LAB MANUAL

for

DATA STRUCTURE USING C

Paper Code: (CC -3)

2nd Semester

B.Sc. in Computer Science (Hons.)

Prepared by

RANU CHOWDHURY
Department of Comp. Sc.

SIR GURUDAS MAHAVIDYALAYA

Lab manual for Data Structure using C

Array :

Definition :

Arrays are defined as the collection of similar type of data items stored at contiguous

Memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. Array is the simplest data structure where each data element can be randomly accessed by using its index number.

For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variable for the marks in different subject. Instead of that, we can define an array which can store the marks in each subject at a the contiguous memory locations.

The array marks[10] defines the marks of the student in 10 different subjects where each

subject marks are located at a particular subscript in the array i.e. marks[0] denotes the

marks in first subject, marks[1] denotes the marks in 2nd subject and so on.

for example, in C language, the syntax of declaring an array is like following:

1. int arr[10]; char arr[10]; float arr[5]

Need of using Array

In computer programming, the most of the cases requires to store the large number of

data of similar type. To store such amount of data, we need to define a large number of

variables. It would be very difficult to remember names of all the variables while writing

the programs. Instead of naming all the variables with a different name, it is better to

define an array and store all the elements into it.

Lab manual for Data Structure using C

1.

Aim: To read n numbers and display it.

```
#include<stdio.h>
#include<conio.h>

void main()
{
int i,n, a[10];
clrscr();
printf("\nEnter the number of element : \n");
scanf("%d",&n);
printf("Enter element: \n");
for(i=0;i<n;i++)
{
printf("a[%d]=",i);
scanf("%d",&a[i]);
}
printf("\n Display array element: \n");
for(i=0;i<n;i++)
{
printf("a[%d]=%d\n",i,a[i]);

}
getch();
}
```

Output:

2.

Aim: To demonstrate the concept of one dimensional array finding the sum of array elements.

```
#include<stdio.h>
#include<conio.h>

void main()

{
int i,n, a[10],s;
clrscr();
printf("Enter the number of element :\n");
scanf("%d",&n);
s=0;
printf("Enter element:\n");
for(i=0;i<n;i++)
{
printf("a[%d]=",i);
scanf("%d",&a[i]);
s=s+a[i];
}
printf("Sum of arrary element:%d",s);
getch();
}
```

Output:

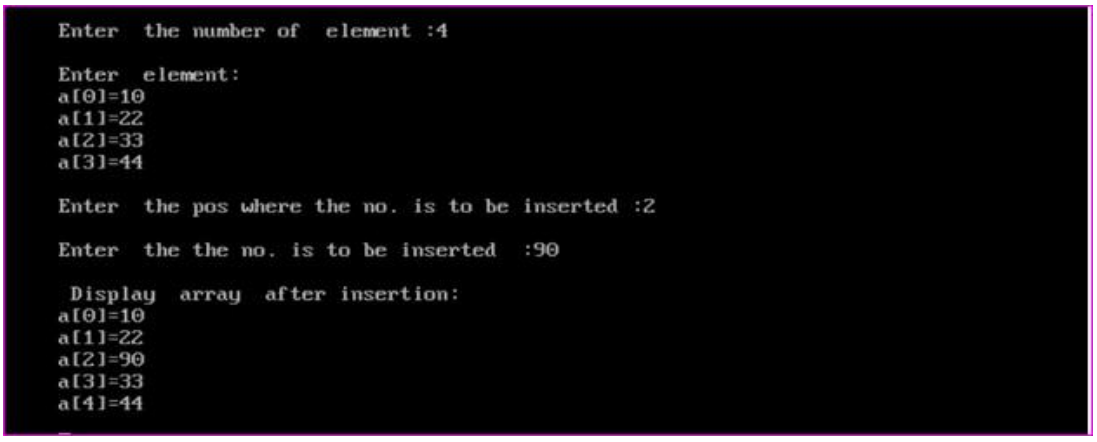
```
Enter the number of element :
5
Enter element:
a[0]=1
a[1]=2
a[2]=3
a[3]=4
a[4]=5

Sum of arrary element:15
```

3.

Aim: To insert an element in an array.

```
#include<stdio.h>
#include<
{
int i,n,pos,num, a[10];
clrscr();
printf("Enter the number of element :\n");
scanf("%d",&n);
printf("Enter element:\n");
for(i=0;i<n;i++)
{
printf("a[%d]=",i);
scanf("%d",&a[i]);
}
printf("\nEnter the pos where the no. is to be inserted :");
scanf("%d",&pos);
printf("\nEnter the the no. is to be inserted :");
scanf("%d",&num);
for(i=n-1;i>=pos;i--)
a[i+1]=a[i];
n=n+1;
a[pos]=num;
printf("\n Display array after insertion:\n");
for(i=0;i<n;i++)
{
printf("a[%d]=%d\n",i,a[i]);
}
getch();
}
```

Output:



4.

To delete an element from an array.

```
#include<stdio.h>
#include<conio.h>

void main()
{
int i,n,pos, a[10];
clrscr();
printf("Enter the number of elements :\n");
scanf("%d",&n);
printf("Enter element: \n ");
for(i=0;i<n;i++)
{
printf("a[%d]=",i);
scanf("%d",&a[i]);
}
printf("\nEnter the pos from which the no. has to be deleted :");
scanf("%d",&pos);
for(i=pos;i<n;i++)
a[i]=a[i+1];
n=n-1;
printf("\n Displar array after deletion: \n ");
for(i=0;i<n;i++)
{
printf("\n a[%d]=%d",i,a[i]);
}
getch();
}
```

Output:

```
Enter  the number of  elements :7
Enter  element:a[0]=12
a[1]=23
a[2]=43
a[3]=25
a[4]=78
a[5]=45
a[6]=14

Enter  the pos from which  the no. has  to be deleted :4

 Displar array after deletion:
 a[0]=12
 a[1]=23
 a[2]=43
 a[3]=25
 a[4]=45
 a[5]=14
```

Passing array to the function :

As we have mentioned earlier that, the name of the array represents the starting address or the address of the first element of the array. All the elements of the array can be traversed by using the base address.

The following example illustrate how the array can be passed to a function.

Example:

```
1. #include <stdio.h>
2. int summation(int[]);
3. void main ()
4. {
5. int arr[5] = {0,1,2,3,4};
6. int sum = summation(arr);
7. printf("%d",sum);
```

```
8. }
10. int summation (int arr[])
11. {
12. int sum=0,i;
13. for (i = 0; i<5; i++)
14. {
15. sum = sum + arr[i];
16. }
17. return sum;
18. }
```

The above program defines a function named as summation which accepts an array as an argument. The function calculates the sum of all the elements of the array and returns it.

2D Array

2D array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns.

However, 2D arrays are created to implement a relational database look alike data structure. It provides ease of holding bulk of data at once which can be passed to any number of functions wherever required.

How to declare 2D Array

The syntax of declaring two dimensional array is very much similar to that of a one dimensional array, given as follows.

```
1. int arr[max_rows][max_columns];
```

however, It produces the data structure which looks like following.

How do we access data in a 2D array

Due to the fact that the elements of 2D arrays can be random accessed. Similar to one

dimensional arrays, we can access the individual cells in a 2D array by using the indices of the cells. There are two indices attached to a particular cell, one is its row number while the other is its column number.

However, we can store the value stored in any particular cell of a 2D array to some

variable x by using the following syntax.

```
1. int x = a[i][j];
```

where i and j is the row and column number of the cell respectively.

We can assign each cell of a 2D array to 0 by using the following code:

```
1. for ( int i=0; i<n ;i++)
2. {
3. for (int j=0; j<n; j++)
4. {
5. a[i][j] = 0;
6. }
7. }
```

Initializing 2D Arrays

We know that, when we declare and initialize one dimensional array in C programming simultaneously, we don't need to specify the size of the array. However this will not work with 2D arrays. We will have to define at least the second dimension of the array.

The syntax to declare and initialize the 2D array is given as follows.

1. int arr[2][2] = {0,1,2,3};

The number of elements that can be present in a 2D array will always be equal to (number

of rows * number of columns).

Example : Storing User's data into a 2D array and printing it.

C Example :

```
1. #include <stdio.h>
2. void main ()
3. {
4. int arr[3][3],i,j;
5. for (i=0;i<3;i++)
6. {
7. for (j=0;j<3;j++)
8. {
9. printf("Enter a[%d][%d]: ",i,j);
10. scanf("%d",&arr[i][j]);
11. }
12. }
13. printf("\n printing the elements ....\n");
14. for(i=0;i<3;i++)
15. {
16. printf("\n");
17. for (j=0;j<3;j++)
```

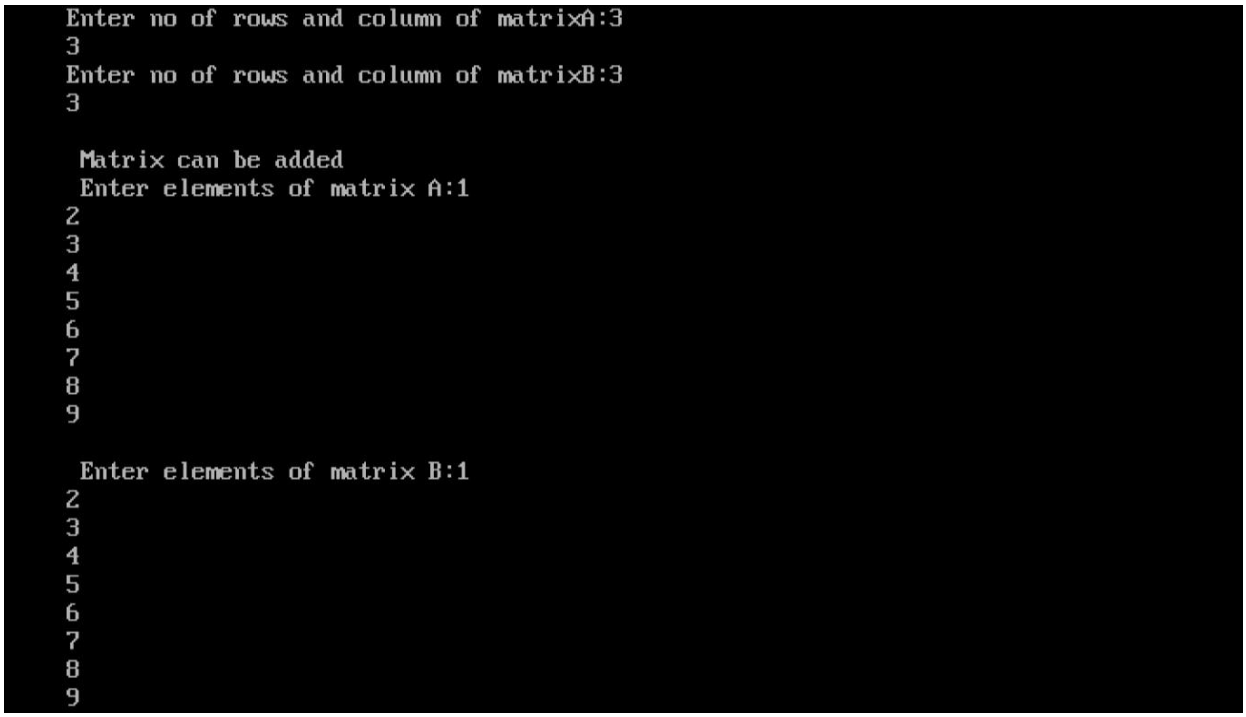
```
18. {
19. printf("%d\t",arr[i][j]);
20. }
21. }
22. }
```

Aim: To add two matrix A and B.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,m,n,p,q;
int a[10][10], b[10][10], c[10][10];
clrscr();
printf("\nEnter no of rows and column of matrixA:");
scanf("%d%d",&m,&n);
printf("\nEnter no of rows and column of matrixB:");
scanf("%d%d",&p,&q);
if(m!=p && n!=q)
{
printf("\n Matrix cannot be added.");
exit(0);
}
printf("\n Matrix can be added");
printf("\n Enter elements of matrix A:");
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter elements of matrix B:");
for(i=0;i<p;i++)
for(j=0;j<q;j++)
scanf("%d",&b[i][j]);
for(i=0;i<m;i++)
for(j=0;j<n;j++)
c[i][j]=a[i][j]+b[i][j];
printf("\n Display matrix A:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
printf("%d\t",a[i][j]);
```

```
printf("\n");
}
printf("\n Display matrix B:\n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
printf("%d\t",b[i][j]);
printf("\n");
}
printf("\n Display matrix C:\n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
printf("%d\t",c[i][j]);
printf("\n");
}
getch();
}
```

Output:



```
 Enter elements of matrix B:1
2
3
4
5
6
7
8
9

 Display matrix A:
1       2       3
4       5       6
7       8       9

 Display matrix B:
1       2       3
4       5       6
7       8       9

 Display matrix C:
2       4       6
8       10      12
14      16      18
```

Aim: To multiply two matrix A and B.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,m,n,p,q,k;
int a[10][10], b[10][10], c[10][10];
clrscr();
printf("\nEnter no of rows and column of matrixA:");
scanf("%d%d",&m,&n);
printf("\nEnter no of rows and column of matrixB:");
scanf("%d%d",&p,&q);
printf("\n Enter elements of matrix A:\n");
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter elements of matrix B:\n");
for(i=0;i<p;i++)
```

```
for(j=0;j<q;j++)
scanf("%d",&b[i][j]);
if(n==p)
{
for(i=0;i<m;i++)
for(j=0;j<q;j++)
{
c[i][j]=0;
for(k=0;k<n;k++)
c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
}
}
else
{
printf("\n Matrix  cannot be multiplied");
exit(1);
}
printf("\n Display matrix A:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
printf("%d\t",a[i][j]);
printf("\n");
}
printf("\n Display matrix B:\n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
printf("%d\t",b[i][j]);
printf("\n");
}
printf("\n Display Product:\n");
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
printf("%d\t",c[i][j]);
printf("\n");
}
getch();
}
```

Output:

```
Enter no of rows and column of matrixA:2
2

Enter no of rows and column of matrixB:2
2

 Enter elements of matrix A:
1
2
3
4

 Enter elements of matrix B:
1
2
3
4_
```

```
 Display matrix A:
1       2
3       4

 Display matrix B:
1       2
3       4

 Display Product:
7       10
15      22
```

Aim: To Concatenate two string.

