SCHOOL OF ENGINEERING & TECHNOLOGY

BACHELOR OF TECHNOLOGY

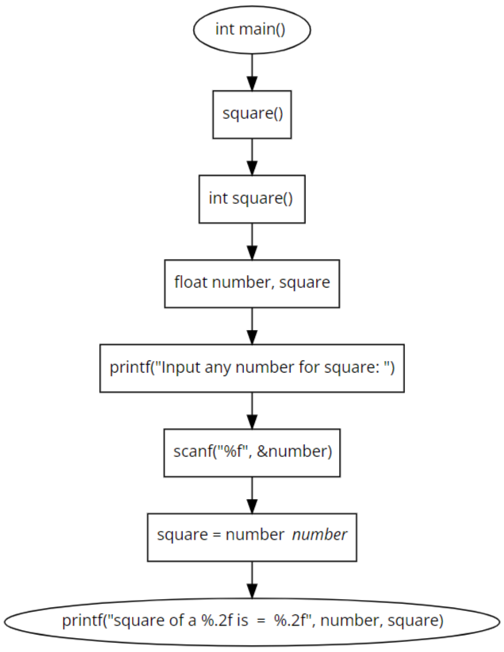
DATA STRUCTURE 3 RD SEMESTER

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Laboratory Manual

Practical 1: Write a program in C to find the square of any number using the function.

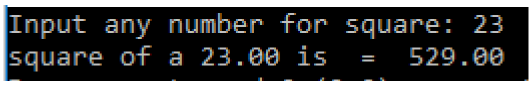
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include<stdio.h>
int main()
{
   square();
}
int square()
{
   float number, square;
   printf("Input any number for square: ");
   scanf("%f", &number);
   square = number * number;
   printf("square of a %.2f is  =  %.2f", number, square);
}
```

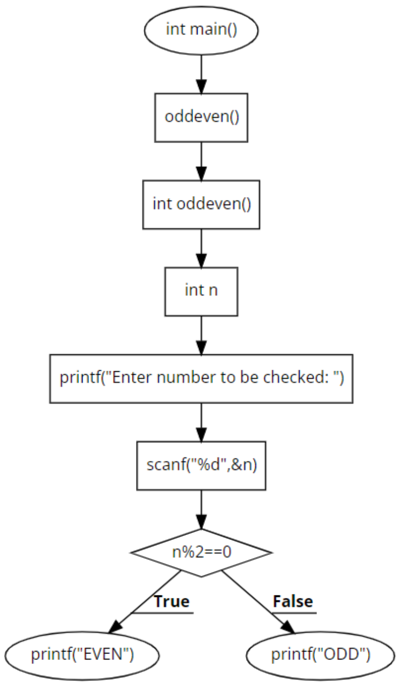
Output:



```
Input any number for square: 23
square of a 23.00 is  =  529.00
```

Practical 2: Write a program in C to check a given number is even or odd using the function.

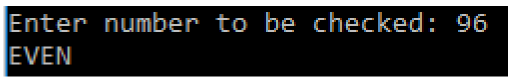
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include<stdio.h>
int main()
{
   oddeven();
}
int oddeven()
{
   int n;
   printf("Enter number to be checked: ");
   scanf("%d",&n);
   if (n%2==0)
   {
      printf("EVEN");
   }
   else
   {
      printf("ODD");
   }
}
```

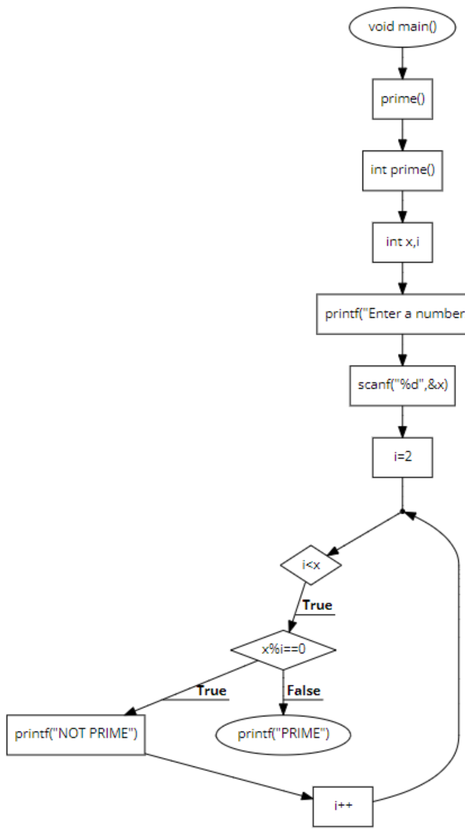
Output:



```
Enter number to be checked: 96
EVEN
```

Practical 3: Write a program in C to check whether a number is a prime number or not using the function.

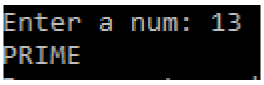
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include<stdio.h>
void main()
{
    prime();
}
int prime()
{
   int x,i;
   printf("Enter a number: ");
   scanf("%d",&x);
   for(i=2;i<x;i++)
   {
       if(x%i==0)
       {
          printf("NOT PRIME");
       }
       else
       {
          printf("PRIME");
          break;
       }

   }
}
```

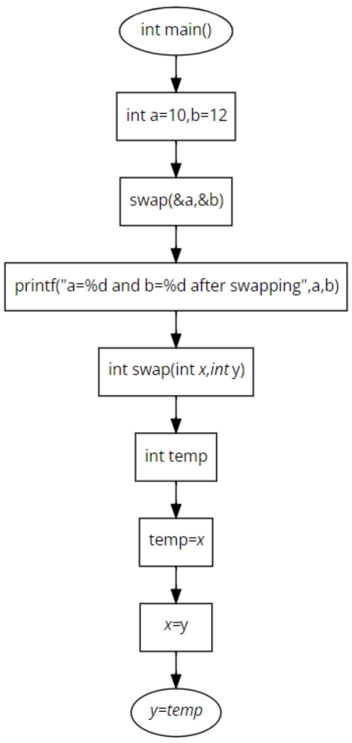
Output:



```
Enter a num: 13
PRIME
```

Practical 4: Write a program in C to swap two numbers using function.

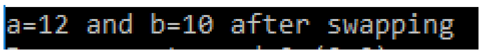
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include<stdio.h>
int main()
{
   int a=10,b=12;
   swap(&a,&b);
   printf("a=%d and b=%d after swapping",a,b);
}
int swap(int *x,int *y)
{
   int temp;
   temp=*x;
   *x=*y;
   *y=temp;
}
```

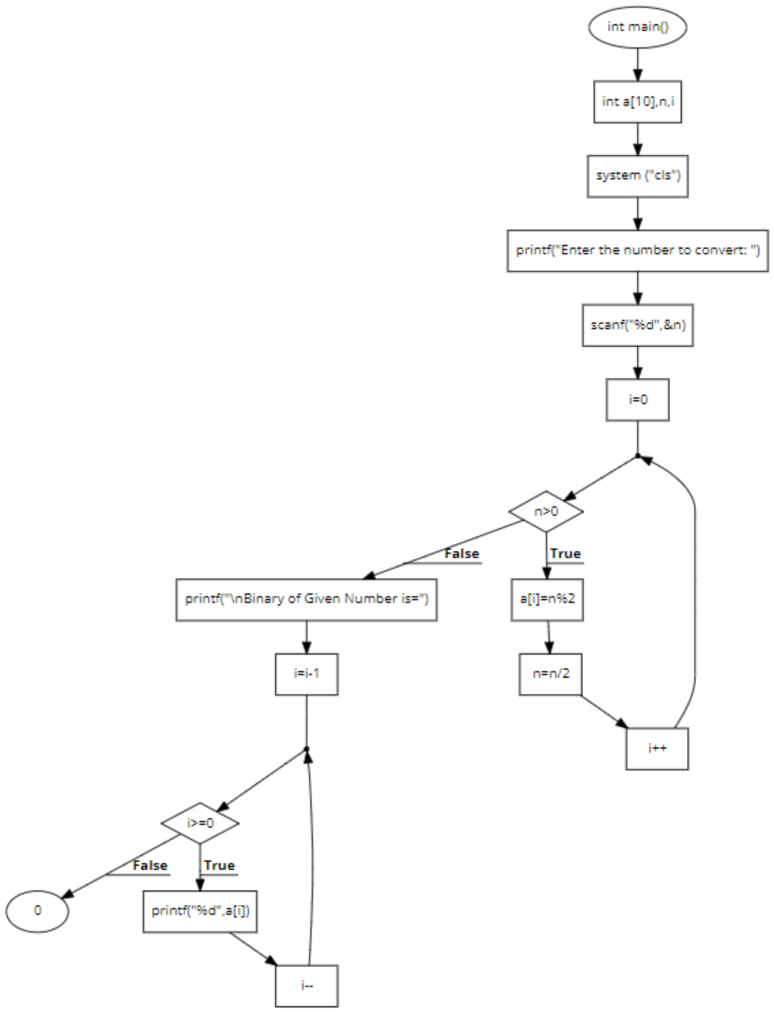
Output:



```
a=12 and b=10 after swapping
```

Practical 6: Write a program in C to convert decimal number to binary number using the function.

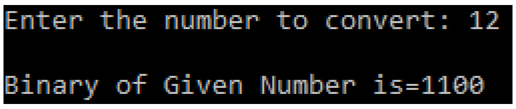
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int a[10],n,i;
   system ("cls");
   printf("Enter the number to convert: ");
   scanf("%d",&n);
   for(i=0;n>0;i++)
   {
      a[i]=n%2;
      n=n/2;
   }
   printf("\nBinary of Given Number is=");
   for(i=i-1;i>=0;i--)
   {
      printf("%d",a[i]);
   }
   return 0;
}
```

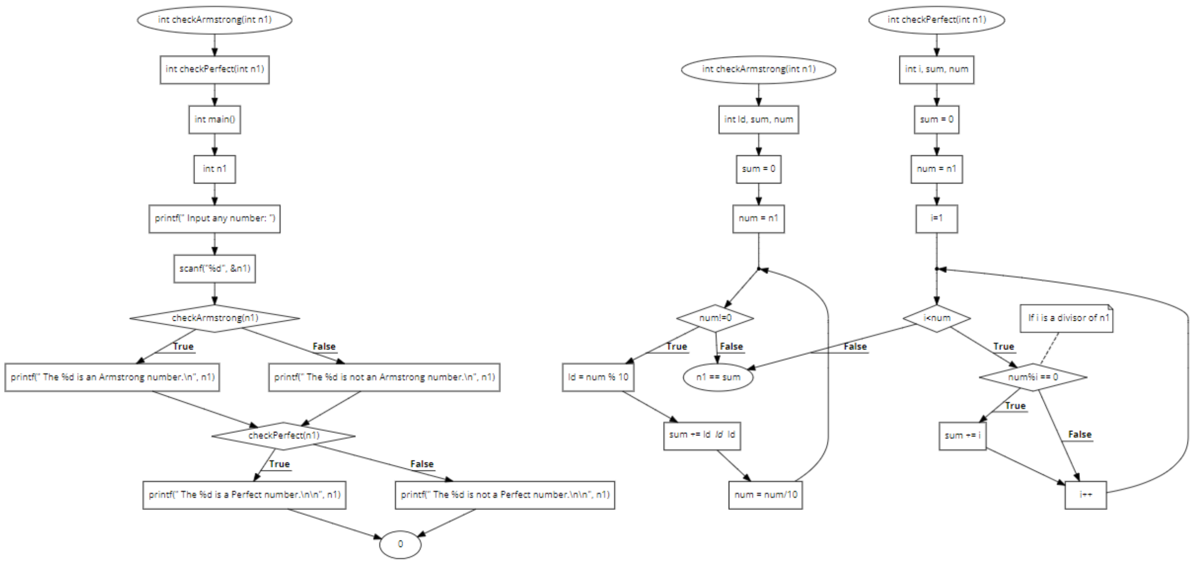
Output:



```
Enter the number to convert: 12

Binary of Given Number is=1100
```

Practical 7: Write a program in C to check armstrong and perfect numbers using the function.

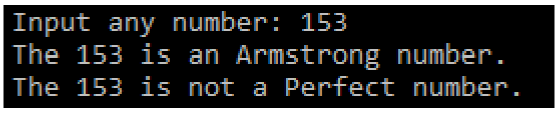
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include <stdio.h>

int checkArmstrong(int n1);
int checkPerfect(int n1);

int main()
{
   int n1;
   printf(" Input any number: ");
   scanf("%d", &n1);

   if(checkArmstrong(n1))
   {
      printf(" The %d is an Armstrong number.\n", n1);
   }
   else
   {
      printf(" The %d is not an Armstrong number.\n", n1);
   }

   if(checkPerfect(n1))
   {
      printf(" The %d is a Perfect number.\n\n", n1);
   }
   else
   {
        printf(" The %d is not a Perfect number.\n\n", n1);
    }
    return 0;
}
int checkArmstrong(int n1)
{
    int ld, sum, num;
    sum = 0;
    num = n1;
    while(num!=0)
    {
        ld = num % 10;
        sum += ld * ld * ld;
        num = num/10;
    }
    return (n1 == sum);
}
int checkPerfect(int n1)
{
    int i, sum, num;
    sum = 0;
    num = n1;
    for(i=1; i<num; i++)
    {
        /* If i is a divisor of n1 */
        if(num%i == 0)
        {
            sum += i;
        }
    }
    return (n1 == sum);
}
```

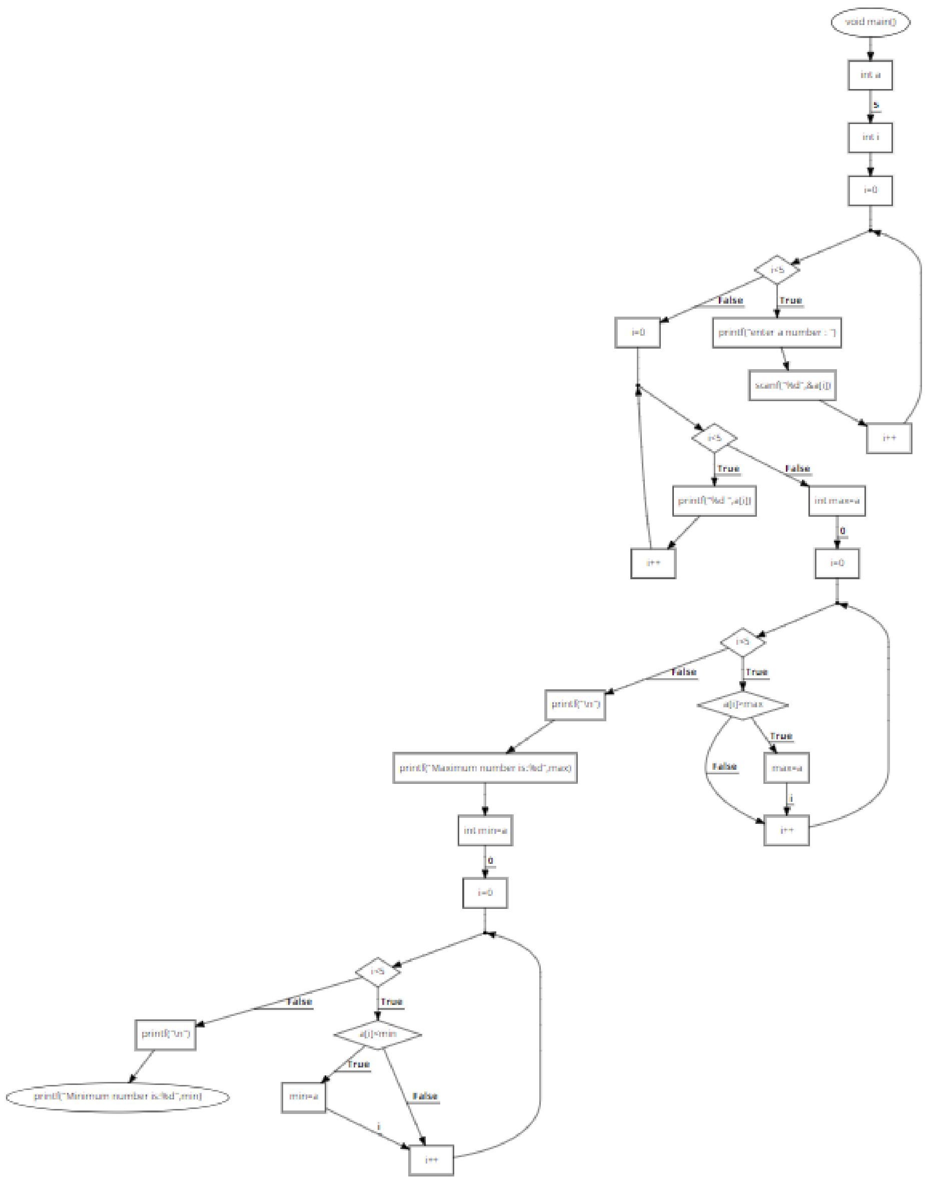
Output:



```
Input any number: 153
The 153 is an Armstrong number.
The 153 is not a Perfect number.
```

Practical 8: Write a program in C to get the largest element of an array using the function.

