S.No: 1	Exp. Name: ***Write a C program to find the Factorial of a given number using Recursion***	Date:

Aim:

Write a program to find the `factorial` of a given number using recursion process.

At the time of execution, the program should print the message on the console as:

```
Enter an integer :
```

For example, if the user gives the **input** as:

```
Enter an integer : 6
```

then the program should **print** the result as:

```
Factorial of 6 is : 720
```

Note: Write the recursive function **factorial()** in `Program901a.c`.

Source Code:

Program901.c

```
#include <stdio.h>
#include "Program901a.c"
void main() {
   long int n;
   printf("Enter an integer : ");
   scanf("%ld", &n);
   printf("Factorial of %ld is : %ld\n", n ,factorial(n));
}
```

Program901a.c

```
long int factorial(long int n)
{
   //Factorial of 0 is 1
   if(n==0)
      return(1);

   //Function calling itself: recursion
   return(n*factorial(n-1));
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter an integer : 5
Factorial of 5 is : 120

Test Case - 2
User Output
Enter an integer : 7
Factorial of 7 is : 5040

Test Case - 3
User Output
Enter an integer : 4
Factorial of 4 is : 24

Test Case - 4
User Output
Enter an integer : 8
Factorial of 8 is : 40320

Test Case - 5
User Output
Enter an integer : 0
Factorial of 0 is : 1

Test Case - 6
User Output
Enter an integer : 9
Factorial of 9 is : 362880

S.No: 2	Exp. Name: ***Write a C program to display the Fibonacci series up to the given number of terms using Recursion***	Date:

Aim:

Write a program to display the `fibonacci series` up to the given number of terms using recursion process.

The fibonacci series is `0 1 1 2 3 5 8 13 21 34.....`.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 6
```

then the program should **print** the result as:

```
The fibonacci series of 6 terms are :  0  1  1  2  3  5
```

Note: Write the recursive function **fib()** in `Program908a.c`.

Source Code:

Program908.c

```
#include <stdio.h>
#include "Program908a.c"
void main() {
   int n, i;
   printf("Enter value of n : ");
   scanf("%d", &n);
   printf("The fibonacci series of %d terms are : ", n);
   for (i = 0; i < n; i++) {
      printf(" %d ", fib(i));
   }
}
```

Program908a.c

```
int fib(int n)
{
   if(n<=1)
   return n;
   else
   return fib(n-1)+fib(n-2);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n : 4

Test Case - 1
The fibonacci series of 4 terms are : 0 1 1 2

Test Case - 2
User Output
Enter value of n : 8
The fibonacci series of 8 terms are : 0 1 1 2 3 5 8 13

Test Case - 3
User Output
Enter value of n : 14
The fibonacci series of 14 terms are : 0 1 1 2 3 5 8 13 21 34 55 89 144 233

Test Case - 4
User Output
Enter value of n : 3
The fibonacci series of 3 terms are : 0 1 1

S.No: 3	Exp. Name: ***Write a C program to find the GCD of two numbers using Recursion***	Date:

Aim:

Write a program to find the `gcd` (Greatest Common Divisor) of a given two numbers using recursion process.

The greatest common divisor (`gcd`) of two or more integers, when at least one of them is not zero, is the largest positive integer that is a divisor of both numbers.

At the time of execution, the program should print the message on the console as:

```
Enter two integer values :
```

For example, if the user gives the **input** as:

```
Enter two integer values : 12 18
```

then the program should **print** the result as:

```
The gcd of two numbers 12 and 18 = 6
```

Note: Write the recursive function **gcd()** in `Program906a.c` .

Source Code:

Program906.c

```
#include <stdio.h>
#include "Program906a.c"
void main() {
   int a, b;
   printf("Enter two integer values : ");
   scanf("%d %d", &a, &b);
   printf("The gcd of two numbers %d and %d = %d\n", a, b, gcd(a, b));
}
```

Program906a.c

```
int gcd(int a,int b)
{
   if(a%b==0)
   return b;
   else
   gcd(b,a%b);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter two integer values : 12 15
The gcd of two numbers 12 and 15 = 3

Test Case - 2
User Output
Enter two integer values : 36 124
The gcd of two numbers 36 and 124 = 4

S.No: 4	Exp. Name: ***Write a C program to solve the Towers of Hanoi problem using Recursion***	Date:

Aim:

Write a program to solve the towers of hanoi (https://en.wikipedia.org/wiki/Tower_of_Hanoi) problem using recursion process.

At the time of execution, the program should print the message on the console as:

```
Enter number of disks :
```

For example, if the user gives the **input** as:

```
Enter number of disks : 3
```

then the program should **print** the result as:

```
Move disk - 1 from pole A to C
Move disk - 2 from pole A to B
Move disk - 1 from pole C to B
Move disk - 3 from pole A to C
Move disk - 1 from pole B to A
Move disk - 2 from pole B to C
Move disk - 1 from pole A to C
```

Note: Write the recursive function **hanoi()** in `Program909a.c` and do use the **printf()** function with a **newline** character (`\n`) at the end.

Source Code:

Program909.c

```
#include <stdio.h>
#include "Program909a.c"
void main() {
   int n;
   printf("Enter number of disks : ");
   scanf("%d", &n);
   hanoi(n, 'A', 'B', 'C');
}
```

Program909a.c

```
void hanoi(int n, char S, char A, char D)
{
    if (n == 1)
     printf("Move disk - %d from pole %c to %c\n", n, S, D);
    else
  {
    hanoi(n - 1, S, D, A);
    printf("Move disk - %d from pole %c to %c\n", n, S, D);
    hanoi(n - 1, A, S, D);
  }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter number of disks : 2
Move disk - 1 from pole A to B
Move disk - 2 from pole A to C
Move disk - 1 from pole B to C

Test Case - 2
User Output
Enter number of disks : 4
Move disk - 1 from pole A to B
Move disk - 2 from pole A to C
Move disk - 1 from pole B to C
Move disk - 3 from pole A to B
Move disk - 1 from pole C to A
Move disk - 2 from pole C to B
Move disk - 1 from pole A to B
Move disk - 4 from pole A to C
Move disk - 1 from pole B to C
Move disk - 2 from pole B to A
Move disk - 1 from pole C to A
Move disk - 3 from pole B to C
Move disk - 1 from pole A to B
Move disk - 2 from pole A to C
Move disk - 1 from pole B to C

S.No: 5	Exp. Name: ***Write a C program to Search an element using Linear Search process***	Date:

Aim:

Write a program to `search` a key element with in the given array of elements using `linear search` process.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 3
```

Next, the program should print the messages one by one on the console as:

```
Enter element for a[0] :
Enter element for a[1] :
Enter element for a[2] :
```

if the user gives the **input** as:

```
Enter element for a[0] : 89
Enter element for a[1] : 33
Enter element for a[2] : 56
```

Next, the program should print the message on the console as:

```
Enter key element :
```

if the user gives the **input** as:

```
Enter key element : 56
```

then the program should **print** the result as:

```
The key element 56 is found at the position 2
```

Similarly if the key element is given as **25** for the above one dimensional array elements then the program should print the output as "**The Key element 25 is not found in the array**".

Note: Do use the **printf()** function with a **newline** character (`\n`) at the end.

Source Code:

Program509.c

```
#include<stdio.h>

int lsr(int a[],int low,int high,int key)

{
if(low>high)
return -1;
if(key==a[low])
return low;
else
return lsr(a,low+1,high,key);
}
```

```
void main()
{
int a[50],n,key,i,pos;
printf("Enter value of n : ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter element for a[%d] : ",i);
scanf("%d",&a[i]);
}
printf("Enter key element : ");
scanf("%d",&key);
pos=lsr(a,0,n-1,key);
if(pos==-1)
printf("The key element %d is not found in the array\n",key);
else
printf("The key element %d is found at the position %d\n",key,pos);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n : 5
Enter element for a[0] : 45
Enter element for a[1] : 67
Enter element for a[2] : 35
Enter element for a[3] : 28
Enter element for a[4] : 16
Enter key element : 28
The key element 28 is found at the position 3

Test Case - 2
User Output
Enter value of n : 5
Enter element for a[0] : 2
Enter element for a[1] : 7
Enter element for a[2] : 5
Enter element for a[3] : 1
Enter element for a[4] : 4
Enter key element : 2
The key element 2 is found at the position 0

Test Case - 3
User Output
Enter value of n : 4
Enter element for a[0] : 452
Enter element for a[1] : 356
Enter element for a[2] : 754
Enter element for a[3] : 127

Test Case - 3
Enter key element : 127
The key element 127 is found at the position 3

Test Case - 4
User Output
Enter value of n : 3
Enter element for a[0] : 5
Enter element for a[1] : 7
Enter element for a[2] : 3
Enter key element : 4
The key element 4 is not found in the array

Test Case - 5
User Output
Enter value of n : 3
Enter element for a[0] : 11
Enter element for a[1] : 45
Enter element for a[2] : 37
Enter key element : 25
The key element 25 is not found in the array

S.No: 6	Exp. Name: ***Write a Program to Search an element using Linear Search and Recursion***	Date:

Aim:

Write a program to `search` the given element from a list of elements with `linear search` technique using **recursion**.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 6
```

Next, the program should print the message on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 12 54 32 9 26
```

Next, the program should print the message on the console as:

```
Enter a key element :
```

if the user gives the **input** as:

```
Enter a key element : 9
```

then the program should **print** the result as:

```
The key element 9 is found at position : 3
```

Similarly, if the key element is given as `18` for the above example then the program should print the output as:

```
The key element 18 is not found
```

Note: Write the functions **read()** and **linearSearch()** in `Program911a.c`

Source Code:

Program911.c

```
#include <stdio.h>
#include "Program911a.c"
void main() {
   int a[20], n, pos, key;
   printf("Enter n value : ");
   scanf("%d", &n);
   read(a, n);
   printf("Enter a key element : ");
   scanf("%d", &key);
   pos = linearSearch(a, 0, n - 1, key);
   if (pos == -1) {
      printf("The key element %d is not found\n", key);
   } else {
      printf("The key element %d is found at position : %d\n", key, pos);
```

```
   }
}
```

Program911a.c

```
void read(int a[], int n){
   int i;
   printf("Enter %d elements : ", n);
   for (i=0; i<n; i++){
      scanf("%d", &a[i]);
   }
}
int linearSearch(int a[], int pos, int m, int key) {
   if (pos>m){
      return -1;
   }
   if (a[pos]==key) {
      return pos;
   }
   else {
      return linearSearch(a, pos+1, m, key);
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter n value : 4
Enter 4 elements : 10 20 15 12
Enter a key element : 15
The key element 15 is found at position : 2

Test Case - 2
User Output
Enter n value : 6
Enter 6 elements : 2 6 4 1 3 7
Enter a key element : 5
The key element 5 is not found

Test Case - 3
User Output
Enter n value : 5
Enter 5 elements : 11 44 33 55 22
Enter a key element : 11
The key element 11 is found at position : 0

Test Case - 4

Test Case - 4
User Output
Enter n value : 5
Enter 5 elements : 99 65 78 34 27
Enter a key element : 26
The key element 26 is not found

S.No: 7	Exp. Name: ***Write a C program to Search an element using Binary Search process***	Date:

Aim:

Write a program to `search` a key element in the given array of elements using `binary search`.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 3
```

Next, the program should print the messages one by one on the console as:

```
Enter element for a[0] :
Enter element for a[1] :
Enter element for a[2] :
```

if the user gives the **input** as:

```
Enter element for a[0] : 89
Enter element for a[1] : 33
Enter element for a[2] : 56
```

Next, the program should print the message on the console as:

```
Enter key element :
```

if the user gives the **input** as:

```
Enter key element : 56
```

then the program should **print** the result as:

```
After sorting the elements in the array are
Value of a[0] = 33
Value of a[1] = 56
Value of a[2] = 89
The key element 56 is found at the position 1
```

Similarly if the key element is given as **25** for the above one dimensional array elements then the program should print the output as "**The Key element 25 is not found in the array**".

Note: Do use the **printf()** function with a **newline** character (`\n`) at the end.

Source Code:

Program510.c

```
#include<stdio.h>
void main() {
   int a[10], n, i, j, key, temp, low, high, mid, flag;
   printf("Enter value of n : ");
   scanf("%d", &n);
   for (i=0; i<n; i++) {
      printf("Enter element for a[%d] : ", i);
      scanf("%d", &a[i]);
   }
```

```
   printf("Enter key element : ");
   scanf("%d", &key);
   for (i=0; i<(n-1); i++) {
      for (j=0; j<(n-i-1); j++){
         if (a[j]>a[j+1]) {
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
         }
      }
   }
   printf("After sorting the elements in the array are\n");
   for(i=0; i<n; i++) {
      printf("Value of a[%d] = %d\n", i, a[i]);
   }
   low =0; high = n-1;
   while(low<=high) {
      mid = (high + low) / 2;
      if(a[mid]==key) {
         flag=1;
         break;
      }
      else if(key>a[mid]) {
         low = mid + 1;
      }
      else {
         high = mid - 1;
      }
   }
   if(flag==1) {
      printf("The key element %d is found at the position %d\n", key, mid);
   }
   else {
      printf("The Key element %d is not found in the array\n", key);
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n : 5
Enter element for a[0] : 4
Enter element for a[1] : 8
Enter element for a[2] : 6
Enter element for a[3] : 2
Enter element for a[4] : 1
Enter key element : 8
After sorting the elements in the array are
Value of a[0] = 1
Value of a[1] = 2
Value of a[2] = 4
Value of a[3] = 6
Value of a[4] = 8

Test Case - 1
The key element 8 is found at the position 4

Test Case - 2
User Output
Enter value of n : 3
Enter element for a[0] : 5
Enter element for a[1] : 8
Enter element for a[2] : 3
Enter key element : 7
After sorting the elements in the array are
Value of a[0] = 3
Value of a[1] = 5
Value of a[2] = 8
The Key element 7 is not found in the array

S.No: 8	Exp. Name: ***Write a Program to Search an element using Binary Search and Recursion***	Date:

Aim:

Write a program to `search` the given element from a list of elements with `binary search` technique using **recursion**.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 5
```

Next, the program should print the following messages one by one on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 33 55 22 44 11
```

then the program should **print** the result as:

```
After sorting the elements are : 11 22 33 44 55
```

Next, the program should print the message on the console as:

```
Enter key element :
```

if the user gives the **input** as:

```
Enter key element : 11
```

then the program should **print** the result as:

```
The given key element 11 is found at position : 0
```

Similarly, if the key element is given as `18` for the above example then the program should print the output as:

```
The given key element 18 is not found
```

Note: Write the functions **read()**, **bubbleSort()**, **display()** and **binarySearch()** in `Program912a.c`

Source Code:

Program912.c

```
#include <stdio.h>
#include "Program912a.c"
void main() {
   int a[20], n, key, flag;
   printf("Enter value of n : ");
   scanf("%d", &n);
   read(a, n);
   bubbleSort(a, n);
   printf("After sorting the elements are : ");
   display(a, n);
   printf("Enter key element : ");
```

```
   scanf("%d", &key);
   flag = binarySearch(a, 0, n - 1, key);
   if (flag == -1) {
      printf("The given key element %d is not found\n", key);
   } else {
      printf("The given key element %d is found at position : %d\n", key, flag);
   }
}
```

Program912a.c

```
void read(int a[], int n) {
   int i;
   printf("Enter %d elements : ", n);
   for(i=0; i<n; i++) {
      scanf("%d", &a[i]);
   }
}
void bubbleSort(int a[], int n) {
   int i, j, temp;
   for (i=0; i<(n-1); i++){
      for(j=0; j<(n-i-1); j++) {
         if (a[j]>a[j+1]) {
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
         }
      }
   }
}
void display(int a[], int n) {
   int i;
   for(i=0; i<n; i++) {
      printf("%d ", a[i]);
   }
   printf("\n");
}
int binarySearch(int a[], int low, int high, int key) {
   int mid;
   if(low>high) {
      return -1;
   }
   mid = (low+high)/2;
   if(a[mid]==key) {
      return mid;
   }
   else if (key>a[mid]) {
      return binarySearch(a, mid+1, high, key);
   }
   else {
      return binarySearch(a, low, mid-1, key);
   }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n : 5
Enter 5 elements : 33 55 22 44 11
After sorting the elements are : 11 22 33 44 55 11
Enter key element : 11
The given key element 11 is found at position : 0

Test Case - 2
User Output
Enter value of n : 4
Enter 4 elements : 23 67 45 18
After sorting the elements are : 18 23 45 67 24
Enter key element : 24
The given key element 24 is not found

Test Case - 3
User Output
Enter value of n : 6
Enter 6 elements : 10 20 18 9 11 15
After sorting the elements are : 9 10 11 15 18 20 18
Enter key element : 18
The given key element 18 is found at position : 4

S.No: 9	Exp. Name: ***Write a C program to Sort given elements using Quick sort***	Date:

Aim:

Write a program to `sort` (`Ascending order`) the given elements using `quick sort` technique.

Note: Pick the first element as pivot. You will not be awarded marks if you do not follow this instruction.

At the time of execution, the program should print the message on the console as:

```
Enter array size :
```

For example, if the user gives the **input** as:

```
Enter array size : 5
```

Next, the program should print the following message on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 34 67 12 45 22
```

then the program should **print** the result as:

```
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

QuickSortMain.c

```
#include <stdio.h>
#include "QuickSortFunctions.c"
void main() {
   int arr[15], i, n;
   printf("Enter array size : ");
   scanf("%d", &n);
   printf("Enter %d elements : ", n);
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
   printf("Before sorting the elements are : ");
   display(arr, n);
   quickSort(arr, 0, n - 1);
   printf("After sorting the elements are : ");
   display(arr, n);
}
```

QuickSortFunctions.c

```
void display(int arr[15], int n) {
   int i;
   for(i=0; i<n; i++) {
      printf("%d ", arr[i]);
```

```
   }
   printf("\n");
}
int partition(int arr[15], int lb, int ub) {
   int p=arr[lb];
   int up=lb, down=ub, temp;
   do{
      do{
         up++;
      }while(arr[up]<p);
      do{
         down--;
      }while(arr[down]>p);
      if(up<down){
         temp = arr[up];
         arr[up]=arr[down];
         arr[down]=temp;
      }
   }while(up<down);
   arr[lb]=arr[down];
   arr[down]=p;
   return down;
}
void quickSort(int arr[15], int low, int high) {
   int p;
   if(low<high){
      p=partition(arr, low, high + 1);
      quickSort(arr, low, p - 1);
      quickSort(arr, p + 1, high);
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size : 5
Enter 5 elements : 34 67 12 45 22
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67

Test Case - 2
User Output
Enter array size : 8
Enter 8 elements : 77 55 22 44 99 33 11 66
Before sorting the elements are : 77 55 22 44 99 33 11 66
After sorting the elements are : 11 22 33 44 55 66 77 99

Test Case - 3
User Output
Enter array size : 5

Test Case - 3
Enter 5 elements : -32 -45 -67 -46 -14
Before sorting the elements are : -32 -45 -67 -46 -14
After sorting the elements are : -67 -46 -45 -32 -14

S.No: 10	Exp. Name: ***Write a C program to Sort given elements using Merge sort***	Date:

Aim:

Write a program to `sort` (`Ascending order`) the given elements using `merge sort` technique.

At the time of execution, the program should print the message on the console as:

```
Enter array size :
```

For example, if the user gives the **input** as:

```
Enter array size : 5
```

Next, the program should print the following message on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 34 67 12 45 22
```

then the program should **print** the result as:

```
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

MergeSortMain.c

```
#include <stdio.h>
#include "MergeSortFunctions.c"
void main() {
   int arr[15], i, n;
   printf("Enter array size : ");
   scanf("%d", &n);
   printf("Enter %d elements : ", n);
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
   printf("Before sorting the elements are : ");
   display(arr, n);
   splitAndMerge(arr, 0, n - 1);
   printf("After sorting the elements are : ");
   display(arr, n);
}
```

MergeSortFunctions.c

```
void display(int arr[15], int n) {
   int i;
   for(i=0; i<n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
```

```
}
void merge(int arr[15], int low, int mid, int high) {
   int i=low, j=mid + 1, index=low, temp[50], k;
   while((i<=mid) && (j<=high)) {
      if (arr[i]<arr[j]) {
         temp[index]=arr[i];
         i++;
      }
      else {
         temp[index]=arr[j];
         j++;
      }
      index++;
   }
   if(i>mid) {
      while(j<=high) {
         temp[index]=arr[j];
         index++;
         j++;
      }
   }
   else {
      while (i<=mid) {
         temp[index] = arr[i];
         index++;
         i++;
      }
   }
   for(k=low; k<index; k++) {
      arr[k] = temp[k];
   }
}
void splitAndMerge(int arr[15], int low, int high) {
   int mid;
   if(low<high) {
      mid = (low+high)/2;
      splitAndMerge(arr, low, mid);
      splitAndMerge(arr, mid + 1, high);
      merge(arr, low, mid, high);
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size : 5
Enter 5 elements : 34 67 12 45 22
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67

Test Case - 2
User Output

Test Case - 2
Enter array size : 8
Enter 8 elements : 77 55 22 44 99 33 11 66
Before sorting the elements are : 77 55 22 44 99 33 11 66
After sorting the elements are : 11 22 33 44 55 66 77 99

Test Case - 3
User Output
Enter array size : 5
Enter 5 elements : -32 -45 -67 -46 -14
Before sorting the elements are : -32 -45 -67 -46 -14
After sorting the elements are : -67 -46 -45 -32 -14

S.No: 11	Exp. Name: ***Write a C program to Create a Singly Linked List***	Date:

Aim:

The `addNodes()` function creates a new list and adds elements to the list until delimiter `-1` is occurred.

The algorithm for `addNodes(NODE first, int x)` is as follows:

```
Step-1: Allocate memory to the node temp.
Step-2: Store an integer value into data field of node temp.
Step-3: If the first is referenced to NULL then assign temp to first node,
        otherwise traverse the list up to the last node (meaning the next field of a
 node contains Null)
        and then assign the temp node to next field of the lastNode.
Step-4: Finally return the first node.
```

The `traverseList()` function traverses and prints all the elements of the list.

The algorithm for `traverseList(NODE first)` is as follows:

```
Step-1: Assign the address contained in first node to temp node.
Step-2: Print the data field of the temp.
Step-3: Move to the next node by placing the address of the next node into the temp
 node.
Step-4: Repeat Stet-2 and Step-3 until temp is NULL.
Step-5: Finally print "NULL" at the end of the list.
```

Fill in the missing code in the below functions `addNodes(NODE first, int x)` and `traverseList(NODE first)` in the file `CreateAndAddNodes.c`.

Source Code:

SingleLL1.c

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

#include "CreateAndAddNodes.c"

void main() {
   NODE first = NULL;
   int x;
   printf("Enter elements up to -1 : ");
   scanf("%d", &x);
   while (x != -1) {
      first = addNodes(first, x);
      scanf("%d", &x);
   }
   if (first == NULL) {
      printf("Single Linked List is empty\n");
   } else {
      printf("The elements in SLL are : ");
      traverseList(first);
   }
}
```

CreateAndAddNodes.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNode() {
   NODE temp;
   temp = (NODE)malloc(sizeof(NODE));
   temp->next = NULL;
   return temp;
}

NODE addNodes(NODE first, int x) {
   NODE temp;
   temp = createNode();
   temp->data = x;
   if (first==NULL) {
      first = temp;
   }
   else {
      NODE lastNode = first;
      while (lastNode->next != NULL){
         lastNode = lastNode->next;
      }
      lastNode->next = temp;
   }
   return first;
}

void traverseList(NODE first) {
   if (first==NULL){
      printf("List is Empty.");
   }
   else {
      NODE temp = first;
      while (temp != NULL) {
         printf("%d --> ", temp->data);
         temp = temp->next;
      }
      printf("NULL");
   }
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter elements up to -1 : 9 18 27 36 45 -1
The elements in SLL are : 9 --> 18 --> 27 --> 36 --> 45 --> NULL

Test Case - 2

Test Case - 2
User Output
Enter elements up to -1 : 12 14 19 23 -1
The elements in SLL are : 12 --> 14 --> 19 --> 23 --> NULL

S.No: 12	Exp. Name: ***Write a C program to Insert an element at Begin and Count number of Nodes in Singly Linked List***	Date:

Aim:

The `insertAtBegin(NODE first, int x)` function inserts a new node at the beginning of the singly linked list.