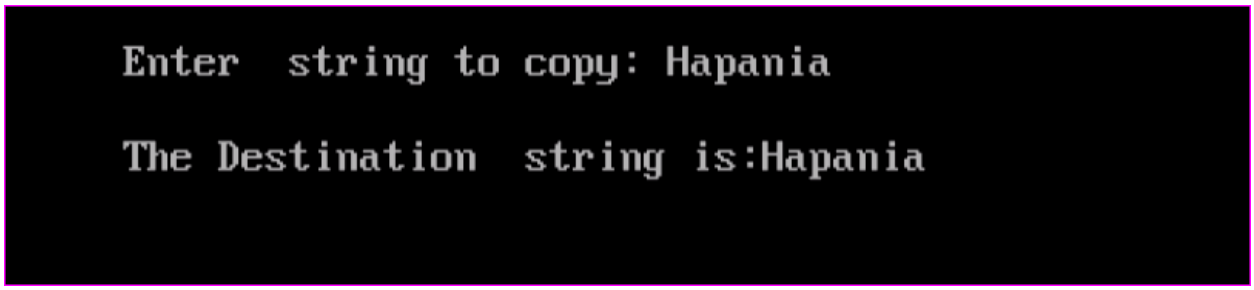
```
#include<stdio.h>
 #include<conio.h>
 #include<string.h>

void  main()
 {
char str[20],str1[20],str2[20];
int i,j;
clrscr();
 i=j=0;
printf("\n Enter 1st string:");
```

```
scanf("%s",&str);
printf("\n Enter 2nd string:");
scanf("%s",&str1);
while(str[i]!='\0')
 {
str2[i]=str[i];
i++;
 }

while(str1[j]!='\0')
 {
str2[i]=str1[j];
i++;
j++;
 }
str2[i]='\0';
printf("\n Resultant string is:%s",str2);
getch();
 }
```

Aim: To copy a string into another string.

```
#include<stdio.h>
 #include<conio.h>
 #include<string.h>

void  main()
 {
char str[20],str1[20];
int i ;
clrscr();
 i=0;
printf("\n Enter  string to copy:");
scanf("%s",&str);
while(str[i]!='\0')
 {
str1[i]=str[i];
i++;
 }
str1[i]='\0';
printf("\n The Destination  string is:%s",str1);
getch();
}
```

Output:



Aim: Implementation of linked list using array.

```
#include<stdio.h>
#include<conio.h>
#define TRUE 1
#define SIZE 10
struct link
{
int info;
int next;
};
struct link node[SIZE];
int Getnode();
void Createlist();
void Freenode(int);
void Display();
void Insert(int,int);
void Delete(int);
int p, avail=0;
void main()
{
int ch=1,i,n,x;
clrscr();
/*Creation of available list*/
for(i=0;i<SIZE-1;i++)
node[i].next=i+1;
node[SIZE-1].next=-1;
printf("\n Create a List:");
Createlist();
while(ch!=4)
{
printf("\n1-DISPLAY");
printf("\n2-INSERT");
printf("\n3-DELETE");
printf("\n4-QUIT");
```

```
printf("\n Enter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1 :
Display();
break;
case 2:
printf("\n Node insertion:after which node:");
scanf("%d",&n);
p=n;
printf("\n Enter the item for insertion:");
scanf("%d",&x);
Insert(p,x);
break;
case 3:
printf("\n Enter the node after which the node will be deleted:");
scanf("%d",&n);
p=n;
Delete(p);
break;
case 4:
break;
default:
printf("\n Wrong choice!Try again:");
}
}
}
int Getnode()
{
if (avail==-1)
{
printf("\n Overflow:");
exit(0);
}
p=avail;
avail=node[avail].next;
return p;
}
void Freenode(int q)
{
node[q].next=avail;
avail=q;
return;
}
void Createlist()
{
int x;
char c;
```

```
p=Getnode();
printf("\n Enter an item to be inserted:");
scanf("%d", &x);
node[p].info=x  ;
node[p].next=-1;
while(TRUE)
{
printf("\n Enter the choice(y/n):");
fflush(stdin);
c=getchar();
if(c=='y'||c=='Y')
{
printf("\n Enter an item to be inserted:");
scanf("%d",&x);
Insert(p,x);
node[p].next= -1;
}
else
return;
}
}
void Display()
{
p=0;
while(node[p].next!=-1)
{
printf("\n%d\t%d\t%d:",p,node[p].info,node[p].next);
p=node[p].next;
}
printf("\n%d\t%d\t%d:",p,node[p].info,node[p].next);
}
void Insert(int r,int x)
{
int q;
if(r==-1)
{
printf("\n void insertion:");
return;
}
q=Getnode();
node[q].info=x;
node[q].next=node[r].next;
node[r].next=q;
return;
}
void Delete(int r)
{
int q;
if(r==-1||node[r].next==-1)
```

```
{
printf("\n void deletion:");
return;
}
q=node[r].next;
node[r].next=node[q].next;
Freenode(q);
return;

}
```

Output:

```
Enter an item to be inserted:4

Enter the choice(y/n):y

Enter an item to be inserted:23

Enter the choice(y/n):y

Enter an item to be inserted:87

Enter the choice(y/n):y

Enter an item to be inserted:22

Enter the choice(y/n):y

Enter an item to be inserted:12

Enter the choice(y/n):n

1-DISPLAY
2-INSERT
3-DELETE
4-QUIT
 Enter your choice:_
```

```
 Enter your choice:2

 Node insertion:after which node:3

 Enter the item for insertion:99

1-DISPLAY
2-INSERT
3-DELETE
4-QUIT
 Enter your choice:1

0       4       1:
1       23      2:
2       87      3:
3       22      5:
5       99      4:
4       12      -1:
1-DISPLAY
2-INSERT
3-DELETE
4-QUIT
 Enter your choice:3

 Enter the node after which the node will be deleted:_
```

```
4       12      -1:
1-DISPLAY
2-INSERT
3-DELETE
4-QUIT
 Enter your choice:3

 Enter the node after which the node will be deleted:1

1-DISPLAY
2-INSERT
3-DELETE
4-QUIT
 Enter your choice:1

0       4       1:
1       23      3:
3       22      5:
5       99      4:
4       12      -1:
1-DISPLAY
2-INSERT
3-DELETE
4-QUIT
 Enter your choice:4
```

Aim: Implementation of stack using array.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAXSTK 100
int top=-1;
int items[MAXSTK];
int Isempty();
int Isfull();
void Push(int);
int Pop();
void Display();
void main()
{
int x;
char ch='1';
clrscr();
while(ch!='4')
{
printf("\n 1-PUSH");
```

```
printf("\n 2-POP");
printf("\n 3-DISPLAY");
printf("\n 4-QUIT");
printf("\n Enter your choice:");
fflush(stdin);
ch=getchar();
switch(ch)
{
case '1':
printf("\n Enter the element to be pushed:");
scanf("%d",&x);
Push(x);
break;
case '2':
x=Pop();
printf("\n Pop element is %d\n:",x);
break;
case '3':
Display();
break;
case '4':
break;
default:
printf("\n Wrong choice!Try again:");
}
}
}
int Isempty()
{
if(top==-1)
return 1;
else
return 0;
}
int Isfull()
{
if(top==MAXSTK-1)
return 1;
else
return 0;
}
void Push(int x)
{
 if(Isfull())
 {
 printf("\n Stack full");
 return;
```

```
 }
 top++;
 items[top]=x;
 }
 int Pop()
 {
 int x;
 if(Isempty())
 {
 printf("\n Stack empty");
 exit(0);
 }
x=items[top];
top--;
return x;
}
void Display()
{
int i;
 if(Isempty())
 {
 printf("\n Stack empty");
 return;
 }
 printf("\n Elements in the Stack are :\n");
 for(i=top;i>=0;i--)
 printf("%d\n",items[i]);
 }
```

Output:

```
 Enter the element to be pushed:78

 1-PUSH
 2-POP
 3-DISPLAY
 4-QUIT
 Enter your choice:1

 Enter the element to be pushed:87

 1-PUSH
 2-POP
 3-DISPLAY
 4-QUIT
 Enter your choice:3

 Elements in the Stack are :
87
78

 1-PUSH
 2-POP
 3-DISPLAY
 4-QUIT
 Enter your choice:_
```

```
87
78

 1-PUSH
 2-POP
 3-DISPLAY
 4-QUIT
 Enter your choice:2

 Pop element is 87
:
 1-PUSH
 2-POP
 3-DISPLAY
 4-QUIT
 Enter your choice:3

 Elements in the Stack are :
78

 1-PUSH
 2-POP
 3-DISPLAY
 4-QUIT
 Enter your choice:4_
```

Lab manual for Data Structure using C

Aim: To Create fibonacci series using recursive function.

```
#include<stdio.h>
#include<conio.h>
int Fibonacci(int);
void main()
{
int i,n;
clrscr();
printf("\n Enter the no of elements to be displayed:");
scanf("%d",&n);
for(i=0;i<n;i++)
printf("%d\t",Fibonacci(i));
getch();
}
int Fibonacci(int n)
{
if(n<=0)
return 0;
else if (n==1)
return 1;
else
return Fibonacci(n-1)+ Fibonacci(n-2);
}
```

Output:

```
 Enter the no of elements to be displayed:13
0       1       1       2       3       5       8       13      21      34
55      89      144
```

Aim: Calculate factorial of a number using recursive function.

```
#include<stdio.h>
#include<conio.h>
```

```
int Factorial(int);
void main()
{
int i,n;
clrscr();
printf("\n Enter the no of elements:");
scanf("%d",&n);
printf("Factorial of %d is %d",n,Factorial(n));
getch();
}
int Factorial(int n)
{
if(n==0)
return 1;
else
return n*Factorial(n-1);
}
```

Output:

Aim: Implementation of queue using array.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAXQ 100
int front=0,rear=-1;
int items[MAXQ];
int Isempty();
int Isfull();
void Insert(int);
int Delete();
void Display();
void main()
{
int x;
```

```
char ch='1';
clrscr();
while(ch!='4')
{
printf("\n 1-INSERT");
printf("\n 2-DELETE");
printf("\n 3-DISPLAY");
printf("\n 4-QUIT");
printf("\n Enter your choice:");
fflush(stdin);
ch=getchar();
switch(ch)
{
case '1':
printf("\n Enter the element to be inserted:");
scanf("%d",&x);
Insert(x);
break;
case '2':
x=Delete();
printf("\n Delete element is %d\n:",x);
break;
case '3':
Display();
break;
case '4':
break;
default:
printf("\n Wrong choice!Try again:");
}
}
getch();
}
int Isempty()
{
if(rear<front)
return 1;
else
return 0;
}
int Isfull()
{
if(rear==MAXQ-1)
return 1;
else
return 0;
}
```

```
void Insert(int x)
{
 if(Isfull())
 {
 printf("\n Queue full");
 return;
 }
 rear++;
 items[rear]=x;
 }
 int Delete()
 {
 int x;
 if(Isempty())
 {
 printf("\n Queue is empty");
 exit(0);
 }
x=items[front];
front++;
return x;
}
void Display()
{
int i;
 if(Isempty())
 {
 printf("\n Queue is empty");
 return;
 }
 printf("\n Elements in the Queue are :\n");
 for(i=front;i<=rear;i++)
 printf("%d\n",items[i]);
 }
```

Output:

```
1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the element to be inserted:30

1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the element to be inserted:40

1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the element to be inserted:50_
```

```
1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:3

Elements in the Queue are :
30
40
50

1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:2

Delete element is 30
:
1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:
```

```
50

 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:2

 Delete element is 30
:
 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:3

 Elements in the Queue are :
40
50

 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:4
```

Aim: Implementation of circular queue using array.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAXQ 100
int front=-1,rear=-1;
int items[MAXQ];
int Isempty();
int Isfull();
void Insert(int);
int Delete();
void Display();
void main()
{
int x;
char ch='1';
clrscr();
while(ch!='4')
```