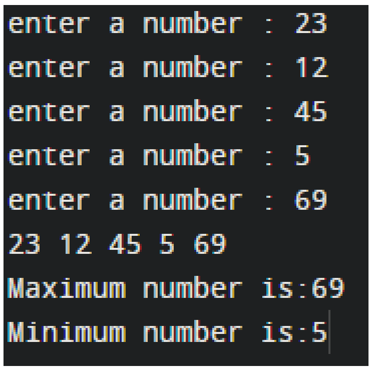
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include<stdio.h>
void main()
{
   int a[5];
   int i;
   for (i=0;i<5;i++)
   {
      printf("enter a number : ");
      scanf("%d",&a[i]);
   }
   for (i=0;i<5;i++)
   {
      printf("%d ",a[i]);
   }
   int max=a[0];
   for (i=0;i<5;i++)
   {
    if (a[i]>max)
    {
      max=a[i];
    }
  }
  printf("\n");
  printf("Maximum number is:%d",max);
  int min=a[0];
  for (i=0;i<5;i++)
  {
    if (a[i]<min)
    {
      min=a[i];
    }
  }
  printf("\n");
  printf("Minimum number is:%d",min);

}
```

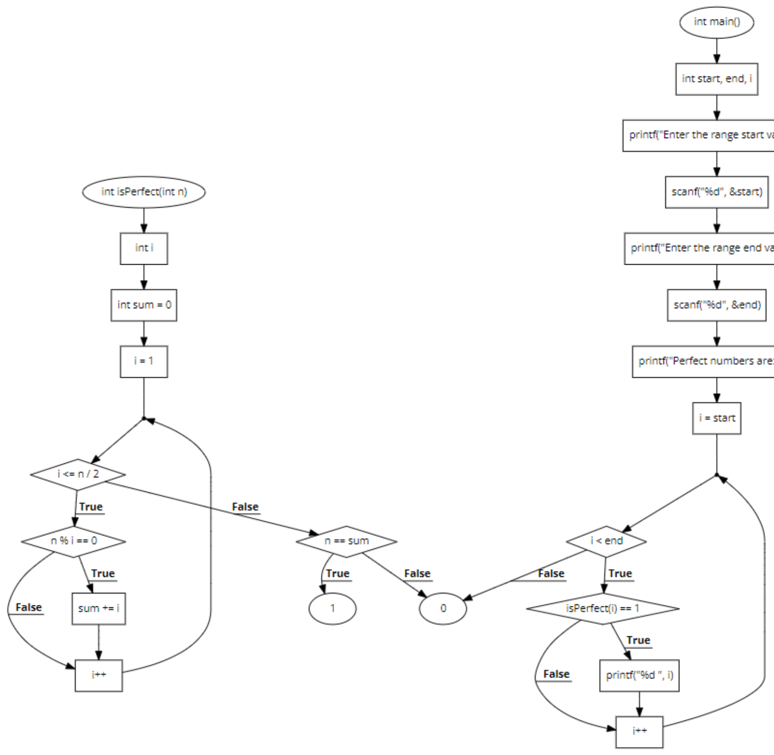
Output:



```
enter a number : 23
enter a number : 12
enter a number : 45
enter a number : 5
enter a number : 69
23 12 45 5 69
Maximum number is:69
Minimum number is:5
```

Practical 9: Write a program in C to print all perfect numbers in given range using the function.

Pseudocode/Logic/Algorithm/flow chart:

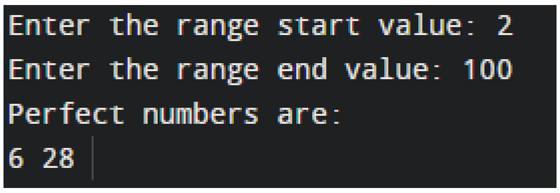


Source Code:

```
#include <stdio.h>

int isPerfect(int n)
{
   int i;
   int sum = 0;

   for (i = 1; i <= n / 2; i++)
   {
      if (n % i == 0)
      {
         sum += i;
      }
   }

   if (n == sum)
   {
      return 1;
   }
   return 0;
}

int main()
{
   int start, end, i;

   printf("Enter the range start value: ");
   scanf("%d", &start);

   printf("Enter the range end value: ");
```

```
    scanf("%d", &end);

    printf("Perfect numbers are: \n");

    for (i = start; i < end; i++)
    {
        if (isPerfect(i) == 1)
        {
            printf("%d ", i);
        }
    }
    return 0;
}
```

Output:



```
Enter the range start value: 2
Enter the range end value: 100
Perfect numbers are:
6 28
```

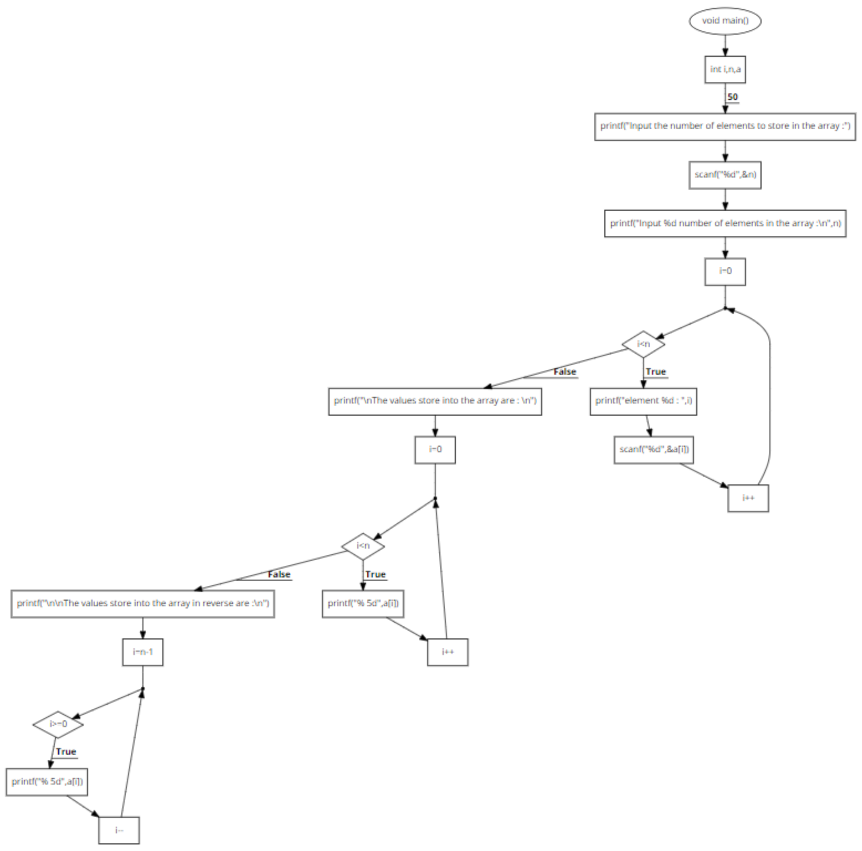
Conclusion:

Functions provide modularity to the program and are very useful to prevent repetitive code. Using modularity, it also becomes easier to write and execute the code. Also, the redundancy decrease speeds up the execution process

LAB II: - Programs Based on Array

Practical 1: Write a program in C to read n number of values in an array and display it in reverse order.

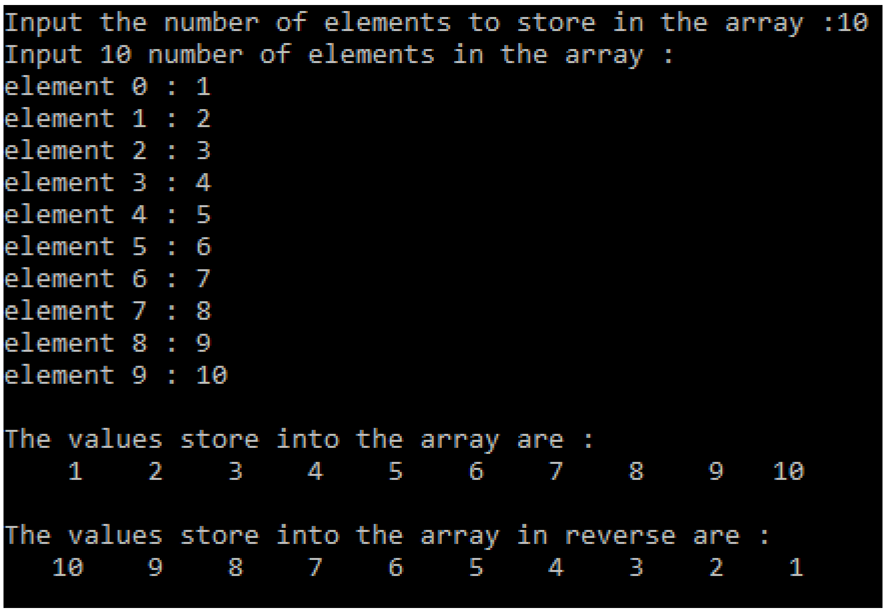
Pseudocode/Logic/Algorithm/flow chart:


void main()
int i,n,a
50
printf("Input the number of elements to store in the array :")
scanf("%d",&n)
printf("Input %d number of elements in the array :\n",n)
i=0
i<n
False
True
printf("\nThe values store into the array are : \n")
printf("element %d : ",i)
scanf("%d",&a[i])
i++
i=0
i<n
False
True
printf("\n\nThe values store into the array in reverse are :\n")
printf("% 5d",a[i])
i++
i=n-1
i>=0
True
printf("% 5d",a[i])
i--

Source Code:

```
#include <stdio.h>

void main()
{
  int i,n,a[50];
  printf("Input the number of elements to store in the array :");
  scanf("%d",&n);

  printf("Input %d number of elements in the array :\n",n);
  for(i=0;i<n;i++)
    {
          printf("element %d : ",i);
          scanf("%d",&a[i]);
          }

  printf("\nThe values store into the array are : \n");
  for(i=0;i<n;i++)
   {
           printf("% 5d",a[i]);
          }

  printf("\n\nThe values store into the array in reverse are :\n");
  for(i=n-1;i>=0;i--)
    {
           printf("% 5d",a[i]);
          }
}
```

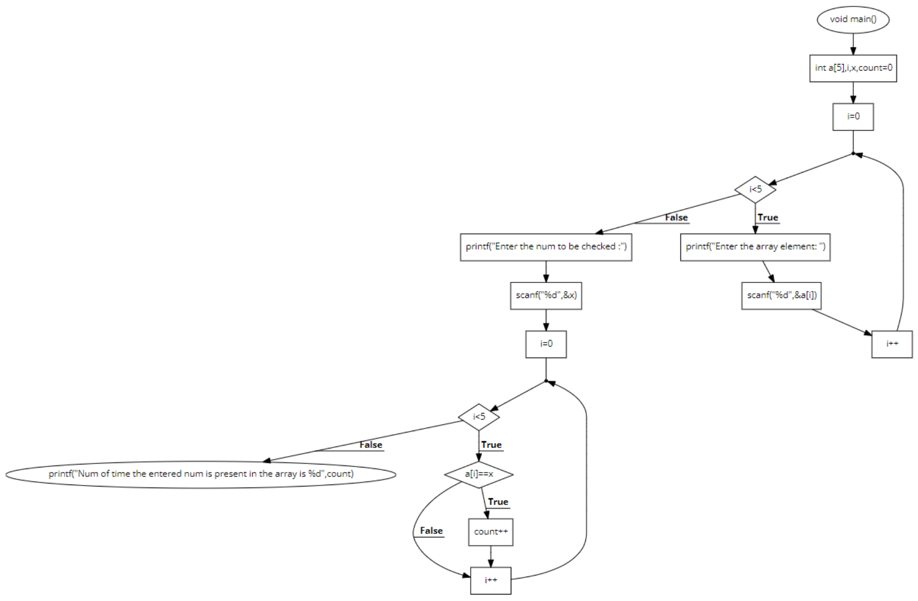
Output:



```
Input the number of elements to store in the array :10
Input 10 number of elements in the array :
element 0 : 1
element 1 : 2
element 2 : 3
element 3 : 4
element 4 : 5
element 5 : 6
element 6 : 7
element 7 : 8
element 8 : 9
element 9 : 10

The values store into the array are :
    1    2    3    4    5    6    7    8    9   10

The values store into the array in reverse are :
   10    9    8    7    6    5    4    3    2    1
```

Practical 2: Write a program in C to count a total number of duplicate elements in an array.

Pseudocode/Logic/Algorithm/flow chart:

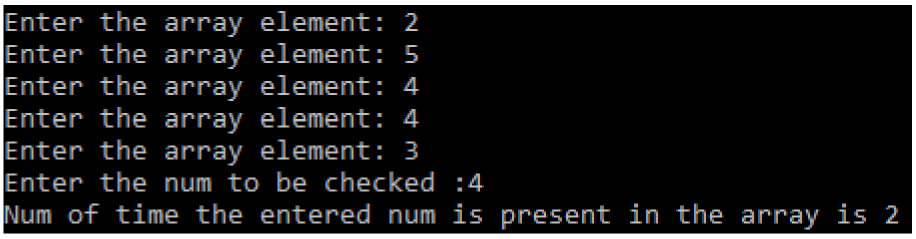

void main()
int a[5],i,x,count=0
i=0
i<5
False
True
printf("Enter the num to be checked :")
printf("Enter the array element: ")
scanf("%d",&x)
scanf("%d",&a[i])
i=0
i++
i<5
False
True
printf("Num of time the entered num is present in the array is %d",count)
a[i]==x
True
False
count++
i++

Source Code:

```
#include<stdio.h>
void main()
{
   int a[5],i,x,count=0;
   for (i=0;i<5;i++){
      printf("Enter the array element: ");
      scanf("%d",&a[i]);
   }
   printf("Enter the num to be checked :");
```

```
    scanf("%d",&x);
    for(i=0;i<5;i++){
        if(a[i]==x){
            count++;
        }
    }
    printf("Num of time the entered num is present in the array is %d",count);
}
```

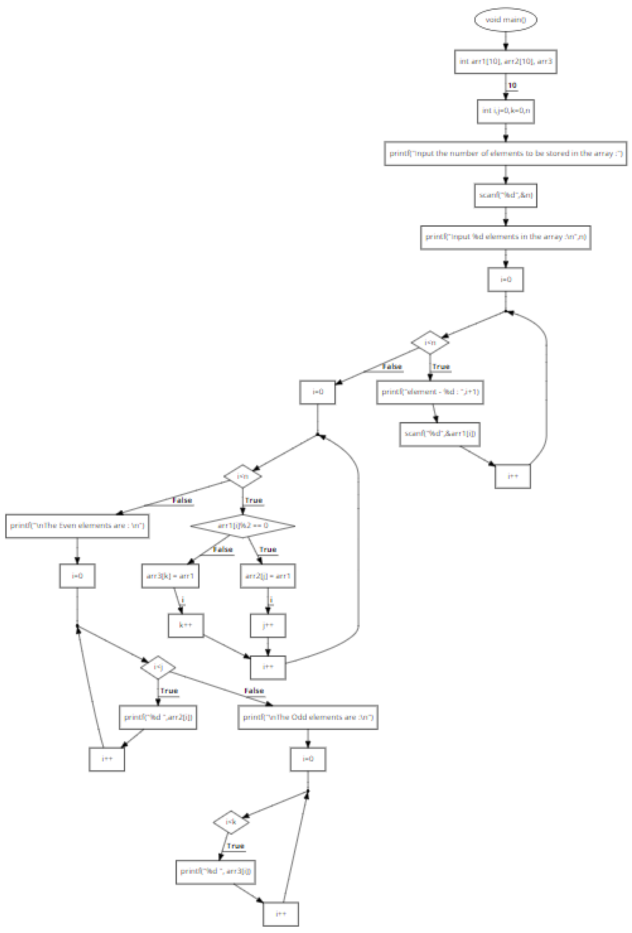
Output:



```
Enter the array element: 2
Enter the array element: 5
Enter the array element: 4
Enter the array element: 4
Enter the array element: 3
Enter the num to be checked :4
Num of time the entered num is present in the array is 2
```

Practical 3: Write a program in C to separate odd and even integers in separate arrays

Pseudocode/Logic/Algorithm/flow chart:



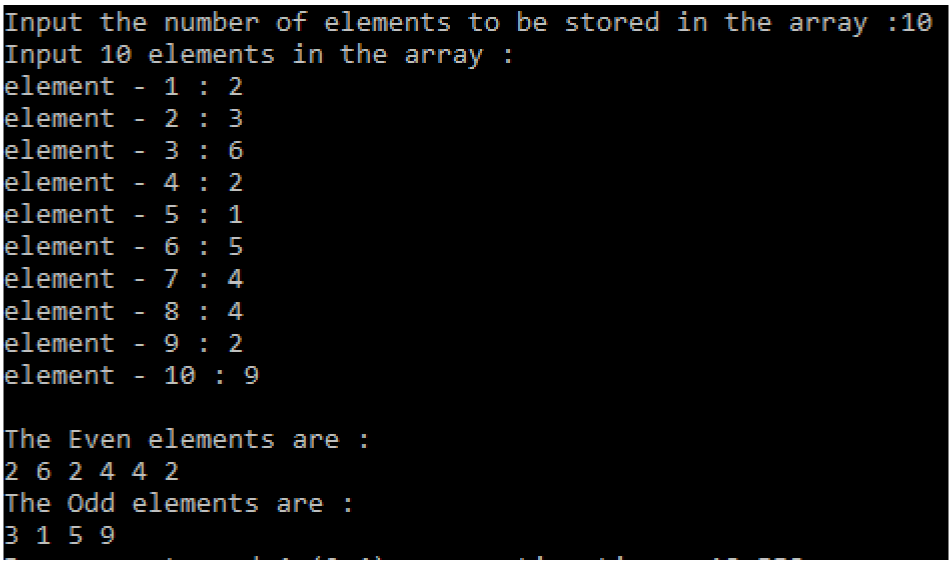
void main()
int arr1[10], arr2[10], arr3
10
int i,j=0,k=0,n
printf("Input the number of elements to be stored in the array :")
scanf("%d",&n)
printf("Input %d elements in the array :\n",n)
i=0
i<n
False
True
i=0
printf("element - %d : ",i+1)
scanf("%d",&arr1[i])
i++
i<n
False
True
printf("\nThe Even elements are : \n")
arr1[i]%2 == 0
False
True
i=0
arr3[k] = arr1
arr2[j] = arr1
i
i
k++
j++
i++
i<j
True
False
printf("%d ",arr2[i])
printf("\nThe Odd elements are :\n")
i++
i=0
i<k
True
printf("%d ", arr3[i])
i++

Source Code:

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

```
void main()
 {
   int arr1[10], arr2[10], arr3[10];
   int i,j=0,k=0,n;
   printf("Input the number of elements to be stored in the array :");
   scanf("%d",&n);

   printf("Input %d elements in the array :\n",n);
   for(i=0;i<n;i++)
   {
      printf("element - %d : ",i+1);
      scanf("%d",&arr1[i]);
   }

   for(i=0;i<n;i++)
   {
            if (arr1[i]%2 == 0)
            {
              arr2[j] = arr1[i];
              j++;
            }
            else
            {
              arr3[k] = arr1[i];
              k++;
            }
   }

   printf("\nThe Even elements are : \n");
   for(i=0;i<j;i++)
   {
            printf("%d ",arr2[i]);
   }

   printf("\nThe Odd elements are :\n");
   for(i=0;i<k;i++)
   {
            printf("%d ", arr3[i]);
   }

 }
```

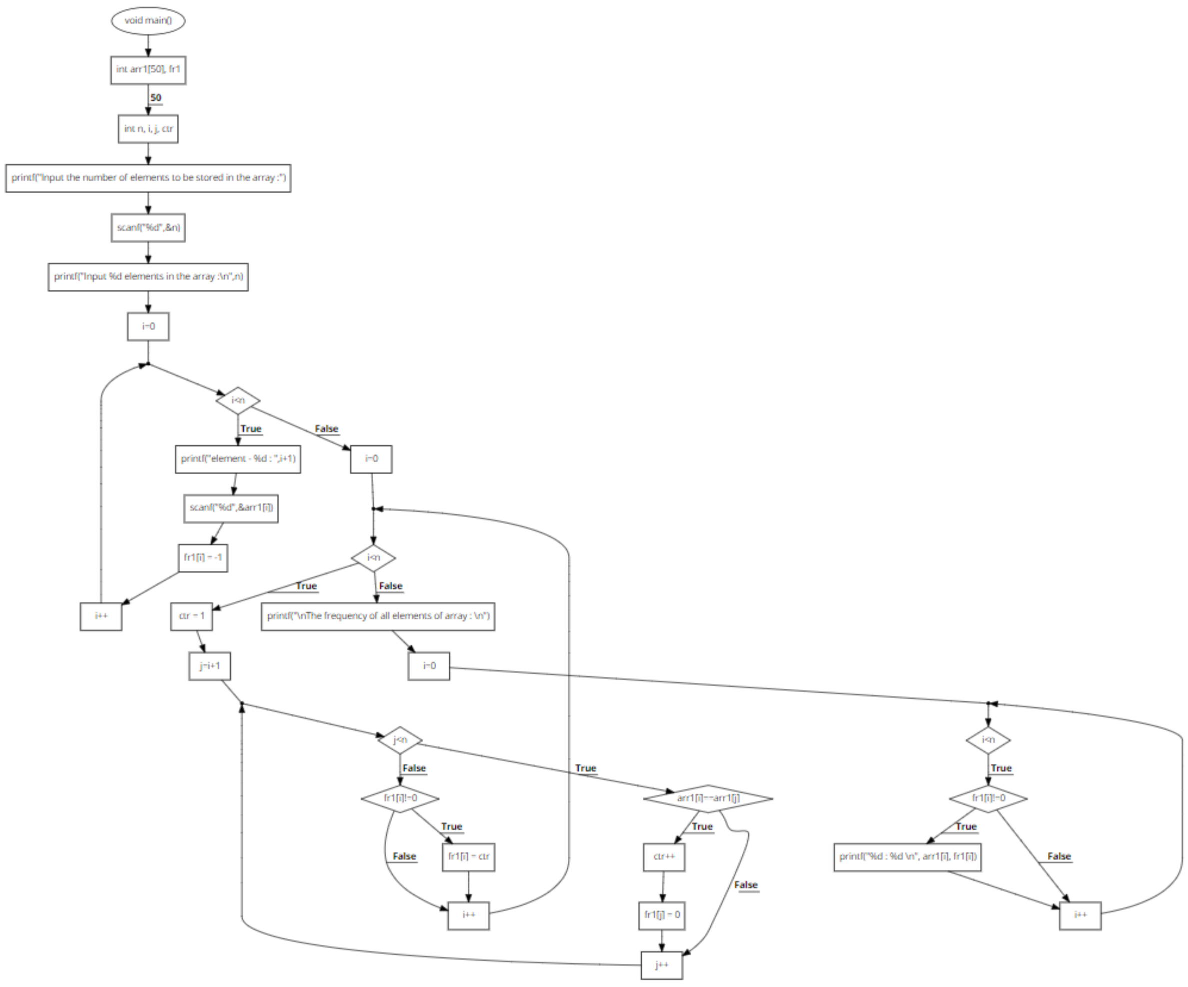
Output:



```
Input the number of elements to be stored in the array :10
Input 10 elements in the array :
element - 1 : 2
element - 2 : 3
element - 3 : 6
element - 4 : 2
element - 5 : 1
element - 6 : 5
element - 7 : 4
element - 8 : 4
element - 9 : 2
element - 10 : 9

The Even elements are :
2 6 2 4 4 2
The Odd elements are :
3 1 5 9
```

Practical 4: Write a program in C to count the frequency of each element of an array.

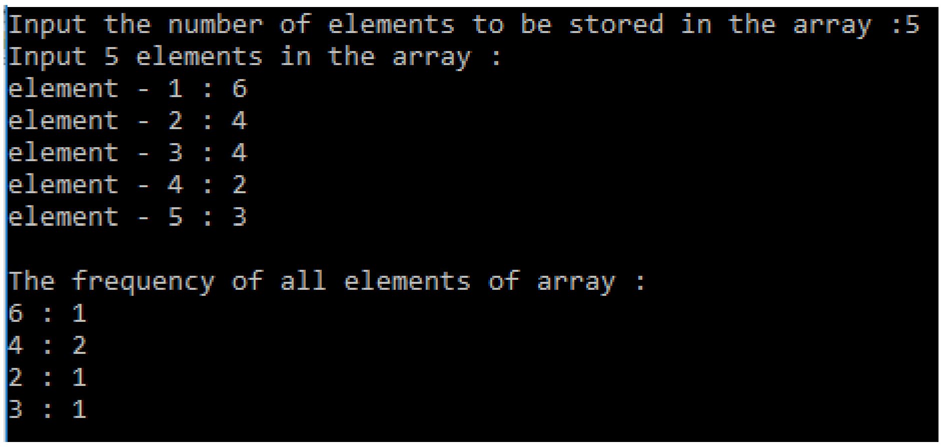
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include <stdio.h>

void main()
{
   int arr1[50], fr1[50];
   int n, i, j, ctr;

   printf("Input the number of elements to be stored in the array :");
   scanf("%d",&n);

   printf("Input %d elements in the array :\n",n);
   for(i=0;i<n;i++)
   {
      printf("element - %d : ",i+1);
      scanf("%d",&arr1[i]);
      fr1[i] = -1;
   }
   for(i=0; i<n; i++)
   {
      ctr = 1;
      for(j=i+1; j<n; j++)
      {
         if(arr1[i]==arr1[j])
         {
            ctr++;
            fr1[j] = 0;
         }
      }

      if(fr1[i]!=0)
      {
         fr1[i] = ctr;
      }
   }
   printf("\nThe frequency of all elements of array : \n");
   for(i=0; i<n; i++)
   {
      if(fr1[i]!=0)
      {
         printf("%d : %d \n", arr1[i], fr1[i]);
      }
   }
}
```

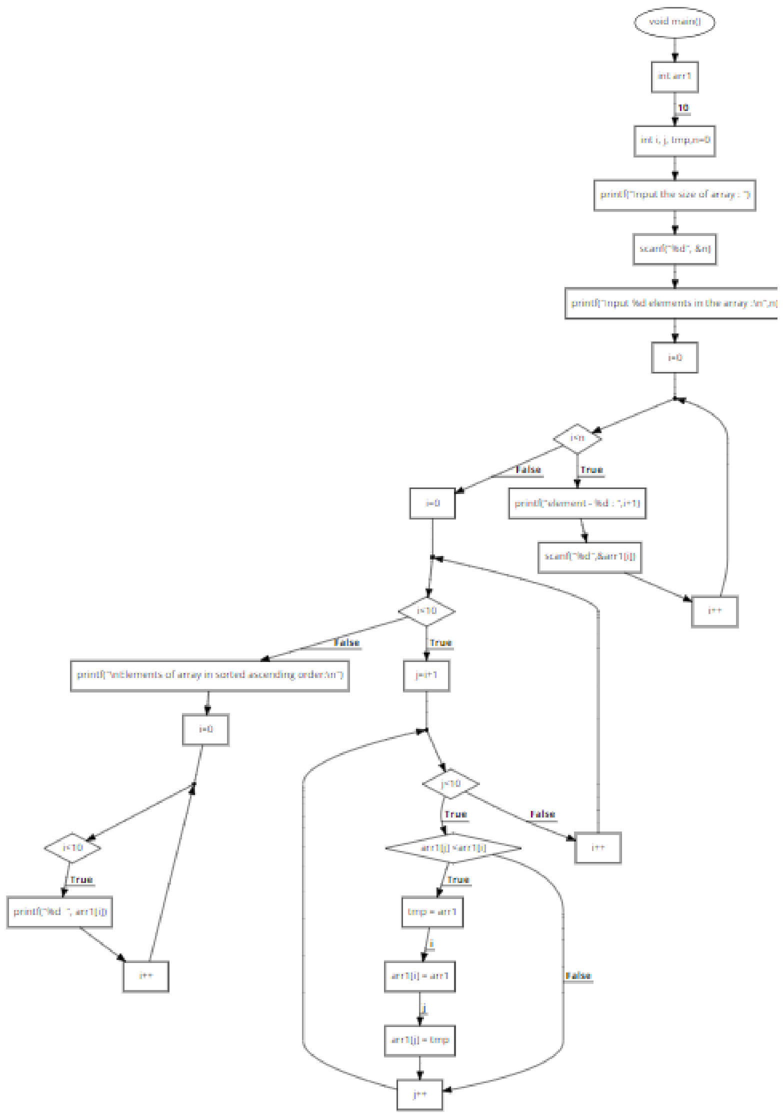
Output:



```
Input the number of elements to be stored in the array :5
Input 5 elements in the array :
element - 1 : 6
element - 2 : 4
element - 3 : 4
element - 4 : 2
element - 5 : 3

The frequency of all elements of array :
6 : 1
4 : 2
2 : 1
3 : 1
```

Practical 5: Write a program in C to sort elements of array in ascending order.

Pseudocode/Logic/Algorithm/flow chart:

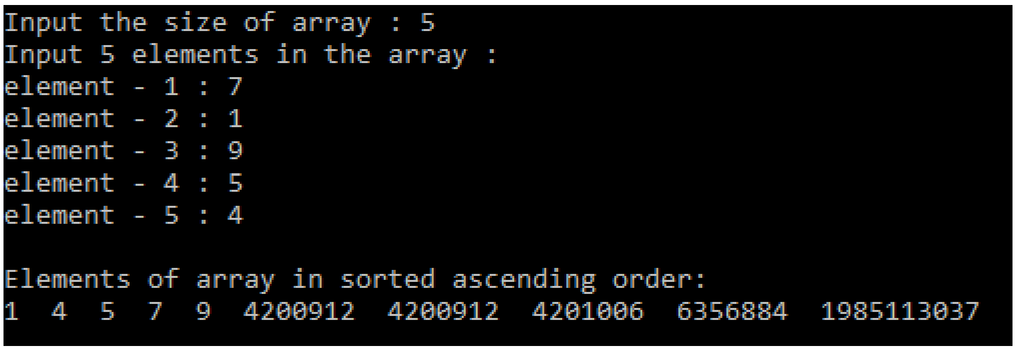


Source Code:

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

```
void main()
{
   int arr1[10];
   int i, j, tmp,n=0;
   printf("Input the size of array : ");
   scanf("%d", &n);
   printf("Input %d elements in the array :\n",n);
   for(i=0;i<n;i++)
   {
      printf("element - %d : ",i+1);
      scanf("%d",&arr1[i]);
   }

   for(i=0; i<10; i++)
   {
      for(j=i+1; j<10; j++)
      {
         if(arr1[j] <arr1[i])
         {
            tmp = arr1[i];
            arr1[i] = arr1[j];
            arr1[j] = tmp;
         }
      }
   }
   printf("\nElements of array in sorted ascending order:\n");
   for(i=0; i<10; i++)
   {
      printf("%d  ", arr1[i]);
   }
}
```

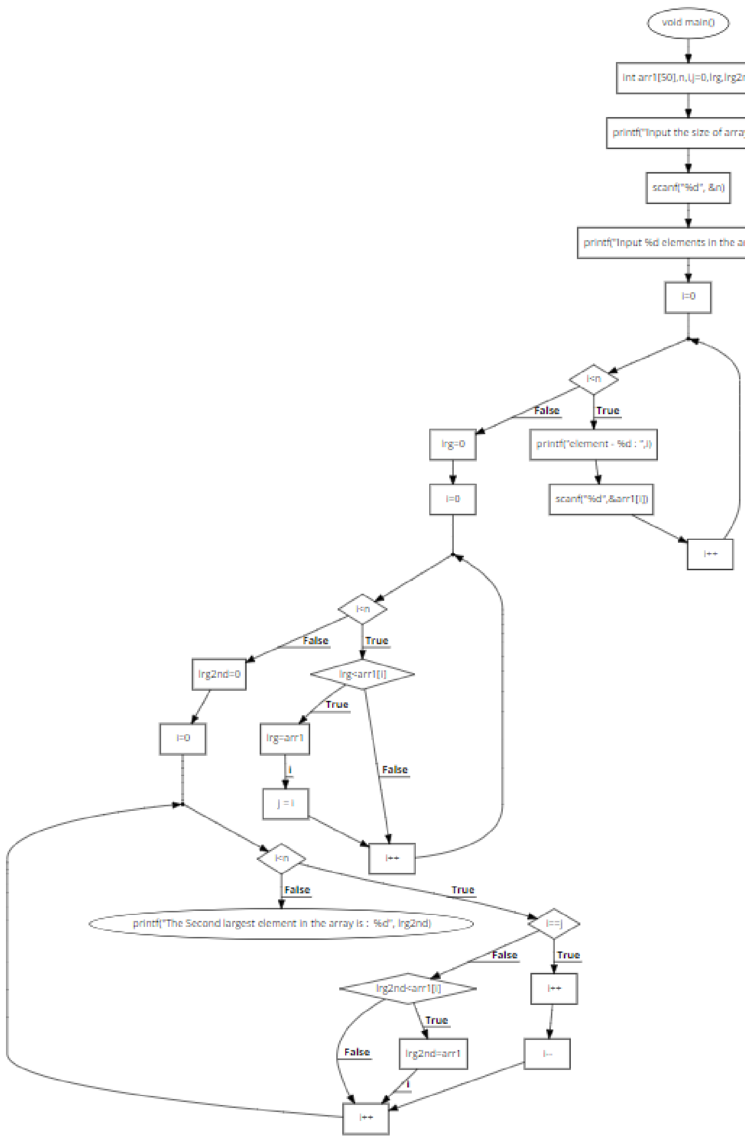
Output:



```
Input the size of array : 5
Input 5 elements in the array :
element - 1 : 7
element - 2 : 1
element - 3 : 9
element - 4 : 5
element - 5 : 4

Elements of array in sorted ascending order:
1  4  5  7  9  4200912  4200912  4201006  6356884  1985113037
```

Practical 6: Write a program in C to find the second largest element in an array.

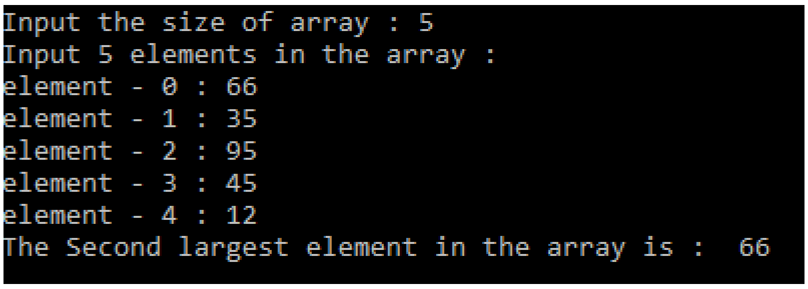
Pseudocode/Logic/Algorithm/flow chart:


void main()
int arr1[50],n,i,j=0,lrg,lrg2nd
printf("Input the size of array : ")
scanf("%d", &n)
printf("Input %d elements in the array :\n",n)
i=0
i<n
False
True
lrg=0
printf("element - %d : ",i)
scanf("%d",&arr1[i])
i=0
i++
i<n
False
True
lrg2nd=0
lrg<arr1[i]
True
False
i=0
lrg=arr1
i
j = i
i++
i<n
False
True
printf("The Second largest element in the array is : %d", lrg2nd)
i==j
False
True
lrg2nd<arr1[i]
i++
True
False
lrg2nd=arr1
i--
i
i++

Source Code:

```
#include <stdio.h>

void main()
{
   int arr1[50],n,i,j=0,lrg,lrg2nd;

   printf("Input the size of array : ");
   scanf("%d", &n);
   printf("Input %d elements in the array :\n",n);
   for(i=0;i<n;i++)
   {
      printf("element - %d : ",i);
      scanf("%d",&arr1[i]);
   }

   lrg=0;
   for(i=0;i<n;i++)
   {
      if(lrg<arr1[i])
      {
          lrg=arr1[i];
          j = i;
      }
   }

   lrg2nd=0;
   for(i=0;i<n;i++)
   {
      if(i==j)
      {
          i++;
          i--;
      }
      else
      {
          if(lrg2nd<arr1[i])
          {
            lrg2nd=arr1[i];
          }
      }
   }

   printf("The Second largest element in the array is :  %d", lrg2nd);
}
```

Output:



```
Input the size of array : 5
Input 5 elements in the array :
element - 0 : 66
element - 1 : 35
element - 2 : 95
element - 3 : 45
element - 4 : 12
The Second largest element in the array is :   66
```

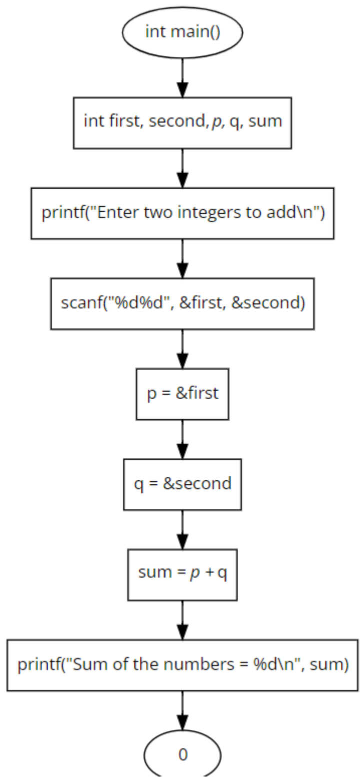
Conclusion:

Arrays store similar type of data in a sequential manner of order. Since all the values are stored sequentially it becomes easier to access data from anywhere in the dataset. Arrays prove to be very useful when working with dataset of same data types.

LAB III: - Programs Based on Functions and Pointers

Practical 1: Write a program in C to add two numbers using pointers.

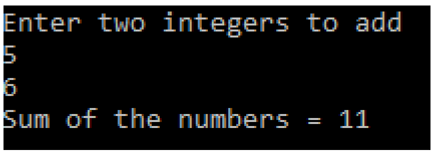
Pseudocode/Logic/Algorithm/flow chart:


int main()
int first, second, *p*, q, sum
printf("Enter two integers to add\n")
scanf("%d%d", &first, &second)
p = &first
q = &second
sum = *p* + q
printf("Sum of the numbers = %d\n", sum)
0

Source Code:

```
#include <stdio.h>
int main()
{
   int first, second, *p, *q, sum;
   printf("Enter two integers to add\n");
   scanf("%d%d", &first, &second);
   p = &first;
   q = &second;
   sum = *p + *q;
   printf("Sum of the numbers = %d\n", sum);
   return 0;
}
```

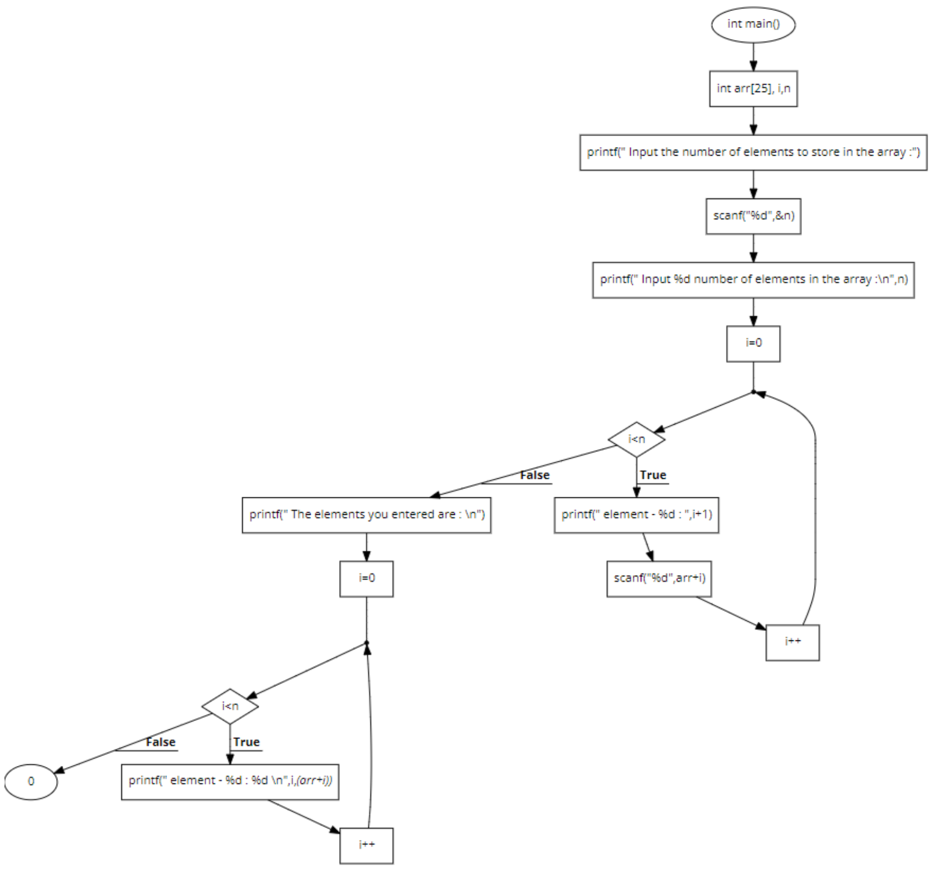
Output:



```
Enter two integers to add
5
6
Sum of the numbers = 11
```

Practical 2: Write a program in C to store n elements in an array and print the elements using pointer.