The algorithm for `insertAtBegin(NODE first, int x)` is as follows:

```
Step-1: Allocate memory to the node temp.
Step-2: Store an integer value into data field of node temp.
Step-3: Assign the address contained in the first node to the next field of temp.
Step-4: Now treat the temp node as first node.
Step-5: Finally return the first node.
```

The `count(NODE first)` function counts the number of nodes linked in a singly linked list.

The algorithm for `count(node first)` is as follows:

```
Step-1: Assign the address contained in first node to temp node.
Step-2: Initialize a variable sum to 0 (zero).
Step-3: Repeat Step-4 and Step-5 until temp reaches the NULL.
Step-4: Increment the sum by 1.
Step-5: Move to the next node by placing the address of the next node in temp node.
Step-6: Finally return sum.
```

Source Code:

SingleLL2.c

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

#include "InsAtBeginAndCount.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBegin(first, x);
               break;
         case 2:  printf("The number of nodes in a SLL are : %d\n", count(first));
               break;
         case 3:  if (first == NULL) {
                  printf("Single Linked List is empty\n");
               } else {
                  printf("The elements in SLL are : ");
                  traverseList(first);
               }
               break;
         case 4:  exit(0);
```

```
      }
   }
}
```


InsAtBeginAndCount.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNode() {
   NODE temp;
   temp = (NODE)malloc(sizeof(NODE));
   temp -> next = NULL;
   return temp;
}

NODE insertAtBegin(NODE first, int x) {
   NODE temp;
   temp = createNode();
   temp -> data = x;
   temp -> next = first;
   first = temp;
   return first;
}

int count(NODE first) {
   NODE temp = first;
   int sum = 0;
   while(temp!=NULL) {
      sum++;
      temp = temp -> next;
   }
   return sum;
}

void traverseList(NODE first) {
   NODE temp = first;
   while (temp != NULL) {
      printf("%d --> ",temp -> data);
      temp = temp -> next;
   }
   printf("NULL\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1

Test Case - 1
Enter an element : 10
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 20
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 30
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option : 2
The number of nodes in a SLL are : 3 3
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 30 --> 20 --> 10 --> NULL 1
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 40
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option : 2
The number of nodes in a SLL are : 4 3
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 40 --> 30 --> 20 --> 10 --> NULL 4
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 4
Enter your option : 4

Test Case - 2
User Output
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 99
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 89
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 89 --> 99 --> NULL 2
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option : 2
The number of nodes in a SLL are : 2 4
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 13	Exp. Name: ***Write a C program to Insert an element at End in Singly Linked List***	Date:

Aim:

Source Code:

SingleLL3.c

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

#include "InsAtEnding.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At End 2.Traverse the List 3.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEnd(first, x);
               break;
         case 2:  if (first == NULL) {
                     printf("Single Linked List is empty\n");
               } else {
                     printf("The elements in SLL are : ");
                     traverseList(first);
               }
               break;
         case 3:  exit(0);
      }
   }
}
```

InsAtEnding.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNode() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtEnd(NODE first, int x) {
   NODE temp;
   temp = createNode();
```

```
   temp -> data = x;
   if(first==NULL) {
      first = temp;
   }
   else {
      NODE lastNode = first;
      while (lastNode -> next != NULL) {
         lastNode = lastNode -> next;
      }
      lastNode -> next = temp;
   }
   return first;
}

void traverseList(NODE first){
   if(first==NULL) {
      printf("List is Enpty.\n");
   }
   else {
      NODE temp = first;
      while(temp!=NULL) {
         printf("%d --> ", temp -> data);
         temp = temp -> next;
      }
      printf("NULL");
   }
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 10
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 20
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 30
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in SLL are : 10 --> 20 --> 30 --> NULL 3
1.Insert At End 2.Traverse the List 3.Exit 3
Enter your option : 3

Test Case - 2
User Output
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2

Test Case - 2
Single Linked List is empty 1
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 99
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 29
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 59
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in SLL are : 99 --> 29 --> 59 --> NULL 3
1.Insert At End 2.Traverse the List 3.Exit 3
Enter your option : 3

S.No: 14	Exp. Name: ***Write a C program to Insert an element at the Specified Position in Singly Linked List***	Date:

Aim:

Source Code:

SingleLL4.c

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

#include "InsAtPosition.c"

void main() {
   NODE first = NULL;
   int x, pos, op;
   while(1) {
      printf("1.Insert At specified position 2.Traverse the List 3.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter a position : ");
               scanf("%d", &pos);
               if (pos <= 0) {
                  printf("No such position in SLL so insertion is not possible\n");
               } else {
                  printf("Enter an element : ");
                  scanf("%d", &x);
                  first = insertAtPosition(first, pos, x);
               }
               break;
         case 2:  if (first == NULL) {
                  printf("Single Linked List is empty\n");
               } else {
                  printf("The elements in SLL are : ");
                  traverseList(first);
               }
               break;
         case 3:  exit(0);
      }
   }
}
```

InsAtPosition.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNode() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
```

```
   temp -> next = NULL;
   return temp;
}

NODE insertAtPosition(NODE first, int pos, int x) {
   if(pos<=0) {
      printf("No such position in SLL so insertion is not possible\n");
      return first;
   }
   int i;
   NODE last = first, prevPos = first, temp;
   for (i=1; i<pos; i++) {
      if(last==NULL) {
         printf("No such position in SLL so insertion is not possible\n");
         return first;
      }
      prevPos = last;
      last = last->next;
   }
   temp = createNode();
   temp->data = x;
   if(pos==1) {
      temp->next = first;
      first = temp;
   }
   else{
      temp->next = prevPos->next;
      prevPos->next = temp;
   }
   return first;
}

void traverseList(NODE first) {
   if(first==NULL) {
      printf("List is Empty.\n");
   }
   else {
      NODE temp = first;
      while(temp!=NULL) {
         printf("%d --> ", temp->data);
         temp = temp->next;
      }
      printf("NULL");
   }
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
Single Linked List is empty 1

Test Case - 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 10
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 2
Enter an element : 30
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in SLL are : 10 --> 30 --> NULL 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 2
Enter an element : 20
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in SLL are : 10 --> 20 --> 30 --> NULL 3
1.Insert At specified position 2.Traverse the List 3.Exit 3
Enter your option : 3

Test Case - 2
User Output
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 3
Enter an element : 12
No such position in SLL so insertion is not possible 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 5
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 2
Enter an element : 7
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in SLL are : 5 --> 7 --> NULL 3
1.Insert At specified position 2.Traverse the List 3.Exit 3
Enter your option : 3

S.No: 15	Exp. Name: ***Write a C program to Delete an element at Begin from Singly Linked List***	Date:

Aim:

Source Code:

SingleLL5.c

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

#include "DelAtBegin.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBegin(first, x);
               break;
         case 2:  if (first == NULL) {
                     printf("Single Linked List is empty so deletion is not possible\n");
               } else {
                     first = deleteAtBegin(first);
               }
               break;
         case 3:  if (first == NULL) {
                     printf("Single Linked List is empty\n");
               } else {
                     printf("The elements in SLL are : ");
                     traverseList(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtBegin.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNode() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
```

```
   return temp;
}

NODE insertAtBegin(NODE first, int x) {
   NODE temp;
   temp = createNode();
   temp -> data = x;
   temp -> next = first;
   first = temp;
   return first;
}

NODE deleteAtBegin(NODE first) {
   if (first==NULL) {
      printf("List is empty, deletion is not possible\n");
   }
   else {
      NODE temp = first;
      first = first -> next;
      printf("The deleted element from SLL : %d\n", temp -> data);
      free(temp);
      return first;
   }
}

void traverseList(NODE first) {
   NODE temp = first;
   while (temp != NULL) {
      printf("%d --> ",temp -> data);
      temp = temp -> next;
   }
   printf("NULL\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 10
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 10 --> NULL 2
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 2
Enter your option : 2
The deleted element from SLL : 10 3
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 3
Enter your option : 3
Single Linked List is empty 4
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 16	Exp. Name: ***Write a C program to Delete an element at End from Singly Linked List***	Date:

Aim:

Source Code:

SingleLL6.c

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

#include "DelAtEnding.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEnd(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Single Linked List is empty so deletion is not possible\n");
               } else {
                  first = deleteAtEnd(first);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Single Linked List is empty\n");
               } else {
                  printf("The elements in SLL are : ");
                  traverseList(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtEnding.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNode() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
```

```
   return temp;
}

NODE insertAtEnd(NODE first, int x) {
   NODE temp;
   temp = createNode();
   temp -> data = x;
   if(first==NULL) {
      first = temp;
   }
   else {
      NODE last = first;
      while(last -> next != NULL) {
         last = last -> next;
      }
      last -> next = temp;
   }
   return first;
}

NODE deleteAtEnd(NODE first) {
   if(first == NULL) {
      printf("List is Empty, deletion is not possible\n");
   }
   else {
      NODE prev, last = first;
      if(last -> next == NULL) {
         first = first -> next;
      }
      else {
         while(last -> next != NULL) {
            prev = last;
            last = last -> next;
         }
         prev -> next = NULL;
      }
      printf("The deleted item from SLL : %d\n", last -> data);
      free (last);
      return first;
   }
}

void traverseList(NODE first) {
   NODE temp = first;
   while (temp != NULL) {
      printf("%d --> ",temp -> data);
      temp = temp -> next;
   }
   printf("NULL\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1

Test Case - 1
User Output
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 55
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 66
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 2
Enter your option : 2
The deleted item from SLL : 66 2
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 2
Enter your option : 2
The deleted item from SLL : 55 3
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 3
Enter your option : 3
Single Linked List is empty 4
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 17	Exp. Name: ***Write a C program to Delete an element at the Specified Position from Singly Linked List***	Date:

Aim:

Source Code:

SingleLL7.c

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

#include "DelAtPosition.c"

void main() {
   NODE first = NULL;
   int x, op, pos;
   while(1) {
      printf("1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEnd(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Single Linked List is empty so deletion is not possible\n");
               } else {
                  printf("Enter position : ");
                  scanf("%d", &pos);
                  first = deleteAtPosition(first, pos);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Single Linked List is empty\n");
               } else {
                  printf("The elements in SLL are : ");
                  traverseList(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtPosition.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNode() {
```

```
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtEnd(NODE first, int x) {
   NODE temp;
   temp = createNode();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
   }
   else {
      NODE last = first;
      while(last -> next != NULL) {
         last = last -> next;
      }
      last -> next = temp;
   }
   return first;
}

NODE deleteAtPosition(NODE first, int pos) {
   if(first==NULL) {
      printf("List is Empty, deletion is not possible\n");
   }
   else{
      NODE prev, last = first;
      if(pos==1) {
         first = first->next;
      }
      else {
         if(pos<=0) {
            printf("No such position in SLL so deletion is not possible\n");
            return first;
         }
         else {
            int i;
            for (i=1; i<pos; i++) {
               if (last==NULL) {
                  printf("No such position in SLL so deletion is not possible\n");
                  return first;
               }
               prev = last;
               last = last->next;
            }
            if (last==NULL) {
               printf("No such position in SLL so deletion is not possible\n");
               return first;
            }
            prev->next = last->next;
         }
      }
      printf("The deleted element from SLL : %d\n", last->data);
      free (last);
      return first;
   }
```

```
}

void traverseList(NODE first) {
   NODE temp = first;
   while (temp != NULL) {
      printf("%d --> ",temp -> data);
      temp = temp -> next;
   }
   printf("NULL\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 11
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 22
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 33
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 3
The deleted element from SLL : 33 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 4
No such position in SLL so deletion is not possible 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 11 --> 22 --> NULL 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 1
The deleted element from SLL : 11 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 22 --> NULL 4
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 4
Enter your option : 4

Test Case - 2
User Output
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 10
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2

Test Case - 2
Enter your option : 2
Enter position : 2
No such position in SLL so deletion is not possible 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 10 --> NULL 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 1
The deleted element from SLL : 10 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
Single Linked List is empty 4
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 18	Exp. Name: ***Write a C program to Search the Position of a given element in SLL***	Date:

Aim:

Source Code:

SingleLL8.c

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

#include "SearchPositionOfEle.c"

void main() {
   NODE first = NULL;
   int x, pos, op;
   while(1) {
      printf("1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit
\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBegin(first, x);
               break;
         case 2:  printf("Enter search element : ");
               scanf("%d", &x);
               pos = searchPosOfEle(first, x);
               if (pos == 0) {
                  printf("The given element %d is not found in the given SLL\n", x);
               } else {
                  printf("The given element %d is found at position : %d\n", x, pos);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Single Linked List is empty\n");
               } else {
                  printf("The elements in SLL are : ");
                  traverseList(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

SearchPositionOfEle.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;
```

```
NODE createNode() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtBegin(NODE first, int x) {
   NODE temp;
   temp = createNode();
   temp -> data = x;
   temp -> next = first;
   first = temp;
   return first;
}

int searchPosOfEle(NODE first, int key) {
   if (first==NULL) {
      return 0;
   }
   else {
      NODE current = first;
      int count=0;
      while(current->data!=key) {
         if(current->next==NULL) {
            return 0;
         }
         else {
            count++;
            current = current->next;
         }
      }
      return (count+1);
   }
}

void traverseList(NODE first) {
   NODE temp = first;
   while (temp != NULL) {
      printf("%d --> ",temp -> data);
      temp = temp -> next;
   }
   printf("NULL\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 10
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1

Test Case - 1
Enter an element : 20
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 30
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter search element : 10
The given element 10 is found at position : 3 3
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 30 --> 20 --> 10 --> NULL 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter search element : 20
The given element 20 is found at position : 2 4
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 4
Enter your option : 4

Test Case - 2
User Output
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 10
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 20
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in SLL are : 20 --> 10 --> NULL 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter search element : 10
The given element 10 is found at position : 2 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter search element : 24
The given element 24 is not found in the given SLL 4
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 19	Exp. Name: ***Write Code for addNodesInCLL() and traverseListInCLL() functions in CLL***	Date:

Aim:

In the below `circular linked list` program we have two files, one file contains the **main program** and the other file contains the **functions**, to be implemented by the **user**.

Here the user has to implement the code for two functions `addNodesInCLL()` and `traverseListInCLL()`.

The `addNodesInCLL()` function creates a new list and adds elements to the list until delimiter `-1` is occurred.

The `traverseListInCLL()` function traverses and prints all the elements of the list.

Fill in the missing code in the below functions `addNodesInCLL(NODE first, int x)` and `traverseListInCLL(NODE first)` in the file `CreateAndAddNodesInCLL.c`.

Source Code:

CircularLL1.c

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

#include "CreateAndAddNodesInCLL.c"

void main() {
   NODE first = NULL;
   int x;
   printf("Enter elements up to -1 : ");
   scanf("%d", &x);
   while (x != -1) {
      first = addNodesInCLL(first, x);
      scanf("%d", &x);
   }
   if (first == NULL) {
      printf("Circular Linked List is empty\n");
   } else {
      printf("The elements in CLL are : ");
      traverseListInCLL(first);
   }
}
```

CreateAndAddNodesInCLL.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNodeInCLL() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
```

```
   return temp;
}

NODE addNodesInCLL(NODE first, int x) {
   NODE temp, last = first;
   temp = createNodeInCLL();
   temp -> data = x;
   if(first == NULL) {
      first = temp;
   }
   else{
      while(last->next!=first) {
         last = last->next;
      }
      last->next = temp;
   }
   temp->next = first;
   return first;
}

void traverseListInCLL(NODE first) {
   NODE temp = first;
   do {
      printf("%d --> ", temp->data);
      temp = temp->next;
   } while(temp != first);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter elements up to -1 : 10 20 30 40 -1
The elements in CLL are : 10 --> 20 --> 30 --> 40 -->

Test Case - 2
User Output
Enter elements up to -1 : 1 2 3 4 -1
The elements in CLL are : 1 --> 2 --> 3 --> 4 -->

Test Case - 3
User Output
Enter elements up to -1 : 11 22 33 -1
The elements in CLL are : 11 --> 22 --> 33 -->

Test Case - 4
User Output
Enter elements up to -1 : -1

Test Case - 4
Circular Linked List is empty

S.No: 20	Exp. Name: ***Write Code for insertAtBeginInCLL() and countInCLL() functions in CLL***	Date:

Aim:

Fill in the missing code in the below functions `insertAtBeginInCLL(NODE first, int x)` and `countInCLL(NODE first)` in the file `InsAtBeginAndCountInCLL.c`.

The `insertAtBeginInCLL(NODE first, int x)` function inserts a new node at the beginning of the circular linked list.

The `countInCLL(NODE first)` function counts the number of nodes linked in a circular linked list.

Source Code:

CircularLL2.c

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

#include "InsAtBeginAndCountInCLL.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBeginInCLL(first, x);
               break;
         case 2:  printf("The number of nodes in a CLL are : %d\n", countInCLL(first));
               break;
         case 3:  if (first == NULL) {
                  printf("Circular Linked List is empty\n");
               } else {
                  printf("The elements in CLL are : ");
                  traverseListInCLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

InsAtBeginAndCountInCLL.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;
```

```
NODE createNodeInCLL() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtBeginInCLL(NODE first, int x) {
   NODE temp, last = first;
   temp = createNodeInCLL();
   temp->data = x;
   if(first==NULL) {
      first = temp;
      temp->next = first;
   }
   else {
      while(last->next!=first) {
         last = last->next;
      }
      temp->next = first;
      first = temp;
      last->next = first;
   }
   return first;
}

int countInCLL(NODE first) {
   int sum = 0;
   NODE temp=first;
   if(first==NULL) {
      return sum;
   }
   else{
      do{
         sum++;
         temp = temp->next;
      } while(temp!=first);
      return sum;
   }
}

void traverseListInCLL(NODE first) {
   NODE temp = first;
   do {
      printf("%d --> ", temp -> data);
      temp = temp -> next;
   } while (temp != first);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output

Test Case - 1
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 11
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 22
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option : 2
The number of nodes in a CLL are : 2 3
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in CLL are : 22 --> 11 --> 1
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 33
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 44
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in CLL are : 44 --> 33 --> 22 --> 11 --> 2
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option : 2
The number of nodes in a CLL are : 4 4
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 4
Enter your option : 4

Test Case - 2
User Output
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option : 3
Circular Linked List is empty 2
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option : 2
The number of nodes in a CLL are : 0 1
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 99
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option : 2
The number of nodes in a CLL are : 1 3
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in CLL are : 99 --> 4
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 21	Exp. Name: ***Write Code for insertAtEndInCLL() function in CLL***	Date:

Aim:

Fill in the missing code in the below finction `insertAtEndInCLL(NODE first, int x)` in the file `InsAtEndingInCLL.c`, which inserts a new node at the end of circular linked list.