```
{
printf("\n 1-INSERT");
printf("\n 2-DELETE");
printf("\n 3-DISPLAY");
printf("\n 4-QUIT");
printf("\n Enter your choice:");
fflush(stdin);
ch=getchar();
switch(ch)
{
case '1':
printf("\n Enter the nos of element to be inserted:");
scanf("%d",&x);
Insert(x);
break;
case '2':
x=Delete();
printf("\n Deleted element is %d\n:",x);
break;
case '3':
Display();
break;
case '4':
break;
default:
printf("\n Wrong choice!Try again:");
}
}
getch();
}
int Isempty()
{
if(front==-1)
return 1;
else
return 0;
}
int Isfull()
{
if(front==(rear+1)%MAXQ)
return 1;
else
return 0;
}
void Insert(int x)
{
 if(Isfull())
```

```
 {
 printf("\n Queue full");
 return;
 }
 if (front==-1)
 {
 front=0;
 rear=0;
 }
 else
 rear=(rear+1)%MAXQ;
 items[rear]=x;
 }
 int Delete()
 {
 int x;
 if(Isempty())
 {
 printf("\n Queue is empty");
 exit(0);
 }
x=items[front];
 if (front==rear)
 {
 front=-1;
 rear=-1;
 }
 else
 front=(front+1)%MAXQ;
 return x;
 }
void Display()
{
int i,n;
 if(Isempty())
 {
 printf("\n Queue is empty");
 return;
 }
 printf("\n Elements in the Queue are :\n");
 if(front<=rear)
 {
 for(i=front;i<=rear;i++)
 printf("%d\n",items[i]);
 }
 else
 {
```

```
 for(i=front;i<=MAXQ-1;i++)
printf("%d\n",items[i]);
for(i=0;i<=rear;i++)
printf("%d\n",items[i]);
 }
 }
```

Output:

```
1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the nos of element to be inserted:20

1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the nos of element to be inserted:30

1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the nos of element to be inserted:40
```

```
 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:1

 Enter the nos of element to be inserted:50

 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:3

 Elements in the Queue are :
20
30
40
50

 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:_
```

```
 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:2

 Deleted element is 20
:
 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:3

 Elements in the Queue are :
30
40
50

 1-INSERT
 2-DELETE
 3-DISPLAY
 4-QUIT
 Enter your choice:
```

Lab manual for Data Structure using C

Aim: Implementation of binary search tree using array.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define TRUE 1
#define TREENODES 100
#define FALSE 0
struct tree
{
int info;
int used;
};
struct tree node[TREENODES];
void Createtree();
void Insert(int);
void Display();
void Setleft(int,int);
void Setright(int,int);
void main()
{
int x;
char ch='1';
clrscr();
printf("\n Enter root node value:");
scanf("%d", &x);
Createtree(x);
while(ch!='3')
{
printf("\n1-INSERT");
printf("\n2-DISPLAY");
printf("\n3-QUIT");
printf("\n Enter your choice:");
fflush(stdin);
ch=getchar();
switch(ch)
{
case '1' :
printf("\n Enter the element to be inserted:");
scanf("%d",&x);
Insert(x);
break;
case '2':
Display();
break;
```

```
case '3':
break;
default:
printf("\n Wrong choice!Try again:");
}
}
}
void Createtree(int x)
{
int i;
node[0].info=x;
node[0].used=TRUE;
for(i=1;i<TREENODES;i++)
node[i].used=FALSE;
}
void Insert(int x)
{
int p,q;
p=q=0;
while(q<TREENODES && node[q].used && x!=node[p].info)
{
p=q;
if(x<node[p].info)
q=2*p+1;
else
q=2*p+2;
}
if(x==node[p].info)
printf("\n %d is a duplicate number\n",x);
else
if(x<node[p].info)
Setleft(p,x);
else
Setright(p,x);
}
void Setleft(int pos,int x)
{
int q;
q=2*pos+1;
if(q>TREENODES)
printf("\n Array overflow.");
else
if(node[q].used==TRUE)
printf("\n Invalid insertion.");
else
{
node[q].info=x;
```

```
node[q].used=TRUE;
}
}
void Setright(int pos,int x)
{
int q;
q=2*pos+2;
if(q>TREENODES)
printf("\n Array overflow.");
else
if(node[q].used==TRUE)
printf("\n Invalid insertion.\n");
else
{
node[q].info=x;
node[q].used=TRUE;
}
}
void Display()
{
int i;
for(i=0;i<TREENODES;i++)
if(node[i].used==TRUE)
printf("%d ",node[i].info);
printf("\n");
}
```

Output:

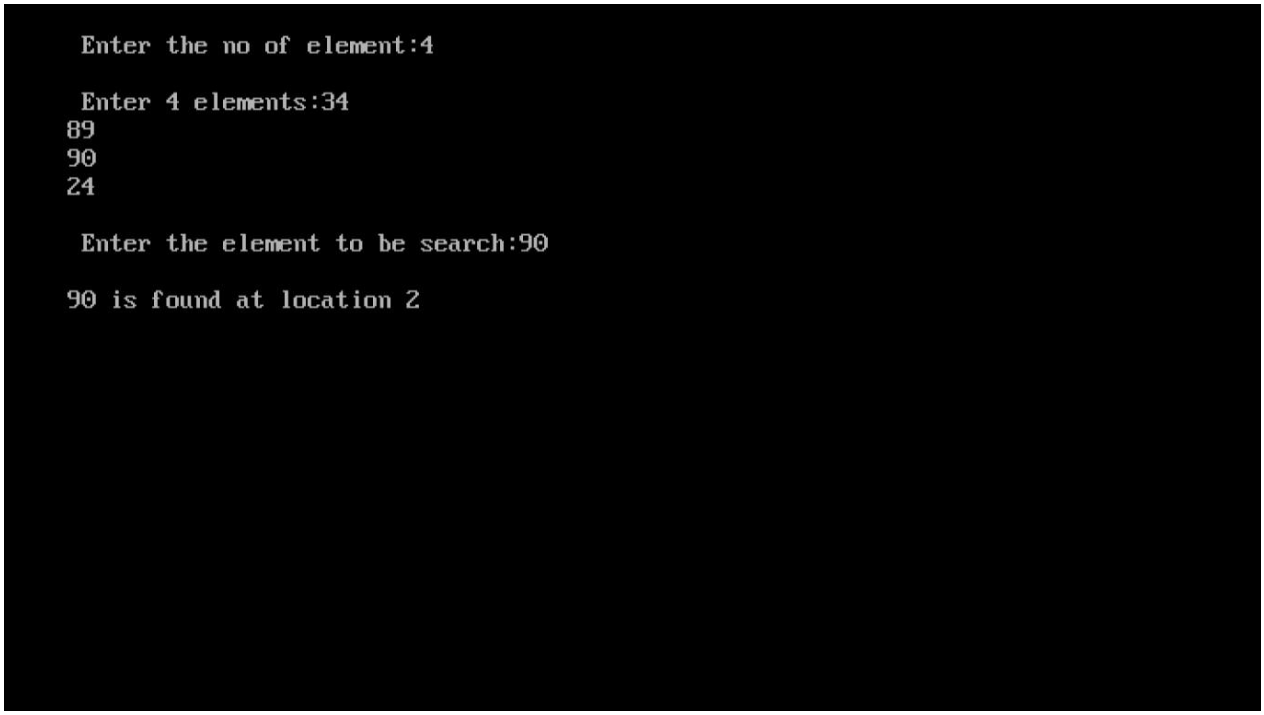
```
 Enter root  node value:60

1-INSERT
2-DISPLAY
3-QUIT
 Enter your choice:1

 Enter the element to be inserted:40

1-INSERT
2-DISPLAY
3-QUIT
 Enter your choice:1

 Enter the element to be inserted:30

1-INSERT
2-DISPLAY
3-QUIT
 Enter your choice:1

 Enter the element to be inserted:70



1-INSERT
2-DISPLAY
3-QUIT
 Enter your choice:1

 Enter the element to be inserted:70

1-INSERT
2-DISPLAY
3-QUIT
 Enter your choice:1

 Enter the element to be inserted:90

1-INSERT
2-DISPLAY
3-QUIT
 Enter your choice:2
60 40 70 30 90

1-INSERT
2-DISPLAY
3-QUIT
 Enter your choice:3_
```

Lab manual for Data Structure using C

Aim: To Search an element using sequential search.

```
#include<stdio.h>
#include<conio.h>

int Sequentialsearch(int[],int,int);
void main()
{
int x[20],i,n,p,key;
clrscr();
printf("\n Enter the no of element:");
scanf("%d",&n);
printf("\n Enter %d elements:",n);
for(i=0;i<n;i++)
scanf("%d",&x[i]);
printf("\n Enter the element to be search:");
scanf("%d",&key);
p=Sequentialsearch(x,n,key);
if(p==-1)
printf("\n The searchis unsuccessful:\n");
else
printf("\n%d is found at location %d",key,p);
getch();
}

int  Sequentialsearch(int a[],int n ,int k)
{
int i;
for(i=0;i<n;i++)
{
if(k==a[i])
return(i);
}
return(-1);
}
```

Output:

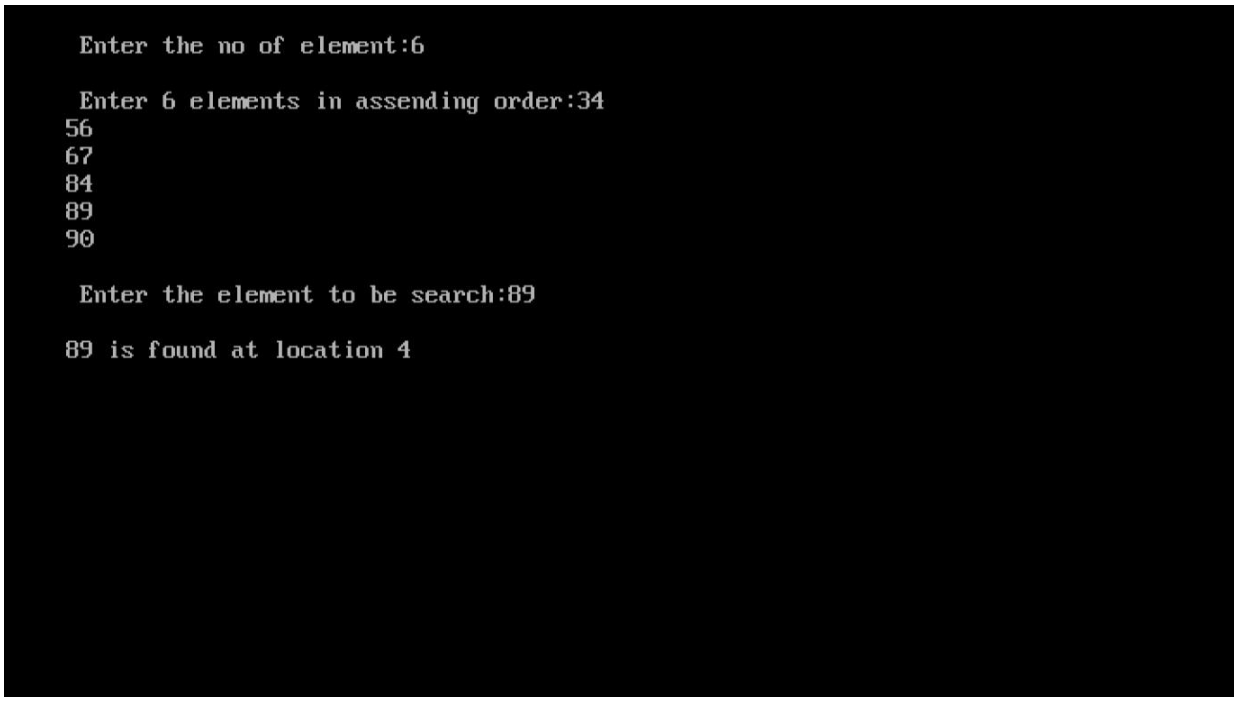


Aim: To Search an element using binary search.

```
#include<stdio.h>
#include<conio.h>

int Binarysearch(int[],int,int);
void main()
{
int x[20],i,n,p,key;
clrscr();
printf("\n Enter the no of element:");
scanf("%d",&n);
printf("\n Enter %d elements in assending order:",n);
for(i=0;i<n;i++)
scanf("%d",&x[i]);
printf("\n Enter the element to be search:");
scanf("%d",&key);
p=Binarysearch(x,n,key);
if(p==-1)
printf("\n The searchis unsuccessful:\n");
else
printf("\n%d is found at location %d",key,p);
```

```
getch();
}

int  Binarysearch(int a[],int n ,int k)
{
int lo,hi,mid;
lo=0;
hi=n-1;
while(lo<=hi)
{
mid=(lo+hi)/2;
if(k==a[mid])
return(mid);
if(k<a[mid])
hi=mid-1;
else
lo=mid+1;
}
return(-1);
}
```

Output:

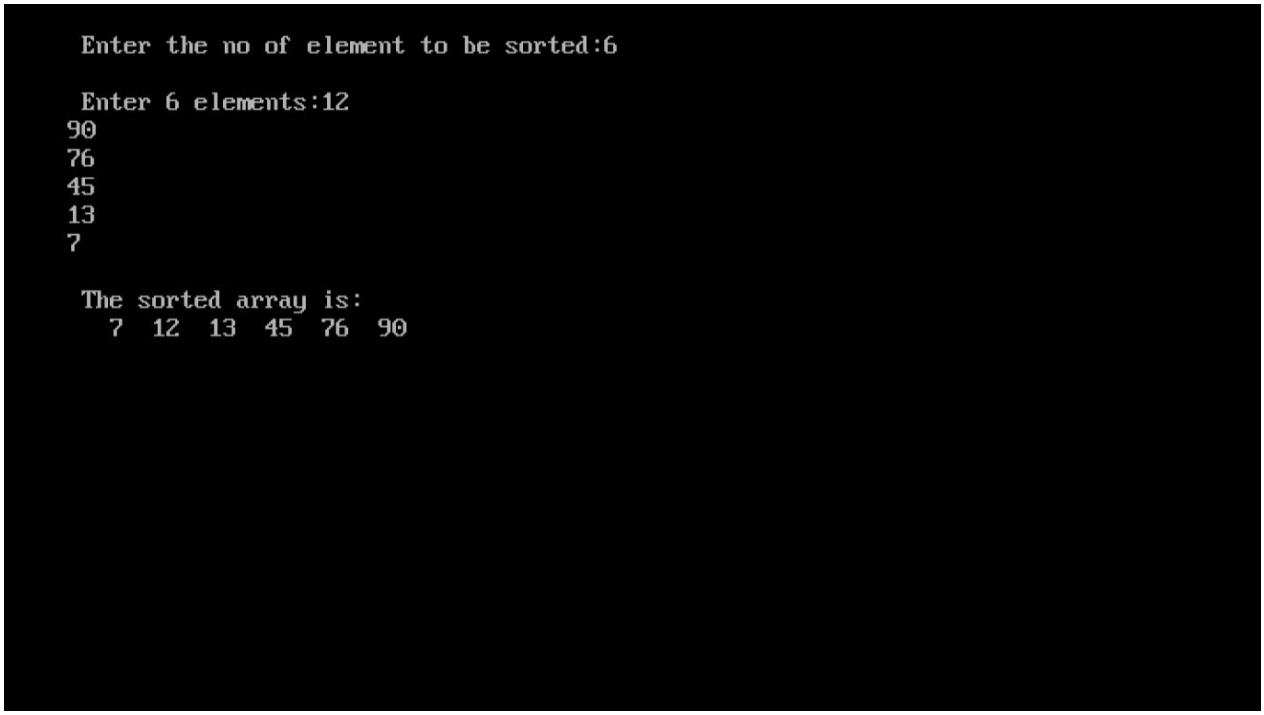


Lab manual for Data Structure using C

Aim: Arrange the list of numbers in ascending order using Bubble Sort.

```
#include<stdio.h>
#include<conio.h>

void Bubblesort(int[],int);
void main()
{
int x[20],i,n;
clrscr();
printf("\n Enter the no of element to be sorted:");
scanf("%d",&n);
printf("\n Enter %d elements:",n);
for(i=0;i<n;i++)
scanf("%d",&x[i]);
Bubblesort(x,n);
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%4d",x[i]);
getch();
}
void Bubblesort(int a[],int n)
{
int temp,pass,i;
for(pass=0;pass<n-1;pass++)
{
for(i=0;i<n-pass-1;i++)
{
if(a[i]>a[i+1])
{
temp=a[i];
a[i]=a[i+1];
a[i+1]=temp;
}
}
}
}
```

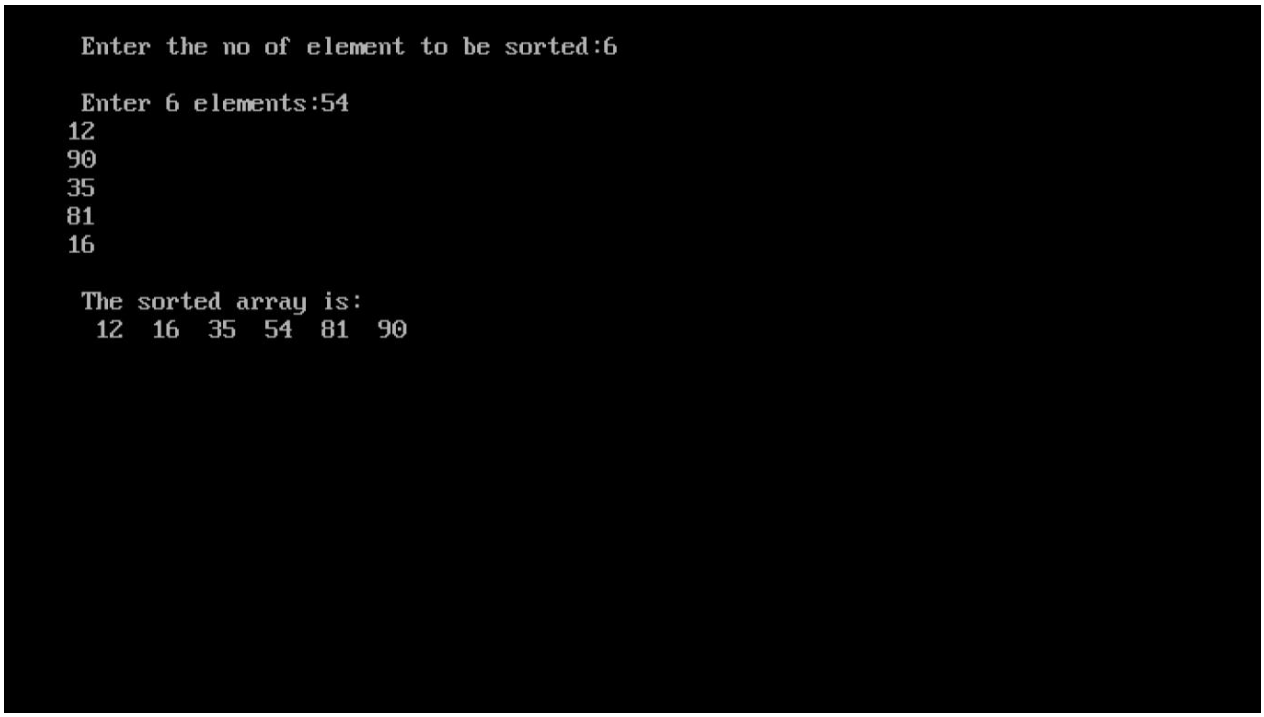
Output:



Aim: Arrange the list of numbers in ascending order using Insertion Sort.

```
#include<stdio.h>
#include<conio.h>

void Insertionsort(int[],int);
void main()
{
int x[20],i,n;
clrscr();
printf("\n Enter the no of element to be sorted:");
scanf("%d",&n);
printf("\n Enter %d elements:",n);
for(i=0;i<n;i++)
scanf("%d",&x[i]);
Insertionsort(x,n);
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%4d",x[i]);
getch();
}
void Insertionsort(int a[],int n)
```

```
{
int i,j,key;
for(j=1;j<n;j++)
{
key=a[j];
i=j-1;
while((i>-1)&&(a[i]>key))
{
a[i+1]=a[i];
i=i-1;
}
a[i+1]=key;
}
}
```

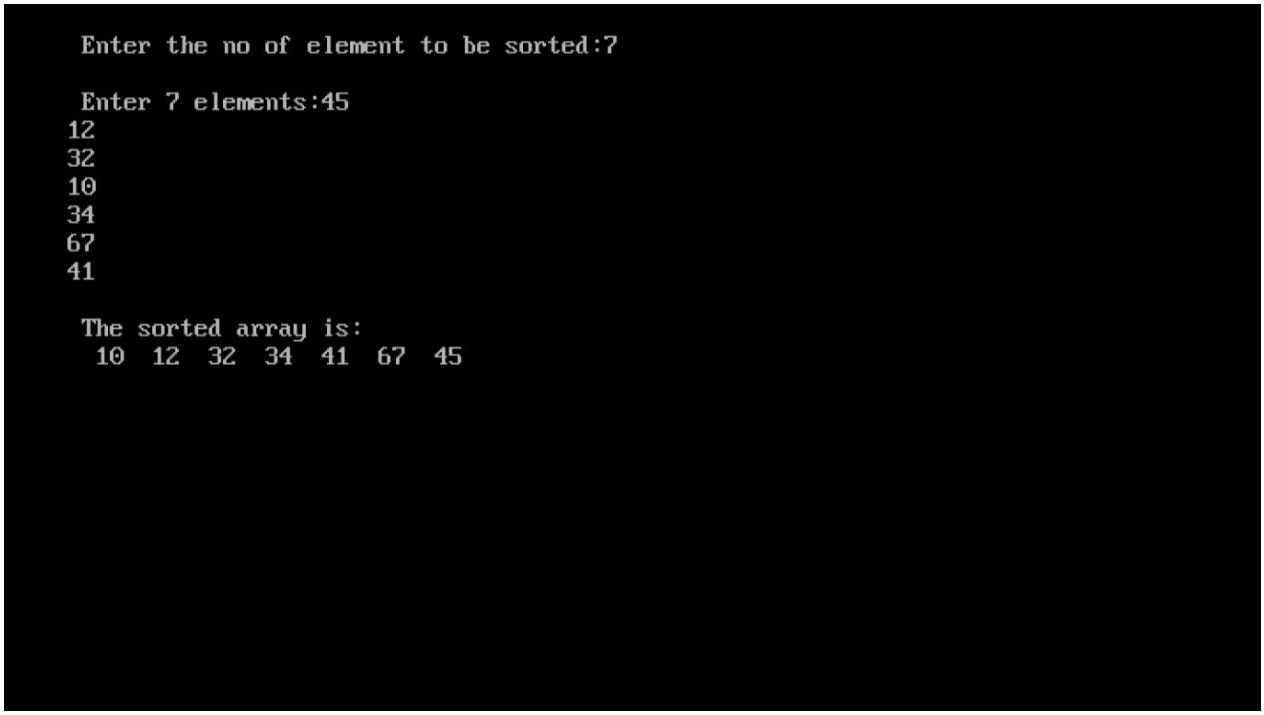


Aim: Arrange the list of numbers in ascending order using Selection Sort.

```
#include<stdio.h>
#include<conio.h>

void Selectionsort(int[],int);
void main()
```

```
{
int x[20],i,n;
clrscr();
printf("\n Enter the no of element to be sorted:");
scanf("%d",&n);
printf("\n Enter %d elements:",n);
for(i=0;i<n;i++)
scanf("%d",&x[i]);
Selectionsort(x,n);
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%4d",x[i]);
getch();
}
void Selectionsort(int a[],int n)
{
int i,j,pos,large;
for(i=n-1;i>0;i--)
{
large=a[0];
pos=0;
for(j=1;j<=i;j++)
{
if (a[i]>large)
{
large=a[j];
pos=j;
}
}
a[pos]=a[i];
a[i]=large;
}
}
```

Output:

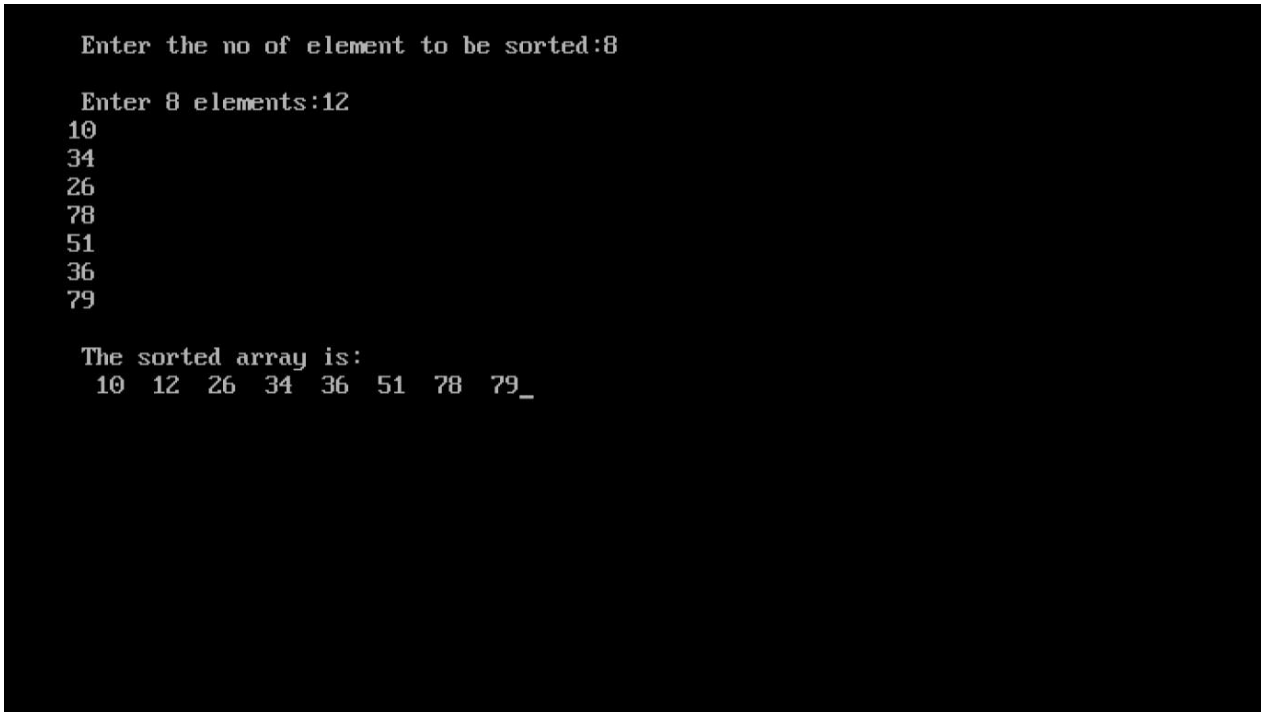


Aim: Arrange the list of numbers in ascending order using Merge Sort.

```
#include<stdio.h>
#include<conio.h>

void Mergesort(int[],int,int);
void  Merge(int[],int,int,int);
void main()
{
int x[20],i,n;
clrscr();
printf("\n Enter the no of element to be sorted:");
scanf("%d",&n);
printf("\n Enter %d elements:",n);
for(i=0;i<n;i++)
scanf("%d",&x[i]);
Mergesort(x,0,n-1);
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%4d",x[i]);
getch();
}
```

```
void Mergesort(int a[],int p,int r)
{
int q;
if(p<r)
{
q=(p+r)/2;
Mergesort(a,p,q);
Mergesort(a,q+1,r);
Merge(a,p,q,r);
}
}
void Merge(int a[], int p, int q,int r)
{
int b[20],l1,r1,i;
l1=p;
r1=q+1;
i=p;
while((l1<=q)&&(r1<=r))
{
if(a[l1]<a[r1])
{
b[i]=a[l1];
l1=l1+1;
i=i+1;
}
else
{
b[i]=a[r1];
r1=r1+1;
i=i+1;
}
}
while(l1<=q)
{
b[i]=a[l1];
l1=l1+1;
i=i+1;
}
while(r1<=r)
{
b[i]=a[r1];
r1=r1+1;
i=i+1;
}
for(i=p;i<=r;i++)
a[i]=b[i];
}
```