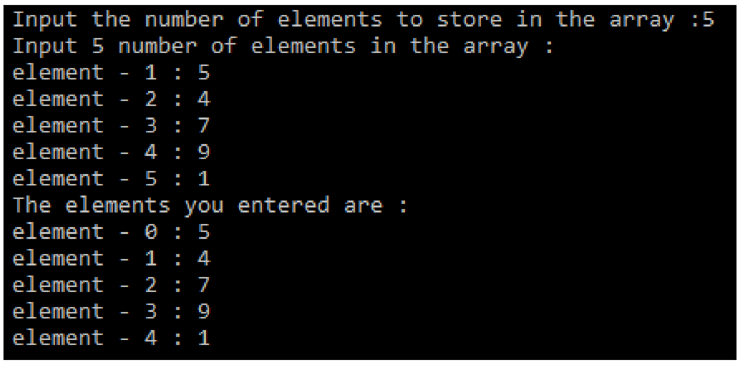
Pseudocode/Logic/Algorithm/flow chart:


int main()
int arr[25], i,n
printf(" Input the number of elements to store in the array :")
scanf("%d",&n)
printf(" Input %d number of elements in the array :\n",n)
i=0
i<n
False
True
printf(" The elements you entered are : \n")
printf(" element - %d : ",i+1)
scanf("%d",arr+i)
i++
i=0
i<n
False
True
0
printf(" element - %d : %d \n",i,(arr+i))
i++

Source Code:

```
#include <stdio.h>
int main()
{
   int arr[25], i,n;
   printf(" Input the number of elements to store in the array :");
   scanf("%d",&n);

   printf(" Input %d number of elements in the array :\n",n);
   for(i=0;i<n;i++)
   {
      printf(" element - %d : ",i+1);
      scanf("%d",arr+i);
   }
   printf(" The elements you entered are : \n");
   for(i=0;i<n;i++)
   {
      printf(" element - %d : %d \n",i,*(arr+i));
   }
   return 0;
}
```

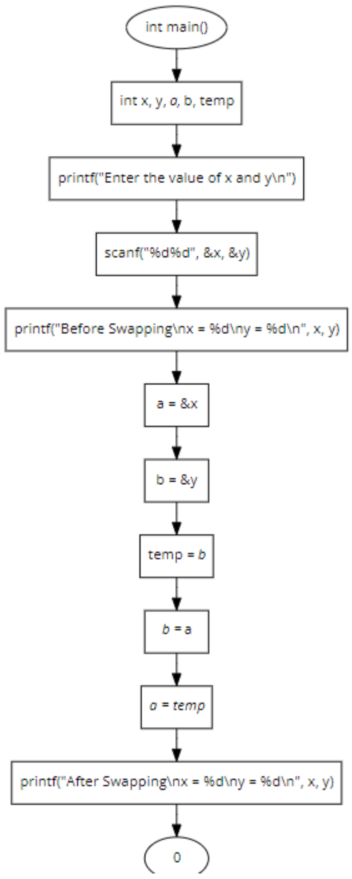
Output:



```
Input the number of elements to store in the array :5
Input 5 number of elements in the array :
element - 1 : 5
element - 2 : 4
element - 3 : 7
element - 4 : 9
element - 5 : 1
The elements you entered are :
element - 0 : 5
element - 1 : 4
element - 2 : 7
element - 3 : 9
element - 4 : 1
```

Practical 3: Write a program in C to swap two numbers using pointers.

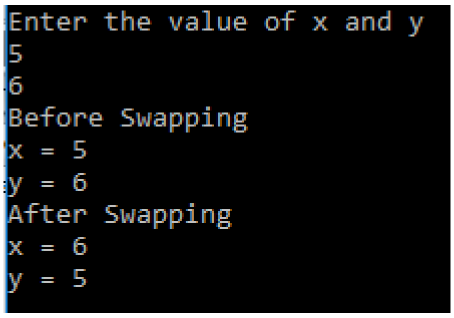
Pseudocode/Logic/Algorithm/flow chart:


int main()
int x, y, a, b, temp
printf("Enter the value of x and y\n")
scanf("%d%d", &x, &y)
printf("Before Swapping\nx = %d\ny = %d\n", x, y)
a = &x
b = &y
temp = b
b = a
a = temp
printf("After Swapping\nx = %d\ny = %d\n", x, y)
0

Source Code:

```
#include <stdio.h>

int main()
{
  int x, y, *a, *b, temp;

  printf("Enter the value of x and y\n");
  scanf("%d%d", &x, &y);

  printf("Before Swapping\nx = %d\ny = %d\n", x, y);
  a = &x;
  b = &y;

  temp = *b;
  *b = *a;
  *a = temp;

  printf("After Swapping\nx = %d\ny = %d\n", x, y);

  return 0;
}
```

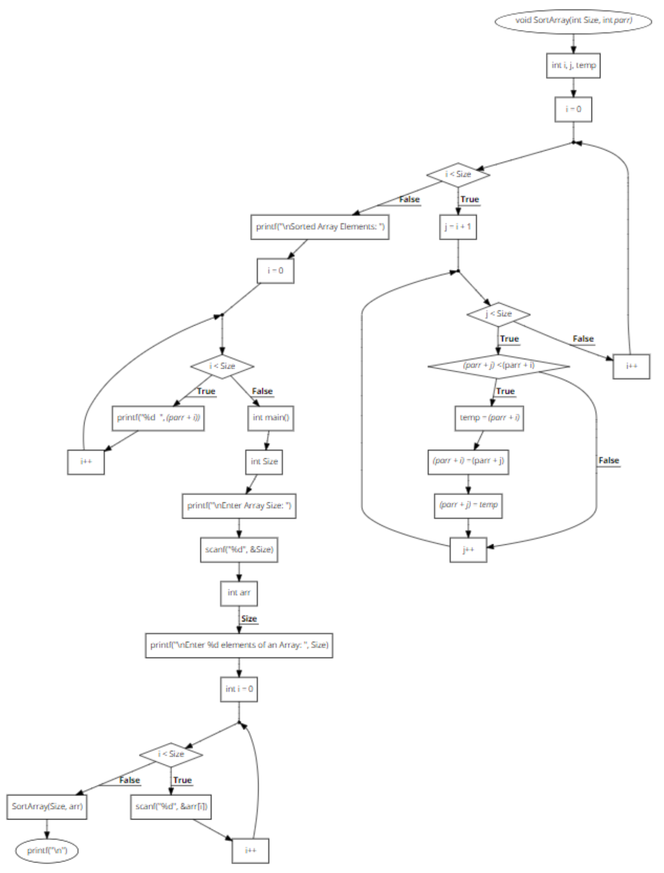
Output:



```
Enter the value of x and y
5
6
Before Swapping
x = 5
y = 6
After Swapping
x = 6
y = 5
```

Practical 4: Write a program in C to sort an array using Pointer.

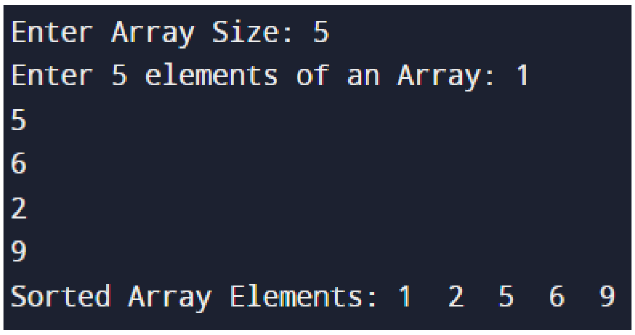
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include <stdio.h>
void SortArray(int Size, int* parr)
{
        int i, j, temp;
        for (i = 0; i < Size; i++)
        {
                for (j = i + 1; j < Size; j++)
                {
                        if(*(parr + j) < *(parr + i))
                        {
                                temp = *(parr + i);
                                *(parr + i) = *(parr + j);
                                *(parr + j) = temp;
                        }
                }
        }
        printf("\nSorted Array Elements: ");
        for(i = 0; i < Size; i++)
        {
                printf("%d  ", *(parr + i));
        }
}

int main()
{
        int Size;
        printf("\nEnter Array Size: ");
        scanf("%d", &Size);
        int arr[Size];
        printf("\nEnter %d elements of an Array: ", Size);
        for (int i = 0; i < Size; i++)
        {
                scanf("%d", &arr[i]);
  }
        SortArray(Size, arr);
        printf("\n");
}
```

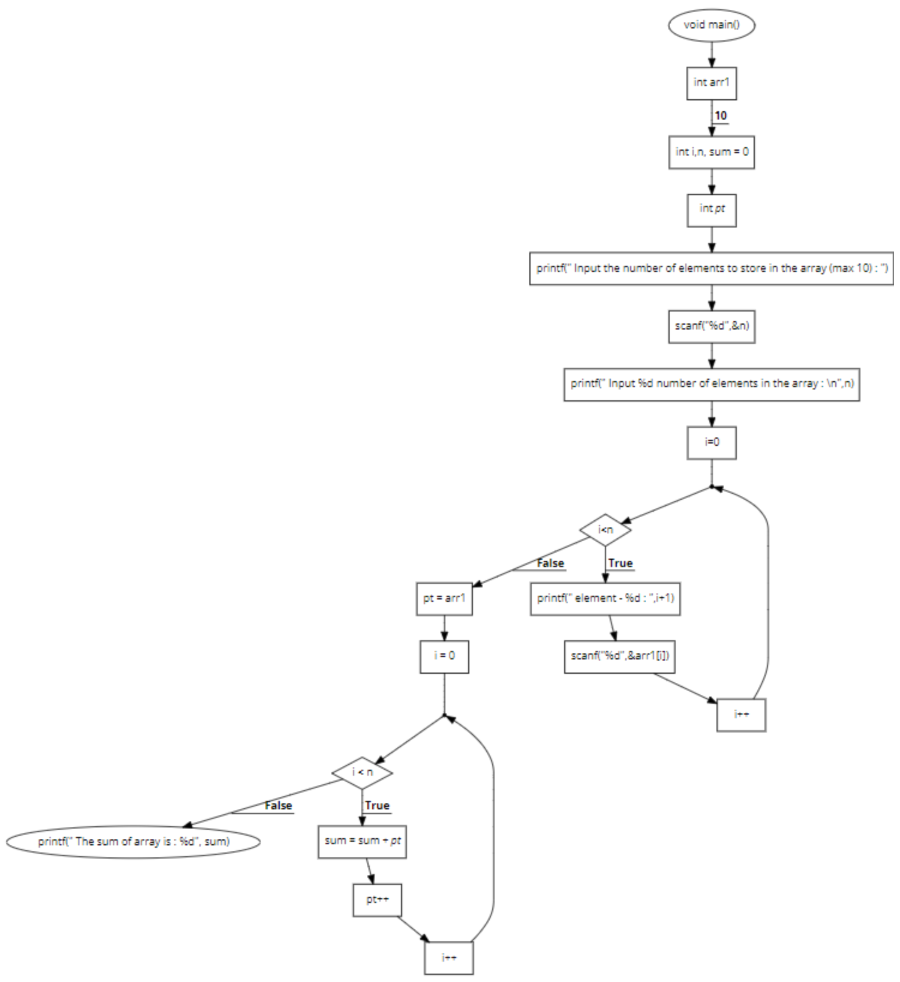
Output:



```
Enter Array Size: 5
Enter 5 elements of an Array: 1
5
6
2
9
Sorted Array Elements: 1  2  5  6  9
```

Practical 5: Write a program in C to compute the sum of all elements in an array using pointers.

Pseudocode/Logic/Algorithm/flow chart:

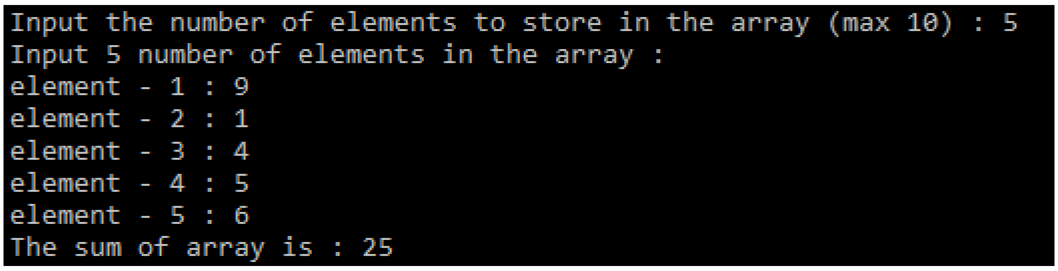

void main()
int arr1
10
int i,n, sum = 0
int pt
printf(" Input the number of elements to store in the array (max 10) : ")
scanf("%d",&n)
printf(" Input %d number of elements in the array : \n",n)
i=0
i<n
False
True
pt = arr1
printf(" element - %d : ",i+1)
i = 0
scanf("%d",&arr1[i])
i++
i < n
False
True
printf(" The sum of array is : %d", sum)
sum = sum + pt
pt++
i++

Source Code:

```
#include <stdio.h>
void main()
{
   int arr1[10];
   int i,n, sum = 0;
   int *pt;
   printf(" Input the number of elements to store in the array (max 10) : ");
   scanf("%d",&n);

   printf(" Input %d number of elements in the array : \n",n);
   for(i=0;i<n;i++)
   {
      printf(" element - %d : ",i+1);
      scanf("%d",&arr1[i]);
   }

   pt = arr1;

   for (i = 0; i < n; i++)
   {
      sum = sum + *pt;
      pt++;
   }
```

```
    printf(" The sum of array is : %d", sum);
}
```

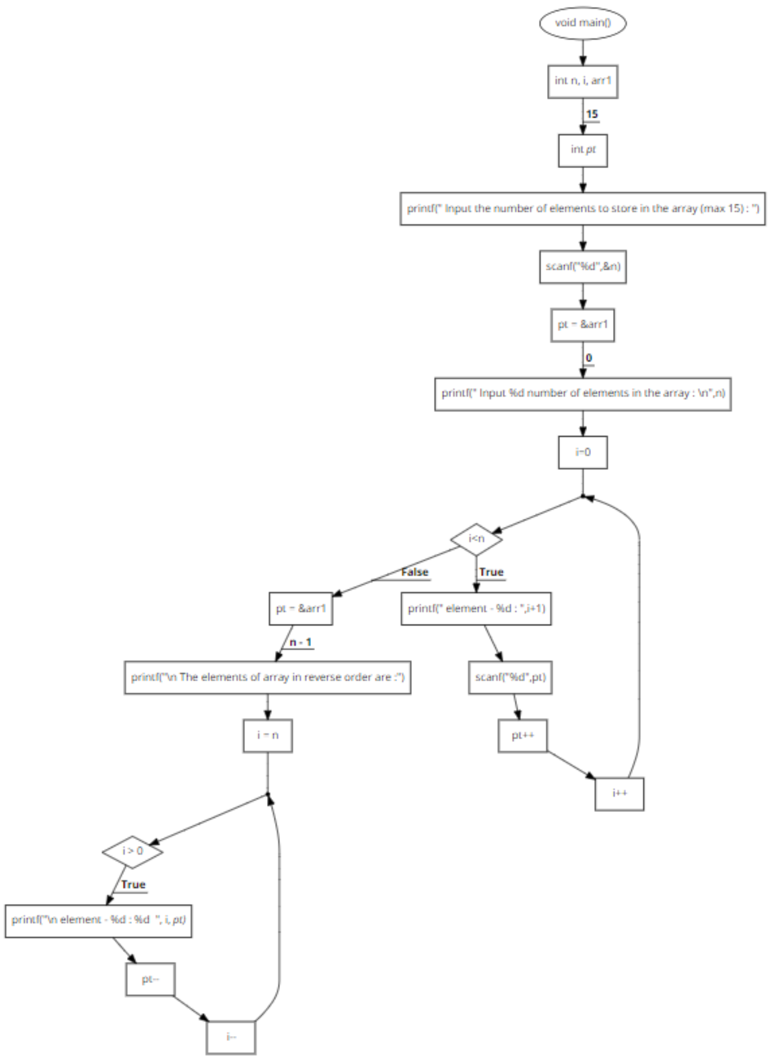
Output:



```
Input the number of elements to store in the array (max 10) : 5
Input 5 number of elements in the array :
element - 1 : 9
element - 2 : 1
element - 3 : 4
element - 4 : 5
element - 5 : 6
The sum of array is : 25
```

Practical 6: Write a program in C to print the elements of an array in reverse order.

Pseudocode/Logic/Algorithm/flow chart:



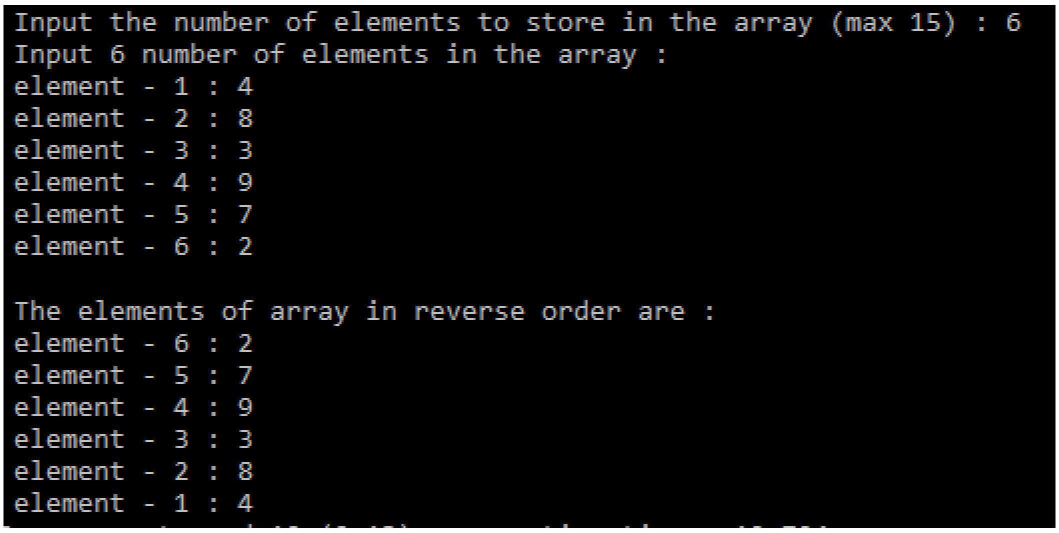
void main()
int n, i, arr1
15
int pt
printf(" Input the number of elements to store in the array (max 15) : ")
scanf("%d",&n)
pt = &arr1
0
printf(" Input %d number of elements in the array : \n",n)
i=0
i<n
False
True
pt = &arr1
printf(" element - %d : ",i+1)
n - 1
printf("\n The elements of array in reverse order are :")
scanf("%d",pt)
i = n
pt++
i++
i > 0
True
printf("\n element - %d : %d ", i, pt)
pt--
i--

Source Code:

```
#include <stdio.h>
void main()
```

```
{
   int n, i, arr1[15];
   int *pt;

   printf(" Input the number of elements to store in the array (max 15) : ");
   scanf("%d",&n);
   pt = &arr1[0];
   printf(" Input %d number of elements in the array : \n",n);
   for(i=0;i<n;i++)
   {
      printf(" element - %d : ",i+1);
      scanf("%d",pt);
      pt++;
   }

   pt = &arr1[n - 1];

   printf("\n The elements of array in reverse order are :");

   for (i = n; i > 0; i--)
   {
      printf("\n element - %d : %d  ", i, *pt);
      pt--;
   }
}
```

Output:



```
Input the number of elements to store in the array (max 15) : 6
Input 6 number of elements in the array :
element - 1 : 4
element - 2 : 8
element - 3 : 3
element - 4 : 9
element - 5 : 7
element - 6 : 2

The elements of array in reverse order are :
element - 6 : 2
element - 5 : 7
element - 4 : 9
element - 3 : 3
element - 2 : 8
element - 1 : 4
```

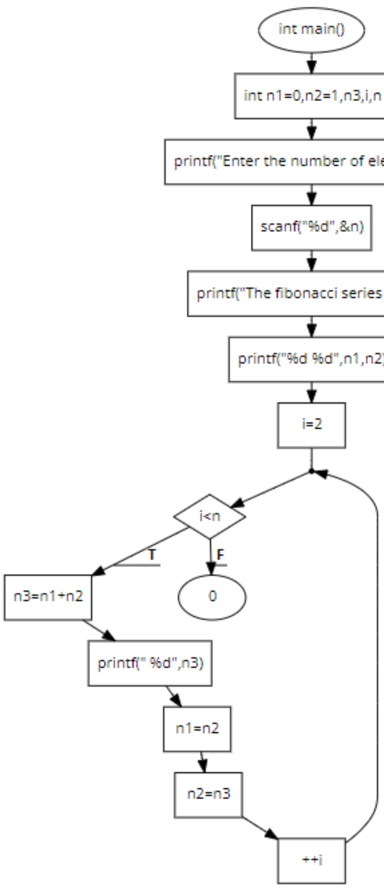
Conclusion:

Pointers store the address of a specific variable or value. Since pointer store the address, it provides for the facility to directly modify the values stores at that given location. We can access this address locations throughout the program. So, pointers provide a great versality to the program.

LAB IV: - Programs Based on Recursion

Practical 1: Write a program in C to Print Fibonacci Series using recursion.

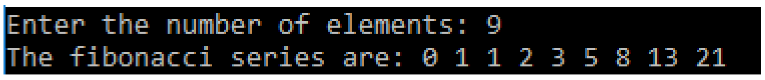
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include<stdio.h>
int main()
{
    int n1=0,n2=1,n3,i,n;
    printf("Enter the number of elements: ");
    scanf("%d",&n);
    printf("The fibonacci series are: ");
    printf("%d %d",n1,n2);
    for(i=2;i<n;++i)
   {
        n3=n1+n2;
        printf(" %d",n3);
        n1=n2;
        n2=n3;
   }
   return 0;
}
```

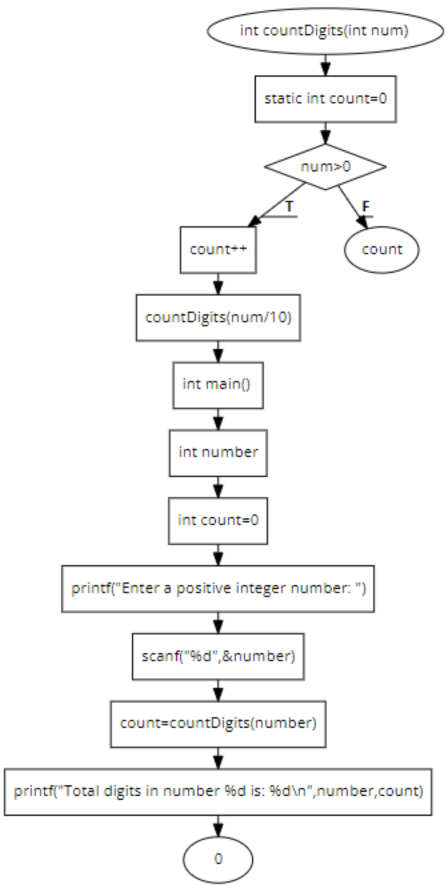
Output:



```
Enter the number of elements: 9
The fibonacci series are: 0 1 1 2 3 5 8 13 21
```

Practical 2: Write a program in C to count the digits of a given number using recursion.

Pseudocode/Logic/Algorithm/flow chart:

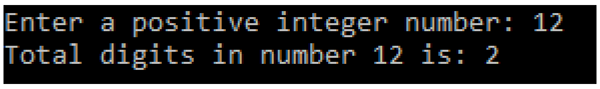


Source Code:

```
#include <stdio.h>
int countDigits(int num)
{
   static int count=0;

   if(num>0)
   {
      count++;
      countDigits(num/10);
   }
   else
   {
      return count;
   }
}
int main()
{
   int number;
   int count=0;
   printf("Enter a positive integer number: ");
```

```
    scanf("%d",&number);
    count=countDigits(number);
    printf("Total digits in number %d is: %d\n",number,count);
    return 0;
}
```

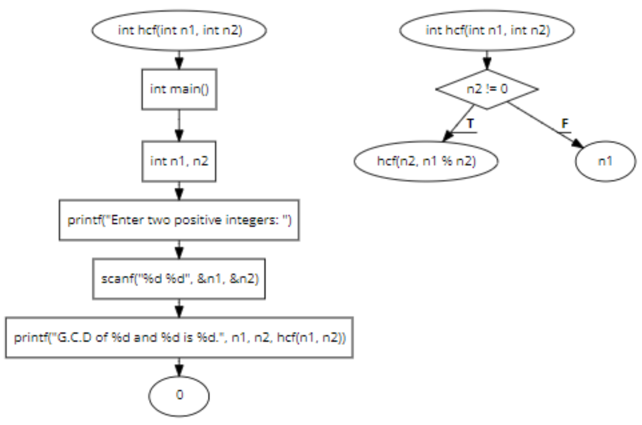
Output:



```
Enter a positive integer number: 12
Total digits in number 12 is: 2
```

Practical 3: Write a program in C to find GCD of two numbers using recursion.

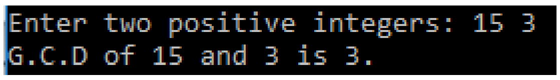
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include <stdio.h>
int hcf(int n1, int n2);
int main() {
    int n1, n2;
    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);
    printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1, n2));
    return 0;
}

int hcf(int n1, int n2) {
    if (n2 != 0)
        return hcf(n2, n1 % n2);
    else
        return n1;
}
```

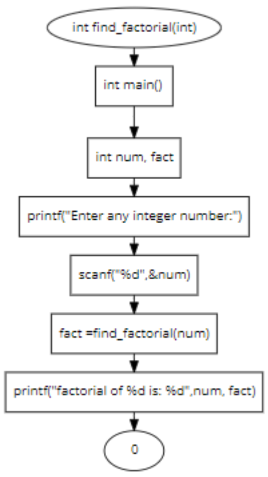
Output:

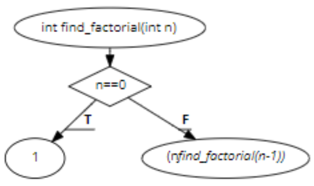


```
Enter two positive integers: 15 3
G.C.D of 15 and 3 is 3.
```

Practical 4: Write a program in C to find the Factorial of a number using recursion.

Pseudocode/Logic/Algorithm/flow chart:

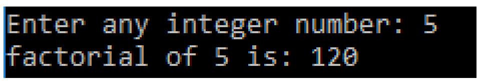

int find_factorial(int)
int main()
int num, fact
printf("Enter any integer number:")
scanf("%d",&num)
fact =find_factorial(num)
printf("factorial of %d is: %d",num, fact)
0



Source code:

```
#include<stdio.h>
int find_factorial(int);
int main()
{
   int num, fact;
   printf("Enter any integer number:");
   scanf("%d",&num);

   fact =find_factorial(num);

   printf("factorial of %d is: %d",num, fact);
   return 0;
}
int find_factorial(int n)
{
   if(n==0)
      return(1);
   return(n*find_factorial(n-1));
}
```

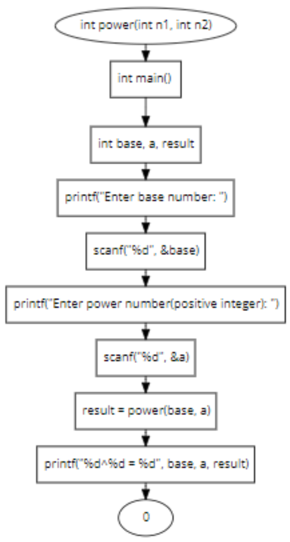
Output:

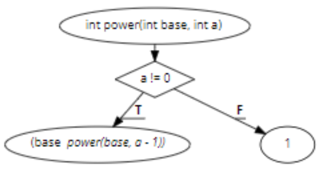


```
Enter any integer number: 5
factorial of 5 is: 120
```

Practical 5: Write a program in C to calculate the power of any number using recursion.

Pseudocode/Logic/Algorithm/flow chart:

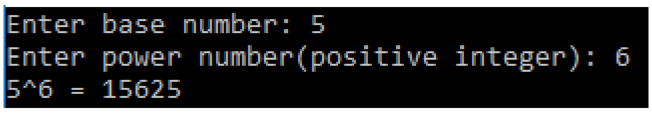

int power(int n1, int n2)
int main()
int base, a, result
printf("Enter base number: ")
scanf("%d", &base)
printf("Enter power number(positive integer): ")
scanf("%d", &a)
result = power(base, a)
printf("%d^%d = %d", base, a, result)
0



Source Code:

```
#include <stdio.h>
int power(int n1, int n2);
int main()
{
    int base, a, result;
    printf("Enter base number: ");
    scanf("%d", &base);
    printf("Enter power number(positive integer): ");
    scanf("%d", &a);
    result = power(base, a);
    printf("%d^%d = %d", base, a, result);
    return 0;
}

int power(int base, int a)
{
    if (a != 0)
        return (base * power(base, a - 1));
    else
        return 1;
}
```

Output:



```
Enter base number: 5
Enter power number(positive integer): 6
5^6 = 15625
```

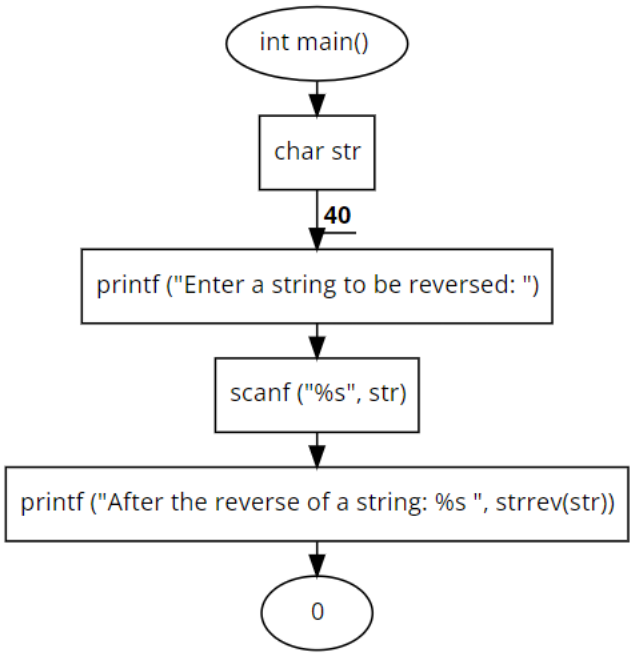
Conclusion:

Recursion refers to the repetitive calling of the same function. The function is designed in such a way that at the end of the sequential calling process we finally get to our desired output. Recursion helps solve many coding problems. It plays a very important role in coding.

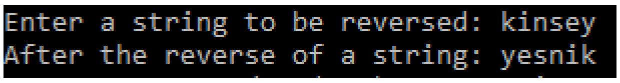
LAB V: - Write Programs based on String

Practical 1: Write a program in C to print individual characters of string in reverse order.

Pseudocode/Logic/Algorithm/flow chart:

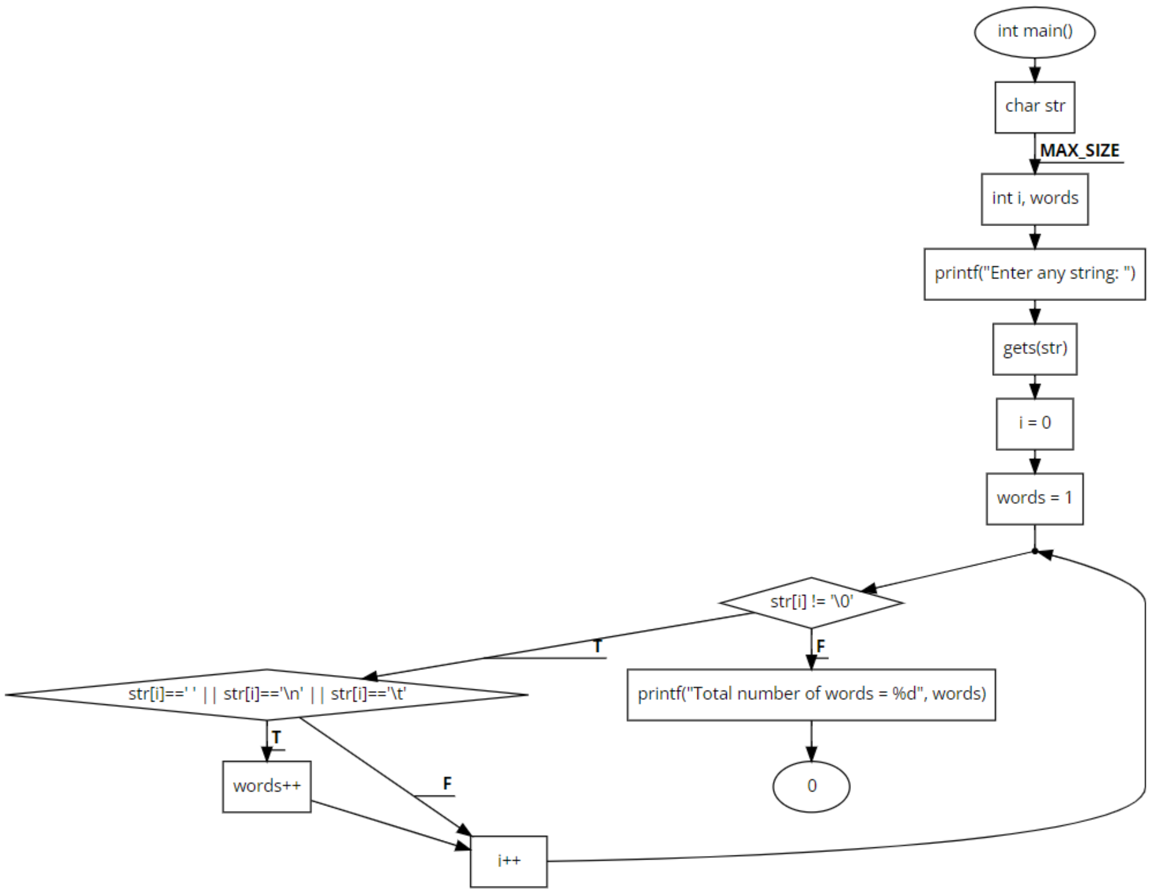

int main()
char str
40
printf ("Enter a string to be reversed: ")
scanf ("%s", str)
printf ("After the reverse of a string: %s ", strrev(str))
0

Source Code:
```
#include <stdio.h>
#include <string.h>
int main()
{
   char str[40];
   printf ("Enter a string to be reversed: ");
   scanf ("%s", str);

   printf ("After the reverse of a string: %s ", strrev(str));
   return 0;
}
```
Output:


```
Enter a string to be reversed: kinsey
After the reverse of a string: yesnik
```

Practical 2: Write a program in C to count the total number of words in a string.

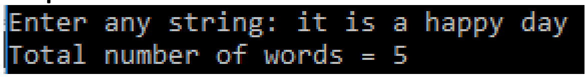
Pseudocode/Logic/Algorithm/flow chart:


int main()
char str
MAX_SIZE
int i, words
printf("Enter any string: ")
gets(str)
i = 0
words = 1
str[i] != '\0'
T
F
str[i]==' ' || str[i]=='\n' || str[i]=='\t'
printf("Total number of words = %d", words)
T
F
words++
0
i++

Source Code:

```
#include <stdio.h>
#define MAX_SIZE 100 // Maximum string size

int main()
{
   char str[MAX_SIZE];
   int i, words;
   printf("Enter any string: ");
   gets(str);
   i = 0;
   words = 1;
   while(str[i] != '\0')
   {
      if(str[i]==' ' || str[i]=='\n' || str[i]=='\t')
      {
         words++;
      }
      i++;
   }
   printf("Total number of words = %d", words);
   return 0;
}
```

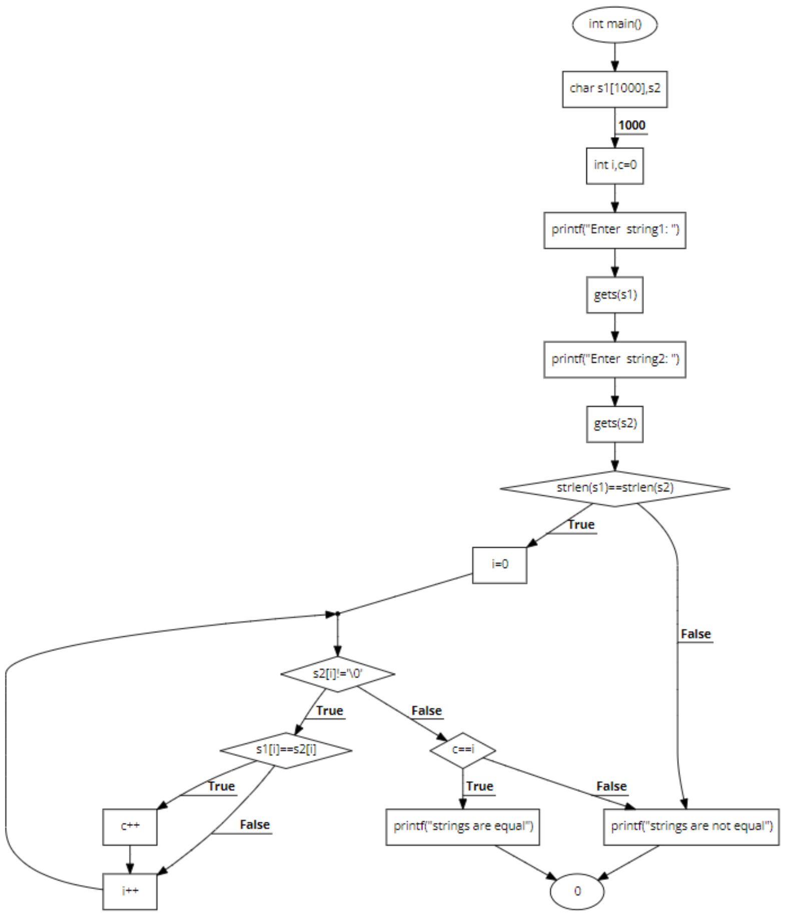
Output:



```
Enter any string: it is a happy day
Total number of words = 5
```

Practical 3: Write a program in C to compare two strings without using string library functions.

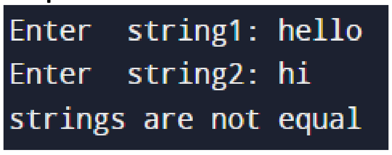
Pseudocode/Logic/Algorithm/flow chart:



Source Code:

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s1[1000],s2[1000];
  int i,c=0;
  printf("Enter  string1: ");
  gets(s1);
  printf("Enter  string2: ");
  gets(s2);
  if(strlen(s1)==strlen(s2))
  {
        for(i=0;s2[i]!='\0';i++)
    {
        if(s1[i]==s2[i])
         c++;
          }
          if(c==i)
     printf("strings are equal");
    else
     printf("strings are not equal");
  }
  else
   printf("strings are not equal");
  return 0;
}
```

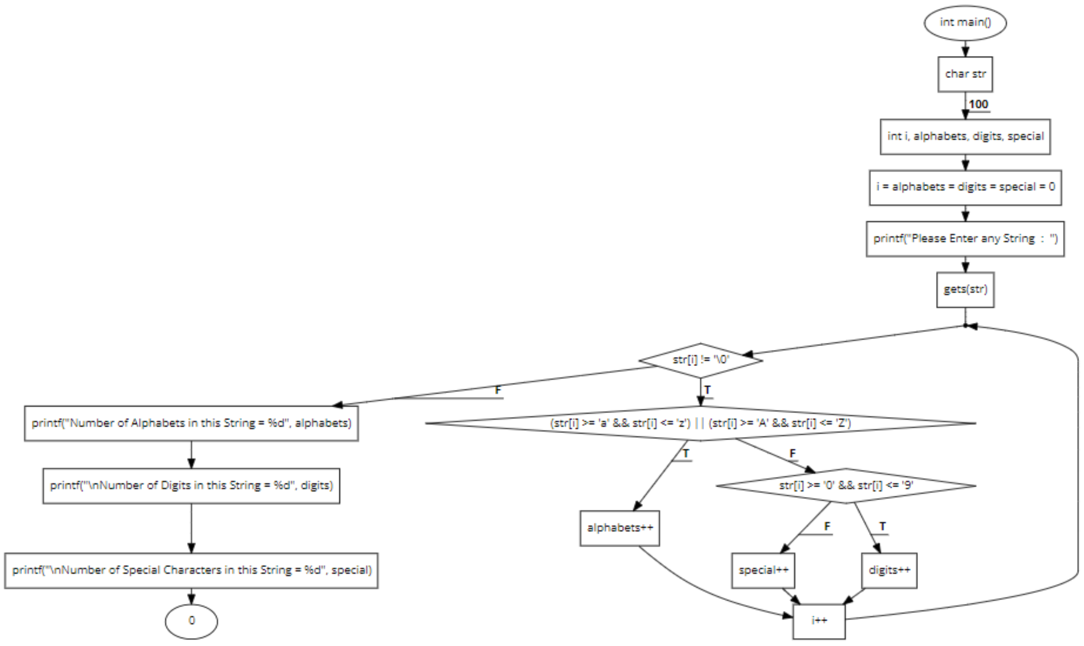
Output:



```
Enter  string1: hello
Enter  string2: hi
strings are not equal
```

Practical 4: Write a program in C to count total number of alphabets, digits and special characters in a string.

Pseudocode/Logic/Algorithm/flow chart:

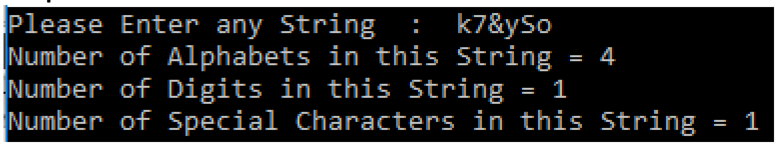

int main()
char str
100
int i, alphabets, digits, special
i = alphabets = digits = special = 0
printf("Please Enter any String : ")
gets(str)
str[i] != '\0'
F
T
printf("Number of Alphabets in this String = %d", alphabets)
(str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z')
T
F
printf("\nNumber of Digits in this String = %d", digits)
str[i] >= '0' && str[i] <= '9'
alphabets++
F
T
printf("\nNumber of Special Characters in this String = %d", special)
special++
digits++
0
i++

Source Code:

```
#include <stdio.h>
int main()
{
            char str[100];
            int i, alphabets, digits, special;
            i = alphabets = digits = special = 0;
            printf("Please Enter any String  :  ");
            gets(str);
            while (str[i] != '\0')
            {
                        if( (str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z') )
                        {
                                    alphabets++;
                        }
                        else if (str[i] >= '0' && str[i] <= '9')
                        {
                                    digits++;
```

```
                }
                else
                special++;
        i++;
        }
  printf("Number of Alphabets in this String = %d", alphabets);
        printf("\nNumber of Digits in this String = %d", digits);
        printf("\nNumber of Special Characters in this String = %d", special);
        return 0;
}
```

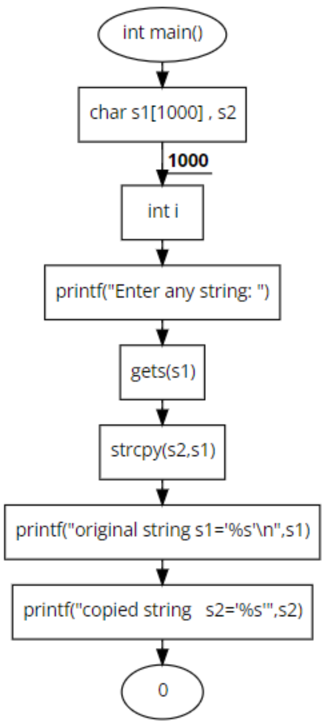
Output:



```
Please Enter any String  :  k7&ySo
Number of Alphabets in this String = 4
Number of Digits in this String = 1
Number of Special Characters in this String = 1
```

Practical 5: Write a program in C to copy one string to another string.

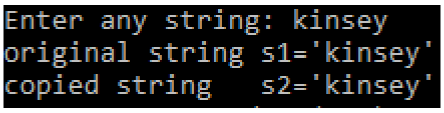
Pseudocode/Logic/Algorithm/flow chart:


int main()
char s1[1000] , s2
1000
int i
printf("Enter any string: ")
gets(s1)
strcpy(s2,s1)
printf("original string s1='%s'\n",s1)
printf("copied string s2='%s'",s2)
0

Source Code:

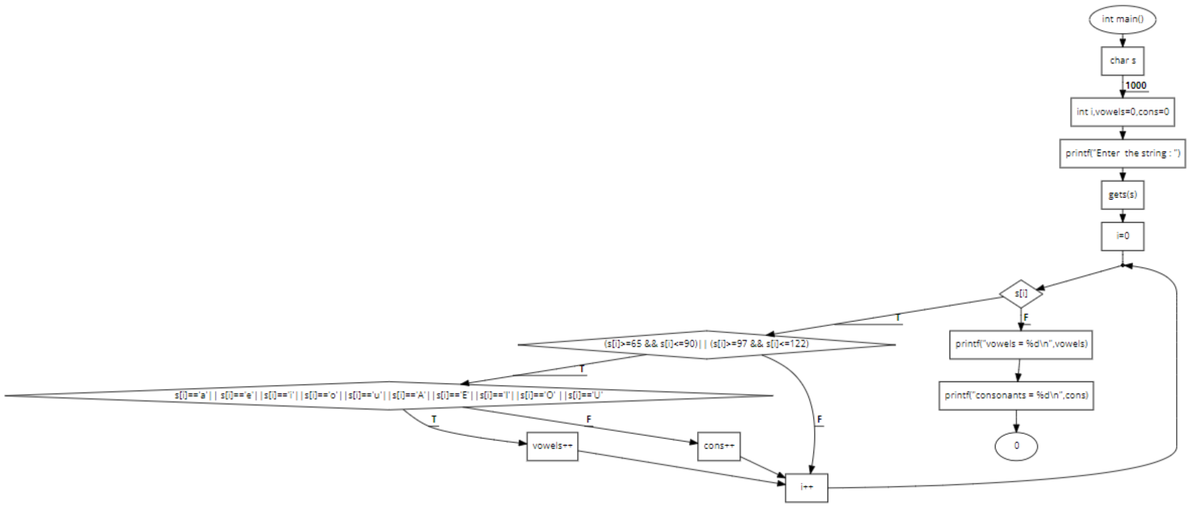
```
#include <stdio.h>
#include <string.h>
int main()
{
  char s1[1000],s2[1000];
  int i;

  printf("Enter any string: ");
  gets(s1);
  strcpy(s2,s1);
  printf("original string s1='%s'\n",s1);
  printf("copied string  s2='%s'",s2);
  return 0;
}
```

Output:



```
Enter any string: kinsey
original string s1='kinsey'
copied string   s2='kinsey'
```

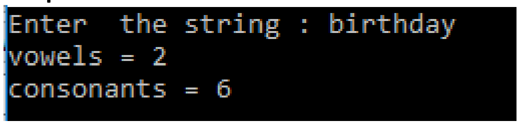
Practical 6: Write a program in C to count total number of vowel or consonant in a string.
Pseudocode/Logic/Algorithm/flow chart:


int main()
char s
1000
int i,vowels=0,cons=0
printf("Enter the string : ")
gets(s)
i=0
s[i]
T
F
(s[i]>=65 && s[i]<=90) | | (s[i]>=97 && s[i]<=122)
printf("vowels = %d\n",vowels)
printf("consonants = %d\n",cons)
0
s[i]=='a' | | s[i]=='e' | | s[i]=='i' | | s[i]=='o' | | s[i]=='u' | | s[i]=='A' | | s[i]=='E' | | s[i]=='I' | | s[i]=='O' | | s[i]=='U'
vowels++
cons++
i++

Source Code:

```
#include <stdio.h>
#include <string.h>
int main()
{
   char s[1000];
   int i,vowels=0,cons=0;
   printf("Enter  the string : ");
   gets(s);
   for(i=0;s[i];i++)
   {
            if((s[i]>=65 && s[i]<=90)|| (s[i]>=97 && s[i]<=122))
            {

         if(s[i]=='a'|| s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u'||s[i]=='A'||s[i]=='E'||s[i]=='I'||s[i]=='O' ||s[i]=='U')
                        vowels++;
         else
          cons++;
      }
            }
   printf("vowels = %d\n",vowels);
   printf("consonants = %d\n",cons);
   return 0;
}
```

Output:



```
Enter  the string : birthday
vowels = 2
consonants = 6
```

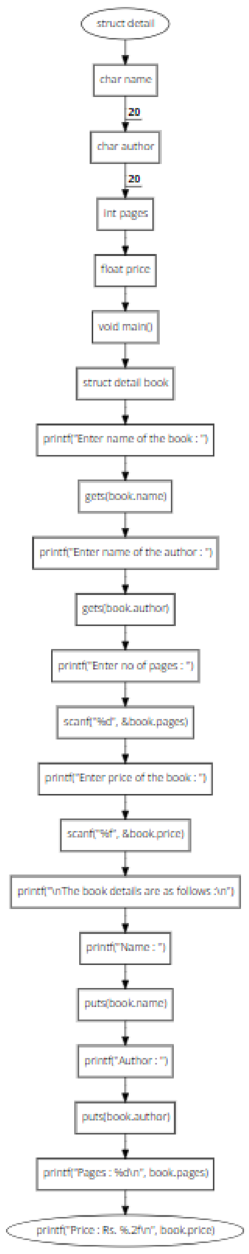
Conclusion:

Strings are a very important aspect of any programming language. Strings in C are implemented using long arrays of characters. Since strings are implemented in form of arrays, they are very manipulable in C language compared to many other programming languages such as python. Strings are very useful while coding

LAB 6: - Programs Based on Structure and Union

Practical 1: Write a C program to create a structure of Book Detail and display the details of the book in appropriate format by passing structure as function argument.

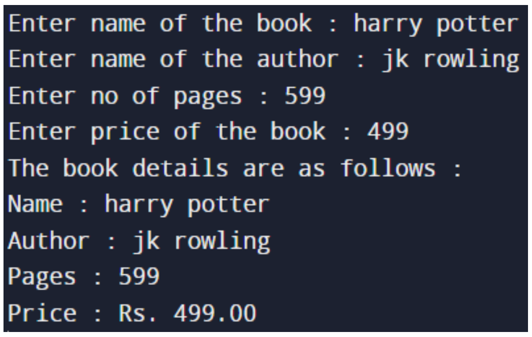
Pseudocode/Logic/Algorithm/flow chart:


struct detail
char name
20
char author
20
int pages
float price
void main()
struct detail book
printf("Enter name of the book : ")
gets(book.name)
printf("Enter name of the author : ")
gets(book.author)
printf("Enter no of pages : ")
scanf("%d", &book.pages)
printf("Enter price of the book : ")
scanf("%f", &book.price)
printf("\nThe book details are as follows :\n")
printf("Name : ")
puts(book.name)
printf("Author : ")
puts(book.author)
printf("Pages : %d\n", book.pages)
printf("Price : Rs. %.2f\n", book.price)

Source Code:

```
#include <stdio.h>
struct detail
{
   char name[20];
   char author[20];
   int pages;
   float price;
};
void main()
{
   struct detail book;
   printf("Enter name of the book : ");
   gets(book.name);
   printf("Enter name of the author : ");
   gets(book.author);
   printf("Enter no of pages : ");
   scanf("%d", &book.pages);
   printf("Enter price of the book : ");
   scanf("%f", &book.price);

   printf("\nThe book details are as follows :\n");
   printf("Name : ");
   puts(book.name);
   printf("Author : ");
   puts(book.author);
   printf("Pages : %d\n", book.pages);
   printf("Price : Rs. %.2f\n", book.price);
}
```

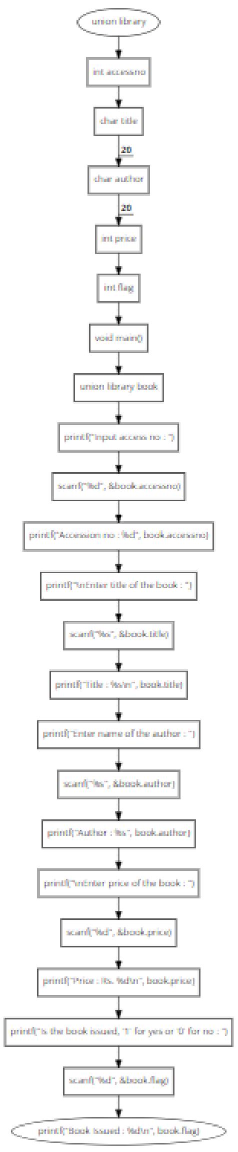
Output:



```
Enter name of the book : harry potter
Enter name of the author : jk rowling
Enter no of pages : 599
Enter price of the book : 499
The book details are as follows :
Name : harry potter
Author : jk rowling
Pages : 599
Price : Rs. 499.00
```

Practical 2: Create a Union called library to hold accession number, title of the book, author name, price of the book and flag indicating whether the book is issued or not. (flag = 1 if the book is issued, flag = 0 otherwise). Write a program to enter data of one book and display the data.

Pseudocode/Logic/Algorithm/flow chart:

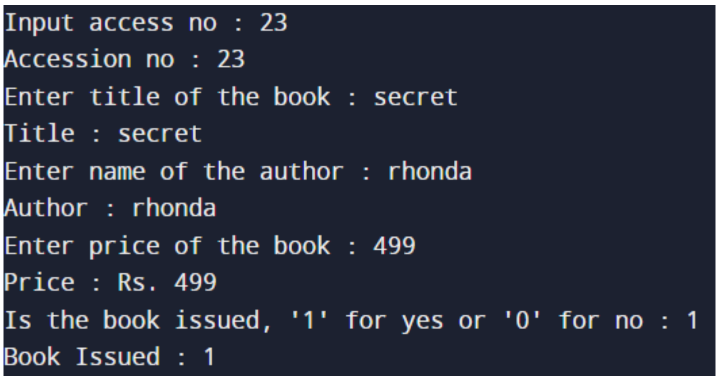

union library
int accessno
char title
20
char author
20
int price
int flag
void main()
union library book
printf("Input access no : ")
scanf("%d", &book.accessno)
printf("Accession no : %d", book.accessno)
printf("\nEnter title of the book : ")
scanf("%s", &book.title)
printf("Title : %s\n", book.title)
printf("Enter name of the author : ")
scanf("%s", &book.author)
printf("Author : %s", book.author)
printf("\nEnter price of the book : ")
scanf("%d", &book.price)
printf("Price : Rs. %d\n", book.price)
printf("Is the book issued, '1' for yes or '0' for no : ")
scanf("%d", &book.flag)
printf("Book issued : %d\n", book.flag)

Source Code:

```
#include <stdio.h>
union library
{
   int accessno;
   char title[20];
   char author[20];
   int price;
   int flag;
};
void main()
{
   union library book;
   printf("Input access no : ");
   scanf("%d", &book.accessno);
   printf("Accession no : %d", book.accessno);
   printf("\nEnter title of the book : ");
   scanf("%s", &book.title);
```

```
    printf("Title : %s\n", book.title);

    printf("Enter name of the author : ");
    scanf("%s", &book.author);
    printf("Author : %s", book.author);

    printf("\nEnter price of the book : ");
    scanf("%d", &book.price);
    printf("Price : Rs. %d\n", book.price);
    printf("Is the book issued, '1' for yes or '0' for no : ");
    scanf("%d", &book.flag);
    printf("Book Issued : %d\n", book.flag);
}
```

Output:



```
Input access no : 23
Accession no : 23
Enter title of the book : secret
Title : secret
Enter name of the author : rhonda
Author : rhonda
Enter price of the book : 499
Price : Rs. 499
Is the book issued, '1' for yes or '0' for no : 1
Book Issued : 1
```

Conclusion:

Structures and Unions provide the facility to aggregate different datatypes into a singular module. This module can be called upon any requirement. It also makes it easier to access and modify data. Structures and Unions may seem similar but serve different purposes.

LAB 7: - Programs Based on File Handling

Q1. Write a program in c to read an existing file.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int num;
    FILE *fptr;
    if ((fptr = fopen("C:\\program.txt","r")) == NULL)
    {
       printf("Error! opening file");

       exit(1);
    }
    fscanf(fptr,"%d", &num);
    printf("Value of n=%d", num);
```

```
   fclose(fptr);
   return 0;
}
```

Output:

Q2. Write a program in C to Find the Number of Lines in a Text File.

Source Code:

```
#include <stdio.h>
#define FSIZE 100
int main()
{
   FILE *fptr;
   int ctr = 0;
   char fname[FSIZE];
   char c;
         printf(" Input the file name to be opened : ");
         scanf("%s",fname);

   fptr = fopen(fname, "r");
   if (fptr == NULL)
   {
      printf("Could not open file %s", fname);
      return 0;
   }
   for (c = getc(fptr); c != EOF; c = getc(fptr))
      if (c == '\n')
         ctr = ctr + 1;
   fclose(fptr);
   printf(" The lines in the file %s are : %d \n \n", fname, ctr-1);
   return 0;
}
```

Output:

Q3. Write a program in C to count a number of words and characters in a file.

Source Code:

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

void main()
{
   FILE *fptr;
   char ch;
   int wrd=1,charctr=1;
   char fname[20];
         printf(" Input the filename to be opened : ");
         scanf("%s",fname);
   fptr=fopen(fname,"r");
   if(fptr==NULL)
   {
        printf(" File does not exist or can not be opened.");
    }
    else
    {
        ch=fgetc(fptr);
        printf(" The content of the file %s are : ",fname);
        while(ch!=EOF)
        {
            printf("%c",ch);
            if(ch==' '||ch=='\n')
            {
                wrd++;
            }
            else
            {
                charctr++;
            }
                ch=fgetc(fptr);
        }
        printf("\n The number of words in the  file %s are : %d\n",fname,wrd-2);
        printf(" The number of characters in the  file %s are : %d\n\n",fname,charctr-1);
    }
    fclose(fptr);
}
```

Q4. Write a program in C to copy a file in another file name.