Source Code:

CircularLL3.c

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

#include "InsAtEndingInCLL.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At End 2.Traverse the List 3.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEndInCLL(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Circular Linked List is empty\n");
               } else {
                  printf("The elements in CLL are : ");
                  traverseListInCLL(first);
               }
               break;
         case 3:  exit(0);
      }
   }
}
```

InsAtEndingInCLL.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNodeInCLL() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}
```

```
NODE insertAtEndInCLL(NODE first, int x) {
   NODE temp, lastNode = first;
   temp = createNodeInCLL();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
   } else {
      while (lastNode -> next != first) {
         lastNode = lastNode -> next;
      }
      lastNode -> next = temp;
   }
   temp -> next = first;
   return first;
}

void traverseListInCLL(NODE first) {
   NODE temp = first;
   do {
      printf("%d --> ", temp -> data);
      temp = temp -> next;
   } while (temp != first);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 12
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 13
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 14
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in CLL are : 12 --> 13 --> 14 --> 1
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 15
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in CLL are : 12 --> 13 --> 14 --> 15 --> 3
1.Insert At End 2.Traverse the List 3.Exit 3
Enter your option : 3

Test Case - 2
User Output

Test Case - 2
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2
Circular Linked List is empty 1
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 111
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 112
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in CLL are : 111 --> 112 --> 3
1.Insert At End 2.Traverse the List 3.Exit 3
Enter your option : 3

S.No: 22	Exp. Name: ***Write Code for insertAtPositionInCLL() function in CLL***	Date:

Aim:

Fill in the missing code in the below function `insertAtPositionInCLL(NODE first, int pos, int x)` in the file `InsAtPositionInCLL.c`, which inserts a new node at the specified position of the circular linked list.

Source Code:

CircularLL4.c

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

#include "InsAtPositionInCLL.c"

void main() {
   NODE first = NULL;
   int x, pos, op;
   while(1) {
      printf("1.Insert At specified position 2.Traverse the List 3.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter a position : ");
               scanf("%d", &pos);
               if (pos <= 0) {
                  printf("No such position in CLL so insertion is not possible\n");
               } else {
                  printf("Enter an element : ");
                  scanf("%d", &x);
                  first = insertAtPositionInCLL(first, pos, x);
               }
               break;
         case 2:  if (first == NULL) {
                  printf("Circular Linked List is empty\n");
               } else {
                  printf("The elements in CLL are : ");
                  traverseListInCLL(first);
               }
               break;
         case 3:  exit(0);
      }
   }
}
```

InsAtPositionInCLL.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;

NODE createNodeInCLL() {
```

```
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtPositionInCLL(NODE first, int pos, int x) {
   NODE temp, lastNode = first;
   int i ;
   for (i = 1; i < (pos - 1); i++) {
      if (lastNode -> next == first) {
         printf("No such position in CLL so insertion is not possible\n");
         return first;
      }
      lastNode = lastNode -> next;
   }
   temp = createNodeInCLL();
   temp -> data = x;
   if (pos == 1) {
      if (first == NULL) {
         first = temp;
         temp -> next = first;
      } else {
         while (lastNode -> next != first) {
            lastNode = lastNode -> next;
         }
         temp -> next = first;
         first = temp;
         lastNode -> next = first;
      }
   } else {
      temp -> next = lastNode -> next;
      lastNode -> next = temp;
   }
   return first;
}

void traverseListInCLL(NODE first) {
   NODE temp = first;
   do {
      printf("%d --> ", temp -> data);
      temp = temp -> next;
   } while (temp != first);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 23

Test Case - 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 2
Enter an element : 34
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 2
Enter an element : 45
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in CLL are : 23 --> 45 --> 34 --> 3
1.Insert At specified position 2.Traverse the List 3.Exit 3
Enter your option : 3

S.No: 23	Exp. Name: ***Write Code for deleteAtBeginInCLL() function in CLL***	Date:

Aim:

Fill in the missing code in the below function `deleteAtBeginInCLL(NODE first)`, which deletes the node at the beginning of circular linked list.

Source Code:

CircularLL5.c

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

#include "DelAtBeginInCLL.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBeginInCLL(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Circular Linked List is empty so deletion is not possible
\n");
               } else {
                  first = deleteAtBeginInCLL(first);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Circular Linked List is empty\n");
               } else {
                  printf("The elements in CLL are : ");
                  traverseListInCLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtBeginInCLL.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;
```

```
NODE createNodeInCLL() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtBeginInCLL(NODE first, int x) {
   NODE temp, lastNode = first;
   temp = createNodeInCLL();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
      temp -> next = first;
   } else {
      while (lastNode -> next != first) {
         lastNode = lastNode -> next;
      }
      temp -> next = first;
      first = temp;
      lastNode -> next = first;
   }
   return first;
}

NODE deleteAtBeginInCLL(NODE first) {
   NODE prev = first, lastNode = first;
   if (prev -> next == first) {
      first = NULL;
   } else {
      while (lastNode -> next != first) {
         lastNode = lastNode -> next;
      }
      first = prev -> next;
      lastNode -> next = first;
   }
   printf("The deleted element from CLL : %d\n" , prev -> data);
   free(prev);
   return first;
}

void traverseListInCLL(NODE first) {
   NODE temp = first;
   do {
      printf("%d --> ", temp -> data);
      temp = temp -> next;
   } while (temp != first);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output

Test Case - 1

```
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  11
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  12
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  13
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in CLL are : 13 --> 12 --> 11 -->  2
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 2
Enter your option :  2
The deleted element from CLL : 13 2
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 2
Enter your option :  2
The deleted element from CLL : 12 3
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in CLL are : 11 -->  4
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 4
Enter your option :  4
```

S.No: 24	Exp. Name: ***Write Code for deleteAtEndInCLL() function in CLL***	Date:

Aim:

Fill in the missing code in the below function `deleteAtEndInCLL(NODE first)`, which deletes the node at the end of singly linked list.

Source Code:

CircularLL6.c

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

#include "DelAtEndingInCLL.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEndInCLL(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Circular Linked List is empty so deletion is not possible
\n");
               } else {
                  first = deleteAtEndInCLL(first);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Circular Linked List is empty\n");
               } else {
                  printf("The elements in CLL are : ");
                  traverseListInCLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtEndingInCLL.c

```
struct node {
   int data;
   struct node *next;
};
typedef struct node *NODE;
```

```
NODE createNodeInCLL() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtEndInCLL(NODE first, int x) {
   NODE temp, lastNode = first;
   temp = createNodeInCLL();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
   } else {
      while (lastNode -> next != first) {
         lastNode = lastNode -> next;
      }
      lastNode -> next = temp;
   }
   temp -> next = first;
   return first;
}

NODE deleteAtEndInCLL(NODE first) {
   NODE prev, lastNode = first;
   if (lastNode -> next == first) {
      first = NULL;
   } else {
      while (lastNode -> next != first) {
         prev = lastNode;
         lastNode = lastNode -> next;
      }
      prev -> next = first;
   }
   printf("The deleted item from CLL : %d\n", lastNode -> data);
   free(lastNode);
   return first;
}

void traverseListInCLL(NODE first) {
   NODE temp = first;
   do {
      printf("%d --> ", temp -> data);
      temp = temp -> next;
   } while (temp != first);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 1
Enter your option : 1

Test Case - 1

```
Enter an element :  12
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  15
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  56
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in CLL are : 12 --> 15 --> 56 -->  2
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 2
Enter your option :  2
The deleted item from CLL : 56 2
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 2
Enter your option :  2
The deleted item from CLL : 15 3
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in CLL are : 12 -->  4
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 4
Enter your option :  4
```

S.No: 25	Exp. Name: ***Write Code for deleteAtPositionInCLL() function in CLL***	Date:

Aim:

Fill in the missing code in the below function `deleteAtPositionInCLL(NODE first, int pos)`, which deletes a node at the specified position of circular linked list.

Source Code:

CircularLL7.c

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

#include "DelAtPositionInCLL.c"

void main() {
   NODE first = NULL;
   int x, op, pos;
   while(1) {
      printf("1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEndInCLL(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Circular Linked List is empty so deletion is not possible
\n");
               } else {
                  printf("Enter position : ");
                  scanf("%d", &pos);
                  first = deleteAtPositionInCLL(first, pos);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Circular Linked List is empty\n");
               } else {
                  printf("The elements in CLL are : ");
                  traverseListInCLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtPositionInCLL.c

```
struct node {
   int data;
   struct node *next;
};
```

```
typedef struct node *NODE;

NODE createNodeInCLL() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtEndInCLL(NODE first, int x) {
   NODE temp, lastNode = first;
   temp = createNodeInCLL();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
   } else {
      while (lastNode -> next != first) {
         lastNode = lastNode -> next;
      }
      lastNode -> next = temp;
   }
   temp -> next = first;
   return first;
}

NODE deleteAtPositionInCLL(NODE first, int pos) {
   NODE prev = first, lastNode = first;
   int i;
   if (pos == 1) {
      if (prev -> next == first) {
         first = NULL;
      } else {
         while (lastNode -> next != first) {
            lastNode = lastNode -> next;
         }
         first = prev -> next;
         lastNode -> next = first;
      }
   } else {
      for (i = 1; i < pos; i++) {
         if (prev -> next == first) {
            printf("No such position in CLL so deletion is not possible\n");
            return first;
         }
         lastNode = prev;
         prev = prev -> next;
      }
      lastNode -> next = prev -> next;
   }
   printf("The deleted element from CLL : %d\n", prev -> data);
   free(prev);
   return first;
}

void traverseListInCLL(NODE first) {
   NODE temp = first;
   do {
      printf("%d --> ", temp -> data);
```

```
      temp = temp -> next;
   } while (temp != first);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 22
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 44
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 55
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in CLL are : 22 --> 44 --> 55 --> 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 3
The deleted element from CLL : 55 1
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 66
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in CLL are : 22 --> 44 --> 66 --> 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 1
The deleted element from CLL : 22 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in CLL are : 44 --> 66 --> 4
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 4
Enter your option : 4

Test Case - 2
User Output
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Circular Linked List is empty so deletion is not possible 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
Circular Linked List is empty 1
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1

Test Case - 2

```
Enter an element :  43
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  65
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in CLL are : 43 --> 65 -->  2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option :  2
Enter position :  2
The deleted element from CLL : 65 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option :  2
Enter position :  3
No such position in CLL so deletion is not possible 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option :  2
Enter position :  1
The deleted element from CLL : 43 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option :  3
Circular Linked List is empty 4
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 4
Enter your option :  4
```

S.No: 26	Exp. Name: ***Write Code for searchPosOfEleInCLL() function in CLL***	Date:

Aim:

Fill in the missing code in `searchPosOfEleInCLL(NODE first, int key)` function which **searches** a given element key in the list of elements and prints the **position** of that element if found.

Source Code:

CircularLL8.c

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

#include "SearchPositionOfEleInCLL.c"

void main() {
   NODE first = NULL;
   int x, pos, op;
   while(1) {
      printf("1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit
\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBeginInCLL(first, x);
               break;
         case 2:  printf("Enter search element : ");
               scanf("%d", &x);
               pos = searchPosOfEleInCLL(first, x);
               if (pos == 0) {
                  printf("The given element %d is not found in the given CLL\n", x);
               } else {
                  printf("The given element %d is found at position : %d\n", x, pos);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Circular Linked List is empty\n");
               } else {
                  printf("The elements in CLL are : ");
                  traverseListInCLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

SearchPositionOfEleInCLL.c

```
struct node {
   int data;
   struct node *next;
```

```
};
typedef struct node *NODE;

NODE createNodeInCLL() {
   NODE temp;
   temp = (NODE) malloc(sizeof(struct node));
   temp -> next = NULL;
   return temp;
}

NODE insertAtBeginInCLL(NODE first, int x) {
   NODE temp, lastNode = first;
   temp = createNodeInCLL();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
      temp -> next = first;
   } else {
      while (lastNode -> next != first) {
         lastNode = lastNode -> next;
      }
      temp -> next = first;
      first = temp;
      lastNode -> next = first;
   }
   return first;
}

int searchPosOfEleInCLL(NODE first, int key) {
   NODE currentNode = first, q = first;
   int count = 0;
   if (currentNode == NULL) {
      return count;
   } else {
      do {
         count++;
         q = currentNode;
         if (currentNode -> next == first && currentNode -> data != key) {
            return 0;
         }
         currentNode = currentNode -> next;
      } while (q -> next != first && q -> data != key);
         return count;
   }
}

void traverseListInCLL(NODE first) {
   NODE temp = first;
   do {
      printf("%d --> ", temp -> data);
      temp = temp -> next;
   } while (temp != first);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 10
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 20
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 30
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 40
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in CLL are : 40 --> 30 --> 20 --> 10 --> 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter search element : 2
The given element 2 is not found in the given CLL 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter search element : 30
The given element 30 is found at position : 2 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter search element : 10
The given element 10 is found at position : 4 4
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 27	Exp. Name: ***Write a Program to Insert an element at Begin and Count number of Nodes in Doubly Linked List***	Date:

Aim:

Fill in the missing code in the `insertAtBeginInDLL()` and `countInDLL()` methods.

The `insertAtBeginInDLL(NODE first, int x)` function inserts a new integer at the beginning of the double linked list.

The `countInDLL(NODE first)` function counts the number of nodes linked in a double linked list.

Source Code:

DoubleLL2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "InsAtBeginAndCountInDLL.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBeginInDLL(first, x);
               break;
         case 2:  printf("The number of nodes in a DLL are : %d\n", countInDLL(first));
               break;
         case 3:  if (first == NULL) {
                  printf("Double Linked List is empty\n");
               } else {
                  printf("The elements in DLL are : ");
                  traverseListInDLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

InsAtBeginAndCountInDLL.c

```
struct node {
   int data;
   struct node *prev;
   struct node *next;
};
typedef struct node * NODE;
```

```
NODE createNodeInDLL() {
   NODE temp;
   temp = (NODE)malloc(sizeof(struct node));
   temp -> prev = NULL;
   temp -> next = NULL;
   return temp;
}

void traverseListInDLL(NODE first) {
   NODE lastNode = first;
   while (lastNode != NULL) {
      printf("%d <--> ", lastNode -> data);
      lastNode = lastNode -> next;
   }
   printf("NULL\n");
}

int countInDLL(NODE first) {
   NODE lastNode = first;
   int sum = 0;
   while (lastNode != NULL) {
      sum++;
      lastNode = lastNode -> next;
   }
   return sum;
}

NODE insertAtBeginInDLL(NODE first, int x) {
   NODE temp;
   temp = createNodeInDLL();
   temp -> data = x;
   if (first != NULL) {
      temp -> next = first;
      first -> prev = temp;
   }
   first = temp;
   return first;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 15
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 16
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 17
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option : 3

Test Case - 1

```
The elements in DLL are : 17 <--> 16 <--> 15 <--> NULL 2
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option :  2
The number of nodes in a DLL are : 3 1
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  18
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  19
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in DLL are : 19 <--> 18 <--> 17 <--> 16 <--> 15 <--> NULL 2
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 2
Enter your option :  2
The number of nodes in a DLL are : 5 4
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit 4
Enter your option :  4
```

S.No: 28	Exp. Name: ***Write a Program to Insert an element at End and Traverse the Nodes in Doubly Linked List***	Date:

Aim:

Fill in the missing code in the `insertAtEndInDLL(NODE first, int x)` and `traverseListInDLL(NODE first)` methods.

The `insertAtEndInDLL()` function adds an element to the end of the list.

The `traverseListInDLL()` function traverses and prints all the elements of the list.

Source Code:

DoubleLL1.c

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

#include "InsertEndAndTraverseInDLL.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At End 2.Traverse the List 3.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEndInDLL(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Double Linked List is empty\n");
               } else {
                  printf("The elements in DLL are : ");
                  traverseListInDLL(first);
               }
               break;
         case 3:  exit(0);
      }
   }
}
```

InsertEndAndTraverseInDLL.c

```
struct node {
   int data;
   struct node *prev;
   struct node *next;
};
typedef struct node * NODE;

NODE createNodeInDLL() {
```

```
   NODE temp;
   temp = (NODE)malloc(sizeof(struct node));
   temp->prev = NULL;
   temp->next = NULL;
   return temp;
}

NODE insertAtEndInDLL(NODE first, int x) {
   NODE temp, lastNode = first;
   temp = createNodeInDLL();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
   } else {
      while (lastNode -> next != NULL) {
         lastNode = lastNode -> next;
      }
      lastNode -> next = temp;
      temp -> prev = lastNode;
   }
   return first;
}

void traverseListInDLL(NODE first) {
   NODE lastNode = first;
   while (lastNode != NULL) {
      printf("%d <--> ", lastNode -> data);
      lastNode = lastNode -> next;
   }
   printf("NULL\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 14
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 67
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 56
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in DLL are : 14 <--> 67 <--> 56 <--> NULL 1
1.Insert At End 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter an element : 34
1.Insert At End 2.Traverse the List 3.Exit 2
Enter your option : 2

Test Case - 1
The elements in DLL are : 14 <--> 67 <--> 56 <--> 34 <--> NULL 3
1.Insert At End 2.Traverse the List 3.Exit 3
Enter your option : 3

S.No: 29	Exp. Name: ***Write a Program to Insert an element at Specified Position in Doubly Linked List***	Date:

Aim:

Fill in the missing code in `insertAtPositionInDLL(NODE first, int pos, int x)` function which inserts a new integer at a particular position in the double linked list.

Source Code:

DoubleLL4.c

```
#include <stdio.h>
#include <stdlib.h>
#include "InsAtPositionInDLL.c"

void main() {
   NODE first = NULL;
   int x, pos, op;
   while(1) {
      printf("1.Insert At specified position 2.Traverse the List 3.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter a position : ");
               scanf("%d", &pos);
               printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtPositionInDLL(first, pos, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Double Linked List is empty\n");
               } else {
                  printf("The elements in DLL are : ");
                  traverseListInDLL(first);
               }
               break;
         case 3:  exit(0);
      }
   }
}
```

InsAtPositionInDLL.c

```
struct node {
   int data;
   struct node *prev;
   struct node *next;
};
typedef struct node * NODE;

NODE createNodeInDLL() {
   NODE temp;
   temp = (NODE)malloc(sizeof(struct node));
   temp->prev = NULL;
```

```
   temp->next = NULL;
   return temp;
}

NODE insertAtPositionInDLL(NODE first, int position, int x) {
    if (position <= 0 || (first == NULL && position > 1 ))  {
      printf("No such position in DLL so insertion is not possible\n");
      return first;
   }
   NODE temp, last = first;
   int i;
   for (i = 1; i < (position - 1); i++) {
      if (last -> next == NULL) {
         printf("No such position in DLL so insertion is not possible\n");
         return first;
      }
      last = last -> next;
   }
   temp = createNodeInDLL();
   temp -> data = x;
   if (position == 1) {
      if (first != NULL) {
         temp -> next = first;
         first -> prev = temp;
      }
      first = temp;
   } else {
      temp -> next = last -> next;
      temp -> prev = last;
      if(last->next != NULL)
         last->next->prev = temp;
      last->next = temp;
   }
   return first;
}

void traverseListInDLL(NODE first) {
   NODE lastNode = first;
   while (lastNode != NULL) {
      printf("%d <--> ", lastNode -> data);
      lastNode = lastNode -> next;
   }
   printf("NULL\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 15
1.Insert At specified position 2.Traverse the List 3.Exit 1

Test Case - 1
Enter your option : 1
Enter a position : 2
Enter an element : 20
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 10
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in DLL are : 10 <--> 15 <--> 20 <--> NULL 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 3
Enter an element : 18
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in DLL are : 10 <--> 15 <--> 18 <--> 20 <--> NULL 3
1.Insert At specified position 2.Traverse the List 3.Exit 3
Enter your option : 3

Test Case - 2
User Output
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 2
Enter an element : 20
No such position in DLL so insertion is not possible 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 10
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 3
Enter an element : 30
No such position in DLL so insertion is not possible 2
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in DLL are : 10 <--> NULL 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 20
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 30
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in DLL are : 30 <--> 20 <--> 10 <--> NULL 3
1.Insert At specified position 2.Traverse the List 3.Exit 3
Enter your option : 3

Test Case - 3
User Output
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 5
Enter an element : 50
No such position in DLL so insertion is not possible 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 10
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 3
Enter an element : 30
No such position in DLL so insertion is not possible 1
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 2
Enter an element : 20
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in DLL are : 10 <--> 20 <--> NULL 3
1.Insert At specified position 2.Traverse the List 3.Exit 3
Enter your option : 3

Test Case - 4
User Output
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 10
1.Insert At specified position 2.Traverse the List 3.Exit 1
Enter your option : 1
Enter a position : 1
Enter an element : 20
1.Insert At specified position 2.Traverse the List 3.Exit 2
Enter your option : 2
The elements in DLL are : 20 <--> 10 <--> NULL 3
1.Insert At specified position 2.Traverse the List 3.Exit 3
Enter your option : 3

S.No: 30	Exp. Name: ***Write a Program to Delete an element at Begin from Doubly Linked List***	Date:

Aim:

Fill in the missing code in `deleteAtBeginInDLL(NODE first)` function which deletes an integer from the beginning of the double linked list.

Source Code:

DoubleLL9.c

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

#include "DelAtBeginInDLL.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBeginInDLL(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Double Linked List is empty so deletion is not possible\n");
               } else {
                  first = deleteAtBeginInDLL(first);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Double Linked List is empty\n");
               } else {
                  printf("The elements in DLL are : ");
                  traverseListInDLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtBeginInDLL.c

```
struct node {
   int data;
   struct node *prev;
   struct node *next;
};
typedef struct node * NODE;
```

```
NODE createNodeInDLL() {
   NODE temp;
   temp = (NODE)malloc(sizeof(struct node));
   temp->prev = NULL;
   temp->next = NULL;
   return temp;
}

NODE insertAtBeginInDLL(NODE first, int x) {
   NODE temp;
   temp = createNodeInDLL();
   temp -> data = x;
   if (first != NULL) {
      temp -> next = first;
      first -> prev = temp;
   }
   first = temp;
   return first;
}

NODE deleteAtBeginInDLL(NODE first) {
   NODE lastNode = first;
   if (lastNode -> next == NULL) {
   first = NULL;
   } else {
      first = first -> next;
      first -> prev = NULL;
   }
   printf("The deleted element from DLL : %d\n", lastNode -> data);
   free(lastNode);
   return first;
}

void traverseListInDLL(NODE first) {
   NODE lastNode = first;
   while (lastNode != NULL) {
      printf("%d <--> ", lastNode -> data);
      lastNode = lastNode -> next;
   }
   printf("NULL\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 10
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 20
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 1
Enter your option : 1

Test Case - 1

```
Enter an element :  30
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in DLL are : 30 <--> 20 <--> 10 <--> NULL 2
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 2
Enter your option :  2
The deleted element from DLL : 30 2
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 2
Enter your option :  2
The deleted element from DLL : 20 3
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in DLL are : 10 <--> NULL 2
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 2
Enter your option :  2
The deleted element from DLL : 10 3
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 3
Enter your option :  3
Double Linked List is empty 2
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 2
Enter your option :  2
Double Linked List is empty so deletion is not possible 4
1.Insert At Begin 2.Delete at Begin 3.Traverse the List 4.Exit 4
Enter your option :  4
```

S.No: 31	Exp. Name: ***Write a Program to Delete an element at End from Doubly Linked List***	Date:

Aim:

Fill in the missing code in `deleteAtEndInDLL(NODE first)` function which deletes an integer from the ending of the double linked list.