Output:

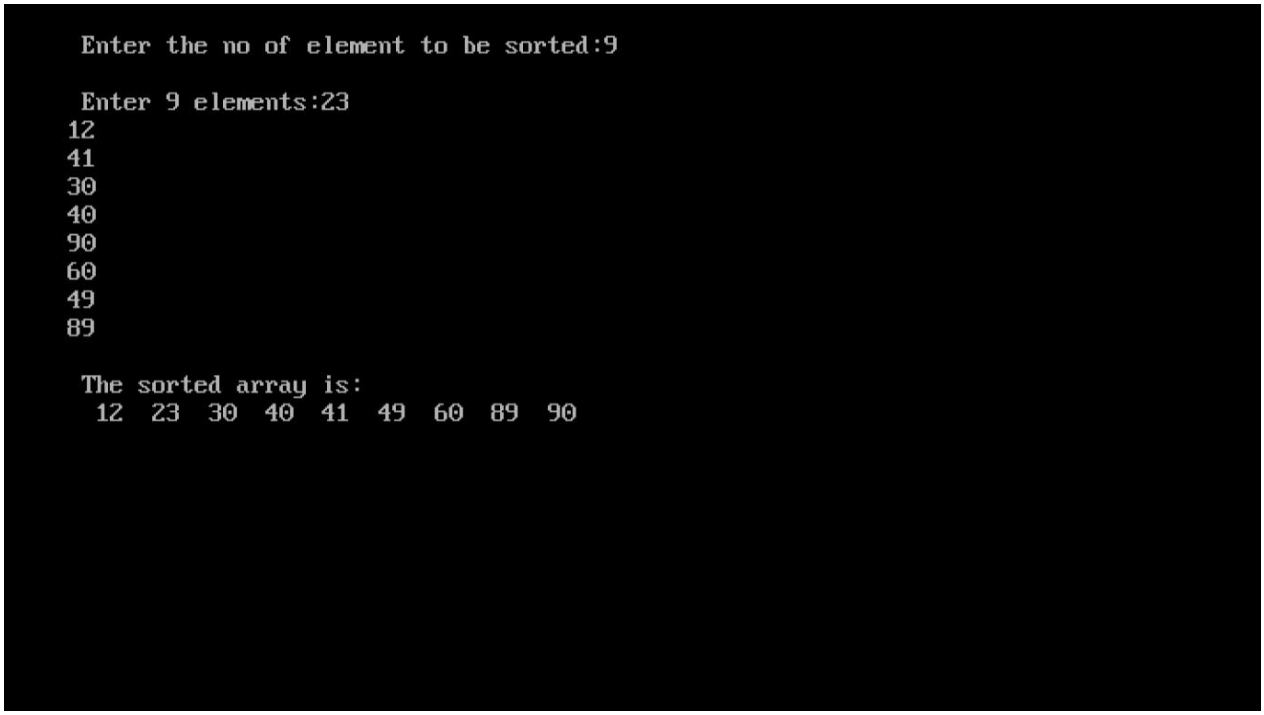


Aim: Arrange the list of numbers in ascending order using Quick Sort.

```
#include<stdio.h>
#include<conio.h>

void Quicksort(int[],int,int);
int partition(int[],int,int);
void main()
{
int x[20],i,n;
clrscr();
printf("\n Enter the no of element to be sorted:");
scanf("%d",&n);
printf("\n Enter %d elements:",n);
for(i=0;i<n;i++)
scanf("%d",&x[i]);
Quicksort(x,0,n-1);
```

```
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%4d",x[i]);
getch();
}
void Quicksort(int a[],int p,int r)
{
int q;
if(p<r)
{
q=Partition(a,p,r);
Quicksort(a,p,q);
Quicksort(a,q+1,r);
}
}
int Partition(int a[], int p,int r)
{
int k,i,j,temp;
k=a[p];
i=p-1;
j=r+1;
while(1)
{
do
{
j=j-1;
}while(a[j]>k);
do
{
i=i+1;
}while(a[i]<k);
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
else
return(j);
}
}
```

Output:

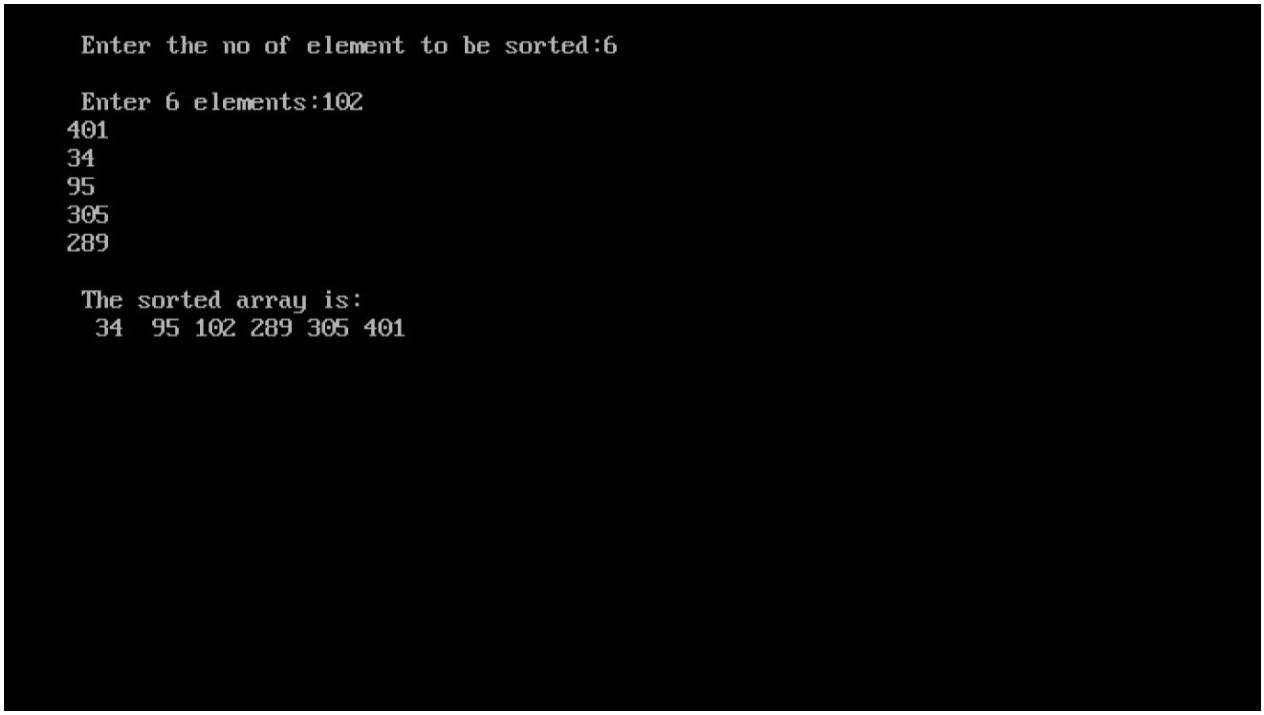


Aim: Arrange the list of numbers in ascending order using Radix Sort.

```
#include<stdio.h>
#include<conio.h>

void Radixsort(int[],int);
void main()
{
int x[20],i,n;
clrscr();
printf("\n Enter the no of element to be sorted:");
scanf("%d",&n);
printf("\n Enter %d elements:",n);
for(i=0;i<n;i++)
scanf("%d",&x[i]);
Radixsort(x,n);
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%4d",x[i]);
getch();
}
void Radixsort(int a[],int n)
```

```
{
int bucket[10][10],buck[10];
int i,j,k,l,num,div,large,pass;
div=1;
num=0;
large=a[0];
for(i=0;i<n;i++)
{
if(a[i]>large)
large=a[i];
}
while(large>0)
{
num=num+1;
large=large/10;
}
for(pass=0;pass<num;pass++)
{
for(k=0;k<10;k++)
buck[k]=0;
for(i=0;i<n;i++)
{
l=(a[i]/div)%10;
bucket[l][buck[l]++]=a[i];
}
i=0;
for(k=0;k<10;k++)
{
for(j=0;j<buck[k];j++)
a[i++]=bucket[k][j];
}
div=div*10;
}
}
```

Output:



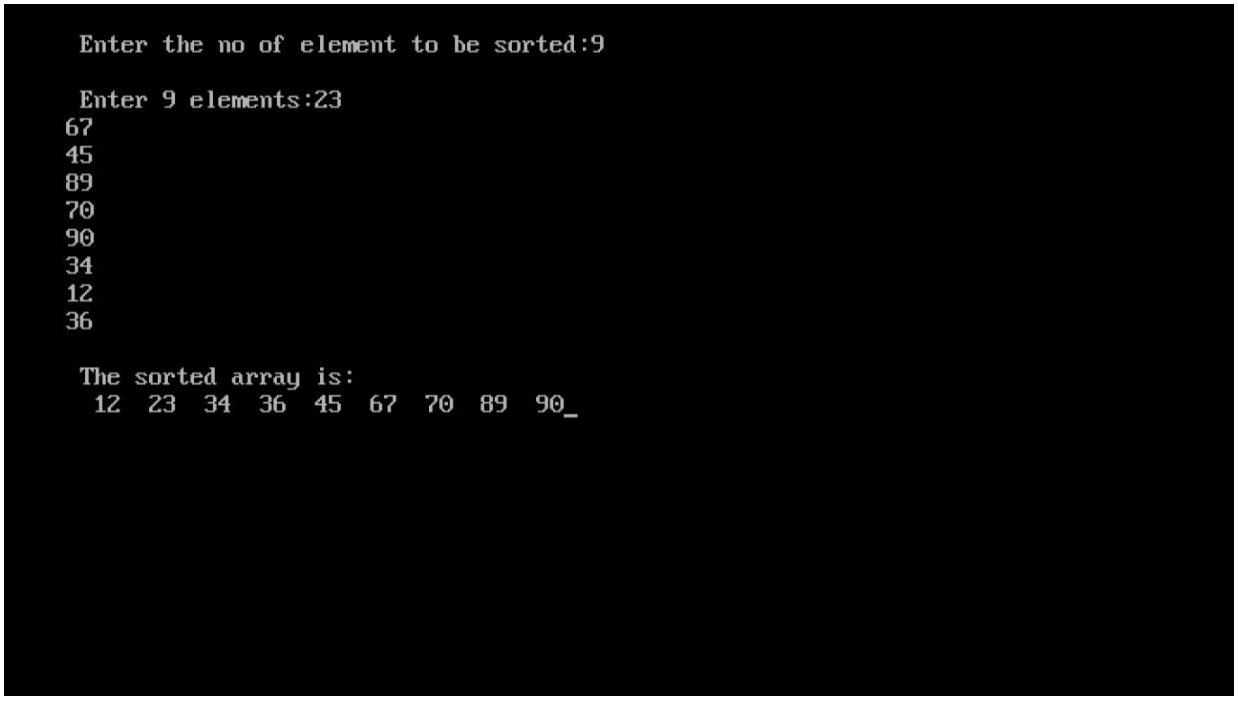
Aim: Arrange the list of numbers in ascending order using Heap Sort.

```
#include<stdio.h>
#include<conio.h>

void Heapsort(int[],int);
int Parent(int);
int Left(int);
int Right(int);
void Heapify(int[],int,int);
void Buildheap(int[],int);
void main()
{
int x[20],i,n;
clrscr();
printf("\n Enter the no of element to be sorted:");
scanf("%d",&n);
printf("\n Enter %d elements:",n);
```

```
for(i=0;i<n;i++)
scanf("%d",&x[i]);
Heapsort(x,n);
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%4d",x[i]);
getch();
}
int Parent(int i)
{
return(i/2);
}
int Left(int i)
{
return(2*i+1);
}
int Right(int i)
{
return(2*i+2);
}
void Heapify(int a[],int i,int n)
{
int l,r,large,temp ;
l=Left(i);
r=Right(i);
if((l<=n-1)&&(a[l]>a[i]))
large=l;
else
large=i;
if((r<=n-1)&&(a[r]>a[large]))
large=r;
if(large!=i)
{
temp=a[i];
a[i]=a[large];
a[large]=temp;
Heapify(a,large,n);
}
}
void Buildheap(int a[],int n)
{
int i;
for(i=(n-1)/2;i>=0;i--)
Heapify(a,i,n);
}
void Heapsort(int a[],int n)
{
```

```
int i,m,temp;
Buildheap(a,n);
m=n;
for(i=n-1;i>=1;i--)
{
temp=a[0];
a[0]=a[i];
a[i]=temp;
m=m-1;
Heapify(a,0,m);
}
}
```

