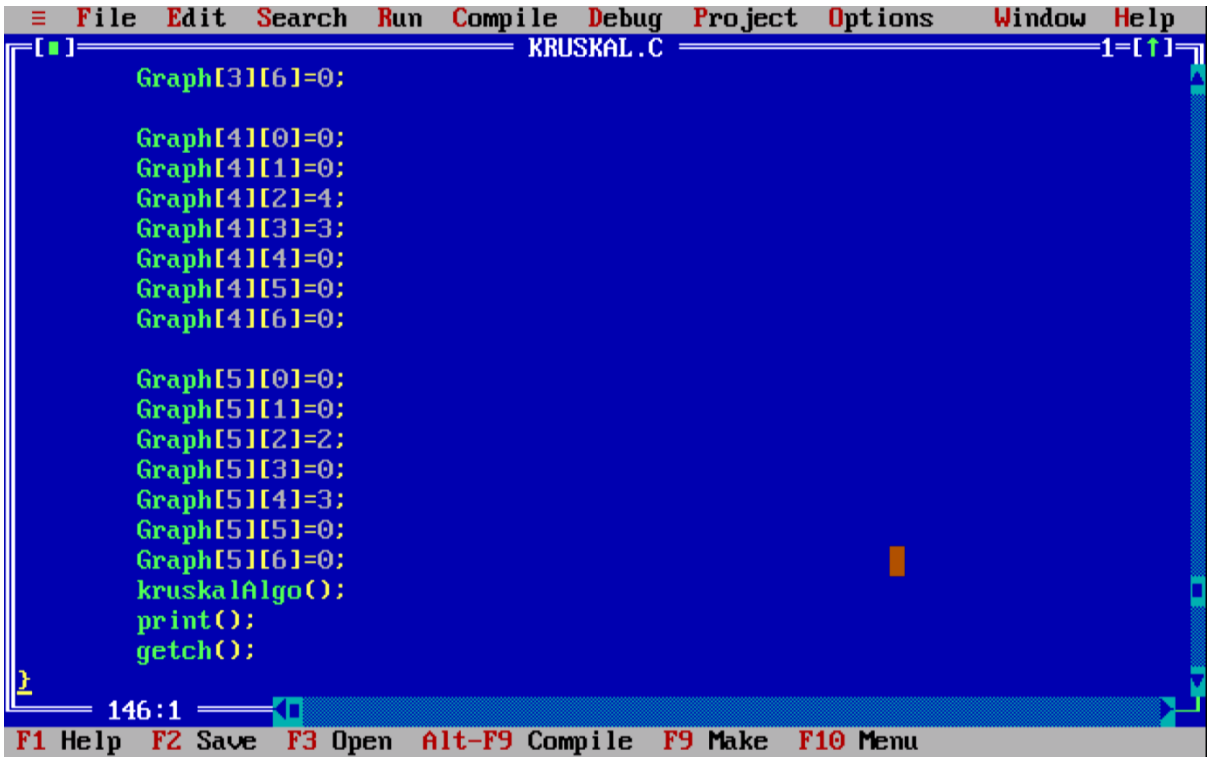

File Edit Search Run Compile Debug Project Options Window Help
KRUSKAL.C
{
cno1=find(belongs,elist.data[i].u);
cno2=find(belongs,elist.data[i].v);
if(cno1!=cno2)
{
spanlist.data[spanlist.n]=elist.data[i];
spanlist.n=spanlist.n+1;
applyUnion(belongs,cno1,cno2);
}
}
}
int find(int belongs[],int vertexno)
{
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;
64:31
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