Source Code:

DoubleLL6.c

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

#include "DelAtEndingInDLL.c"

void main() {
   NODE first = NULL;
   int x, op;
   while(1) {
      printf("1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEndInDLL(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Double Linked List is empty so deletion is not possible\n");
               } else {
                  first = deleteAtEndInDLL(first);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Double Linked List is empty\n");
               } else {
                  printf("The elements in DLL are : ");
                  traverseListInDLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtEndingInDLL.c

```
struct node {
   int data;
   struct node *prev;
   struct node *next;
};
typedef struct node * NODE;
```

```
NODE createNodeInDLL() {
   NODE temp;
   temp = (NODE)malloc(sizeof(struct node));
   temp->prev = NULL;
   temp->next = NULL;
   return temp;
}

NODE insertAtEndInDLL(NODE first, int x) {
   NODE temp, lastNode = first;
   temp = createNodeInDLL();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
   } else {
      while (lastNode -> next != NULL) {
         lastNode = lastNode -> next;
      }
      lastNode -> next = temp;
      temp -> prev = lastNode;
   }
   return first;
}

NODE deleteAtEndInDLL(NODE first) {
   NODE temp, lastNode = first;
   if (lastNode -> next == NULL) {
   first = NULL;
   } else {
      while (lastNode -> next != NULL) {
         temp = lastNode;
         lastNode = lastNode -> next;
      }
      temp -> next = NULL;
   }
   printf("The deleted element from DLL : %d\n", lastNode -> data);
   free(lastNode);
   return first;
}

void traverseListInDLL(NODE first) {
   NODE lastNode = first;
   while (lastNode != NULL) {
      printf("%d <--> ", lastNode -> data);
      lastNode = lastNode -> next;
   }
   printf("NULL\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 1

Test Case - 1
Enter your option : 1
Enter an element : 10
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 20
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 30
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in DLL are : 10 <--> 20 <--> 30 <--> NULL 2
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 2
Enter your option : 2
The deleted element from DLL : 30 2
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 2
Enter your option : 2
The deleted element from DLL : 20 3
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in DLL are : 10 <--> NULL 4
1.Insert At End 2.Delete at End 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 32	Exp. Name: ***Write a Program to Delete an element at Specified Position from Doubly Linked List***	Date:

Aim:

Fill in the missing code in `deleteAtPositionInDLL(NODE first, int pos)` function which deletes an integer from a given position in the double linked list.

Source Code:

DoubleLL7.c

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

#include "DelAtPositionInDLL.c"

void main() {
   NODE first = NULL;
   int x, op, pos;
   while(1) {
      printf("1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtEndInDLL(first, x);
               break;
         case 2:  if (first == NULL) {
                  printf("Double Linked List is empty so deletion is not possible\n");
               } else {
                  printf("Enter position : ");
                  scanf("%d", &pos);
                  first = deleteAtPositionInDLL(first, pos);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Double Linked List is empty\n");
               } else {
                  printf("The elements in DLL are : ");
                  traverseListInDLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

DelAtPositionInDLL.c

```
struct node {
   int data;
   struct node *prev;
   struct node *next;
```

```
};
typedef struct node * NODE;

NODE createNodeInDLL() {
   NODE temp;
   temp = (NODE)malloc(sizeof(struct node));
   temp->prev = NULL;
   temp->next = NULL;
   return temp;
}

void traverseListInDLL(NODE first) {
   NODE lastNode = first;
   while (lastNode != NULL) {
      printf("%d <--> ", lastNode -> data);
      lastNode = lastNode -> next;
   }
   printf("NULL\n");
}

NODE insertAtEndInDLL(NODE first, int x) {
   NODE temp, lastNode = first;
   temp = createNodeInDLL();
   temp -> data = x;
   if (first == NULL) {
      first = temp;
   } else {
      while (lastNode -> next != NULL) {
         lastNode = lastNode -> next;
      }
      lastNode -> next = temp;
      temp -> prev = lastNode;
   }
   return first;
}

NODE deleteAtPositionInDLL(NODE first, int position) {
   NODE prev;
   if (first == NULL) {
      printf("Double Linked List is empty so deletion is not possible\n");
   } else {
      NODE last = first;
      if (position == 1) {
         if (last -> next == NULL) {
            first = NULL;
         }  else {
            first = first -> next;
            first -> prev = NULL;
         }
      } else {
         int i;
         for (i = 1; i < position; i++) {
            if (last == NULL) {
               printf("No such position in DLL so deletion is not possible\n");
               return first;
            } else {
               prev = last;
               last = last -> next;
```

```
            }
         }
         if (last == NULL || position <= 0 ) {
            printf("No such position in DLL so deletion is not possible\n");
            return first;
         } else if (last -> next == NULL) {
            prev -> next = NULL;
         } else {
            prev -> next = last -> next;
            prev -> next -> prev = last -> prev;
         }
      }
      printf("The deleted element from DLL : %d\n", last -> data);
      free(last);
      return first;
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 11
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 22
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 33
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in DLL are : 11 <--> 22 <--> 33 <--> NULL 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 2
The deleted element from DLL : 22 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in DLL are : 11 <--> 33 <--> NULL 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 1
The deleted element from DLL : 11 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in DLL are : 33 <--> NULL 4
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 4
Enter your option : 4

Test Case - 2

Test Case - 2

User Output
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Double Linked List is empty so deletion is not possible 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
Double Linked List is empty 1
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 1
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 4
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 1
Enter your option : 1
Enter an element : 5
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in DLL are : 1 <--> 4 <--> 5 <--> NULL 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 4
No such position in DLL so deletion is not possible 2
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 2
Enter your option : 2
Enter position : 3
The deleted element from DLL : 5 3
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 3
Enter your option : 3
The elements in DLL are : 1 <--> 4 <--> NULL 4
1.Insert At End 2.Delete at Position 3.Traverse the List 4.Exit 4
Enter your option : 4

S.No: 33	Exp. Name: ***Write Code for searchPosOfEleInDLL() function in DLL***	Date:

Aim:

Fill in the missing code in `searchPosOfEleInDLL(NODE first, int element)` function which **searches** a given element in the list of elements and returns the **position** of that element if found.

Source Code:

DoubleLL8.c

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

#include "SearchPositionOfEleInDLL.c"

void main() {
   NODE first = NULL;
   int x, pos, op;
   while(1) {
      printf("1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit
\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element : ");
               scanf("%d", &x);
               first = insertAtBeginInDLL(first, x);
               break;
         case 2:  printf("Enter search element : ");
               scanf("%d", &x);
               pos = searchPosOfEleInDLL(first, x);
               if (pos == 0) {
                  printf("The given element %d is not found in the given DLL\n", x);
               } else {
                  printf("The given element %d is found at position : %d\n", x, pos);
               }
               break;
         case 3:  if (first == NULL) {
                  printf("Double Linked List is empty\n");
               } else {
                  printf("The elements in DLL are : ");
                  traverseListInDLL(first);
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

SearchPositionOfEleInDLL.c

```
struct node {
   int data;
   struct node *prev;
```

```
   struct node *next;
};

typedef struct node * NODE;

NODE createNodeInDLL() {
   NODE temp;
   temp = (NODE)malloc(sizeof(struct node));
   temp->prev = NULL;
   temp->next = NULL;
   return temp;
}

void traverseListInDLL(NODE first) {
   NODE lastNode = first;
   while (lastNode != NULL) {
      printf("%d <--> ", lastNode -> data);
      lastNode = lastNode -> next;
   }
   printf("NULL\n");
}

NODE insertAtBeginInDLL(NODE first, int x) {
   NODE temp;
   temp = createNodeInDLL();
   temp -> data = x;
   if (first != NULL) {
      temp -> next = first;
      first -> prev = temp;
   }
   first = temp;
   return first;
}
int searchPosOfEleInDLL(NODE first, int element) {
   NODE currentNode = first;
   int count = 0;
   if (currentNode == NULL) {
      return count;
   }
   while (currentNode != NULL && currentNode -> data != element) {
      if (currentNode -> next == NULL) {
         return 0;
      }
      count++;
      currentNode = currentNode -> next;
   }
   return(count + 1);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1

Test Case - 1

```
Enter your option :  1
Enter an element :  5
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  6
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  3
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 1
Enter your option :  1
Enter an element :  4
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 3
Enter your option :  3
The elements in DLL are : 4 <--> 3 <--> 6 <--> 5 <--> NULL 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option :  2
Enter search element :  5
The given element 5 is found at position : 4 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option :  2
Enter search element :  3
The given element 3 is found at position : 2 2
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 2
Enter your option :  2
Enter search element :  7
The given element 7 is not found in the given DLL 4
1.Insert At Begin 2.Search an element Position 3.Traverse the List 4.Exit 4
Enter your option :  4
```

S.No: 34	Exp. Name: ***Write a C program to implement different Operations on Stack using Array representation***	Date:

Aim:

Write a program to implement `stack` using **arrays**.

```
Sample Input and Output:
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 4
    Stack is empty.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 2
    Stack is underflow.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 3
    Stack is empty.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 5
    Stack is underflow.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 25
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 26
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 3
    Elements of the stack are : 26 25
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 2
    Popped value = 26
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 4
    Stack is not empty.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 5
    Peek value = 25
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 6
```

Source Code:

StackUsingArray.c

```
#include <stdio.h>
#include <stdlib.h>
#define STACK_MAX_SIZE 10
#include "StackOperations.c"

int main() {
   int op, x;
   while(1) {
      printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
```

```
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:
            printf("Enter element : ");
            scanf("%d", &x);
            push(x);
            break;
         case 2:
            pop();
            break;
         case 3:
            display();
            break;
         case 4:
            isEmpty();
            break;
         case 5:
            peek();
            break;
         case 6:
            exit(0);
      }
   }
}
```

StackOperations.c

```
int arr[STACK_MAX_SIZE];
int top = -1;

void push(int element) {
   if(top == STACK_MAX_SIZE - 1) {
      printf("Stack is overflow.\n");
   } else {
      top = top + 1;
      arr[top] = element;
      printf("Successfully pushed.\n");
   }
}

void display() {
   if (top < 0) {
      printf("Stack is empty.\n");
   } else {
      printf("Elements of the stack are : " );
      for(int i = top; i >= 0; i--) {
         printf("%d ", arr[i]);
      }
      printf("\n");
   }
}

void pop() {
   int x;
   if(top < 0) {
```

```
      printf("Stack is underflow.\n");
   } else {
      x = arr[top];
      top = top - 1;
      printf("Popped value = %d\n",x);
   }
}

void peek() {
   int x;
   if(top < 0) {
      printf("Stack is underflow.\n");
   } else {
      x = arr[top];
      printf("Peek value = %d\n",x);
   }
}

void isEmpty() {
   if (top < 0) {
      printf("Stack is empty.\n");
   } else {
      printf("Stack is not empty.\n");
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 10
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 20
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 30
Successfully pushed. 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Elements of the stack are : 30 20 10 5
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 5
Enter your option : 5
Peek value = 30 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Popped value = 30 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2

Test Case - 1
Popped value = 20 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Elements of the stack are : 10 5
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 5
Enter your option : 5
Peek value = 10 4
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 4
Enter your option : 4
Stack is not empty. 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Popped value = 10 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Stack is empty. 4
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 4
Enter your option : 4
Stack is empty. 6
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 6
Enter your option : 6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 4
Enter your option : 4
Stack is empty. 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Stack is underflow. 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Stack is empty. 5
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 5
Enter your option : 5
Stack is underflow. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 25
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 26
Successfully pushed. 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Elements of the stack are : 26 25 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Popped value = 26 4
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 4
Enter your option : 4

Test Case - 2
Stack is not empty. 5
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 5
Enter your option : 5
Peek value = 25 6
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 6
Enter your option : 6

Test Case - 3
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 11
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 12
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 13
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 14
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 15
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 16
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 17
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 18
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 19
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 12
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 13

Test Case - 3
Stack is overflow. 6
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 6
Enter your option : 6

S.No: 35	Exp. Name: ***Write a C program to implement different Operations on Stack using Linked Lists***	Date:

Aim:

Write a program to implement `stack` using **linked lists**.

```
Sample Input and Output:
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 33
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 22
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 55
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 1
    Enter element : 66
    Successfully pushed.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 3
    Elements of the stack are : 66 55 22 33
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 2
    Popped value = 66
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 2
    Popped value = 55
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 3
    Elements of the stack are : 22 33
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 5
    Peek value = 22
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 4
    Stack is not empty.
    1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
    Enter your option : 6
```

Source Code:

StackUsingLL.c

```
#include <stdio.h>
#include <stdlib.h>
#include "StackOperationsLL.c"

int main() {
   int op, x;
   while(1) {
```

```
      printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:
            printf("Enter element : ");
            scanf("%d", &x);
            push(x);
            break;
         case 2:
            pop();
            break;
         case 3:
            display();
            break;
         case 4:
            isEmpty();
            break;
         case 5:
            peek();
            break;
         case 6:
            exit(0);
      }
   }
}
```

StackOperationsLL.c

```
#include <stdio.h>
#include <stdlib.h>
struct stack {
   int data;
   struct stack *next;
};

typedef struct stack *stk;
stk top = NULL;

stk push(int x) {
   stk temp;
   temp = (stk)malloc(sizeof(struct stack));
   if(temp == NULL) {
      printf("Stack is overflow.\n");
   } else {
      temp -> data = x;
      temp -> next = top;
      top = temp;
      printf("Successfully pushed.\n");
   }
}

void display() {
   stk temp = top;
   if(temp == NULL) {
      printf("Stack is empty.\n");
```

```
   } else {
      printf("Elements of the stack are : ");
      while(temp != NULL) {
         printf("%d ", temp -> data);
         temp = temp -> next;
      }
      printf("\n");
   }
}

stk pop() {
   stk temp;
   if(top == NULL) {
      printf("Stack is underflow.\n");
   } else {
      temp = top;
      top = top -> next;
      printf("Popped value = %d\n", temp -> data);
      free(temp);
   }
}

void peek() {
   stk temp;
   if(top == NULL) {
      printf("Stack is underflow.\n");
   } else {
      temp = top;
      printf("Peek value = %d\n", temp -> data);
   }
}

void isEmpty() {
   if(top == NULL) {
      printf("Stack is empty.\n");
   } else {
      printf("Stack is not empty.\n");
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 33
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 22
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1

Test Case - 1
Enter element : 55
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 66
Successfully pushed. 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Elements of the stack are : 66 55 22 33 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Popped value = 66 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Popped value = 55 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Elements of the stack are : 22 33 5
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 5
Enter your option : 5
Peek value = 22 4
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 4
Enter your option : 4
Stack is not empty. 6
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 6
Enter your option : 6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Stack is underflow. 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Stack is empty. 5
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 5
Enter your option : 5
Stack is underflow. 4
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 4
Enter your option : 4
Stack is empty. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 23
Successfully pushed. 1
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 1
Enter your option : 1
Enter element : 24
Successfully pushed. 3
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 3
Enter your option : 3
Elements of the stack are : 24 23 5

Test Case - 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 5
Enter your option : 5
Peek value = 24 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Popped value = 24 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Popped value = 23 2
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 2
Enter your option : 2
Stack is underflow. 4
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 4
Enter your option : 4
Stack is empty. 6
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit 6
Enter your option : 6

S.No: 36	Exp. Name: ***Write a C program to Convert an Infix expression into Postfix expression***	Date:

Aim:

Algorithm for converting a infix expression to postfix expression.

```
Step-1: Create a stack.
Step-2: For each character c in the infix expression
    if c is an operand
        print c
    if c is an right parenthesis [An closing parenthesis ')' is encountered]
        pop up and print all the operators of the stack until the left matching parent
hesis is encountered. Don't print the matching left parenthesis just pop it.
    else [c is an operator or left parenthesis]
        print "Invalid symbols in infix expression. Only alphanumeric and { '+', '-
','*', '%%', '/' } are allowed." if c is not any of  {'+', '-','*','/','%'} and STOP.
        else pop and print operators until one of lower priority than c is encountere
d or a left parentheses is encountered or the stack is empty.
        push c into the stack.
Step-3: pop and print all the operators from the stack until stack is empty. If any le
ft parenthesis is found then print "Invalid infix expression : unbalanced parenthesi
s." and STOP.
Step-4: print the postfix expression.
```

Source Code:

Infix2PostfixMain.c

```
#include "Infix2PostfixOperation.c"
int main() {
   char exp[20];
   char *e, x;
   printf("Enter the expression : ");
   scanf("%s",exp);
   e = exp;
   convertInfix(e);
}
```

Infix2PostfixOperation.c

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include<ctype.h>
#define STACK_MAX_SIZE 20
char stack [STACK_MAX_SIZE];
int top = -1;

//Return 1 if stack is empty else return 0.
int isEmpty() {
   if(top<0)
      return 1;
   else
      return 0;
}

//Push the character into stack
```

```
void push(char x) {
   if(top == STACK_MAX_SIZE - 1) {
      printf("Stack is overflow.\n");
   } else {
      top = top + 1;
      stack[top] = x;
   }
}

//pop a character from stack
char pop() {
    if(top < 0) {
        printf("Stack is underflow : unbalanced parenthesis\n");
        exit(0);
    }
    else
        return stack[top--];
}

// Return 0 if char is '('
// Return 1 if char is '+' or '-'
// Return 2 if char is '*' or '/' or '%'
int priority(char x) {
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/' || x == '%')
        return 2;
}

//Output Format
//if expression is correct then output will be Postfix Expression : <postfix notation>
//If expression contains invalid operators then output will be "Invalid symbols in infi
x expression. Only alphanumberic and { '+', '-','*', '%%', '/' } are allowed."
//If the expression contains unbalanced paranthesis the output will be "Invalid infix e
xpression : unbalanced parenthesis."
void convertInfix(char * e) {
   int x;
   int k=0;
   char * p = (char *)malloc(sizeof(char)*strlen(e));
   while(*e != '\0') {
        if(isalnum(*e))
            p[k++]=*e;
        else if(*e == '(')
            push(*e);
        else if(*e == ')') {
            while(!isEmpty() && (x = pop()) != '(')
                p[k++]=x;
        }
        else if (*e == '+' || *e == '-' || *e == '*' || *e == '/' || *e == '%') {
            while(priority(stack[top]) >= priority(*e))
                p[k++]=pop();
            push(*e);
        }
        else {
         printf("Invalid symbols in infix expression. Only alphanumeric and { '+', '-
','*', '%%', '/' } are allowed.\n");
         exit(0);
```

```
        }
        e++;
    }
    while(top != -1) {
      x=pop();
      if(x == '(') {
         printf("Invalid infix expression : unbalanced parenthesis.\n");
         exit(0);
      }
        p[k++] = x;
    }
    p[k++]='\0';
    printf("Postfix expression : %s\n",p);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the expression : A+B*(C-D)
Postfix expression : ABCD-*+

Test Case - 2
User Output
Enter the expression : A+B*C
Postfix expression : ABC*+

Test Case - 3
User Output
Enter the expression : A+B*(C+D)*E+(F*G
Invalid infix expression : unbalanced parenthesis.

Test Case - 4
User Output
Enter the expression : A+B*(S+W&L)
Invalid symbols in infix expression. Only alphanumeric and { '+', '-','*', '%', '/' } are allowed.

S.No: 37	Exp. Name: ***Write a C program to evaluate a Postfix expression***	Date:

Aim:

The algorithm for evaluating the given **postfix** expression is as follows:

```
Step-1: Create a stack.
Step-2: For each character c in the postfix expression
    if c is an operand
        push c into stack
    if c is an operator
        pop an element from stack and assign it to A
        pop one more element from stack and assign it to B
        If the stack becomes empty before popping two elements the print "Invalid post
fix expression." and STOP.
        calculate result = B c A , where c is the operator.
        push result back to stack.
Step-3: After reaching the end of postfix expression, pop and print the element from t
he stack as the result.
Step-4: If stack becomes empty before reaching the end of expression then print "Inval
id postfix expression." and also the stack should have one element after reaching the
 end of expression else print "Invalid postfix expression."
```

Source Code:

PostfixEvaluation.c

```
#include <ctype.h>
#include <stdio.h>
#define STACK_MAX_SIZE 20
int stack [STACK_MAX_SIZE];
int top = -1;

//Return 1 if stack is empty else return 0.
int isEmpty() {
   if(top<0)
      return 1;
   else
      return 0;
}

//Push the character into stack
void push(int x) {
   if(top == STACK_MAX_SIZE - 1) {
      printf("Stack is overflow.\n");
   } else {
      top = top + 1;
      stack[top] = x;
   }
}

//pop a character from stack
int pop() {
    if(top < 0) {
        return -1;
    }
    else
```

```
        return stack[top--];
}

//Output Format - Result : <result> if the input postfix expression is vaild.
//Output Format - Invalid postfix expression,. - if the input expression is invalid.
//postfix expression is given as the parameter.
void evaluatePostfix(char * e) {
   int a,b,result;
   while(*e != '\0') {
        if(isdigit(*e))
            push(*e-'0');
        else if (*e == '+' || *e == '-' || *e == '*' || *e == '/' || *e == '%'){
            if(isEmpty()){
               printf("Invalid postfix expression.\n");
               return;
            }
            a = pop();
            if(isEmpty()){
               printf("Invalid postfix expression.\n");
               return;
            }
            b = pop();
            switch(*e) {
               case '+':
                  result = b + a;
                  break;
               case '-':
                  result = b - a;
                  break;
               case '*':
                  result = b * a;
                  break;
               case '/':
                  result = b / a;
                  break;
               case '%':
                  result = b % a;
                  break;
            }
            push(result);
        }
        e++;
    }
    if(isEmpty()){
      printf("Invalid postfix expression.\n");
      return;
   }
   result = pop();
   if(!isEmpty()) {
      printf("Invalid postfix expression.\n");
      return;
   }
   printf("Result : %d\n",result);
}

//Read a postfix expression and evaluate it.
int main() {
   char exp[20];
```

```
   char *e, x;
   printf("Enter the postfix expression : ");
   scanf("%s",exp);
   e = exp;
   evaluatePostfix(e);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the postfix expression : 234+-
Result : -5

Test Case - 2
User Output
Enter the postfix expression : 345+-34-
Invalid postfix expression.