Output:



Implementation of stack using Linked List

```
1. void push ()
2. {
3. int val;
4. struct node *ptr =(struct node*)malloc(sizeof(struct node));
5. if(ptr == NULL)
6. {
7. printf("not able to push the element");
8. }
9. else
10. {
11. printf("Enter the value");
12. scanf("%d",&val);
13. if(head==NULL)
14. {
15. ptr->val = val;
16. ptr -> next = NULL;
17. head=ptr;
18. }
19. else
20. {
21. ptr->val = val;
22. ptr->next = head;
23. head=ptr;
24.
25. }
26. printf("Item pushed");
27.
28. }
29. }
```

Deleting a node from the stack (POP operation)

```
void pop()
33. {
34. int item;
35. struct node *ptr;
36. if (head == NULL)
37. {
38. printf("Underflow");
39. }
40. else
41. {
42. item = head->val;
43. ptr = head;
44. head = head->next;
45. free(ptr);
```

```
46. printf("Item popped");
47. }
48. }
```

Menu Driven program in C implementing all the stack operations using linked list :

```
#include <stdio.h>
72. #include <stdlib.h>
73. void push();
74. void pop();
75. void display();
76. struct node
77. {
78. int val;
79. struct node *next;
80. };
81. struct node *head;
82.
83. void main ()
84. {
85. int choice=0;
86. printf("\n*********Stack operations using linked list*********\n");
87. printf("\n----------------------------------------------\n");
88. while(choice != 4)
89. {
90. printf("\n\nChose one from the below options...\n");
91. printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
92. printf("\n Enter your choice \n");
93. scanf("%d",&choice);
94. switch(choice)
95. {
96. case 1:
97. {
98. push();
99. break;
100. }
101. case 2:
102. {
103. pop();
104. break;
105. }
106. case 3:
107. {
108. display();
109. break;
110. }
```

```
111. case 4:
112. {
113. printf("Exiting....");
114. break;
115. }
116. default:
117. {
118. printf("Please Enter valid choice ");
119. }
120. };
121. }
122. }

123. void push ()
124. {
125. int val;
126. struct node *ptr = (struct node*)malloc(sizeof(struct node));
127. if(ptr == NULL)
128. {
129. printf("not able to push the element");
130. }
131. else
132. {
133. printf("Enter the value");
134. scanf("%d",&val);
135. if(head==NULL)
136. {
137. ptr->val = val;
138. ptr -> next = NULL;
139. head=ptr;
140. }
141. else
142. {
143. ptr->val = val;
144. ptr->next = head;
145. head=ptr;
146.
147. }
148. printf("Item pushed");
149.
150. }
151. }
152.
153. void pop()
154. {
155. int item;
156. struct node *ptr;
157. if (head == NULL)
158. {
```

```
159. printf("Underflow");
160. }
161. else
162. {
163. item = head->val;
164. ptr = head;
165. head = head->next;
166. free(ptr);

167. printf("Item popped");
168.
169. }
170. }
171. void display()
172. {
173. int i;
174. struct node *ptr;
175. ptr=head;
176. if(ptr == NULL)
177. {
178. printf("Stack is empty\n");
179. }
180. else
181. {
182. printf("Printing Stack elements \n");
183. while(ptr!=NULL)
184. {
185. printf("%d\n",ptr->val);
186. ptr = ptr->next;
187. }
188. }
189. }
```

Menu-Driven Program implementing all the operations on Linear Queue using linked list

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. struct node
4. {
5. int data;
6. struct node *next;
7. };
8. struct node *front;
9. struct node *rear;
10. void insert();
11. void delete();
```

```
12. void display();
13. void main ()
14. {
15. int choice;
16. while(choice != 4)
17. {
18. printf("\n*************************Main Menu*************************
*****\n");
19. printf("\n===========================================
====================\n");
20. printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit
\n");
21. printf("\nEnter your choice ?");
22. scanf("%d",& choice);
23. switch(choice)
24. {
25. case 1:
26. insert();
27. break;
28. case 2:
29. delete();
30. break;
31. case 3:
32. display();
33. break;
34. case 4:
35. exit(0);
36. break;
37. default:
38. printf("\nEnter valid choice??\n");

39. }
40. }
41. }
42. void insert()
43. {
44. struct node *ptr;
45. int item;
46.
47. ptr = (struct node *) malloc (sizeof(struct node));
48. if(ptr == NULL)
49. {
50. printf("\nOVERFLOW\n");
51. return;
52. }
53. else
54. {
55. printf("\nEnter value?\n");
56. scanf("%d",&item);
```

```
ptr -> data = item;
if(front == NULL)
{
front = ptr;
rear = ptr;
front -> next = NULL;
rear -> next = NULL;
}
else
{
rear -> next = ptr;
rear = ptr;
rear->next = NULL;
}
}
}
void delete ()
{
struct node *ptr;
if(front == NULL)
{
printf("\nUNDERFLOW\n");
return;
}
else
{

ptr = front;
front = front -> next;
free(ptr);
}
}
void display()
{
struct node *ptr;
ptr = front;
if(front == NULL)
{
printf("\nEmpty queue\n");
}
else
{ printf("\nprinting values .....\n");
while(ptr != NULL)
{
printf("\n%d\n",ptr -> data);
ptr = ptr -> next;
}
}
}
```

Output:

```
**********Main Menu**********

==============================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1
Enter value?
123

**********Main Menu**********

==============================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1
Enter value?
90

**********Main Menu**********

==============================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?3
printing values .....
123

90

**********Main Menu**********

==============================

1.insert an element
2.Delete an element
3.Display the queue
```

```
4.Exit

Enter your choice ?2
**********Main Menu**********

==============================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?3
printing values .....
90

**********Main Menu**********

==============================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?4
```

Implementation of circular queue using linked list

As we know that linked list is a linear data structure that stores two parts, i.e., data part and the address part where address part contains the address of the next node. Here, linked list is used to implement the circular queue; therefore, the linked list follows the properties of the Queue. When we are implementing the circular queue using linked list then both the enqueue and dequeue operations take O(1) time.

```
1. #include <stdio.h>
2. // Declaration of struct type node
3. struct node
4. {
5. int data;
6. struct node *next;
7. };
8. struct node *front=-1;
9. struct node *rear=-1;
10. // function to insert the element in the Queue
11. void enqueue(int x)
12. {
13. struct node *newnode; // declaration of pointer of struct node type.
```

```
14. newnode=(struct node *)malloc(sizeof(struct node)); // allocating the memory t
o the newnode
15. newnode->data=x;
16. newnode->next=0;
17. if(rear==-1) // checking whether the Queue is empty or not.
18. {
19. front=rear=newnode;
20. rear->next=front;
21. }
22. else
23. {
24. rear->next=newnode;
25. rear=newnode;
26. rear->next=front;
27. }
28. }
29.
30. // function to delete the element from the queue
31. void dequeue()
32. {
33. struct node *temp; // declaration of pointer of node type
34. temp=front;
35. if((front==-1)&&(rear==-1)) // checking whether the queue is empty or not
36. {
37. printf("\nQueue is empty");
38. }
39. else if(front==rear) // checking whether the single element is left in the queue
40. {
41. front=rear=-1;
42. free(temp);
43. }
44. else
45. {
46. front=front->next;
47. rear->next=front;
48. free(temp);
49. }
50. }
51.
52. // function to get the front of the queue
53. int peek()
54. {
55. if((front==-1) &&(rear==-1))
56. {
57. printf("\nQueue is empty");
58. }
59. else
60. {
61. printf("\nThe front element is %d", front->data);
```

```
62. }
63. }
64.
65. // function to display all the elements of the queue
66. void display()
67. {
68. struct node *temp;
69. temp=front;
70. printf("\n The elements in a Queue are : ");
71. if((front==-1) && (rear==-1))
72. {
73. printf("Queue is empty");
74. }
75.
76. else
77. {
78. while(temp->next!=front)
79. {
80. printf("%d,", temp->data);
81. temp=temp->next;
82. }
83. printf("%d", temp->data);
84. }
85. }
86.
87. void main()
88. {
89. enqueue(34);
90. enqueue(10);
91. enqueue(23);
92. display();
93. dequeue();
94. peek();
95. }
```

DEQUE

Let's create a program of deque.
The following are the six functions that we have used in the below program:
enqueue_front(): It is used to insert the element from the front end.
enqueue_rear(): It is used to insert the element from the rear end.
dequeue_front(): It is used to delete the element from the front end.
dequeue_rear(): It is used to delete the element from the rear end.
getfront(): It is used to return the front element of the deque.
getrear(): It is used to return the rear element of the deque.

```
1. #define size 5
2. #include <stdio.h>
```

```
3. int deque[size];
4. int f=-1, r=-1;
5. // enqueue_front function will insert the value from the front
6. void enqueue_front(int x)
7. {
8. if((f==0 && r==size-1) || (f==r+1))
9. {
10. printf("deque is full");
11. }

12. else if((f==-1) && (r==-1))
13. {
14. f=r=0;
15. deque[f]=x;
16. }
17. else if(f==0)
18. {
19. f=size-1;
20. deque[f]=x;
21. }
22. else
23. {
24. f=f-1;
25. deque[f]=x;
26. }
27. }
28.
29. // enqueue_rear function will insert the value from the rear
30. void enqueue_rear(int x)
31. {
32. if((f==0 && r==size-1) || (f==r+1))
33. {
34. printf("deque is full");
35. }
36. else if((f==-1) && (r==-1))
37. {
38. r=0;
39. deque[r]=x;
40. }
41. else if(r==size-1)
42. {
43. r=0;
44. deque[r]=x;
45. }
46. else
47. {
48. r++;
49. deque[r]=x;
50. }
```

```
51.
52. }
53.
54. // display function prints all the value of deque.
55. void display()

56. {
57. int i=f;
58. printf("\n Elements in a deque : ");
59.
60. while(i!=r)
61. {
62. printf("%d ",deque[i]);
63. i=(i+1)%size;
64. }
65. printf("%d",deque[r]);
66. }
67.
68. // getfront function retrieves the first value of the deque.
69. void getfront()
70. {
71. if((f==-1) && (r==-1))
72. {
73. printf("Deque is empty");
74. }
75. else
76. {
77. printf("\nThe value of the front is: %d", deque[f]);
78. }
79.
80. }
81.
82. // getrear function retrieves the last value of the deque.
83. void getrear()
84. {
85. if((f==-1) && (r==-1))
86. {
87. printf("Deque is empty");
88. }
89. else
90. {
91. printf("\nThe value of the rear is: %d", deque[r]);
92. }
93.
94. }
95.
96. // dequeue_front() function deletes the element from the front
97. void dequeue_front()
98. {
```

```
99. if((f==-1) && (r==-1))

100. {
101. printf("Deque is empty");
102. }
103. else if(f==r)
104. {
105. printf("\nThe deleted element is %d", deque[f]);
106. f=-1;
107. r=-1;
108.
109. }
110. else if(f==(size-1))
111. {
112. printf("\nThe deleted element is %d", deque[f]);
113. f=0;
114. }
115. else
116. {
117. printf("\nThe deleted element is %d", deque[f]);
118. f=f+1;
119. }
120. }
121.
122. // dequeue_rear() function deletes the element from the rear
123. void dequeue_rear()
124. {
125. if((f==-1) && (r==-1))
126. {
127. printf("Deque is empty");
128. }
129. else if(f==r)
130. {
131. printf("\nThe deleted element is %d", deque[r]);
132. f=-1;
133. r=-1;
134.
135. }
136. else if(r==0)
137. {
138. printf("\nThe deleted element is %d", deque[r]);
139. r=size-1;
140. }
141. else
142. {
143. printf("\nThe deleted element is %d", deque[r]);