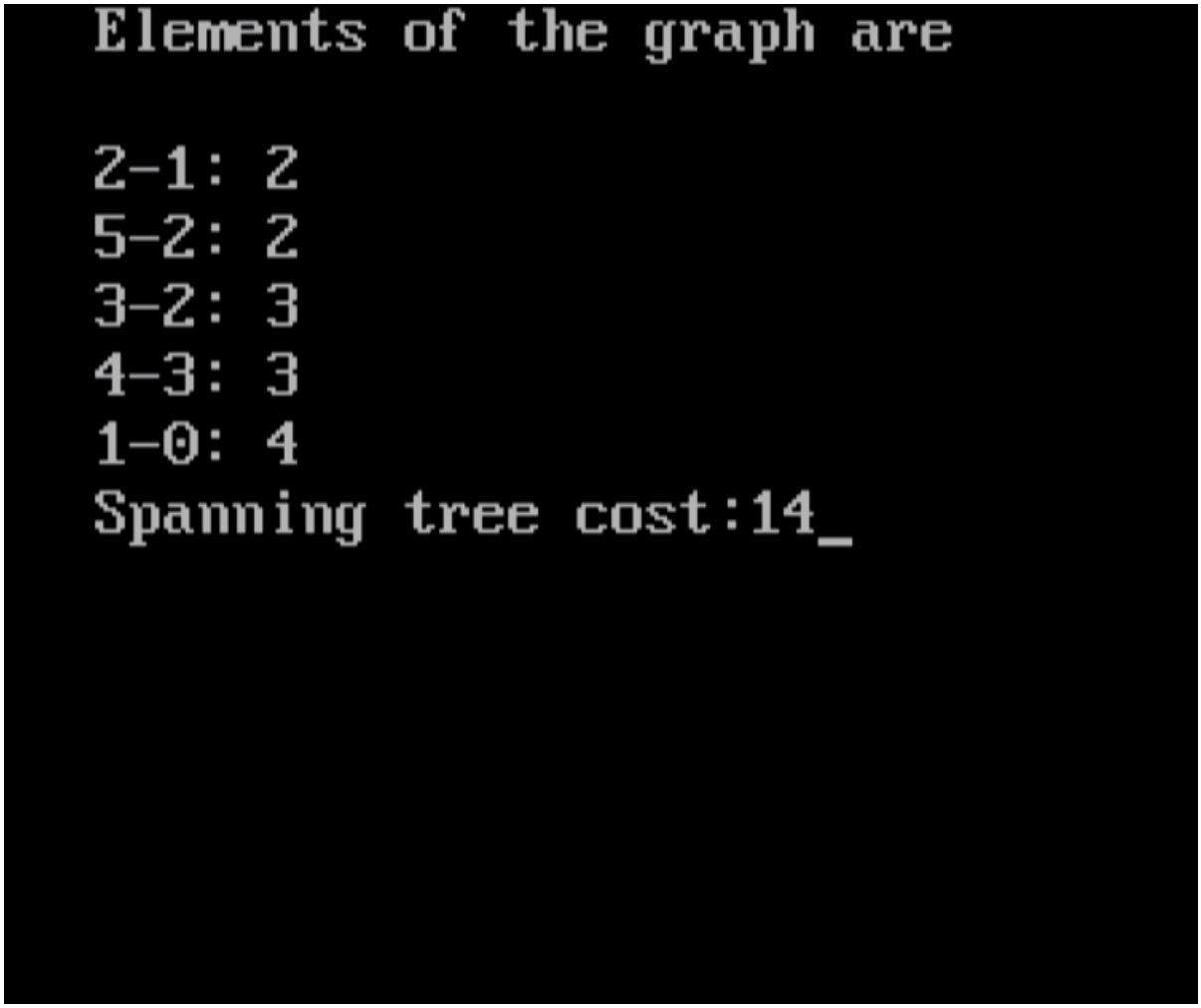

File Edit Search Run Compile Debug Project Options Window Help
KRUSKAL.C
}
//Sorting algo
void sort()
{
int i,j;
edge temp;
for(i=1;i<elist.n;i++)
for(j=0;j<elist.n-1;j++)
if(elist.data[j].w>elist.data[j+1].w)
{
temp=elist.data[j];
elist.data[j]=elist.data[j+1];
elist.data[j+1]=temp;
}
}
//Printing the result
void print()
{
int i,cost=0;
for(i=0;i<spanlist.n;i++)
{
85:31
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
KRUSKAL.C
for(i=0;i<spanlist.n;i++)
{
printf("\n%d-%d: %d",spanlist.data[i].u,spanlist.data[i].v,sp
cost=cost+spanlist.data[i].w;
}
printf("\nSpanning tree cost:%d",cost);
}
void main()
{
int i,j,total_cost;
clrscr();
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;
104:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
KRUSKAL.C
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;
123:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
KRUSKAL.C
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();
143:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
KRUSKAL.C
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();
}
146:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


Elements of the graph are
2-1: 2
5-2: 2
3-2: 3
4-3: 3
1-0: 4
Spanning tree cost:14_