Test Case - 3
User Output
Enter the postfix expression : 2345+--23+-
Result : 3

S.No: 38	Exp. Name: ***Write a C program to implement different Operations on Queue using Array representation***	Date:

Aim:

Write a program to implement `queue` using **arrays**.

```
Sample Input and Output:
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 23
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 56
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 3
    Elements in the queue : 23 56
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 4
    Queue is not empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 5
    Queue size : 2
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted element = 23
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted element = 56
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 4
    Queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 6
```

Source Code:

QueueUsingArray.c

```
#include <conio.h>
#include <stdio.h>
#include "QueueOperations.c"
int main() {
   int op, x;
   while(1) {
      printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
      printf("Enter your option : ");
      scanf("%d",&op);
      switch(op) {
         case 1:
            printf("Enter element : ");
            scanf("%d",&x);
            enqueue(x);
            break;
```

```
         case 2:
            dequeue();
            break;
         case 3:
            display();
            break;
         case 4:
            isEmpty();
            break;
         case 5:
            size();
            break;
         case 6: exit(0);
      }
   }
}
```

QueueOperations.c

```
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;

void enqueue(int x) {
   if (rear == MAX - 1) {
      printf("Queue is overflow.\n");
   } else {
      rear++;
      queue[rear] = x;
      printf("Successfully inserted.\n");
   }
   if (front == -1) {
      front++;
   }
}

void dequeue() {
   if (front == -1) {
      printf("Queue is underflow.\n");
   } else {
      printf("Deleted element = %d\n",queue[front]);
      if (rear == front) {
         rear = front = -1;
      } else {
         front++;
      }
   }
}

void display() {
   if (front == -1 && rear == -1) {
      printf("Queue is empty.\n");
   } else {
      printf("Elements in the queue : ");
      for (int i = front; i <= rear; i++) {
         printf("%d ",queue[i]);
```

```
      }
      printf("\n");
   }
}

void size() {
   if(front == -1 && rear == -1)
      printf("Queue size : 0\n");
   else
      printf("Queue size : %d\n",rear-front+1);
}

void isEmpty() {
   if(front == -1 && rear == -1)
      printf("Queue is empty.\n");
   else
      printf("Queue is not empty.\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 23
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 56
Successfully inserted. 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the queue : 23 56 4
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 4
Enter your option : 4
Queue is not empty. 5
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 5
Enter your option : 5
Queue size : 2 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 2
Enter your option : 2
Deleted element = 23 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 2
Enter your option : 2
Deleted element = 56 4
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 4
Enter your option : 4
Queue is empty. 6
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 6
Enter your option : 6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 2
Enter your option : 2
Queue is underflow. 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 3
Enter your option : 3
Queue is empty. 4
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 4
Enter your option : 4
Queue is empty. 5
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 5
Enter your option : 5
Queue size : 0 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 14
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 78
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 53
Successfully inserted. 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the queue : 14 78 53 5
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 5
Enter your option : 5
Queue size : 3 6
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 6
Enter your option : 6

S.No: 39	Exp. Name: ***Write a C program to implement different Operations on Queue using Linked Lists***	Date:

Aim:

Write a program to implement `queue` using **linked lists**.

```
Sample Input and Output:
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 57
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 87
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 5
    Queue size : 2
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 3
    Elements in the queue : 57 87
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 57
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 87
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 3
    Queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 5
    Queue size : 0
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 6
```

Source Code:

QueueUsingLL.c

```
#include <conio.h>
#include <stdio.h>
#include "QueueOperationsLL.c"
int main() {
   int op, x;
   while(1) {
      printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
      printf("Enter your option : ");
      scanf("%d",&op);
      switch(op) {
         case 1:
            printf("Enter element : ");
            scanf("%d",&x);
            enqueue(x);
            break;
```

```
         case 2:
            dequeue();
            break;
         case 3:
            display();
            break;
         case 4:
            isEmpty();
            break;
         case 5:
            size();
            break;
         case 6: exit(0);
      }
   }
}
```

QueueOperationsLL.c

```
struct queue {
   int data;
   struct queue *next;
};

typedef struct queue *Q;
Q front = NULL, rear = NULL;

void enqueue(int element) {
   Q temp = NULL;
   temp = (Q)malloc(sizeof(struct queue));
   if(temp == NULL) {
      printf("Queue is overflow.\n");
   } else {
      temp -> data = element;
      temp -> next = NULL;

      if(front == NULL) {
         front = temp;
      } else {
         rear -> next = temp;
      }
      rear = temp;
      printf("Successfully inserted.\n");
   }
}

void dequeue() {
   Q temp = NULL;
   if(front == NULL) {
      printf("Queue is underflow.\n");
   } else {
      temp = front;
      if (front == rear) {
         front = rear = NULL;
      } else {
         front = front -> next;
```

```
      }
      printf("Deleted value = %d\n", temp -> data);
      free(temp);
   }
}
void display() {
   if(front == NULL) {
      printf("Queue is empty.\n");
   } else {
      Q temp = front;
      printf("Elements in the queue : ");
      while(temp  != NULL) {
         printf("%d ", temp -> data);
         temp = temp -> next;
      }
      printf("\n");
   }
}
void size() {
   int count =0;
   if(front == NULL) {
      printf("Queue size : 0\n");
   } else {
      Q temp = front;
      while(temp  != NULL) {
         temp = temp -> next;
         count = count + 1;
      }
      printf("Queue size : %d\n",count);
   }
}

void isEmpty() {
   if(front == NULL ) {
      printf("Queue is empty.\n");
   } else {
      printf("Queue is not empty.\n");
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 2
Enter your option : 2
Queue is underflow. 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 3
Enter your option : 3
Queue is empty. 4
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 4
Enter your option : 4
Queue is empty. 5
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 5

Test Case - 1
Enter your option : 5
Queue size : 0 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 44
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 55
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 66
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 67
Successfully inserted. 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the queue : 44 55 66 67 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 44 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 55 5
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 5
Enter your option : 5
Queue size : 2 4
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 4
Enter your option : 4
Queue is not empty. 6
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 6
Enter your option : 6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 23
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 234
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 45
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 1
Enter your option : 1

Test Case - 2
Enter element : 456
Successfully inserted. 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 23 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the queue : 234 45 456 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 234 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the queue : 45 456 4
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 4
Enter your option : 4
Queue is not empty. 5
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 5
Enter your option : 5
Queue size : 2 6
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit 6
Enter your option : 6

S.No: 40	Exp. Name: ***Implementation of Circular Queue using Arrays***	Date:

Aim:

Write a program to implement `circular queue` using **arrays**.

Note: Define the **MAX** value as `5`.

```
Sample Input and Output:
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Circular queue is underflow.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 3
    Circular queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 4
    Circular queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 5
    Circular queue size : 0
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 11
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 12
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 3
    Elements in the circular queue : 11 12
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 4
    Circular queue is not empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 5
    Circular queue size : 2
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Deleted element = 11
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Deleted element = 12
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Circular queue is underflow.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 6
```

Source Code:

CQueueUsingArray.c

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

```
#include "CQueueOperations.c"

int main() {
   int op, x;
   while(1) {
      printf("1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit\n");
      printf("Enter your option : ");
      scanf("%d",&op);
      switch(op) {
         case 1:
            printf("Enter element : ");
            scanf("%d",&x);
            enqueue(x);
            break;
         case 2:
            dequeue();
            break;
         case 3:
            display();
            break;
         case 4:
            isEmpty();
            break;
         case 5:
            size();
            break;
         case 6: exit(0);
      }
   }
}
```

CQueueOperations.c

```
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;

void dequeue() {
   if (front == -1) {
      printf("Circular queue is underflow.\n");
   } else {
      printf("Deleted element = %d\n", queue[front]);
      if (rear == front) {
         rear = front = -1;
      } else if (front == MAX - 1) {
         front = 0;
      } else {
         front++;
      }
   }
}

void enqueue(int x) {
   if (((rear == MAX - 1) && (front == 0)) || (rear + 1 == front)) {
      printf("Circular queue is overflow.\n");
   } else {
```

```
      if (rear == MAX - 1) {
         rear = -1;
      } else if (front == -1) {
         front = 0;
      }
      rear++;
      queue[rear] = x;
      printf("Successfully inserted.\n");
   }
}
void display() {
   int i;
   if (front == -1 && rear == -1) {
      printf("Circular queue is empty.\n");
   } else {
      printf("Elements in the circular queue : ");
      if (front <= rear) {
         for (i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
         }
      } else {
         for (i = front; i <= MAX - 1; i++) {
            printf("%d ", queue[i]);
         }
         for (i = 0; i <= rear; i++) {
            printf("%d ", queue[i]);
         }
      }
      printf("\n");
   }
}

void size() {
   if(front == -1 && rear == -1)
      printf("Circular queue size : 0\n");
   else {
      if(front <= rear )
         printf("Circular queue size : %d\n",rear-front+1);
      else
         printf("Circular queue size : %d\n",MAX-front+rear+1);
   }
}
void isEmpty() {
   if(front == -1 && rear == -1)
      printf("Circular queue is empty.\n");
   else
      printf("Circular queue is not empty.\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1

Test Case - 1
Enter your option : 1
Enter element : 34
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 55
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 26
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 77
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 38
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 59
Circular queue is overflow. 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the circular queue : 34 55 26 77 38 5
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 5
Enter your option : 5
Circular queue size : 5 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted element = 34 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted element = 55 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted element = 26 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the circular queue : 77 38 4
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 4
Enter your option : 4
Circular queue is not empty. 5
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 5
Enter your option : 5
Circular queue size : 2 6
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 6
Enter your option : 6

Test Case - 2
User Output

Test Case - 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Circular queue is underflow. 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Circular queue is empty. 4
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 4
Enter your option : 4
Circular queue is empty. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 12
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 34
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 56
Successfully inserted. 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the circular queue : 12 34 56 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 38
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 25
Successfully inserted. 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the circular queue : 12 34 56 38 25 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 56
Circular queue is overflow. 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted element = 12 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted element = 34 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the circular queue : 56 38 25 4
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 4
Enter your option : 4
Circular queue is not empty. 5
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 5
Enter your option : 5

Test Case - 2
Circular queue size : 3 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 11
Successfully inserted. 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the circular queue : 56 38 25 11 6
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 6
Enter your option : 6

S.No: 41	Exp. Name: ***Implementation of Circular Queue using Linked List***	Date:

Aim:

Write a program to implement `circular queue` using **linked lists**.

```
Sample Input and Output:
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 15
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 16
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 17
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 3
    Elements in the circular queue : 15 16 17
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 5
    Circular queue size : 3
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 15
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 16
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 17
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 3
    Circular queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 4
    Circular queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 5
    Circular queue size : 0
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 6
```

Source Code:

CQueueLL.c

```
#include <stdlib.h>
#include <stdio.h>
#include "CQueueOperationsLL.c"
int main() {
   int op, x;
   while(1) {
```

```
      printf("1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit\n");
      printf("Enter your option : ");
      scanf("%d",&op);
      switch(op) {
         case 1:
            printf("Enter element : ");
            scanf("%d",&x);
            enqueue(x);
            break;
         case 2:
            dequeue();
            break;
         case 3:
            display();
            break;
         case 4:
            isEmpty();
            break;
         case 5:
            size();
            break;
         case 6: exit(0);
      }
   }
}
```

CQueueOperationsLL.c

```
struct queue {
   int data;
   struct queue *next;
};
typedef struct queue *CircularQueue;
CircularQueue front = NULL, rear = NULL;

void dequeue() {
   CircularQueue temp = NULL;
   if(front == NULL) {
      printf("Circular queue is underflow.\n");
   } else {
      temp = front;
      if (front == rear) {
         front = rear = NULL;
      } else {
         front = front -> next;
         rear->next = front;
      }
      printf("Deleted value = %d\n", temp -> data);
      free(temp);
   }
}

void size() {
   int count =0;
   if(front == NULL) {
      printf("Circular queue size : 0\n");
```

```
      return;
   }
   CircularQueue temp = front;
   do {
      temp = temp -> next;
      count = count + 1;
   } while(temp != front);
   printf("Circular queue size : %d\n",count);
}

void isEmpty() {
   if(front == NULL ) {
      printf("Circular queue is empty.\n");
   } else {
      printf("Circular queue is not empty.\n");
   }
}

void enqueue(int element) {
   CircularQueue temp = NULL;
   temp = (CircularQueue)malloc(sizeof(struct queue));
   if(temp == NULL) {
      printf("Circular queue is overflow.\n");
   } else {
      temp -> data = element;
      temp -> next = NULL;
      if(front == NULL) {
         front = temp;
      } else {
         rear -> next = temp;
      }
      rear = temp;
      rear -> next = front;
      printf("Successfully inserted.\n");
   }
}

void display() {
   if(front == NULL) {
      printf("Circular queue is empty.\n");
   } else {
      CircularQueue temp = front;
      printf("Elements in the circular queue : ");
      do {
         printf("%d ", temp -> data);
         temp = temp -> next;
      } while(temp != front);
      printf("\n");
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 15
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 16
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 17
Successfully inserted. 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the circular queue : 15 16 17 5
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 5
Enter your option : 5
Circular queue size : 3 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 15 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 16 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 17 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Circular queue is empty. 4
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 4
Enter your option : 4
Circular queue is empty. 5
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 5
Enter your option : 5
Circular queue size : 0 6
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 6
Enter your option : 6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Circular queue is underflow. 5
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 5
Enter your option : 5
Circular queue size : 0 4
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 4
Enter your option : 4
Circular queue is empty. 3

Test Case - 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Circular queue is empty. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 143
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 153
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 163
Successfully inserted. 1
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 1
Enter your option : 1
Enter element : 173
Successfully inserted. 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 3
Enter your option : 3
Elements in the circular queue : 143 153 163 173 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 143 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 2
Enter your option : 2
Deleted value = 153 5
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 5
Enter your option : 5
Circular queue size : 2 4
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 4
Enter your option : 4
Circular queue is not empty. 6
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit 6
Enter your option : 6

S.No: 42	Exp. Name: ***BST Operations - Creating node, insertion and in-order traversal***	Date:

Aim:

Each individual node of a tree is defined as a structure as follows :

```
struct node {
    int data;
    struct node * left; //reference to the left child of the node
    struct node * right; //reference to the left child of the node
 };
typedef struct node * BSTNODE; // typedef defines a new type BSTNODE as pointer of
data type struct node
// Instead of struct node * we can use BSTNODE because of the typedef declaration.
```

The `newNodeInBST(int item)` which allocates memory to the Binary Search Tree node is defined as follows :

```
BSTNODE newNodeInBST(int item) {
    BSTNODE temp =  (BSTNODE)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}
```

The function `insertNodeInBST(BSTNODE root, int ele)` inserts an element `ele` into the Binary Search Tree.

The algorithm for `insertNodeInBST(BSTNODE root, int ele)` is as follows :

```
Step-1: Create a new node temp with given value ele and set its left and right fie
ld to NULL.
Step-2: If root is equal to NULL set temp as root.
Step-3: If root is not equal to empty , check if the ele is smaller or larger than th
e root node
Step-4: If ele is smaller than or equal to the node, then move to its left child. If
 ele is larger than the node, then move to its right child.
Step-5: Repeat the steps 3 and 4 until we reach a leaf node.
Step-6: After reaching a leaf node, then insert the newNode as left child if newNode i
s smaller or equal to that leaf else insert it as right child.
```

The pseudo code for `BSTNODE insertNodeInBST(BSTNODE node,int ele)` is as follows

```
BSTNODE insertNodeInBST(BSTNODE node, int ele) {
    if ( node is equal to NULL) {
        print "Successfully inserted."
        return call : newNodeInBST( ele );
    }
    if( ele is less than the data field of node )
        left field of node = call : insertNodeInBST( node->left , ele )
    else if ( ele greater than the data field of node )
        right field of node = call :
insertNodeInBST( node->right , ele )
    else
        print "Element already exists in BST."
    return node ;
}
```

The `inorderInBST(BSTNODE root)` performs an in-order traversal on the tree.

The algorithm for `inorderInBST(BSTNODE root)` is as follows

```
Step-1: Traverse the elements in the left subtree of the root node. i.e call inorderI
nBST( root->left ).
Step-2: Visit the root node i.e. print the data field of the root
Step-3: Traverse the elements in the right subtree of the root node. i.e call inorder
InBST( root->right ).
```

Source Code:

BSTmain1.c

```
#include<stdio.h>
#include<stdlib.h>
#include "BSTInsertAndInorder.c"

void main() {
   int x, op;
   BSTNODE root = NULL;
   while(1)
   {
      printf("1.Insert 2.Inorder Traversal 3.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element to be inserted : ");
               scanf("%d", &x);
               root = insertNodeInBST(root,x);
               break;
         case 2:
               if(root == NULL) {
                  printf("Binary Search Tree is empty.\n");
               }
               else {
                  printf("Elements of the tree (in-order traversal): ");
                  inorderInBST(root);
                  printf("\n");
                  }
               break;
```

```
         case 3:
               exit(0);
      }
   }
}
```


BSTInsertAndInorder.c

```
struct node {
   int data;
   struct node *left, *right;
};

typedef struct node *BSTNODE;

BSTNODE newNodeInBST(int item) {
   BSTNODE temp =  (BSTNODE)malloc(sizeof(struct node));
   temp->data = item;
   temp->left = temp->right = NULL;
   return temp;
}

void inorderInBST(BSTNODE root) {
   if (root != NULL) {
      inorderInBST(root->left);
      printf("%d ", root->data);
      inorderInBST(root->right);
   }
}

BSTNODE insertNodeInBST(BSTNODE node, int ele) {
   if (node == NULL) {
      printf("Successfully inserted.\n");
      return newNodeInBST(ele);
   }
   if (ele < node->data)
      node->left  = insertNodeInBST(node->left,ele);
   else if (ele > node->data)
      node->right = insertNodeInBST(node->right,ele);
   else
      printf("Element already exists in BST.\n");
   return node;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 23
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Exit 1

Test Case - 1
Enter your option : 1
Enter an element to be inserted : 34
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 26
Successfully inserted. 2
1.Insert 2.Inorder Traversal 3.Exit 2
Enter your option : 2
Elements of the tree (in-order traversal): 23 26 34 2
1.Insert 2.Inorder Traversal 3.Exit 2
Enter your option : 2
Elements of the tree (in-order traversal): 23 26 34 3
1.Insert 2.Inorder Traversal 3.Exit 3
Enter your option : 3

Test Case - 2
User Output
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 54
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 34
Successfully inserted. 2
1.Insert 2.Inorder Traversal 3.Exit 2
Enter your option : 2
Elements of the tree (in-order traversal): 34 54 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 65
Successfully inserted. 2
1.Insert 2.Inorder Traversal 3.Exit 2
Enter your option : 2
Elements of the tree (in-order traversal): 34 54 65 3
1.Insert 2.Inorder Traversal 3.Exit 3
Enter your option : 3

Test Case - 3
User Output
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 23
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 12
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Exit 1

Test Case - 3
Enter your option : 1
Enter an element to be inserted : 11
Successfully inserted. 2
1.Insert 2.Inorder Traversal 3.Exit 2
Enter your option : 2
Elements of the tree (in-order traversal): 11 12 23 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 23
Element already exists in BST. 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 34
Successfully inserted. 2
1.Insert 2.Inorder Traversal 3.Exit 2
Enter your option : 2
Elements of the tree (in-order traversal): 11 12 23 34 3
1.Insert 2.Inorder Traversal 3.Exit 3
Enter your option : 3

Test Case - 4
User Output
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 1
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 2
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 3
Successfully inserted. 2
1.Insert 2.Inorder Traversal 3.Exit 2
Enter your option : 2
Elements of the tree (in-order traversal): 1 2 3 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 4
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Exit 1
Enter your option : 1
Enter an element to be inserted : 5
Successfully inserted. 2
1.Insert 2.Inorder Traversal 3.Exit 2
Enter your option : 2
Elements of the tree (in-order traversal): 1 2 3 4 5 3
1.Insert 2.Inorder Traversal 3.Exit 3
Enter your option : 3

S.No: 43	Exp. Name: ***Program to insert into BST and traversal using In-order, Pre-order and Post-order***	Date:

Aim:

Write a program to create a binary search tree of integers and perform the following operations

1. insert a node
2. in-order traversal
3. pre-order traversal
4. post-order traversal

Source Code:

BinarySearchTree.c