144. r=r-1;
145. }
```

```
146. }
147.
148. int main()
149. {
150. // inserting a value from the front.
151. enqueue_front(2);
152. // inserting a value from the front.
153. enqueue_front(1);
154. // inserting a value from the rear.
155. enqueue_rear(3);
156. // inserting a value from the rear.
157. enqueue_rear(5);
158. // inserting a value from the rear.
159. enqueue_rear(8);
160. // Calling the display function to retrieve the values of deque
161. display();
162. // Retrieve the front value
163. getfront();
164. // Retrieve the rear value.
165. getrear();
166. // deleting a value from the front
167. dequeue_front();
168. //deleting a value from the rear
169. dequeue_rear();
170. // Calling the display function to retrieve the values of deque
171. display();
172. return 0;
173. }
```

Deque implementation using Linked List

```
#include <stdio.h>
 #include <stdlib.h>
 struct node {
      int data;
      struct node *prev, *next;
 };
 struct node *head = NULL, *tail = NULL;
 struct node * createNode(int data) {
      struct node *newnode = (struct node *)malloc(sizeof (struct node));
      newnode-&gt;data = data;
      newnode-&gt;next = newnode-&gt;prev = NULL;
      return (newnode);
 }
  /*
   * create sentinel(dummy head &amp;amp; tail) that
   * helps us to do insertion and deletion
   * operation at front and rear so easily.  And
```

```
 * these dummy head and tail wont get deleted
 * till the end of execution of this program
 */
void createSentinels() {
    head = createNode(0);
    tail = createNode(0);
    head-&gt;next = tail;
    tail-&gt;prev = head;
}
/* insertion at the front of the queue */
void enqueueAtFront(int data) {
    struct node *newnode, *temp;
    newnode = createNode(data);
    temp = head-&gt;next;
    head-&gt;next = newnode;
    newnode-&gt;prev = head;
    newnode-&gt;next = temp;
    temp-&gt;prev = newnode;
}
/*insertion at the rear of the queue */
void enqueueAtRear(int data) {
    struct node *newnode, *temp;
    newnode = createNode(data);

    temp = tail-&gt;prev;
    tail-&gt;prev = newnode;
    newnode-&gt;next = tail;
    newnode-&gt;prev = temp;
    temp-&gt;next = newnode;
}
/* deletion at the front of the queue */
void dequeueAtFront() {
    struct node *temp;
    if (head-&gt;next == tail) {
        printf(&quot;Queue is empty\n&quot;);
    } else {
        temp = head-&gt;next;
        head-&gt;next = temp-&gt;next;
        temp-&gt;next-&gt;prev = head;
        free(temp);
    }
    return;
}

/* deletion at the rear of the queue */
void dequeueAtRear()  {
    struct node *temp;
    if (tail-&gt;prev == head) {
        printf(&quot;Queue is empty\n&quot;);
```

```
    } else {
        temp = tail-&gt;prev;
        tail-&gt;prev = temp-&gt;prev;
        temp-&gt;prev-&gt;next = tail;
        free(temp);
    }
    return;
}
/* display elements present in the queue */
void display() {
    struct node *temp;
    if (head-&gt;next == tail) {
        printf(&quot;Queue is empty\n&quot;);
        return;
    }
    temp = head-&gt;next;
    while (temp != tail) {
        printf(&quot;%-3d&quot;, temp-&gt;data);
        temp = temp-&gt;next;
    }
    printf(&quot;\n&quot;);

}
int main() {
    int data, ch;
    createSentinels();
    while (1) {
        printf(&quot;1. Enqueue at front\n2. Enqueue at rear\n&quot;);
        printf(&quot;3. Dequeue at front\n4. Dequeue at rear\n&quot;);
        printf(&quot;5. Display\n6. Exit\n&quot;);
        printf(&quot;Enter your choice:&quot;);
        scanf(&quot;%d&quot;, &amp;ch);
        switch (ch) {
            case 1:
                printf(&quot;Enter the data to insert:&quot;);
                scanf(&quot;%d&quot;, &amp;data);
                enqueueAtFront(data);
                break;
            case 2:
                printf(&quot;Enter ur data to insert:&quot;);
                scanf(&quot;%d&quot;, &amp;data);
                enqueueAtRear(data);
                break;
            case 3:
                dequeueAtFront();
                break;
            case 4:
                dequeueAtRear();
                break;
```

```
                case 5:
                        display();
                        break;
                case 6:
                        exit(0);
                default:
                        printf(&amp;quot;Pls. enter correct option\n&amp;quot;);
                        break;
        }
    }
    return 0;
}
```

Write a program to create a Binary Search Tree and include following operations in tree:

(a) Insertion
(b) Deletion.
(c) Search a node in BST.
(d) Display its preorder, postorder and inorder traversals

```
1.
2. /*
3. * C Program to Construct a Binary Search Tree and perform

insertion, deletion, searching and inorder, pre-order, post-
order traversal on it.

4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct btnode
9. {
10. int value;
11. struct btnode *l;
12. struct btnode *r;
13. }*root = NULL,

*temp = NULL,
*t2,
*t1;

14.
15. void delete1();
16. void insert();
17. void delete();
18. void inorder(struct btnode *t);
19. void create();
20. void search(struct btnode *t);
21. void preorder(struct btnode *t);
22. void postorder(struct btnode *t);
23. void search1(struct btnode *t,int data);
24. int smallest(struct btnode *t);
25. int largest(struct btnode *t);
26.
27. int flag = 1;
28.
29. void main()
30. {
31. int ch;
```

```
32.
33. printf("\nOPERATIONS ---");
34. printf("\n1 - Insert an element into tree\n");

35. printf("2 - Delete an element from the tree\n");
36. printf("3 - Inorder Traversal\n");
37. printf("4 - Preorder Traversal\n");
38. printf("5 - Postorder Traversal\n");
39. printf("6 - Exit\n");
40. while(1)
41. {
42. printf("\nEnter your choice : ");
43. scanf("%d", &ch);
44. switch (ch)
45. {
46. case 1:
47. insert();
48. break;
49. case 2:
50. delete();
51. break;
52. case 3:
53. inorder(root);
54. break;
55. case 4:
56. preorder(root);
57. break;
58. case 5:
59. postorder(root);
60. break;
61. case 6:
62. exit(0);
63. default :
64. printf("Wrong choice, Please enter correct
choice ");
65. break;
66. }
67. }
68. }
69.
70. /* To insert a node in the tree */
71. void insert()
72. {
73. create();
74. if (root == NULL)
75. root = temp;

76. else
77. search(root);
```

```
78. }
79.
80. /* To create a node */
81. void create()
82. {
83. int data;
84.
85. printf("Enter data of node to be inserted : ");
86. scanf("%d", &data);
87. temp = (struct btnode *)malloc(1*sizeof(struct
btnode));
88. temp->value = data;
89. temp->l = temp->r = NULL;
90. }
91.
92. /* Function to search the appropriate position to insert
the new node */
93. void search(struct btnode *t)
94. {
95. if ((temp->value > t->value) && (t->r != NULL))
/* value more than root node value insert at right */
96. search(t->r);
97. else if ((temp->value > t->value) && (t->r == NULL))
98. t->r = temp;
99. else if ((temp->value < t->value) && (t->l != NULL))
/* value less than root node value insert at left */
100. search(t->l);
101. else if ((temp->value < t->value) && (t->l == NULL))
102. t->l = temp;
103. }
104.
105. /* recursive function to perform inorder traversal of
tree */
106. void inorder(struct btnode *t)
107. {
108. if (root == NULL)
109. {
110. printf("No elements in a tree to display");
111. return;
112. }
113. if (t->l != NULL)

114. inorder(t->l);
115. printf("%d -> ", t->value);
116. if (t->r != NULL)
117. inorder(t->r);
118. }
119.
120. /* To check for the deleted node */
```

```
121. void delete()
122. {
123. int data;
124.
125. if (root == NULL)
126. {
127. printf("No elements in a tree to delete");
128. return;
129. }
130. printf("Enter the data to be deleted : ");
131. scanf("%d", &data);
132. t1 = root;
133. t2 = root;
134. search1(root, data);
135. }
136.
137. /* To find the preorder traversal */
138. void preorder(struct btnode *t)
139. {
140. if (root == NULL)
141. {
142. printf("No elements in a tree to display");
143. return;
144. }
145. printf("%d -> ", t->value);
146. if (t->l != NULL)
147. preorder(t->l);
148. if (t->r != NULL)
149. preorder(t->r);
150. }
151.
152. /* To find the postorder traversal */
153. void postorder(struct btnode *t)
154. {
155. if (root == NULL)
156. {

157. printf("No elements in a tree to display ");
158. return;
159. }
160. if (t->l != NULL)
161. postorder(t->l);
162. if (t->r != NULL)
163. postorder(t->r);
164. printf("%d -> ", t->value);
165. }
166.
167. /* Search for the appropriate position to insert the new
node */
```

```
168. void search1(struct btnode *t, int data)
169. {
170. if ((data>t->value))
171. {
172. t1 = t;
173. search1(t->r, data);
174. }
175. else if ((data < t->value))
176. {
177. t1 = t;
178. search1(t->l, data);
179. }
180. else if ((data==t->value))
181. {
182. delete1(t);
183. }
184. }
185.
186. /* To delete a node */
187. void delete1(struct btnode *t)
188. {
189. int k;
190.
191. /* To delete leaf node */
192. if ((t->l == NULL) && (t->r == NULL))
193. {
194. if (t1->l == t)
195. {
196. t1->l = NULL;
197. }
198. else

199. {
200. t1->r = NULL;
201. }
202. t = NULL;
203. free(t);
204. return;
205. }
206.
207. /* To delete node having one left hand child */
208. else if ((t->r == NULL))
209. {
210. if (t1 == t)
211. {
212. root = t->l;
213. t1 = root;
214. }
215. else if (t1->l == t)
```

```
216. {
217. t1->l = t->l;
218.
219. }
220. else
221. {
222. t1->r = t->l;
223. }
224. t = NULL;
225. free(t);
226. return;
227. }
228.
229. /* To delete node having right hand child */
230. else if (t->l == NULL)
231. {
232. if (t1 == t)
233. {
234. root = t->r;
235. t1 = root;
236. }
237. else if (t1->r == t)
238. t1->r = t->r;
239. else
240. t1->l = t->r;

241. t == NULL;
242. free(t);
243. return;
244. }
245.
246. /* To delete node having two child */
247. else if ((t->l != NULL) && (t->r != NULL))
248. {
249. t2 = root;
250. if (t->r != NULL)
251. {
252. k = smallest(t->r);
253. flag = 1;
254. }
255. else
256. {
257. k =largest(t->l);
258. flag = 2;
259. }
260. search1(root, k);
261. t->value = k;
262. }
263.
```

```
264. }
265.
266. /* To find the smallest element in the right sub tree */
267. int smallest(struct btnode *t)
268. {
269. t2 = t;
270. if (t->l != NULL)
271. {
272. t2 = t;
273. return(smallest(t->l));
274. }
275. else
276. return (t->value);
277. }
278.
279. /* To find the largest element in the left sub tree */
280. int largest(struct btnode *t)
281. {
282. if (t->r != NULL)
283. {

284. t2 = t;
285. return(largest(t->r));
286. }
287. else
288. return(t->value);
289. }
```

```
$ cc tree43.c
$ a.out

OPERATIONS ---
1 - Insert an element into tree
2 - Delete an element from the tree
3 - Inorder Traversal
4 - Preorder Traversal
5 - Postorder Traversal
6 – Exit

Enter your choice : 1
Enter data of node to be inserted : 40
Enter your choice : 1
Enter data of node to be inserted : 20
Enter your choice : 1
Enter data of node to be inserted : 10
Enter your choice : 1
Enter data of node to be inserted : 30
Enter your choice : 1
Enter data of node to be inserted : 60
```

```
Enter your choice : 1
Enter data of node to be inserted : 80
Enter your choice : 1
Enter data of node to be inserted : 90
Enter your choice : 3
10 -> 20 -> 30 -> 40 -> 60 -> 80 -> 90 ->
40
/\
/ \
20 60

/ \ \
10 30 80
\
90
```

Write a program to create a Binary Search Tree and to count the number of leaf nodes.

```
1. /* C Program to find the number of leaf nodes in a Tree */
2. #include <stdio.h>
3. #include <stdlib.h>

4.
5. struct node
6. {
7. int info;
8. struct node* left, *right;
9.
10. };
11.
12. /*
13. * Function to create new nodes
14. */
15.
16. struct node* createnode(int key)
17. {
18. struct node* newnode = (struct node*)malloc(sizeof(struct node));
19. newnode->info = key;
20. newnode->left = NULL;
21. newnode->right = NULL;
22.
23. return(newnode);
24. }
25.
26. /*
27. * Function to count number of leaf nodes
28. */
29.
```

```
30. int count = 0;
31. int leafnodes(struct node* newnode)
32. {
33.
34. if(newnode != NULL)
35. {
36. leafnodes(newnode->left);
37. if((newnode->left == NULL) && (newnode->right == NULL))
38. {
39. count++;
40. }
41. leafnodes(newnode->right);
42. }
43. return count;
44.
45. }
46.
47. /*
48. * Main Function
49. */
50.
51. int main()

52. {
53. /* Creating first Tree.*/
54.
55. struct node *newnode = createnode(25);
56. newnode->left = createnode(27);
57. newnode->right = createnode(19);
58. newnode->left->left = createnode(17);
59. newnode->left->right = createnode(91);
60. newnode->right->left = createnode(13);
61. newnode->right->right = createnode(55);
62.
63. /* Sample Tree 1- Balanced Tree
64.
65.
66. 25
67. / \
68. 27 19
69. / \ / \
70. 17 91 13 55
71.
72. */
73. printf("Number of leaf nodes in first Tree are\t%d\n",leafnodes(newnode));
74. count = 0;
75.
76. struct node *node = createnode(1);
77. node->right = createnode(2);
```

```
78. node->right->right = createnode(3);
79. node->right->right->right = createnode(4);
80. node->right->right->right->right = createnode(5);
81.
82. /* Sample Tree 2- Right Skewed Tree (Unbalanced).
83.
84. 1
85. \
86. 2
87. \
88. 3
89. \
90. 4
91. \
92. 5
93. */
94.
95. printf("\nNumber of leaf nodes in second tree are\t%d\n",leafnodes(node));
96. count = 0;
97.
98. /*Creating third Tree. */
99.