Source Code:

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

void main()
{
        FILE *fptr1, *fptr2;
        char ch, fname1[20], fname2[20];
        printf(" Input the source file name : ");
        scanf("%s",fname1);
        fptr1=fopen(fname1, "r");
        if(fptr1==NULL)
        {
                printf(" File does not found or error in opening.!!");
                exit(1);
        }
        printf(" Input the new file name : ");
        scanf("%s",fname2);
        fptr2=fopen(fname2, "w");
        if(fptr2==NULL)
        {
                printf(" File does not found or error in opening.!!");
                fclose(fptr1);
                exit(2);
        }
        while(1)
        {
                ch=fgetc(fptr1);
                if(ch==EOF)
                {
                        break;
                }
                else
                {
                        fputc(ch, fptr2);
                }
        }
        printf(" The file %s  copied successfully in the file %s. \n\n",fname1,fname2);
        fclose(fptr1);
        fclose(fptr2);
        getchar();
}
```

Output:

Q5. Write a program in C to merge two files and write it in a new file.

Source Code:

```
#include<stdio.h>
#include<conio.h>
int main()
{
  FILE *fsOne, *fsTwo, *fTarget;
  char fName1[20], fName2[20], fName3[30], ch;
  printf("Enter the Name of First Source File: ");
  gets(fName1);
  printf("Enter the Name of Second Source File: ");
  gets(fName2);
  printf("\nEnter the Name of Target File: ");
  gets(fName3);
  fsOne = fopen(fName1, "r");
  fsTwo = fopen(fName2, "r");
  if(fsOne==NULL || fsTwo==NULL)
  {
    printf("\nError Occurred while Opening the Source File!");
  }
  else
  {
    fTarget = fopen(fName3, "w");
    if(fTarget==NULL)
    {
      printf("\nError Occurred while Opening the Target File!");
    }
    else
    {
      ch = fgetc(fsOne);
      while(ch!=EOF)
      {
        fputc(ch, fTarget);
        ch = fgetc(fsOne);
      }
      ch = fgetc(fsTwo);
      while(ch!=EOF)
      {
           fputc(ch, fTarget);
           ch = fgetc(fsTwo);
      }
      printf("\n%s and %s Merged into %s Successfully!", fName1, fName2, fName3);
    }
  }
  fclose(fsOne);
  fclose(fsTwo);
  fclose(fTarget);
  getch();
  return 0;
}
```

Output:

LAB 8: - Write Menu Driven Programs

Q1. Design, Develop and Implement a menu driven Program in C for the following array operation.

- **Creating an array of N Integer Elements**
- **Display of array Elements with Suitable Headings**
- **Inserting an Element (ELEM) at a given valid Position (POS)**
- **Deleting an Element at a given valid Position (POS)**
- **Exit.**

Support the program with functions for each of the above operations.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int a[20];
int n, val, i, pos;
void create();
void display();
void insert();
void Delete();
int main()
{
   int choice = 1;
   while (choice)
   {
      printf("\n\n———MENU————\n");
      printf("1.CREATE\n");
      printf("2.DISPLAY\n");
      printf("3.INSERT\n");
      printf("4.DELETE\n");
      printf("5.EXIT\n");
      printf("————————“);
      printf("\nENTER YOUR CHOICE:\t");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
          create();
          break;
      case 2:
          display();
          break;
      case 3:
          insert();
          break;
      case 4:
          Delete();
          break;
      case 5:
          exit(0);
      default:
          printf("\nInvalid choice:\n");
          break;
      }
   }
   return 0;
}
//creating an array
void create()
{
   printf("\nEnter the size of the array elements:\t");
   scanf("%d", &n);
   printf("\nEnter the elements for the array:\n");
   for (i = 0; i < n; i++)
   {
      scanf("%d", &a[i]);
   }
}
//displaying an array elements
void display()
{
   // int i;
   printf("\nThe array elements are:\n");
   for (i = 0; i < n; i++)
   {
      printf("%d\t", a[i]);
   }
}
//inserting an element into an array
void insert()
{
   printf("\nEnter the position for the new element:\t");
   scanf("%d", &pos);
   printf("\nEnter the element to be inserted :\t");
   scanf("%d", &val);
   for (i = n – 1; i >= pos; i–)
   {
      a[i + 1] = a[i];
   }
   a[pos] = val;
   n = n + 1;
}
//deleting an array element
```

```
void Delete()
{
   printf("\nEnter the position of the element to be deleted:\t");
   scanf("%d", &pos);
   val = a[pos];
   for (i = pos; i < n – 1; i++)
   {
      a[i] = a[i + 1];
   }
   n = n – 1;
   printf("\nThe deleted element is =%d", val);
}
```

Output:

Q2. Write a menu-driven program to perform following Matrix operations (2-D array implementation):

- Sum
- Difference
- Product
- Transpose

Source Code:

LAB 8: - Programs Based on Linked Lists

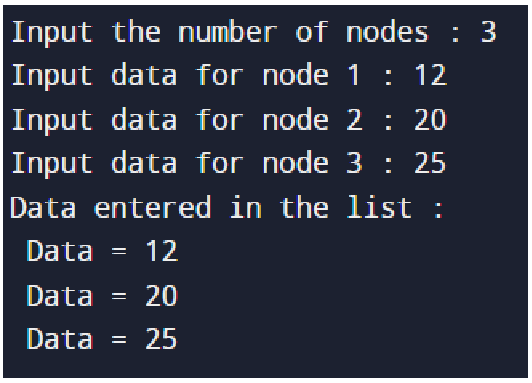
Q1. Write a program in C to create and display Singly Linked List.

Source Code:

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

struct node
{
   int num;
   struct node *nextptr;
}*stnode;

void createNodeList(int n);
void displayList();

int main()
{
   int n;
   printf(" Input the number of nodes : ");
   scanf("%d", &n);
   createNodeList(n);
   printf("\n Data entered in the list : \n");
   displayList();
   return 0;
}
void createNodeList(int n)
{
   struct node *fnNode, *tmp;
```

```
    int num, i;
    stnode = (struct node *)malloc(sizeof(struct node));

    if(stnode == NULL)
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        stnode->num = num;
        stnode->nextptr = NULL;
        tmp = stnode;
        for(i=2; i<=n; i++)
        {
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode == NULL)
            {
                printf(" Memory can not be allocated.");
                break;
            }
            else
            {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);

                fnNode->num = num;
                fnNode->nextptr = NULL;
                tmp->nextptr = fnNode;
                tmp = tmp->nextptr;
            }
        }
    }
}
void displayList()
{
    struct node *tmp;
    if(stnode == NULL)
    {
        printf(" List is empty.");
    }
    else
    {
        tmp = stnode;
        while(tmp != NULL)
        {
            printf(" Data = %d\n", tmp->num);
            tmp = tmp->nextptr;
        }
    }
}
```

Output:



```
Input the number of nodes : 3
Input data for node 1 : 12
Input data for node 2 : 20
Input data for node 3 : 25
Data entered in the list :
 Data = 12
 Data = 20
 Data = 25
```

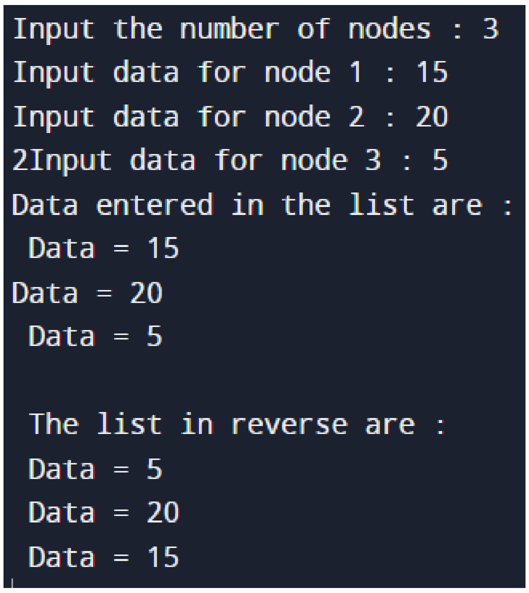
Q2. Write a program in C to create a singly linked list of n nodes and display it in reverse order.

Source Code:

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

struct node
{
   int num;
   struct node *nextptr;
}*stnode;

void createNodeList(int n);
void reverseDispList();
void displayList();

int main()
{
   int n;
   printf(" Input the number of nodes : ");
   scanf("%d", &n);
   createNodeList(n);
   printf("\n Data entered in the list are : \n");
   displayList();
   reverseDispList();
   printf("\n The list in reverse are :  \n");
   displayList();
   return 0;
}

void createNodeList(int n)
{
   struct node *fnNode, *tmp;
   int num, i;
   stnode = (struct node *)malloc(sizeof(struct node));
   if(stnode == NULL)
   {
      printf(" Memory can not be allocated.");
   }
   else
   {
      printf(" Input data for node 1 : ");
      scanf("%d", &num);
```

```
        stnode-> num = num;
        stnode-> nextptr = NULL;
        tmp = stnode;
        for(i=2; i<=n; i++)
        {
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode == NULL)
            {
                printf(" Memory can not be allocated.");
                break;
            }
            else
            {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);
                fnNode->num = num;
                fnNode->nextptr = NULL;
                tmp->nextptr = fnNode;
                tmp = tmp->nextptr;
            }
        }
    }
}

void reverseDispList()
{
    struct node *prevNode, *curNode;

    if(stnode != NULL)
    {
        prevNode = stnode;
        curNode = stnode->nextptr;
        stnode = stnode->nextptr;

        prevNode->nextptr = NULL;
        while(stnode != NULL)
        {
            stnode = stnode->nextptr;
            curNode->nextptr = prevNode;

            prevNode = curNode;
            curNode = stnode;
        }
        stnode = prevNode;
    }
}

void displayList()
{
    struct node *tmp;
    if(stnode == NULL)
    {
        printf(" No data found in the list.");
    }
    else
    {
        tmp = stnode;
        while(tmp != NULL)
        {
            printf(" Data = %d\n", tmp->num);
            tmp = tmp->nextptr;
        }
    }
}
```

Output:



```
Input the number of nodes : 3
Input data for node 1 : 15
Input data for node 2 : 20
2Input data for node 3 : 5
Data entered in the list are :
 Data = 15
Data = 20
 Data = 5

 The list in reverse are :
 Data = 5
 Data = 20
 Data = 15
```

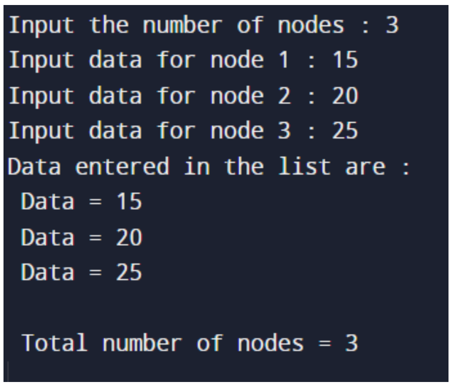
Q3. Write a program in C to create a singly linked list of n nodes and count the number of nodes.

Source Code:

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


struct node
{
    int num;
    struct node *nextptr;
}*stnode;

void createNodeList(int n);
int NodeCount();
void displayList();

int main()
{
    int n,totalNode;
    printf(" Input the number of nodes : ");
    scanf("%d", &n);
    createNodeList(n);
    printf("\n Data entered in the list are : \n");
    displayList();
    totalNode = NodeCount();
    printf("\n Total number of nodes = %d\n", totalNode);
    return 0;
}
void createNodeList(int n)
{
    struct node *fnNode, *tmp;
    int num, i;
    stnode = (struct node *)malloc(sizeof(struct node));
    if(stnode == NULL)
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        stnode-> num = num;
        stnode-> nextptr = NULL;
        tmp = stnode;
        for(i=2; i<=n; i++)
        {
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode == NULL)
            {
                printf(" Memory can not be allocated.");
                break;
            }
            else
            {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);
                fnNode->num = num;
                fnNode->nextptr = NULL;
                tmp->nextptr = fnNode;
                tmp = tmp->nextptr;
            }
        }
    }
}

int NodeCount()
{
    int ctr = 0;
    struct node *tmp;
    tmp = stnode;
    while(tmp != NULL)
    {
        ctr++;
        tmp = tmp->nextptr;
    }
    return ctr;
}
void displayList()
{
    struct node *tmp;
    if(stnode == NULL)
    {
        printf(" No data found in the list.");
```

```
  }
  else
  {
    tmp = stnode;
    while(tmp != NULL)
    {
      printf(" Data = %d\n", tmp->num);
      tmp = tmp->nextptr;
    }
  }
}
```

Output:



```
Input the number of nodes : 3
Input data for node 1 : 15
Input data for node 2 : 20
Input data for node 3 : 25
Data entered in the list are :
 Data = 15
 Data = 20
 Data = 25

 Total number of nodes = 3
```

Q4. Write a menu driven program to implement various operations of Singly Linked List.

Source Code:

```
// C program for the all operations in
// the Singly Linked List
#include <stdio.h>
#include <stdlib.h>
// Linked List Node
struct node {
        int info;
        struct node* link;
};
struct node* start = NULL;

// Function to create list with n nodes initially
void createList()
{
        if (start == NULL) {
                int n;
                printf("\nEnter the number of nodes: ");
                scanf("%d", &n);
                if (n != 0) {
                        int data;
                        struct node* newnode;
                        struct node* temp;
                        newnode = malloc(sizeof(struct node));
                        start = newnode;
                        temp = start;
                        printf("\nEnter number to"
                                " be inserted : ");
                        scanf("%d", &data);
                        start->info = data;

                        for (int i = 2; i <= n; i++) {
                                newnode = malloc(sizeof(struct node));
                                temp->link = newnode;
                                printf("\nEnter number to"
                                        " be inserted : ");
                                scanf("%d", &data);
                                newnode->info = data;
                                temp = temp->link;
                        }
                }
                printf("\nThe list is created\n");
        }
        else
                printf("\nThe list is already created\n");
}

// Function to traverse the linked list
void traverse()
{
        struct node* temp;

        // List is empty
        if (start == NULL)
                printf("\nList is empty\n");

        // Else print the LL
        else {
                temp = start;
                while (temp != NULL) {
                        printf("Data = %d\n", temp->info);
                        temp = temp->link;
                }
        }
}

// Function to insert at the front
// of the linked list
void insertAtFront()
{
        int data;
        struct node* temp;
        temp = malloc(sizeof(struct node));
        printf("\nEnter number to"
                " be inserted : ");
        scanf("%d", &data);
        temp->info = data;

        // Pointer of temp will be
        // assigned to start
        temp->link = start;
        start = temp;
}
```

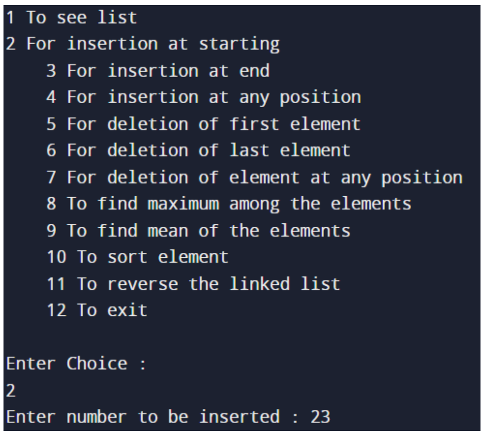
```
// Function to insert at the end of
// the linked list
void insertAtEnd()
{
        int data;
        struct node *temp, *head;
        temp = malloc(sizeof(struct node));

        // Enter the number
        printf("\nEnter number to"
                " be inserted : ");
        scanf("%d", &data);

        // Changes links
        temp->link = 0;
        temp->info = data;
        head = start;
        while (head->link != NULL) {
                head = head->link;
        }
        head->link = temp;
}

// Function to insert at any specified
// position in the linked list
void insertAtPosition()
{
        struct node *temp, *newnode;
        int pos, data, i = 1;
        newnode = malloc(sizeof(struct node));

        // Enter the position and data
        printf("\nEnter position and data :");
        scanf("%d %d", &pos, &data);

        // Change Links
        temp = start;
        newnode->info = data;
        newnode->link = 0;
        while (i < pos - 1) {
                temp = temp->link;
                i++;
        }
        newnode->link = temp->link;
        temp->link = newnode;
}

// Function to delete from the front
// of the linked list
void deleteFirst()
{
        struct node* temp;
        if (start == NULL)
                printf("\nList is empty\n");
        else {
                temp = start;
                start = start->link;
                free(temp);
        }
}

// Function to delete from the end
// of the linked list
void deleteEnd()
{
        struct node *temp, *prevnode;
        if (start == NULL)
                printf("\nList is Empty\n");
        else {
                temp = start;
                while (temp->link != 0) {
                        prevnode = temp;
                        temp = temp->link;
                }
                free(temp);
                prevnode->link = 0;
        }
}

// Function to delete from any specified
// position from the linked list
void deletePosition()
{
        struct node *temp, *position;
        int i = 1, pos;

        // If LL is empty
        if (start == NULL)
                printf("\nList is empty\n");

        // Otherwise
        else {
                printf("\nEnter index : ");

                // Position to be deleted
                scanf("%d", &pos);
                position = malloc(sizeof(struct node));
                temp = start;

                // Traverse till position
                while (i < pos - 1) {
                        temp = temp->link;
                        i++;
                }

                // Change Links
                position = temp->link;
                temp->link = position->link;

                // Free memory
                free(position);
        }
}
```

```
// Function to find the maximum element
// in the linked list
void maximum()
{
	int a[10];
	int i;
	struct node* temp;

	// If LL is empty
	if (start == NULL)
		printf("\nList is empty\n");

	// Otherwise
	else {
		temp = start;
		int max = temp->info;

		// Traverse LL and update the
		// maximum element
		while (temp != NULL) {

			// Update the maximum
			// element
			if (max < temp->info)
				max = temp->info;
			temp = temp->link;
		}
		printf("\nMaximum number "
			"is : %d ",
			max);
	}
}

// Function to find the mean of the
// elements in the linked list
void mean()
{
	int a[10];
	int i;
	struct node* temp;

	// If LL is empty
	if (start == NULL)
		printf("\nList is empty\n");

	// Otherwise
	else {
		temp = start;

		// Stores the sum and count of
		// element in the LL
		int sum = 0, count = 0;
		float m;

		// Traverse the LL
		while (temp != NULL) {
```

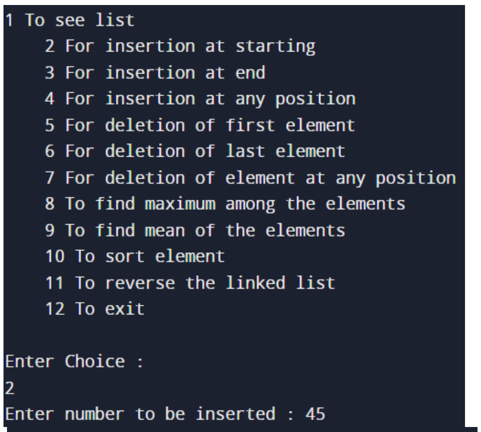
```
                        // Update the sum
                        sum = sum + temp->info;
                        temp = temp->link;
                        count++;
                }