```
#include<stdio.h>
#include<stdlib.h>
#include "InsertAndTraversals.c"

void main() {
   int x, op;
   BSTNODE root = NULL;
   while(1) {
      printf("1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal
5.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element to be inserted : ");
               scanf("%d", &x);
               root = insertNodeInBST(root,x);
               break;
         case 2:
               if(root == NULL) {
                  printf("Binary Search Tree is empty.\n");
               }
               else {
                  printf("Elements of the BST (in-order traversal): ");
                  inorderInBST(root);
                  printf("\n");
                  }
               break;
         case 3:
               if(root == NULL) {
                  printf("Binary Search Tree is empty.\n");
               }
               else {
                  printf("Elements of the BST (pre-order traversal): ");
                  preorderInBST(root);
                  printf("\n");
                  }
               break;
         case 4:
               if(root == NULL) {
                  printf("Binary Search Tree is empty.\n");
               }
               else {
```

```
                  printf("Elements of the BST (post-order traversal): ");
                  postorderInBST(root);
                  printf("\n");
                  }
               break;
         case 5:
               exit(0);
      }
   }
}
```

InsertAndTraversals.c

```
struct node {
   int data;
   struct node *left, *right;
};

typedef struct node *BSTNODE;

BSTNODE newNodeInBST(int item) {
   BSTNODE temp =  (BSTNODE)malloc(sizeof(struct node));
   temp->data = item;
   temp->left = temp->right = NULL;
   return temp;
}

void inorderInBST(BSTNODE root) {
   if (root != NULL) {
      inorderInBST(root->left);
      printf("%d ", root->data);
      inorderInBST(root->right);
   }
}

void preorderInBST(BSTNODE root) {
   if (root != NULL) {
      printf("%d ", root->data);
      preorderInBST(root->left);
      preorderInBST(root->right);
    }
}

void postorderInBST(BSTNODE root) {
    if (root != NULL) {
      postorderInBST(root->left);
        postorderInBST(root->right);
        printf("%d ", root->data);
    }
}

BSTNODE insertNodeInBST(BSTNODE node, int ele) {
   if (node == NULL) {
      printf("Successfully inserted.\n");
      return newNodeInBST(ele);
   }
```

```
   if (ele < node->data)
      node->left  = insertNodeInBST(node->left,ele);
   else if (ele > node->data)
      node->right = insertNodeInBST(node->right,ele);
   else
      printf("Element already exists in BST.\n");
   return node;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 54
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 28
Successfully inserted. 1
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 62
Successfully inserted. 2
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit 2
Enter your option : 2
Elements of the BST (in-order traversal): 28 54 62 3
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit 3
Enter your option : 3
Elements of the BST (pre-order traversal): 54 28 62 4
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit 4
Enter your option : 4
Elements of the BST (post-order traversal): 28 62 54 5
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit 5
Enter your option : 5

S.No: 44	Exp. Name: ***Implementation of BST Operations - Deletion and Pre-order traversal***	Date:

Aim:

Source Code:

BSTmain2.c

```
#include<stdio.h>
#include<stdlib.h>
#include "BSTDeleteAndPreOrder.c"

void main() {
   int x, op;
   BSTNODE root = NULL;
   while(1)
   {
      printf("1.Insert 2.Delete 3.Preorder Traversal 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element to be inserted : ");
               scanf("%d", &x);
               root = insertNodeInBST(root,x);
               break;
         case 2:  printf("Enter an element to be deleted : ");
               scanf("%d", &x);
               root = deleteNodeInBST(root,x);
               break;
         case 3:
               if(root == NULL) {
                  printf("Binary Search Tree is empty.\n");
               }
               else {
                  printf("Elements of the BST (pre-order traversal): ");
                  preorderInBST(root);
                  printf("\n");
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

BSTDeleteAndPreOrder.c

```
struct node {
   int data;
   struct node *left, *right;
};

typedef struct node * BSTNODE;

BSTNODE newNodeInBST(int item) {
   BSTNODE temp =  (BSTNODE)malloc(sizeof(struct node));
```

```
   temp->data = item;
   temp->left = temp->right = NULL;
   return temp;
}

void preorderInBST(BSTNODE root) {
   if (root != NULL) {
      printf("%d ", root->data);
      preorderInBST(root->left);
      preorderInBST(root->right);
    }
}

BSTNODE insertNodeInBST(BSTNODE node, int ele) {
   if (node == NULL) {
      printf("Successfully inserted.\n");
      return newNodeInBST(ele);
   }
   if (ele < node->data)
      node->left  = insertNodeInBST(node->left,ele);
   else if (ele > node->data)
      node->right = insertNodeInBST(node->right,ele);
   else
      printf("Element already exists in BST.\n");
   return node;
}


BSTNODE minValueNode(BSTNODE node) {
   BSTNODE current = node;
   while (current->left != NULL)
      current = current->left;
   return current;
}

BSTNODE deleteNodeInBST(BSTNODE root, int ele) {
   if (root == NULL) {
      printf("Cannot find %d in the binary search tree.\n",ele);
      return root;
   }
   if (ele < root->data)
      root->left = deleteNodeInBST(root->left,ele);
   else if (ele > root->data)
      root->right = deleteNodeInBST(root->right,ele);
   else {
      if (root->left == NULL) {
         BSTNODE temp = root->right;
         printf("Deleted %d from binary search tree.\n",ele);
         free(root);
         return temp;
      }
      else if (root->right == NULL) {
         BSTNODE temp = root->left;
         printf("Deleted %d from binary search tree.\n",ele);
         free(root);
         return temp;
      }
      BSTNODE temp = minValueNode(root->right);
```

```
      root->data = temp->data;
      temp->data = ele;
      root->right = deleteNodeInBST(root->right,ele);
    }
   return root;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 56
Successfully inserted. 2
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 2
Enter your option : 2
Enter an element to be deleted : 35
Cannot find 35 in the binary search tree. 3
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 3
Enter your option : 3
Elements of the BST (pre-order traversal): 56 4
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 4
Enter your option : 4

Test Case - 2
User Output
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 25
Successfully inserted. 1
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 65
Successfully inserted. 2
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 2
Enter your option : 2
Enter an element to be deleted : 65
Deleted 65 from binary search tree. 3
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 3
Enter your option : 3
Elements of the BST (pre-order traversal): 25 4
1.Insert 2.Delete 3.Preorder Traversal 4.Exit 4
Enter your option : 4

S.No: 45	Exp. Name: ***Implementation of BST Operations - Search and post order traversal***	Date:

Aim:

The `searchNodeInBST(BSTNODE root, int ele)` function searches for a node in the binary search tree. Return NULL if not found in the binary search tree

The `postorderInBST(BSTNODE root)` performs and post-order traversal on the tree.

Source Code:

BSTmain3.c

```
#include<stdio.h>
#include<stdlib.h>
#include "BSTSearchAndPostOrder.c"

void main() {
   int x, op;
   BSTNODE root = NULL;
   while(1)
   {
      printf("1.Insert 2.Search 3.Postorder Traversal 4.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:  printf("Enter an element to be inserted : ");
               scanf("%d", &x);
               root = insertNodeInBST(root,x);
               break;
         case 2:
               printf("Enter an element to be searched : ");
               scanf("%d", &x);
               if( searchNodeInBST(root,x) == NULL)
                  printf("Element not found in the binary search tree.\n");
               else
                  printf("Element found in the binary search tree.\n");
               break;
         case 3:
               if(root == NULL) {
                  printf("Binary Search Tree is empty.\n");
               }
               else {
                  printf("Elements of the BST (post-order traversal): ");
                  postorderInBST(root);
                  printf("\n");
               }
               break;
         case 4:  exit(0);
      }
   }
}
```

BSTSearchAndPostOrder.c

```
struct node {
    int data;
    struct node *left, *right;
};

typedef struct node * BSTNODE;

BSTNODE newNodeInBST(int item) {
   BSTNODE temp =  (BSTNODE)malloc(sizeof(struct node));
   temp->data = item;
   temp->left = temp->right = NULL;
   return temp;
}

void postorderInBST(BSTNODE root) {
   if (root != NULL) {
      postorderInBST(root->left);
      postorderInBST(root->right);
      printf("%d ", root->data);
    }
}

BSTNODE insertNodeInBST(BSTNODE node, int ele) {
   if (node == NULL) return newNodeInBST(ele);
   if (ele < node->data)
      node->left  = insertNodeInBST(node->left,ele);
   else if (ele > node->data)
      node->right = insertNodeInBST(node->right,ele);
   return node;
}

BSTNODE searchNodeInBST(BSTNODE root, int ele) {
   if (root == NULL || root->data == ele)
      return root;
   if (root->data < ele)
      return searchNodeInBST(root->right, ele);
   else
      return searchNodeInBST(root->left, ele);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Search 3.Postorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 56
1.Insert 2.Search 3.Postorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 45
1.Insert 2.Search 3.Postorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 25

Test Case - 1
1.Insert 2.Search 3.Postorder Traversal 4.Exit 132
Enter your option : 132
1.Insert 2.Search 3.Postorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 589
1.Insert 2.Search 3.Postorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 451
1.Insert 2.Search 3.Postorder Traversal 4.Exit 47
Enter your option : 47
1.Insert 2.Search 3.Postorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 25
1.Insert 2.Search 3.Postorder Traversal 4.Exit 12
Enter your option : 12
1.Insert 2.Search 3.Postorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 5
1.Insert 2.Search 3.Postorder Traversal 4.Exit 3
Enter your option : 3
Elements of the BST (post-order traversal): 5 25 45 451 589 56 4
1.Insert 2.Search 3.Postorder Traversal 4.Exit 4
Enter your option : 4

Test Case - 2
User Output
1.Insert 2.Search 3.Postorder Traversal 4.Exit 1
Enter your option : 1
Enter an element to be inserted : 23
1.Insert 2.Search 3.Postorder Traversal 4.Exit 2
Enter your option : 2
Enter an element to be searched : 41
Element not found in the binary search tree. 3
1.Insert 2.Search 3.Postorder Traversal 4.Exit 3
Enter your option : 3
Elements of the BST (post-order traversal): 23 4
1.Insert 2.Search 3.Postorder Traversal 4.Exit 4
Enter your option : 4

S.No: 46	Exp. Name: ***Implementing a directed graph and its operations using adjacency matrix***	Date:

Aim:

Implementing a directed graph and its operations using adjacency matrix

In the adjacency matix representation, the graph is represented as a square matrix of **V** X **V** where **V** is the number of vertices in the graph.

The operations performed on the graph are as follows :

- Print adjacency matrix.
- Insert a vertex
- Insert an edge
- Delete a vertex
- Delete an edge

The pseudo code for the `void print(int * N)` which prints out an adjacency of size `N` as follows :

```
void print(int * N) {
    Declare to variables i and j
.
    if(*N is equal to 0) {
        print "Graph is empty."
        return;
    }
    for( i from 1 to N ) {
        for(j from 1 to N) {
            print graph[i][j]
        }
    }
    print "newline"
}
```

The pseudo code for `insertVertex(int * N)` which inserts a vertex to the adjacency matrix is as follows

```
void insertVertex(int * N) {
    Declare two integer arrays x and y.
    Declare t and i , the variables of type int.
    Calculate *N as *N+1.
    Read s the number of edges that are from existing vertices to the new verte
x.
    for(i from 1 to s)
        Read x[i].
    Read t the number of edges that are from new vertex to existing vertices,
    for(i from 1 to t) {
        Read y[i]
    for(i from 1 to *N) {
        Set graph[i][*N] to 0.
        Set graph[*N][i] to 0.
    }
    for(i from 1 to s) {
        if(x[i] is less than *N) {
            Set graph[x[i]][*N] to 1;
        }
        else {
            print "Invalid vertex."
        }
    }
    for(i from 1 to t) {
        if(y[i] is less than *N) {
            Set graph[*N][y[i]] to 1;
        }
        else {
            print "Invalid vertex."
        }
    }
    printf "After inserting vertex the adjacency matrix is : ".
    call : print(N); // Display the current status of the adjacency matrix.
}
```

The pseudo code for the function `void insertEdge(int * N)` which inserts an edge to the adjacency matrix is as follows.

```
 void insertEdge(int *N) {
    Declare two variable v1 and v2.
    Read v1 as the source vertex of the edge to be inserted.
    Read v2 as the destination vertex of the edge to be inserte
d.
    if(v1 is less than *N && v2 is less than *N) {
        Set graph[v1][v2] to = 1.;
    } else {
        print "Invalid vertex."
        return;
    }
    print "After inserting edge the adjacency matrix is : \n");
    call : print(N);
}
```

The pseudo code for the function `void deleteVertex(int * N)` which deletes a vertex from the adjacency matrix is as follows.

```
void deleteVertex(int *N) {
    Declare variables vd, i, j and k.
    if(*N is equal to 0) {
        print "Graph is empty."
        return;
    }
    Print "Enter the vertex to be deleted : "
    Reead the vertex in a varaible vd.
    if(vd is greater than *N) {
        Print "Invalid vertex."
        return;
    }
    Assign vd to a variable j.
    // Removing column in the adjacency matrix
    for(i from j to *N-1) {
        for(k from 1 to *N) {
            Assign graph[k][i+1] to graph[k][i].
        }
    }
    //Removing the row in the adjacency matrix.
    for(i from j to *N-1) {
        for(k from 1 to *N) {
            Assign graph[i+1][k] to graph[i][k].
        }
    }
    Reduce *N by 1.
    Print "After deleting vertex the adjacency matrix is : "
    call : print(N);
}
```

The pseudo code for the function `void deleteEdge(int * N)` which deletes an edge from the adjacency matrix is as follows.

```
void deleteEdge(int *N) {
    Declare toe variables v1 and v2.
    Read v1 which is the source vertex of the edge to be deleted"
    Read v2 which is the destination vertex of the edge to be deleted"
    if(v1 is less than of equal to *N and  && v2 is less than are oine <= *
N) {
        if(graph[v1][v2] is equal to 0) {
            print "Edge does not exist.";
            return;
        }
        set graph[v1][v2] to  = 0;
    } else {
        print "Invalid vertex.";
        return;
    }
    print "After deleting edge the adjacency matrix is "
    call : print(N);
}
```

Source Code:

GraphsMain1.c

```
#include<stdio.h>
#include<stdlib.h>
#include "GraphsAdjacencyMatrixDirectedGraph.c"

void main() {
   int x, op;
   int N,E,s,d,i,j;
   printf("Enter the number of vertices : ");
   scanf("%d",&N);
   printf("Enter the number of edges : ");
   scanf("%d",&E);
   for(i=1;i<=E;i++) {
      printf("Enter source : ");
      scanf("%d",&s);
      printf("Enter destination : ");
      scanf("%d",&d);
      if(s > N || d > N || s<=0 || d<=0) {
         printf("Invalid index. Try again.\n");
         i--;
         continue;
      } else {
         graph[s][d] = 1;
      }
   }
   while(1)
   {
      printf("1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjac
ency matrix 6.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:
               insertVertex(&N);
               break;
         case 2:
               insertEdge(&N);
               break;
         case 3:
               deleteVertex(&N);
               break;
         case 4:
               deleteEdge(&N);
               break;
         case 5:
               print(&N);
               break;
         case 6:
               exit(0);
      }
   }
}
```

GraphsAdjacencyMatrixDirectedGraph.c

```
int graph [20][20];
void print(int * N) {
```

```
   int i,j;
   if(*N == 0) {
      printf("Graph is empty.\n");
      return;
   }
   for(i=1;i<=*N;i++) {
      for(j=1;j<=*N;j++) {
         printf("%d\t",graph[i][j]);
      }
      printf("\n");
   }
}

void insertVertex(int * N) {
   int x[20],s,y[20],t,i=0;
   *N = *N + 1;
   printf("Enter the number edges from the existing vertices to new vertex : ");
   scanf("%d",&s);
   printf("Enter the source of each edge : ");
   for(i=1;i<=s;i++) {
      scanf("%d",&x[i]);
   }
   printf("Enter the number edges from the new vertex to existing vertices : ");
   scanf("%d",&t);
   printf("Enter the destination of each edge : ");
   for(i=1;i<=t;i++) {
      scanf("%d",&y[i]);
   }
   for(i=1;i<=*N;i++) {
      graph[i][*N] = 0;
      graph[*N][i] = 0;
   }
   for(i=1;i<=s;i++) {
      if(x[i]<=*N) {
         graph[x[i]][*N] = 1;
      }
      else {
         printf("Invalid vertex.\n");
      }
   }
   for(i=1;i<=t;i++) {
      if(y[i]<= *N) {
         graph[*N][y[i]] = 1;
      }
      else {
         printf("Invalid vertex.\n");
      }
   }
   printf("After inserting vertex the adjacency matrix is : \n");
   print(N);
}
void insertEdge(int *N) {
   int v1,v2;
   printf("Enter the source vertex of the edge : ");
   scanf("%d",&v1);
   printf("Enter the destination vertex of the edge : ");
   scanf("%d",&v2);
   if(v1 <= *N && v2 <= *N) {
```

```
      graph[v1][v2] = 1;
   } else {
      printf("Invalid vertex.\n");
      return;
   }
   printf("After inserting edge the adjacency matrix is : \n");
   print(N);
}
void deleteVertex(int *N) {
   int vd,i,j,k;
   if(*N == 0) {
      printf("Graph is empty.\n");
      return;
   }
   printf("Enter the vertex to be deleted : ");
   scanf("%d",&vd);
   if(vd > *N) {
      printf("Invalid vertex.\n");
      return;
   }
   j = vd;
   for(i = j ; i<= *N-1 ; i++) {
      for(k = 1 ; k <= *N ; k++ ) {
         graph[k][i] = graph [k][i+1];
      }
   }
   for(i = j ; i <= *N-1; i++) {
      for(k = 1 ; k <= *N ; k++) {
         graph[i][k] = graph [i+1][k];
      }
   }
   *N = *N - 1;
   printf("After deleting vertex the adjacency matrix is : \n");
   print(N);
}
void deleteEdge(int *N) {
   int v1,v2;
   printf("Enter the source vertex of the edge : ");
   scanf("%d",&v1);
   printf("Enter the destination vertex of the edge : ");
   scanf("%d",&v2);
   if(v1 <= *N && v2 <= *N) {
      if(graph[v1][v2] == 0) {
         printf("Edge does not exist.\n");
         return;
      }
      graph[v1][v2] = 0;
   } else {
      printf("Invalid vertex.\n");
      return;
   }
   printf("After deleting edge the adjacency matrix is : \n");
   print(N);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of vertices :  3
Enter the number of edges :  3
Enter source :  1
Enter destination :  2
Enter source :  1
Enter destination :  3
Enter source :  3
Enter destination :  2
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 5
Enter your option :  5
0       1       1       4
0       0       0       4
0       1       0       4
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 4
Enter your option :  4
Enter the source vertex of the edge :  3
Enter the destination vertex of the edge :  3
Edge does not exist. 4
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 4
Enter your option :  4
Enter the source vertex of the edge :  3
Enter the destination vertex of the edge :  2
After deleting edge the adjacency matrix is :  2
0       1       1       2
0       0       0       2
0       0       0       2
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 2
Enter your option :  2
Enter the source vertex of the edge :  2
Enter the destination vertex of the edge :  3
After inserting edge the adjacency matrix is :  3
0       1       1       3
0       0       1       3
0       0       0       3
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 3
Enter your option :  3
Enter the vertex to be deleted :  3
After deleting vertex the adjacency matrix is :  1
0       1        1
0       0        1
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 1
Enter your option :  1
Enter the number edges from the existing vertices to new vertex :  1
Enter the source of each edge :  1
Enter the number edges from the new vertex to existing vertices :  1
Enter the destination of each edge :  2
After inserting vertex the adjacency matrix is :  3
0       1       1       3
0       0       0       3
0       1       0       3
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 6
```

Test Case - 1

```
Enter your option :  3
Enter the vertex to be deleted :  3
After deleting vertex the adjacency matrix is :  3
0       1       3
0       0       3
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 3
Enter your option :  3
Enter the vertex to be deleted :  2
After deleting vertex the adjacency matrix is :  3
0       3
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 3
Enter your option :  3
Enter the vertex to be deleted :  1
After deleting vertex the adjacency matrix is :  3
Graph is empty. 3
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 3
Enter your option :  3
Graph is empty. 6
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency matrix 6.Exit 6
Enter your option :  6
```

S.No: 47	Exp. Name: ***Graph implementation - Adjacency list Directed graph***	Date:

Aim:

Implementation of directed graph and its operations using adjacent lists

The node of the adjacency list is defined as follows :

```
struct node {
    struct node *next;
    int vertex;
};
typedef struct node *GNODE;
```

Declaring the array of adjacent lists is as follows

```
GNODE graph[20];
```

The pseudo code for `void print(int *N)` which prints the adjacency list is as follows :

```
void print(int * N) {
    Declare an integer variable i and set it to 0.
    Declare a variable p of type GNODE.
    for(i from 1 to *N) {
        Assign graph[i] to p.
        if(p is not equal to NULL) {
            Print "[<i>]=>" i.e if i = 3 print "[3]=>"
        }
        else {
            continue;
        }
        while(p is not equal to NULL) {
            Print vertex field of p along with a trailing tab spa
ce.
        }
        Print newline
    }
}
```

The pseudo code for `void insertVertex(int *N)` which inserts a vertex into the adjacency list is as follows :

```
void insertVertex(int * N) {
    Declare two integer arrays x and y.
    Declare variables s, t, d and i.
    Declare two variables p and q of type GNODE.
    Increment *N.
    Print "Enter the number edges from the existing vertices to new vertex : "
    Read the number of edges into s
    for(i from 1 to s) {
        Read the sources vertices into x[i]
    }
    Print "Enter the number edges from the new vertex to existing vertices : "
    Read the number of edges into  t
    for(i from 1 to s) {
        Read the destination vertices into y[i]
    }
    for(i from 1 to s) {
        if(x[i] is less than *N) {
            Allocate memory to q.
            Set vertex field of q to *N.
            Set next field of q to NULL.
        }
        if(graph[x[i]] is equal to NULL) {
            Assign q to graph[x[i]]
        }
        else {
            Assign graph[x[i]] to p.
            while(next field of p not equal to NULL) {
                Assign next field of p to p.
            }
            Set q to p->next.
        }
        else {
            Print "Invalid vertex."
        }
    }
    for(i from 1 to t) {
        if(y[i] is less than or equal to *N) {
            Allocate memory to q.
            Assign y[i] as vertex field of q.
            Set next field of q to NULL.
            if(graph[*N] is equal to NULL) {
                Assign q to graph[*N].
            }
            else {
                Assign graph[*N] to p.
                while(next field of p not equal to NULL) {
                    Assign next field of p to p.
                }
                Assign q to next field of p.
            }
        }
        else {
            Print "Invalid vertex."
        }
    }
    printf "After inserting vertex the adjacency list is : "
```