100. struct node *root = createnode(15);
101.
102. /* Sample Tree 3- Tree having just one root node.
103.
104. 15
105. */
106.
107. printf("\nNumber of leaf nodes in third tree are\t%d",leafnodes(root));
108.
109. return 0;
110. }
```

Program Explanation

1. In this program we have used recursion to find the total number of leaf nodes present in a tree. A Leaf Node is one whose left and right child are NULL.
2. We have created a function called leafnodes() which takes in root of the tree as a parameter and returns the total number of leaf nodes it has.
3. The basic idea is to traverse the tree using any traversal so as to visit each and every node and check the condition for leaf node for each node, that is what we have done in leafnodes() function.
4. In the leafnodes() function we have used the inorder traversal, by first traversing the left subtree, then instead of printing the root->data as a second step of inorder traversal, we have checked the leaf node condition and then at last we have traversed the right subtree by passing root->right as a parameter.

Write a program to create a Binary Search Tree and to count
the number of non-leaf nodes.

Lab manual for Data Structure using C

```
1.
2.
3. /* C Program to find the number of non leaf nodes in a Tree */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9. int info;
10. struct node* left, *right;
11.
12. };
13.
14. /*
15. * Function to create new nodes
16. */
17.
18. struct node* createnode(int key)
19. {
20. struct node* newnode = (struct node*)malloc(sizeof(struct node));
21. newnode->info = key;
22. newnode->left = NULL;
23. newnode->right = NULL;
24.
25. return(newnode);
26. }
27.
28. /*
29. * Function to count number of leaf nodes
30. */
31.
32. int count = 0;
33. int nonleafnodes(struct node* newnode)
34. {
35.
36. if(newnode != NULL)
37. {

38. nonleafnodes(newnode->left);
39. if((newnode->left != NULL) || (newnode->right != NULL))
40. {
41. count++;
42. }
43. nonleafnodes(newnode->right);
44. }
45. return count;
46.
47. }
48.
```

```
/*
* Main Function
*/

int main()
{
/* Creating first Tree.*/

struct node *newnode = createnode(25);
newnode->left = createnode(27);
newnode->right = createnode(19);
newnode->left->left = createnode(17);
newnode->left->right = createnode(91);
newnode->right->left = createnode(13);
newnode->right->right = createnode(55);

/* Sample Tree 1- Balanced Tree


      25
     / \
   27 19
  / \ / \
 17 91 13 55

*/
printf("Number of non leaf nodes in first Tree are\t%d",nonleafnodes(newnode));
printf("\n");
count = 0;

struct node *node = createnode(1);
node->right = createnode(2);
node->right->right = createnode(3);
node->right->right->right = createnode(4);
node->right->right->right->right = createnode(5);

/* Sample Tree 2- Right Skewed Tree (Unbalanced).


  1
   \
    2
     \
      3
       \
        4
         \
          5
*/
```

```
97. printf("\n");
98. printf("Number of non leaf nodes in second tree are\t%d",nonleafnodes(node));
99. printf("\n");
100. count = 0;
101.
102. /*Creating third Tree. */
103.
104. struct node *root = createnode(15);
105.
106. /* Sample Tree 3- Tree having just one root node.
107.
108. 15
109. */
110. printf("\n");
111. printf("Number of non leaf nodes in third tree are\t%d",nonleafnodes(root));
112.
113. return 0;
114. }
```

Program Explanation

1. In this program we have used recursion to find the total number of non leaf nodes present in a tree. A non leaf Node is one whose left or the right child is not NULL.
2. We have created a function called nonleafnodes() which takes in root of the tree as a parameter and returns the total number of non leaf nodes it has.
3. The basic idea is to traverse the tree using any traversal so as to visit each and every node and check the condition for non leaf node for each node, that is what we have done in nonleafnodes() function.
4. In the nonleafnodes() function we have used the inorder traversal, by first traversing the left subtree, then instead of printing the root->data as a second step of inorder traversal, we have checked the nonleaf node condition and then at last we have traversed the right subtree by passing root->right as a parameter.

Write a program to create a Binary Search Tree and to Print Height and Depth of given Binary Tree.

```
1.
2.
3. /*
4. * C Program to Print Height and Depth of given Binary Tree
5. * 50
6. * / \
7. * 20 30
8. * / \
9. * 70 80
10. * / \ \
11. * 10 40 60
12. */
13. #include <stdio.h>
14. #include <stdlib.h>
```

```
15.
16. struct btnode {
17. int value;
18. struct btnode *l;
19. struct btnode *r;
20. };
21. struct btnode *root;
22. typedef struct btnode bt;
23. bt *new,*ptr,*ptr1,*ptr2;

24.
25. bt* create()
26. {
27. new = (bt *)malloc(sizeof(bt));
28. new->l = NULL;
29. new->r = NULL;
30. return new;
31. }
32.
33. void construct_binary_tree()
34. {
35. root = create();
36. root->value = 50;
37.
38. ptr = create();
39. root->l = ptr;
40. ptr->value = 20;
41.
42. ptr1 = create();
43. ptr->l = ptr1;
44. ptr1->value = 70;
45.
46. ptr2 = create();
47. ptr1->l = ptr2;
48. ptr2->value = 10;
49.
50. ptr2 = create();
51. ptr1->r = ptr2;
52. ptr2->value = 40;
53.
54. ptr1 = create();
55. ptr->r = ptr1;
56. ptr1->value = 80;
57.
58. ptr2 = create();
59. ptr1->r = ptr2;
60. ptr2->value = 60;
61.
62. ptr = create();
```

```
63. root->r = ptr;
64. ptr->value = 30;
65. }
66.
67. void main()
68. {
69. int depth1 = 0, depth2 = 0;
70.
71. construct_binary_tree();

72. ptr = root;
73. while (ptr->l != NULL || ptr->r != NULL)
74. {
75. depth1++;
76. if (ptr->l == NULL)
77. ptr = ptr->r;
78. else
79. ptr = ptr->l;
80. }
81. ptr = root;
82. while (ptr->l != NULL || ptr->r != NULL)
83. {
84. depth2++;
85. if (ptr->r == NULL)
86. ptr = ptr->l;
87. else
88. ptr = ptr->r;
89. }
90. /*
91. *DEPTH IS CONSIDERED FROM LEVEL-0 ALSO HEIGHT IS CONSIDERED AS NUMBER OF
EDGES
92. */
93. if (depth1 > depth2)
94. printf("height of the tree is %d\ndepth of the tree is %d",depth1,depth1);
95. else
96. printf("height of the tree is %d\ndepth of the tree is %d",depth2,depth2);
97. }
$ cc tree18.c
$ a.out
height of the tree is 3
depth of the tree is 3
```

C Program for level order (level by level) traversal of a Tree

```
*/
1.#include <stdio.h>
2.#include <stdlib.h>
3.struct node
4.{
5. int info;
```

```
6. struct node *left, *right;
7.};
8.struct node *createnode(int key)
9.{
10. struct node *newnode = (struct node*)malloc(sizeof(struct node));

11. newnode->info = key;
12. newnode->left = NULL;
13. newnode->right = NULL;
14. return(newnode);
15.}
16./*
17. * Function to ascertain the height of a Tree
18. */
19.int heightoftree(struct node *root)
20.{
21. int max;
22. if (root!=NULL)
23. {
24. /* Finding the height of left subtree. */
25. int leftsubtree = heightoftree(root->left);
26. /* Finding the height of right subtree. */
27. int rightsubtree = heightoftree(root->right);
28. if (leftsubtree > rightsubtree)
29. {
30. max = leftsubtree + 1;
31. return max;
32. }
33. else
34. {
35. max = rightsubtree + 1;
36. return max;
37. }
38. }
39.}
40./*
41. * Function to print all the nodes left to right of the current level
42. */
43.void currentlevel(struct node *root, int level)
44.{
45. if (root != NULL)
46. {
47. if (level == 1)
48. {
49. printf("%d ", root->info);
50. }
51. else if (level > 1)
52. {
53. currentlevel(root->left, level-1);
```

```
54. currentlevel(root->right, level-1);
55. }
56. }
57.}
58./*

59. * Main Function
60. */
61.int main()
62.{
63. /* Creating first Tree. */
64. struct node *newnode = createnode(25);
65. newnode->left = createnode(27);
66. newnode->right = createnode(19);
67. newnode->left->left = createnode(17);
68. newnode->left->right = createnode(91);
69. newnode->right->left = createnode(13);
70. newnode->right->right = createnode(55);
71. /* Sample Tree 1:
72. * 25
73. * / \
74. * 27 19
75. * / \ / \
76. * 17 91 13 55
77. */
78. printf("Level Order Traversal of Tree 1 is \n");
79. int i;
80. int height = heightoftree(newnode);
81. /* Calling current level function, by passing levels one by one. */
82. for(i = 1; i <= height; i++)
83. {
84. currentlevel(newnode,i);
85. }
86.
87. /* Creating second Tree. */
88. struct node *node = createnode(1);
89. node->right = createnode(2);
90. node->right->right = createnode(3);
91. node->right->right->right = createnode(4);
92. node->right->right->right->right = createnode(5);
93. /* Sample Tree 2: Right Skewed Tree (Unbalanced).
94. * 1
95. * \
96. * 2
97. * \
98. * 3
99. * \
100. * 4
101. * \
```

```
102. * 5
103. */
104. printf("\n\nLevel Order Traversal of Tree 2 is \n");
105. height = heightoftree(node);
106. /* Calling current level function, by passing levels one by one. */

107. for(i = 1; i <= height; i++)
108. {
109. currentlevel(node,i);
110. }
111.
112. /* Creating third Tree. */
113. struct node *root = createnode(15);
114. /* Sample Tree 3- Tree having just one root node.
115. * 15
116. */
117. printf("\n\nLevel Order Traversal of Tree 3 is \n");
118. height = heightoftree(root);
119. /* calling current level function, by passing levels one by one. */
120. for(i = 1; i <= height; i++)
121. {
122. currentlevel(root,i);
123. }
124. return 0;
125.}

/*
1. * C Program to Create a Mirror Copy of a Tree
2. * 40
3. /\
4. 20 60
5. /\ \
6. 10 30 80
7. \
8. 90
9. */
10. #include <stdio.h>
11. #include <stdlib.h>
12.
13. struct btnode
14. {

15. int value;
16. struct btnode *left, *right;
17. };
18. typedef struct btnode node;
19. node *root = NULL, *temp, *list;
20.
21. /* function prototypes */
```

```
22. void insert(node *, node *);
23. void inorder(node *);
24. void mirror(node *);
25.
26. void main()
27. {
28. node *new = NULL;
29. int num = 1;
30.
31. printf("Enter the elements of the tree(enter 0 to exit)\n");
32. while (1)
33. {
34. scanf("%d", &num);
35. if (num == 0)
36. break;
37. new = malloc(sizeof(node));
38. new->left = new->right = NULL;
39. new->value = num;
40. if (root == NULL)
41. root = new;
42. else
43. {
44. insert(new, root);
45. }
46. }
47. printf("elements in a tree in inorder are\n");
48. inorder(root);
49. temp = root;
50. printf("\nmirror image of the elements are\n");
51. mirror(temp);
52. inorder(temp);
53. printf("\n");
54. }
55.
56. /* displaying nodes of a tree using inorder */
57. void inorder(node *list)
58. {
59. if (list != NULL)
60. {
61. inorder(list->left);
62. printf("%d\t", list->value);

63. inorder(list->right);
64. }
65. }
66.
67. /* inserting nodes into the tree */
68. void insert(node * new , node *root)
69. {
```

```
70. if (new->value>root->value)
71. {
72. if (root->right == NULL)
73. root->right = new;
74. else
75. insert (new, root->right);
76. }
77. if (new->value < root->value)
78. {
79. if (root->left == NULL)
80. root->left = new;
81. else
82. insert (new, root->left);
83. }
84. }
85.
86. /* mirror image of nodes */
87. void mirror(node *temp)
88. {
89. node *temp1;
90. if (temp == NULL)
91. return;
92. temp1 = temp->left;
93. temp->left = temp->right;
94. temp->right = temp1;
95. mirror(temp->left);
96. mirror(temp->right);
97. }
```

```
$ cc tree16.c
$ a.out
Enter the elements of the tree(enter 0 to exit)
40
20
60
10
30
80
90
0
elements in a tree in inorder are
10 20 30 40 60 80 90

mirror image of the elements are
90 80 60 40 30 20 10
```