```
    call : print(N);
}
```

The pseudo code for `insertEdge(int *N)` which inserts an edge into adjacency list is as follows :

```
void insertEdge(int *N) {
    Declare two variables v1 and v2.
    Declare two variables p and q of type GNODE.
    Printf "Enter the source vertex of the edge : "
    Read the source vertex into v1.
    Print "Enter the destination vertex of the edge : "
    Read the destination vertex into v2
    if( v1 and v2 are less than *N) {
        Allocate memory to q.
        Assign v2 to vertex field of q.
        Assign next field of q
        if(graph[v1] is equal to NULL) {
            Assign p to graph[v1].
        }
        else {
            Assign graph[v1] to p.
            while(p->next not  equal to NULL) {
                Assign next field of p to p.
            }
            Assign q to next field of p.
        }
    } else {
        print "Invalid vertex. "
        return;
    }
    print "After inserting edge the adjacency list is : "
    call : print(N);
}
```

The pseudo code for `void deleteVertex(int *N)` which deletes an element from the adjacency list is as follows :

```
void deleteVertex(int *N) {
    Declare four variables vd, i, j and k.
    Declare temp and prev of type GNDOE.
    if( *N is equal to 0 ) {
        print "Graph is empty."
        return.
    }
    Print "Enter the vertex to be deleted : "
    Read the vertex to be deleted in vb.
    if( vd is greater than top ) {
        print "Invalid vertex.\n";
        return;
    }
    Assign NULL to vd.
    Decrement *N.
    for (int i from 1 to *N ) {
        Assign graph[i]  to temp.
        if( temp not equal to NULL and vertex field of temp is equal to vd ) {
            Assign next field o f temp to graph[i].
            free ( temp )
            continue;
        }
        while( temp is not equal to NULL and vertex field of temp is equal to vd )
{
                Assign temp to prev.
                Assign next field of temp to temp.
        }
        if( temp not equal to NULL and vertex field of temp == vd) {
            Assign next field of temp to next field of prev.
            Free the memory occupied by them
            free(temp);
        }

    }
    Print "After deleting vertex the adjacency list is : ".
    call : print(N);
}
```

The pseudo code for `void deleteEdge(int *N)` which deletes an edge from the adjacency matrix is as follows :

```
void deleteEdge(int *N) {
    Declare two integer variables v1 and v2.
    Declare two variables temp and prev of type GNODE
    Print "Enter the source vertex of the edge : ";
    Read the source vertex into the v1
    Printf "Enter the destination vertex of the edge : "
    Read the destination into a variable v2;
    Assign graph[v1] to temp
    if( vertex field of temp is v2 ) {
        assign the next field of temp to graph[v1]
        free temp
    }
    while( temp not equal to NULL and vertex field of temp != v
2) {
        Assign temp to prev
        Assign next field of temp to temp.
    }
    if( vertex field of temp is v2 ) {
        Assign next field of temp to next field of prev.
        Deallocate the memory occupied by temp
        free(temp);
    }
    Print ("After deleting edge the adjacency list is : \n").
    call : print(N);
}
```

Source Code:

GraphMain3.c

```
#include<stdio.h>
#include<stdlib.h>
#include "GraphsAdjacencyListDirectedGraph.c"

void main() {
   int x, op;
   int N,E,s,d,i,j;
   GNODE p,q;
   printf("Enter the number of vertices : ");
   scanf("%d",&N);
   printf("Enter the number of edges : ");
   scanf("%d",&E);
   for(i=1;i<=E;i++) {
      printf("Enter source : ");
      scanf("%d",&s);
      printf("Enter destination : ");
      scanf("%d",&d);
      q=(GNODE)malloc(sizeof(struct node));
      q->vertex=d;
      q->next=NULL;
      if(graph[s]==NULL)
         graph[s]=q;
      else {
         p=graph[s];
         while(p->next!=NULL)
            p=p->next;
         p->next=q;
```

```
      }
   }
   while(1) {
      printf("1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjac
ency list 6.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch(op) {
         case 1:
               insertVertex(&N);
               break;
         case 2:
               insertEdge(&N);
               break;
         case 3:
               deleteVertex(&N);
               break;
         case 4:
               deleteEdge(&N);
               break;
         case 5:
               print(&N);
               break;
         case 6:
               exit(0);
      }
   }
}
```

GraphsAdjacencyListDirectedGraph.c

```
struct node {
   struct node *next;
   int vertex;
};
typedef struct node *GNODE;
GNODE graph[20];

void print(int * N) {
   int i=1;
   GNODE p;
   for(i=1;i<=*N;i++) {
      p = graph[i];
      if(p != NULL)
         printf("%d=>",i);
      else
         continue;
      while(p != NULL) {
         printf("%d\t",p->vertex);
         p = p->next;

      }
      printf("\n");
   }
}
```

```
void insertVertex(int * N) {
   int x[20],y[20];
   int s,t,d,i;
   GNODE p,q;
   *N = *N + 1;
   printf("Enter the number edges from the existing vertices to new vertex : ");
   scanf("%d",&s);
   printf("Enter the source of each edge : ");
   for(i=1;i<=s;i++) {
      scanf("%d",&x[i]);
   }
   printf("Enter the number edges from the new vertex to existing vertices : ");
   scanf("%d",&t);
   printf("Enter the destination of each edge : ");
   for(i=1;i<=t;i++) {
      scanf("%d",&y[i]);
   }
   for(i=1;i<=s;i++) {
      if(x[i]<=*N) {
         q=(GNODE)malloc(sizeof(struct node));
         q->vertex=*N;
         q->next=NULL;
         if(graph[x[i]]==NULL)
            graph[x[i]]=q;
         else {
            p=graph[x[i]];
            while(p->next!=NULL)
               p=p->next;
            p->next=q;
         }
      }
      else {
         printf("Invalid vertex.\n");
      }
   }
   for(i=1;i<=t;i++) {
      if(y[i]<= *N) {
         q = (GNODE)malloc(sizeof(struct node));
         q->vertex = y[i];
         q->next = NULL;
         if(graph[*N] == NULL)
            graph[*N] = q;
         else {
            p = graph[*N];
            while(p->next != NULL)
               p = p->next;
            p->next=q;
         }
      }
      else {
         printf("Invalid vertex.\n");
      }
   }
   printf("After inserting vertex the adjacency list is : \n");
   print(N);
}

void insertEdge(int *N) {
```

```
   int v1,v2;
   GNODE p,q;
   printf("Enter the source vertex of the edge : ");
   scanf("%d",&v1);
   printf("Enter the destination vertex of the edge : ");
   scanf("%d",&v2);
   if(v1 <= *N && v2 <= *N) {
      q=(GNODE)malloc(sizeof(struct node));
      q->vertex=v2;
      q->next=NULL;
      if(graph[v1]==NULL)
         graph[v1]=q;
        else {
         p=graph[v1];
         while(p->next!=NULL)
            p=p->next;
         p->next=q;
      }
   } else {
      printf("Invalid vertex.\n");
      return;
   }
   printf("After inserting edge the adjacency list is : \n");
   print(N);
}

void deleteVertex(int *N) {
   int vd,i,j,k;
   GNODE temp,prev;
   if(*N == 0) {
      printf("Graph is empty.\n");
      return;
   }
   printf("Enter the vertex to be deleted : ");
   scanf("%d",&vd);
   if(vd > *N) {
      printf("Invalid vertex.\n");
      return;
   }
   graph[vd] = NULL;
   *N = *N -1;
   for(i=1;i<=*N;i++) {
      temp = graph[i];
      if(temp!=NULL && temp->vertex == vd) {
         graph[i]= temp->next;
         free(temp);
         continue;
      }
      while(temp!= NULL && temp->vertex!= vd) {
         prev = temp;
         temp = temp->next;
      }

      if(temp != NULL && temp->vertex == vd) {
         prev->next = temp->next;
         free(temp);
      }
   }
```

```
   printf("After deleting vertex the adjacency list is : \n");
   print(N);
}

void deleteEdge(int *N) {
   int v1,v2;
   GNODE temp,prev;
   printf("Enter the source vertex of the edge : ");
   scanf("%d",&v1);
   printf("Enter the destination vertex of the edge : ");
   scanf("%d",&v2);
   temp = graph[v1];
   if(temp->vertex == v2) {
      graph[v1]= temp->next;
      free(temp);
   }
   while(temp!= NULL && temp->vertex!= v2) {
      prev = temp;
      temp = temp->next;
   }
   if(temp->vertex == v2) {
      prev->next = temp->next;
      free(temp);
   }
   printf("After deleting edge the adjacency list is : \n");
   print(N);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices : 3
Enter the number of edges : 3
Enter source : 1
Enter destination : 2
Enter source : 1
Enter destination : 3
Enter source : 3
Enter destination : 2
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency list 6.Exit 5
Enter your option : 5
1=>2 3 2
3=>2 2
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency list 6.Exit 2
Enter your option : 2
Enter the source vertex of the edge : 3
Enter the destination vertex of the edge : 2
After inserting edge the adjacency list is : 3
1=>2 3 3
3=>2 2 3
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency list 6.Exit 3
Enter your option : 3

Test Case - 1

```
Enter the vertex to be deleted :  3
After deleting vertex the adjacency list is :  3
1=>2    3
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency list 6.Exit 3
Enter your option :  3
Enter the vertex to be deleted :  2
After deleting vertex the adjacency list is :  3
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency list 6.Exit 3
Enter your option :  3
Enter the vertex to be deleted :  1
After deleting vertex the adjacency list is :  6
1.Insert vertex 2.Insert edge 3.Delete vertex 4.Delete edge 5.Print adjacency list 6.Exit 6
Enter your option :  6
```

S.No: 48	Exp. Name: ***Graph traversals implementation - Depth First Search***	Date:

Aim:

Graph traversals

Graph traversal is technique used for searching a vertex in a graph. The graph traversal is decide the order of visiting of the vertices in the search process. A graph traversal finds the edges to be used in the search process without creating loops that means using graph traversal we visit all vertices of graph without getting into looping path.

There are two major graph traversal techniques

- Depth First Search (DFS)
- Breadth First Search (BFS)

Depth First Search

DFS traversal of a graph, produces a **spanning tree** as final result. Spanning tree is a graph without any loops. We use recursion or stack data structure with maximum size of total number of vertices in the graph to implement DFS traversal of a graph.

Implementation of Graph traversal - Depth First Search

Click on the 👁 Live Demo button to know about the **DFS**.

Consider the graph is represented using the Adjacency list method. The node of the Adjacency list is defined as follows :

```
struct node {
    struct node *next;
    int vertex;
};
typedef struct node * GNODE;
```

The graph array is declared as follows :

```
GNODE graph[20];
```

A visited array is used to maintain the list of all the nodes that are already visited. This visited status is useful to identify the graph vertices that are already visited and thus avoiding falling into loops.

```
int visited[20];
```

The pseudo code for `DFS (int i)` which performs a DFS traversal.

```
void DFS(int i) {
    Declare a variable p of type GNODE.
    Print "%d" followed by tab space.
        Assign graph[i] to p.
    Assign graph[i]  to p
    Set visited[i] = 1
     while( p not equal to NULL ) {
        Assign vertex field of p to i
        if( visited of i^th index is not 1 ) {
            call : DFS(i) // Recursive calling
        }
            Assign next field of p to p
    }
}
```


Source Code:

GraphsDFS.c

```
#include<stdio.h>
#include<stdlib.h>
struct node {
    struct node *next;
    int vertex;
};
typedef struct node * GNODE;
GNODE graph[20];
int visited[20];
int n;

void DFS(int i) {
    GNODE p;
    printf("\n%d",i);
    p=graph[i];
    visited[i]=1;
    while(p!=NULL) {
       i=p->vertex;
       if(!visited[i])
            DFS(i);
        p=p->next;
    }

}

void main() {
    int N,E,i,s,d,v;
    GNODE q,p;
    printf("Enter the number of vertices : ");
   scanf("%d",&N);
   printf("Enter the number of edges : ");
   scanf("%d",&E);
   for(i=1;i<=E;i++) {
      printf("Enter source : ");
      scanf("%d",&s);
      printf("Enter destination : ");
      scanf("%d",&d);
      q=(GNODE)malloc(sizeof(struct node));
```

```
      q->vertex=d;
      q->next=NULL;
      if(graph[s]==NULL)
         graph[s]=q;
        else {
         p=graph[s];
         while(p->next!=NULL)
               p=p->next;
         p->next=q;
      }
   }
    for(i=0;i<n;i++)
        visited[i]=0;
    printf("Enter Start Vertex for DFS : ");
    scanf("%d", &v);
    printf("DFS of graph : ");
    DFS(v);
    printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices : 6
Enter the number of edges : 7
Enter source : 1
Enter destination : 2
Enter source : 1
Enter destination : 4
Enter source : 4
Enter destination : 2
Enter source : 2
Enter destination : 3
Enter source : 4
Enter destination : 5
Enter source : 4
Enter destination : 3
Enter source : 3
Enter destination : 6
Enter Start Vertex for DFS : 1
DFS of graph :
1
2
3
6
4
5

Test Case - 2
User Output

Test Case - 2
Enter the number of vertices : 5
Enter the number of edges : 5
Enter source : 1
Enter destination : 2
Enter source : 1
Enter destination : 4
Enter source : 4
Enter destination : 2
Enter source : 2
Enter destination : 3
Enter source : 4
Enter destination : 5
Enter Start Vertex for DFS : 1
DFS of graph :
1
2
3
4
5

S.No: 49	Exp. Name: ***Graph traversals implementation - Breadth First Search***	Date:

Aim:

Breadth First Search

BFS traversal of a graph, produces a spanning tree as final result. Spanning Tree is a graph without any loops. We use Queue data structure with maximum size of total number of vertices in the graph to implement BFS traversal of a graph.

Click on the Live Demo button to know about the **BFS**.

Implementation of Graph traversal - Breadth First Search

Consider the graph is represented using the Adjacency list method. The node of the Adjacency list is defined as follows :

```
struct node {
    struct node *next;
    int vertex;
};
typedef struct node * GNODE;
```

The graph array is declared as follows :

```
GNODE graph[20];
```

A visited array is used to maintain the list of all the nodes that are already visited. This visited status is useful to identify the graph vertices that are already visited and thus avoiding falling into loops.

```
int visited[20];
```

A queue implementation using arrays is used for queue operations. The queue array along with rear and front variables are declared as follows:

```
#define MAX 99
int queue[MAX], front = -1, rear = -1;
```

The pseudo code for `int isEmptyQueue()` , which identifies wether a queue is empty or not is defined as follows.
Note: This function returns `1` id `CSE` is empty else if return `0`

```
int isEmptyQueue() {
    if(front is equal to -1 or front is greater than rear)
        return 1;
    else
        return 0;
}
```

The pseudo code for `void insertQueue(int vertex)` which inserts a vertex into the queue is defined as follows:

```
void insertQueue(int vertex) {
    if( rear is equal to MAX-1 )
        print "Queue Overflow."
    else {
        if( front is equal to -1 )
            Initialize front with 0;
        Increment rear
        Assign vertex to queue[rear];
    }
}
```

The pseudo code for `int deleteQueue()` which deletes the element from the queue is defined as follows:

```
int deleteQueue() {
    Declare a variable deleteItem
    if( front is equal to -1 or front is greater than rear
) {
        print "Queue Underflow."
        exit(0);
    }
    Assign queue[front] as deleteItem.
    Increment front.
    return deleteItem
}
```

The pseudo code for `BFS (int v)` which performs a BFS traversal as below

```
void BFS(int v) {
    Declare a variable w.
    call : InsertQueue( v0 ) {
        v = call: deleteQueue(v);
        print the vertex value.
    }
        Set visited[v] to 1.
    Assign graph[v] to a variable g of type
 GNODE
    for(;g!=NULL;g=g->next) {
        Assign vertex field of g to w
        if( visited[w] is equal to 0 ) {
            call : insertQueue(w);
                Set visited[w] to 1;
                print w.
            }
        }
    }
    print newline;
}
```

Source Code:

GraphsBFS.c

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 99
struct node {
   struct node *next;
```

```
   int vertex;
};
typedef struct node * GNODE;
GNODE graph[20];
int visited[20];
int queue[MAX], front = -1,rear = -1;
int n;

void insertQueue(int vertex) {
   if(rear == MAX-1)
      printf("Queue Overflow.\n");
   else {
      if(front == -1)
         front = 0;
      rear = rear+1;
      queue[rear] = vertex ;
   }
}

int isEmptyQueue() {
   if(front == -1 || front > rear)
      return 1;
   else
      return 0;
}

int deleteQueue() {
   int deleteItem;
   if(front == -1 || front > rear) {
      printf("Queue Underflow\n");
      exit(1);
   }
   deleteItem = queue[front];
   front = front+1;
   return deleteItem;
}

void BFS(int v) {
   int w;
   insertQueue(v);
   while(!isEmptyQueue()) {
      v = deleteQueue( );
      printf("\n%d",v);
      visited[v]=1;
      GNODE g = graph[v];
      for(;g!=NULL;g=g->next) {
         w=g->vertex;
         if(visited[w]==0) {
            insertQueue(w);
            visited[w]=1;
         }
      }
   }
}

void main() {
   int N, E, s, d, i, j, v;
   GNODE p, q;
```

```
   printf("Enter the number of vertices : ");
   scanf("%d",&N);
   printf("Enter the number of edges : ");
   scanf("%d",&E);
   for(i=1;i<=E;i++) {
      printf("Enter source : ");
      scanf("%d",&s);
      printf("Enter destination : ");
      scanf("%d",&d);
      q=(GNODE)malloc(sizeof(struct node));
      q->vertex=d;
      q->next=NULL;
      if(graph[s]==NULL) {
         graph[s]=q;
      } else {
         p=graph[s];
         while(p->next!=NULL)
            p=p->next;
         p->next=q;
      }
   }
   for(i=1;i<=n;i++)
      visited[i]=0;
   printf("Enter Start Vertex for BFS : ");
   scanf("%d", &v);
   printf("BFS of graph : ");
   BFS(v);
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices : 5
Enter the number of edges : 5
Enter source : 1
Enter destination : 2
Enter source : 1
Enter destination : 4
Enter source : 4
Enter destination : 2
Enter source : 2
Enter destination : 3
Enter source : 4
Enter destination : 5
Enter Start Vertex for BFS : 1
BFS of graph :
1
2
4
3
5

S.No: 50	Exp. Name: ***Minimum spanning tree - Kruskal's Algorithm***	Date:

Aim:

Spanning Tree

Given an undirected and connected graph **G = (V, E)** , a spanning tree of the graph **G** is a tree that spans all vertices of **G** (that is, it includes every vertex of **G**) and is a subgraph of **G** (every edge in the tree belongs to **G**).

Cost of spanning tree

The **cost** of the spanning tree is the sum of the weights of all the edges in the spanning tree.

There can be many spanning trees.

Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees.

There also can be many minimum spanning trees.

Minimum spanning tree has many applications in the design of networks. It is used in algorithms like travelling salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching.

Other practical applications are:

- Cluster Analysis
- Handwriting recognition
- Image segmentation

There are two famous algorithms for finding the Minimum Spanning Tree(**MST**):

1. Kruskal's Algorithm
2. Prim's Algorithm

Kruskal's Algorithm

Kruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy (https://en.wikipedia.org/wiki/Greedy_algorithm) approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.

Algorithm Steps:

1. Sort the graph edges with respect to their weights.
2. Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
3. Only add edges which doesn't form a cycle , edges which connect only disconnected components.

In Kruskal's algorithm, the timing complexity is **O(E log V)** where **E** is the number of edges and **V** is the number of vertices.

The pseudo code for the minimum spanning tree Kruskal's algorithm is as follows :

`ne` - number of edges visited.
`n` - number of edges.
`cost[][]` - An array that stores the weights of the edges.
`parent[]` - An array that stores immediate parents of each vertex. Initially all values are set to `0`

```
void kruskal() {
    while( ne is less than n ) {
        Set min to 999.
        for each i from 1 to n {
            for each j from 1 to n {
                if( cost[i][j] is less than min ) {
                    Set min to cost[i][j]. [weight]
                    Set a = u = i. [source]
                    Set b = v = j. [destination]
                }
            }
        }
        u = call : find( u );
        v = call : find( v );
        if(call : uni( u, v )) {
            print "Edge cost from <source> to <destination> : <weight>
            mincost = mincost + min;
            Increment ne
        }
        Set cost[a][b] =cost[b][a]= 999; // Removing the edge for the next iteratio
n.
    }
    print "Minimum cost of spanning tree = < mincost >"
}
```

The pseudo code for the helper function `int find(int i)` is as follows:

```
int find (int i) {
    while( parent[i] )
        Set i =
 parent[i]
    return i;
}
```

The pseudo code for the helper function `int uni(int i, int j)` is as follows:

```
int uni(int i, int j) {
    if( i is not equal to j
) {
        Set parent[j] = i;
        return 1;
    }
    return 0;
}
```

Click on the Live Demo to explore Kruskal's algorithm.

Fill the function `void kruskal()` in the following code.

The **sample output** of the minimum spanning tree is as follows:

```
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 2 : 1
Edge cost from 4 to 5 : 2
Edge cost from 2 to 5 : 3
Edge cost from 2 to 3 : 5
Minimum cost of spanning tree = 11
```

Source Code:

KruskalSpanningTree.c

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,e,s,d,w,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int i) {
   while(parent[i])
      i=parent[i];
   return i;
}
int uni(int i,int j) {
   if(i!=j) {
      parent[j]=i;
      return 1;
   }
   return 0;
}
void kruskal() {
   while(ne < n) {
      for(i=1,min=999;i<=n;i++) {
         for(j=1;j <= n;j++) {
            if(cost[i][j] < min) {
               min=cost[i][j];
               a=u=i;
               b=v=j;
            }
         }
      }
      u=find(u);
      v=find(v);
      if(uni(u,v)) {
         printf("Edge cost from %d to %d : %d\n",a,b,min);
         mincost +=min;
         ne++;
      }
      cost[a][b]=cost[b][a]=999;
   }
   printf("Minimum cost of spanning tree = %d\n",mincost);
}

void main() {
   printf("Enter the number of vertices : ");
   scanf("%d",&n);
   printf("Enter the number of edges : ");
   scanf("%d",&e);
   for(i=1;i<=e;i++) {
```

```
      printf("Enter source : ");
      scanf("%d",&s);
      printf("Enter destination : ");
      scanf("%d",&d);
      printf("Enter weight : ");
      scanf("%d",&w);
      if(s<=0 || d<=0 || s> n || d > n || w < 0 ) {
         printf("Invalid data.Try again.\n");
         i--;
         continue;
      }
      cost[s][d]=w;
   }
   for(i=1;i<=n;i++) {
      for(j=1;j<=n;j++) {
         if(cost[i][j]==0)
            cost[i][j]=999;
      }
   }
   printf("The edges of Minimum Cost Spanning Tree are : \n");
   kruskal();
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices : 7
Enter the number of edges : 9
Enter source : 2
Enter destination : 7
Enter weight : 14
Enter source : 1
Enter destination : 2
Enter weight : 28
Enter source : 1
Enter destination : 6
Enter weight : 10
Enter source : 5
Enter destination : 6
Enter weight : 25
Enter source : 4
Enter destination : 5
Enter weight : 22
Enter source : 4
Enter destination : 7
Enter weight : 18
Enter source : 2
Enter destination : 3
Enter weight : 16
Enter source : 3
Enter destination : 4
Enter weight : 12

Test Case - 1
Enter source : 5
Enter destination : 7
Enter weight : 24
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 6 : 10
Edge cost from 3 to 4 : 12
Edge cost from 2 to 7 : 14
Edge cost from 2 to 3 : 16
Edge cost from 4 to 5 : 22
Edge cost from 5 to 6 : 25
Minimum cost of spanning tree = 99

Test Case - 2
User Output
Enter the number of vertices : 7
Enter the number of edges : 11
Enter source : 1
Enter destination : 2
Enter weight : 7
Enter source : 2
Enter destination : 3
Enter weight : 8
Enter source : 1
Enter destination : 4
Enter weight : 5
Enter source : 2
Enter destination : 4
Enter weight : 9
Enter source : 2
Enter destination : 5
Enter weight : 7
Enter source : 3
Enter destination : 5
Enter weight : 5
Enter source : 4
Enter destination : 5
Enter weight : 15
Enter source : 4
Enter destination : 6
Enter weight : 6
Enter source : 5
Enter destination : 6
Enter weight : 8
Enter source : 5
Enter destination : 7
Enter weight : 9
Enter source : 6
Enter destination : 7
Enter weight : 11
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 4 : 5
Edge cost from 3 to 5 : 5

Test Case - 2
Edge cost from 4 to 6 : 6
Edge cost from 1 to 2 : 7
Edge cost from 2 to 5 : 7
Edge cost from 5 to 7 : 9
Minimum cost of spanning tree = 39

S.No: 51	Exp. Name: ***Minimum spanning tree - Prim's algorithm***	Date:

Aim:

Prim's Algorithm

Prim’s Algorithm also a greedy algorithm to find the minimum spanning tree.

In Prim’s Algorithm, we start with an empty spanning tree and then we grow the spanning tree from a starting position. Unlike an edge in Kruskal's, we add vertex to the growing spanning tree in Prim's.

The pseudo code for the Prim's algorithm is as follows :
`ne` - number of edges visited.
`n` - number of edges.
`cost[][]` - An array that stores the weights of the edges.
`visited []` - An array with one value per vertex, `1` represents the vertex is visited and `0` represents it is not visited.

```
void prims() {
    Set visited[1] to 1
    while( ne is less than n) {
        Set min to 999.
        for each ( i from 1 to n) {
            if(visited [ i ] is equal to 1) {
                for each ( j from 1 to n) {
                    if( visited[j] is equal to 0 and cost[i][j] is less than min
) {
                        Assign cost[i][j] to min.
                        Set a = u = i.
                        Set b = v = j.
                    }
                }
            }
        }
        print "Edge cost from <source> to <destination> : <weight>
        Increment ne.
        Set mincost = mincost + cost[a][b].
        Set visited[b] to 1.
        Set cost[a][b] and cost[b][a] to  999.
    }
    print "Minimum cost of spanning tree = <mincost>"
}
```

Source Code:

PrimsMinimumSpanningTree.c

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1,e,s,d,w;
int visited[10]={0},min,mincost=0,cost[10][10];
void prims() {
   visited[1]=1;
   while(ne < n) {
      min = 999;
      for(i=1;i<=n;i++) {
```

```
         if(visited[i] == 1) {
            for(j=1;j<=n;j++) {
               if(visited[j] == 0 && cost[i][j] < min ) {
                  min = cost[i][j];
                  a = i;
                  b = j;
               }
            }
         }
      }
      printf("Edge cost from %d to %d : %d\n",a,b,cost[a][b]);
      ne++;
      mincost+=cost[a][b];
      visited[b]=1;
      cost[a][b]=cost[b][a]=999;
   }
   printf("Minimum cost of spanning tree = %d\n",mincost);
}
void main() {
   printf("Enter the number of vertices : ");
   scanf("%d",&n);
   printf("Enter the number of edges : ");
   scanf("%d",&e);
   for(i=1;i<=e;i++) {
      printf("Enter source : ");
      scanf("%d",&s);
      printf("Enter destination : ");
      scanf("%d",&d);
      printf("Enter weight : ");
      scanf("%d",&w);
      if(s<=0 || d<=0 || s> n || d > n || w < 0 ) {
         printf("Invalid data.Try again.\n");
         i--;
         continue;
      }
      cost[s][d]=w;
      cost[d][s]=w;
   }
   for(i=1;i<=n;i++) {
       for(j=1;j<=n;j++) {
         if(cost[i][j]==0)
             cost[i][j]=999;
       }
   }
   printf("The edges of Minimum Cost Spanning Tree are : \n");
   prims();
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices : 7
Enter the number of edges : 11

Test Case - 1
Enter source : 1
Enter destination : 2
Enter weight : 7
Enter source : 2
Enter destination : 3
Enter weight : 8
Enter source : 1
Enter destination : 4
Enter weight : 5
Enter source : 2
Enter destination : 4
Enter weight : 9
Enter source : 2
Enter destination : 5
Enter weight : 7
Enter source : 3
Enter destination : 5
Enter weight : 5
Enter source : 4
Enter destination : 5
Enter weight : 15
Enter source : 4
Enter destination : 6
Enter weight : 6
Enter source : 5
Enter destination : 6
Enter weight : 8
Enter source : 5
Enter destination : 7
Enter weight : 9
Enter source : 6
Enter destination : 7
Enter weight : 11
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 4 : 5
Edge cost from 4 to 6 : 6
Edge cost from 1 to 2 : 7
Edge cost from 2 to 5 : 7
Edge cost from 5 to 3 : 5
Edge cost from 5 to 7 : 9
Minimum cost of spanning tree = 39

Test Case - 2
User Output
Enter the number of vertices : 7
Enter the number of edges : 9
Enter source : 2
Enter destination : 7
Enter weight : 14
Enter source : 1
Enter destination : 2
Enter weight : 28

Test Case - 2
Enter source : 1
Enter destination : 6
Enter weight : 10
Enter source : 5
Enter destination : 6
Enter weight : 25
Enter source : 4
Enter destination : 5
Enter weight : 22
Enter source : 4
Enter destination : 7
Enter weight : 18
Enter source : 2
Enter destination : 3
Enter weight : 16
Enter source : 3
Enter destination : 4
Enter weight : 12
Enter source : 5
Enter destination : 7
Enter weight : 24
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 6 : 10
Edge cost from 6 to 5 : 25
Edge cost from 5 to 4 : 22
Edge cost from 4 to 3 : 12
Edge cost from 3 to 2 : 16
Edge cost from 2 to 7 : 14
Minimum cost of spanning tree = 99

Test Case - 3
User Output
Enter the number of vertices : 5
Enter the number of edges : 7
Enter source : 1
Enter destination : 2
Enter weight : 2
Enter source : 1
Enter destination : 3
Enter weight : 5
Enter source : 2
Enter destination : 3
Enter weight : 8
Enter source : 2
Enter destination : 4
Enter weight : 4
Enter source : 3
Enter destination : 4
Enter weight : 5
Enter source : 3
Enter destination : 5
Enter weight : 6

Test Case - 3
Enter source : 4
Enter destination : 5
Enter weight : 4
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 2 : 2
Edge cost from 2 to 4 : 4
Edge cost from 4 to 5 : 4
Edge cost from 1 to 3 : 5
Minimum cost of spanning tree = 15

Test Case - 4
User Output
Enter the number of vertices : 5
Enter the number of edges : 7
Enter source : 1
Enter destination : 2
Enter weight : 2
Enter source : 1
Enter destination : 3
Enter weight : 5
Enter source : 2
Enter destination : 3
Enter weight : 8
Enter source : 2
Enter destination : 4
Enter weight : 4
Enter source : 3
Enter destination : 5
Enter weight : 6
Enter source : 4
Enter destination : 5
Enter weight : 4
Enter source : 3
Enter destination : 4
Enter weight : 5
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 2 : 2
Edge cost from 2 to 4 : 4
Edge cost from 4 to 5 : 4
Edge cost from 1 to 3 : 5
Minimum cost of spanning tree = 15

S.No: 52	Exp. Name: ***Dijkastras Shortest path algorithm***	Date:

Aim:

Given a graph G and source vertex S, Dijkstra's shortest path algorithm is used to find shortest paths from source S to all vertices in the given graph.
Dijkstra algorithm is also called single source shortest path algorithm. It is based on greedy technique. Little variation in the algorithm can find the shortest path from the source nodes to all the other nodes in the graph.

Sample Input and Output:

```
Enter the number of vertices : 4
Enter the number of edges : 5
Enter source : 1
Enter destination : 2
Enter weight : 4
Enter source : 1
Enter destination : 4
Enter weight : 10
Enter source : 1
Enter destination : 3
Enter weight : 6
Enter source : 2
Enter destination : 4
Enter weight : 5
Enter source : 3
Enter destination : 4
Enter weight : 2
Enter the source :1
Node    Distance        Path
  2         4   2<-1
  3         6   3<-1
  4         8   4<-3<-1
```

Source Code:

Dijkstras.c

```
#include <limits.h>
#include <stdio.h>
#define MAX 20
int V,E;
int graph[MAX][MAX];
#define INFINITY 99999
void dijkstra(int G[MAX][MAX],int n,int startnode) {
   int cost[MAX][MAX],distance[MAX],pred[MAX];
   int visited[MAX],count,mindistance,nextnode,i,j;
   for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
         if(G[i][j]==0)
            cost[i][j]=INFINITY;
         else
            cost[i][j]=G[i][j];
   for(i=1;i<=n;i++) {
      distance[i]=cost[startnode][i];
```

```
      pred[i]=startnode;
      visited[i]=0;
   }
   distance[startnode]=0;
   visited[startnode]=1;
   count=1;
   while(count<n-1){
      mindistance=INFINITY;
      for(i=1;i<=n;i++)
         if(distance[i]<mindistance&&!visited[i]) {
            mindistance=distance[i];
            nextnode=i;
         }
         visited[nextnode]=1;
         for(i=1;i<=n;i++)
            if(!visited[i])
               if(mindistance+cost[nextnode][i]<distance[i])
               {
                  distance[i]=mindistance+cost[nextnode][i];
                  pred[i]=nextnode;
               }
      count++;
   }
   printf("Node\tDistance\tPath\n");
   for(i=1;i<=n;i++)
      if(i!=startnode)
      {
         if(distance[i] == INFINITY) {
            printf("%4d\t%8s",i,"INF");
            printf("\tNO PATH\n");
         }
         else {
            printf("%4d\t%8d",i,distance[i]);
            printf("\t%d",i);
            j=i;
            do
            {
               j=pred[j];
               printf("<-%d",j);
            }while(j!=startnode);
            printf("\n");
         }
   }
}
int main() {
   int s,d,w,i,j;
   printf("Enter the number of vertices : ");
   scanf("%d",&V);
   printf("Enter the number of edges : ");
   scanf("%d",&E);
   for(i = 1 ; i <= V; i++) {
      for(j=1; j <= V; j++ ) {
         graph[i][i] = 0;
      }
   }
   for(i=1;i<=E;i++) {
      printf("Enter source : ");
      scanf("%d",&s);
```

```
      printf("Enter destination : ");
      scanf("%d",&d);
      printf("Enter weight : ");
      scanf("%d",&w);
      if(s > V || d > V || s<=0 || d<=0) {
         printf("Invalid index. Try again.\n");
         i--;
         continue;
      } else {
         graph[s][d] = w;
      }
   }
   printf("Enter the source :");
   scanf("%d",&s);
   dijkstra(graph, V,s);
   return 0;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices : 4
Enter the number of edges : 5
Enter source : 1
Enter destination : 2
Enter weight : 4
Enter source : 1
Enter destination : 4
Enter weight : 10
Enter source : 1
Enter destination : 3
Enter weight : 6
Enter source : 2
Enter destination : 4
Enter weight : 5
Enter source : 3
Enter destination : 4
Enter weight : 2
Enter the source :1

Node	Distance	Path
2	4	2<-1
3	6	3<-1
4	8	4<-3<-1

Test Case - 2
User Output
Enter the number of vertices : 5
Enter the number of edges : 6
Enter source : 1
Enter destination : 2

Test Case - 2

```
Enter weight :  2
Enter source :  1
Enter destination :  5
Enter weight :  3
Enter source :  2
Enter destination :  4
Enter weight :  4
Enter source :  2
Enter destination :  3
Enter weight :  7
Enter source :  4
Enter destination :  3
Enter weight :  2
Enter source :  5
Enter destination :  4
Enter weight :  1
Enter the source : 2
Node    Distance        Path
   1         INF        NO PATH
   3           6        3<-4<-2
   4           4        4<-2
   5         INF        NO PATH
```

Test Case - 3

User Output

```
Enter the number of vertices :  4
Enter the number of edges :  5
Enter source :  1
Enter destination :  2
Enter weight :  4
Enter source :  3
Enter destination :  2
Enter weight :  5
Enter source :  4
Enter destination :  1
Enter weight :  1
Enter source :  4
Enter destination :  2
Enter weight :  3
Enter source :  4
Enter destination :  3
Enter weight :  8
Enter the source : 1
Node    Distance        Path
   2           4        2<-1
   3         INF        NO PATH
   4         INF        NO PATH
```

S.No: 53	Exp. Name: ***Write a C program to sort the given elements using Heap sort***	Date:

Aim:

Write a program to `sort` (`ascending order`) the given elements using `heap sort` technique.

At the time of execution, the program should print the message on the console as:

```
Enter array size :
```

For example, if the user gives the **input** as:

```
Enter array size : 5
```

Next, the program should print the following message on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 34 67 12 45 22
```

then the program should **print** the result as:

```
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

HeapSortMain.c

```
#include <stdio.h>
#include "HeapSortFunctions.c"
void main() {
   int arr[15], i, n;
   printf("Enter array size : ");
   scanf("%d", &n);
   printf("Enter %d elements : ", n);
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
   printf("Before sorting the elements are : ");
   display(arr, n);
   heapsort(arr,n);
   printf("After sorting the elements are : ");
   display(arr, n);
}
```

HeapSortFunctions.c

```
void display(int arr[15], int n) {
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}
```

```
void heapify(int arr[], int n, int i)  {
   int largest = i;
   int l = 2*i + 1;
   int r = 2*i + 2;
   int temp;
   if (l < n && arr[l] > arr[largest])
      largest = l;
   if (r < n && arr[r] > arr[largest])
      largest = r;
    if (largest != i) {
      temp = arr[i];
      arr[i] = arr[largest];
      arr[largest] = temp;
      heapify(arr, n, largest);
   }
}
void heapsort(int arr[], int n) {
   int i,temp;
   for(i = n/2-1; i >=0 ; i--) {
      heapify(arr,n,i);
   }
   for(i = n-1; i >= 0; i--) {
      temp = arr[0];
      arr[0] = arr[i];
      arr[i] = temp;
      heapify(arr,i,0);
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size : 5
Enter 5 elements : 23 54 22 44 12
Before sorting the elements are : 23 54 22 44 12
After sorting the elements are : 12 22 23 44 54

Test Case - 2
User Output
Enter array size : 6
Enter 6 elements : 12 65 23 98 35 98
Before sorting the elements are : 12 65 23 98 35 98
After sorting the elements are : 12 23 35 65 98 98

Test Case - 3
User Output
Enter array size : 4
Enter 4 elements : -23 -45 -12 -36
Before sorting the elements are : -23 -45 -12 -36
After sorting the elements are : -45 -36 -23 -12

Test Case - 4
User Output
Enter array size : 6
Enter 6 elements : 1 -3 8 -4 -2 5
Before sorting the elements are : 1 -3 8 -4 -2 5
After sorting the elements are : -4 -3 -2 1 5 8

S.No: 54	Exp. Name: ***Collision resolution techniques : linear probing***	Date:

Aim:

Fill the missing code in the file `HashingLinearProbing.c`.

The function `insert(int x)` inserts an element `x` into the hash table using linear probing.

The function `delete(int x)` deletes an element `x` from the hash table using linear probing.

The function `search(int x)` searches for an element `x` from the hash table using linear probing.

The function `print()` prints the elements of the hash table.

Source Code:

HashingMain2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "HashingLinearProbing.c"
int main() {
   int x, op, i = 0;
   for (i = 0; i < SIZE; i++)
      HashTable[i] = -1;
   while (1) {
      printf("1.Insert 2.Delete 3.Search 4.Print 5.Exit\n");
      printf("Enter your option : ");
      scanf("%d", &op);
      switch (op) {
         case 1:  printf("Enter an element to be inserted : ");
               scanf("%d", &x);
               insert(x);
               break;
         case 2:
               printf("Enter an element to be deleted : ");
               scanf("%d", &x);
               delete(x);
               break;
         case 3:
               printf("Enter an element to be searched : ");
               scanf("%d", &x);
               search(x);
               break;
         case 4:
               print();
               break;
         case 5:  exit(0);
      }
   }
}
```

HashingLinearProbing.c

```
#define SIZE 10

int HashTable[SIZE];

int hash(int x) {
   return x % SIZE;
}
void insert(int x) {
   int index = hash(x);
   int start = index;
   while (HashTable[index] != -1) {
       if (HashTable[index] == -1) {
         break;
       }
       index = (index+1) % SIZE;
       if(index == start) {
         printf("Hash table is full.\n");
         return;
       }
   }
   HashTable[index] = x;
   printf("Successfully inserted.\n");
}
void delete(int x) {
   int index = hash(x);
   int start = index;
   while (HashTable[index] != x) {
       if (HashTable[index] == x) {
         break;
       }
       index = (index+1) % SIZE;
       if(index == start) {
         printf("Element not found. So cannot delete the element.\n");
         return;
       }
   }
   HashTable[index] = -1;
   printf("Successfully deleted.\n");
}
void search(int x) {
   int index = hash(x);
   int start = index;
   while (HashTable[index] != x) {
       if (HashTable[index] == x) {
         break;
       }
       index = (index+1) % SIZE;
       if(index == start) {
         printf("Element not found.\n");
         return;
       }
   }
   printf("Element found.\n");
}
void print() {
   int i=0;
   for(;i<SIZE;i++) {
```

```
      if(HashTable[i] != -1)
         printf("[%d]=>%d ",i,HashTable[i]);
   }
   printf("\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Delete 3.Search 4.Print 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 11
Successfully inserted. 1
1.Insert 2.Delete 3.Search 4.Print 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 22
Successfully inserted. 1
1.Insert 2.Delete 3.Search 4.Print 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 33
Successfully inserted. 1
1.Insert 2.Delete 3.Search 4.Print 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 43
Successfully inserted. 1
1.Insert 2.Delete 3.Search 4.Print 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 53
Successfully inserted. 4
1.Insert 2.Delete 3.Search 4.Print 5.Exit 4
Enter your option : 4
[1]=>11 [2]=>22 [3]=>33 [4]=>43 [5]=>53 1
1.Insert 2.Delete 3.Search 4.Print 5.Exit 1
Enter your option : 1
Enter an element to be inserted : 44
Successfully inserted. 4
1.Insert 2.Delete 3.Search 4.Print 5.Exit 4
Enter your option : 4
[1]=>11 [2]=>22 [3]=>33 [4]=>43 [5]=>53 [6]=>44 3
1.Insert 2.Delete 3.Search 4.Print 5.Exit 3
Enter your option : 3
Enter an element to be searched : 34
Element not found. 2
1.Insert 2.Delete 3.Search 4.Print 5.Exit 2
Enter your option : 2
Enter an element to be deleted : 33
Successfully deleted. 4
1.Insert 2.Delete 3.Search 4.Print 5.Exit 4
Enter your option : 4
[1]=>11 [2]=>22 [4]=>43 [5]=>53 [6]=>44 5
1.Insert 2.Delete 3.Search 4.Print 5.Exit 5

Test Case - 1

```
Enter your option :  5
```