                // Find the mean
                m = sum / count;

                // Print the mean value
                printf("\nMean is %f ", m);
        }
}

// Function to sort the linked list
// in ascending order
void sort()
{
        struct node* current = start;
        struct node* index = NULL;
        int temp;

        // If LL is empty
        if (start == NULL) {
                return;
        }

        // Else
        else {

                // Traverse the LL
                while (current != NULL) {
                        index = current->link;

                        // Traverse the LL nestedly
                        // and find the minimum
                        // element
                        while (index != NULL) {

                                // Swap with it the value
                                // at current
                                if (current->info > index->info) {
                                        temp = current->info;
                                        current->info = index->info;
                                        index->info = temp;
                                }
                                index = index->link;
                        }

                        // Update the current
                        current = current->link;
                }
        }
}

// Function to reverse the linked list
void reverseLL()
```

```
{
	struct node *t1, *t2, *temp;
	t1 = t2 = NULL;

	// If LL is empty
	if (start == NULL)
		printf("List is empty\n");

	// Else
	else {

		// Traverse the LL
		while (start != NULL) {

			// reversing of points
			t2 = start->link;
			start->link = t1;
			t1 = start;
			start = t2;
		}
		start = t1;

		// New head Node
		temp = start;

		printf("Reversed linked "
			"list is : ");

		// Print the LL
		while (temp != NULL) {
			printf("%d ", temp->info);
			temp = temp->link;
		}
	}
}

// Driver Code
int main()
{
	int choice;
	while (1) {

		printf("\n\t1 To see list\n");
		printf("\t2 For insertion at"
			" starting\n");
		printf("\t3 For insertion at"
			" end\n");
		printf("\t4 For insertion at "
			"any position\n");
		printf("\t5 For deletion of "
			"first element\n");
		printf("\t6 For deletion of "
			"last element\n");
		printf("\t7 For deletion of "
			"element at any position\n");
		printf("\t8 To find maximum among"
			" the elements\n");
		printf("\t9 To find mean of "
			"the elements\n");
		printf("\t10 To sort element\n");
		printf("\t11 To reverse the "
			"linked list\n");
		printf("\t12 To exit\n");
		printf("\nEnter Choice :\n");
		scanf("%d", &choice);

		switch (choice) {
		case 1:
			traverse();
			break;
		case 2:
			insertAtFront();
			break;
		case 3:
			insertAtEnd();
			break;
		case 4:
			insertAtPosition();
			break;
		case 5:
			deleteFirst();
			break;
		case 6:
			deleteEnd();
			break;
		case 7:
			deletePosition();
			break;
		case 8:
			maximum();
			break;
		case 9:
			mean();
			break;
		case 10:
			sort();
			break;
		case 11:
			reverseLL();
			break;
		case 12:
			exit(1);
			break;
		default:
			printf("Incorrect Choice\n");
		}
	}
	return 0;
}
```

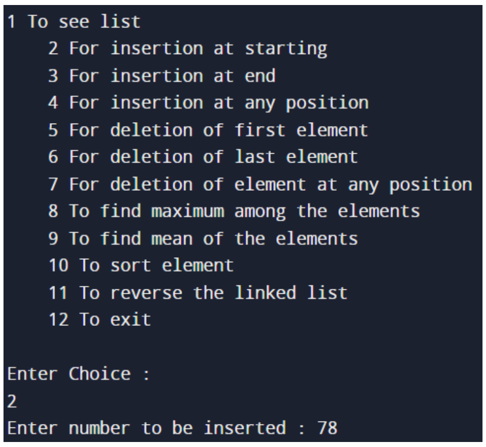
Output:



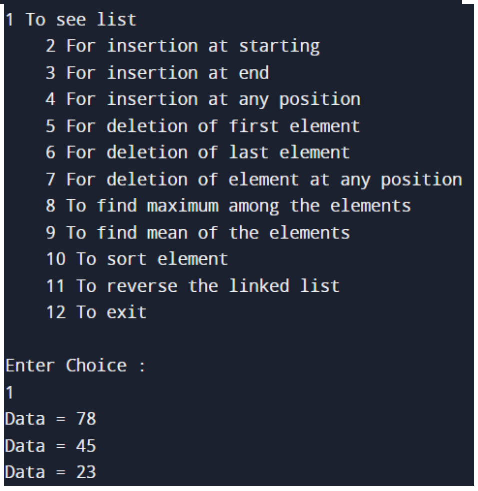
```
1 To see list
2 For insertion at starting
    3 For insertion at end
    4 For insertion at any position
    5 For deletion of first element
    6 For deletion of last element
    7 For deletion of element at any position
    8 To find maximum among the elements
    9 To find mean of the elements
    10 To sort element
    11 To reverse the linked list
    12 To exit

Enter Choice :
2
Enter number to be inserted : 23
```



```
1 To see list
    2 For insertion at starting
    3 For insertion at end
    4 For insertion at any position
    5 For deletion of first element
    6 For deletion of last element
    7 For deletion of element at any position
    8 To find maximum among the elements
    9 To find mean of the elements
    10 To sort element
    11 To reverse the linked list
    12 To exit

Enter Choice :
2
Enter number to be inserted : 45
```



```
1 To see list
    2 For insertion at starting
    3 For insertion at end
    4 For insertion at any position
    5 For deletion of first element
    6 For deletion of last element
    7 For deletion of element at any position
    8 To find maximum among the elements
    9 To find mean of the elements
    10 To sort element
    11 To reverse the linked list
    12 To exit

Enter Choice :
2
Enter number to be inserted : 78
```



```
1 To see list
    2 For insertion at starting
    3 For insertion at end
    4 For insertion at any position
    5 For deletion of first element
    6 For deletion of last element
    7 For deletion of element at any position
    8 To find maximum among the elements
    9 To find mean of the elements
    10 To sort element
    11 To reverse the linked list
    12 To exit

Enter Choice :
1
Data = 78
Data = 45
Data = 23
```

Q5. Write a program in C to create a doubly linked list and display in reverse order.

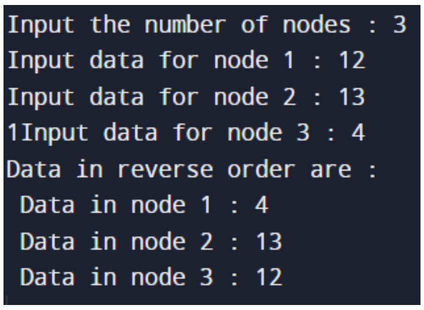
Source Code:

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

struct node {
   int num;
   struct node * preptr;
   struct node * nextptr;
}*stnode, *ennode;


void DlListcreation(int n);
void displayDlListRev();

int main()
{
   int n;
   stnode = NULL;
   ennode = NULL;
   printf(" Input the number of nodes : ");
   scanf("%d", &n);

   DlListcreation(n);
   displayDlListRev();
   return 0;
```

```
}

void DlListcreation(int n)
{
   int i, num;
   struct node *fnNode;

   if(n >= 1)
   {
      stnode = (struct node *)malloc(sizeof(struct node));

      if(stnode != NULL)
      {
         printf(" Input data for node 1 : ");
         scanf("%d", &num);

         stnode->num = num;
         stnode->preptr = NULL;
         stnode->nextptr = NULL;
         ennode = stnode;
         for(i=2; i<=n; i++)
         {
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode != NULL)
            {
               printf(" Input data for node %d : ", i);
               scanf("%d", &num);
               fnNode->num = num;
               fnNode->preptr = ennode;
               fnNode->nextptr = NULL;

               ennode->nextptr = fnNode;
               ennode = fnNode;
            }
            else
            {
               printf(" Memory can not be allocated.");
               break;
            }
         }
      }
      else
      {
         printf(" Memory can not be allocated.");
      }
   }
}

void displayDlListRev()
{
   struct node * tmp;
   int n = 0;

   if(ennode == NULL)
   {
      printf(" No data found in the List yet.");
   }
```

```
    else
    {
        tmp = ennode;
        printf("\n Data in reverse order are :\n");
        while(tmp != NULL)
        {
            printf(" Data in node %d : %d\n", n+1, tmp->num);
            n++;
            tmp = tmp->preptr;
        }
    }
}
```

Output:



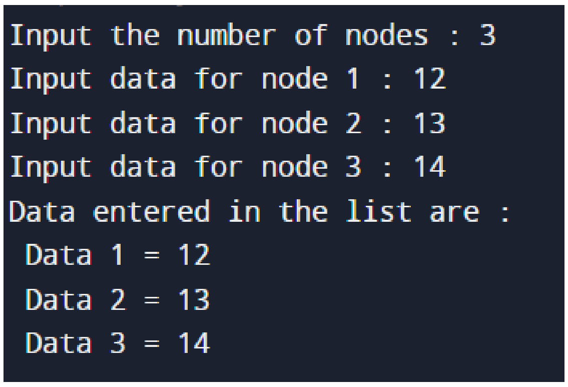
```
Input the number of nodes : 3
Input data for node 1 : 12
Input data for node 2 : 13
1Input data for node 3 : 4
Data in reverse order are :
 Data in node 1 : 4
 Data in node 2 : 13
 Data in node 3 : 12
```

Q6. Write a program in C to create and displaya circular linked list.

Source Code:

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

struct node {
    int num;
    struct node * nextptr;
}*stnode;


void ClListcreation(int n);
void displayClList();

int main()
{
    int n;
    stnode = NULL;
    printf(" Input the number of nodes : ");
    scanf("%d", &n);

    ClListcreation(n);
    displayClList();
    return 0;
}

void ClListcreation(int n)
{
    int i, num;
    struct node *preptr, *newnode;

    if(n >= 1)
    {
        stnode = (struct node *)malloc(sizeof(struct node));

        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        stnode->num = num;
        stnode->nextptr = NULL;
        preptr = stnode;
        for(i=2; i<=n; i++)
        {
            newnode = (struct node *)malloc(sizeof(struct node));
            printf(" Input data for node %d : ", i);
            scanf("%d", &num);
            newnode->num = num;
            newnode->nextptr = NULL;   // next address of new node set as NULL
            preptr->nextptr = newnode; // previous node is linking with new node
            preptr = newnode;                // previous node is advanced
        }
        preptr->nextptr = stnode;            //last node is linking with first node
    }
}

void displayClList()
{
    struct node *tmp;
    int n = 1;

    if(stnode == NULL)
    {
        printf(" No data found in the List yet.");
    }
    else
    {
        tmp = stnode;
        printf("\n\n Data entered in the list are :\n");

        do {
            printf(" Data %d = %d\n", n, tmp->num);

            tmp = tmp->nextptr;
            n++;
        }while(tmp != stnode);
    }
}
```

Output:



```
Input the number of nodes : 3
Input data for node 1 : 12
Input data for node 2 : 13
Input data for node 3 : 14
Data entered in the list are :
 Data 1 = 12
 Data 2 = 13
 Data 3 = 14
```

Q7. Write a program in C to insert a node at the end of a circular linked list.

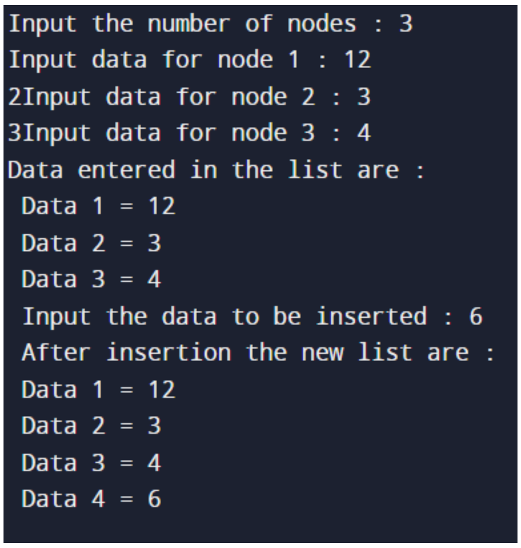
Source Code:

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

struct node {
   int num;
   struct node * nextptr;
}*stnode;

struct node *tail,*p,*q,*store;

void ClListcreation(int n);
void ClLinsertNodeAtEnd(int num);
void displayClList(int a);

int main()
{
   int n,num1,a,insPlc;
   stnode = NULL;
   printf(" Input the number of nodes : ");
   scanf("%d", &n);
   ClListcreation(n);
   a=1;
   displayClList(a);
   printf(" Input the data to be inserted : ");
   scanf("%d", &num1);
   ClLinsertNodeAtEnd(num1);
   a=2;
   displayClList(a);
   return 0;
}

void ClListcreation(int n)
{
   int i, num;
   struct node *preptr, *newnode;
   if(n >= 1)
   {
      stnode = (struct node *)malloc(sizeof(struct node));
      printf(" Input data for node 1 : ");
      scanf("%d", &num);
      stnode->num = num;
```

```
        stnode->nextptr = NULL;
        preptr = stnode;
        for(i=2; i<=n; i++)
        {
            newnode = (struct node *)malloc(sizeof(struct node));
            printf(" Input data for node %d : ", i);
            scanf("%d", &num);
            newnode->num = num;
            newnode->nextptr = NULL;   // next address of new node set as NULL
            preptr->nextptr = newnode; // previous node is linking with new node
            preptr = newnode;                      // previous node is advanced
        }
        preptr->nextptr = stnode;                  //last node is linking with first node
    }
}

void ClLinsertNodeAtEnd(int num1)
{
                        int a;
                        a=num1;
                        struct node *temp=(struct node*)malloc(sizeof(struct node));
                        temp->num=a;
                        p=stnode;
                        while(p->nextptr!=stnode)
                        {
                                    p=p->nextptr;
                        }
                        p->nextptr=temp;
                        temp->nextptr=stnode;
}

void displayClList(int m)
{
    struct node *tmp;
    int n = 1;
    if(stnode == NULL)
    {
        printf(" No data found in the List yet.");
    }
    else
    {
        tmp = stnode;
        if (m==1)
        {
        printf("\n Data entered in the list are :\n");
        }
        else
        {
         printf("\n After insertion the new list are :\n");
        }
        do {
            printf(" Data %d = %d\n", n, tmp->num);

            tmp = tmp->nextptr;
            n++;
        }while(tmp != stnode);
    }
```

}

Output:



```
Input the number of nodes : 3
Input data for node 1 : 12
2Input data for node 2 : 3
3Input data for node 3 : 4
Data entered in the list are :
 Data 1 = 12
 Data 2 = 3
 Data 3 = 4
 Input the data to be inserted : 6
 After insertion the new list are :
 Data 1 = 12
 Data 2 = 3
 Data 3 = 4
 Data 4 = 6
```

LAB 10: - Write Stack and Queue programs

Q1. Write menu driven program to implement various operations of Stack.

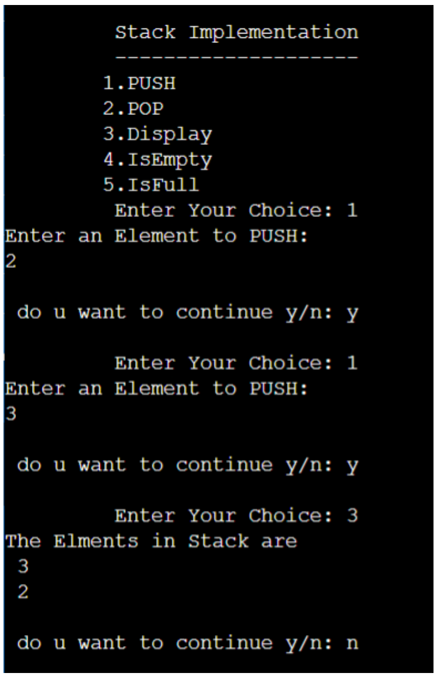
Source Code:

```
#include<stdio.h>
#include<conio.h>
#define max 20
int stack[max];
int top=0,x;
void push(int);
int pop();
void display();
int isempty();
int isfull();
void main()
{
   int ch,item,d;
   char a;
   printf("\n\t Stack Implementation");
   printf("\n\t --------------------");
   printf("\n\t1.PUSH");
   printf("\n\t2.POP");
   printf("\n\t3.Display");
   printf("\n\t4.IsEmpty");
   printf("\n\t5.IsFull");
   do{
   printf("\n\t Enter Your Choice: ");
   scanf("%d",&ch);
   switch(ch)
   {
```

```
   case 1:printf("Enter an Element to PUSH: \n");
          scanf("%d",&item);
          push(item);
          break;
   case 2:x=pop();
          printf("The element POP out from Stack is %d",x);
          break;
   case 3:display();
          break;
   case 4:x=isempty();
         if(x==1)
         printf("Stack is Empty");
         else
         printf("Stack is Not Empty");
         break;
   case 5:x=isfull();
         if(x==1)
         printf("Stack is Full");
         else
         printf("Stack is Not Full");
         break;
   default:printf("INVALID Choice\n");
   }
   printf("\n do u want to continue y/n: ");
   scanf(" %c",&a);
   }while((a=='y')||(a=='Y'));
   getch();
}

   void push(int x)          // PUSH function
   {
      if(top>=max)
      printf("Stack is OVERTFLOW\n");
      else
      {
      stack[top]=x;
      top++;
      }
   }

   int pop()          // POP function
   {
      if(top<=0)
      printf("Stack is UNDERFLOW\n");
      else
      {
      top--;
      x=stack[top];
      }
      return x;
   }

   void display()          // Display function
   {
      int i;
      i=top-1;
      if(top<=0)
```

```
    printf("Stack is Empty");
    else
    printf("The Elments in Stack are \n");
    while(i>=0)
    {
    printf(" %d\n",stack[i--]);
    }
  }

  int isempty()        // isempty function
  {
    if(top<=0)
    return 1;
    else
    return 0;
  }

  int isfull()        // isfull function
  {
    if(top>=max)
    return 1;
    else
    return 0;
  }
```

Output:



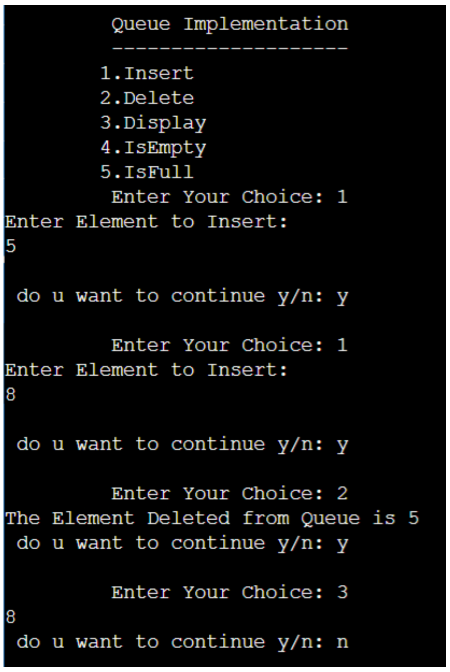
```
         Stack Implementation
         --------------------
        1.PUSH
        2.POP
        3.Display
        4.IsEmpty
        5.IsFull
         Enter Your Choice: 1
Enter an Element to PUSH:
2

 do u want to continue y/n: y

         Enter Your Choice: 1
Enter an Element to PUSH:
3

 do u want to continue y/n: y

         Enter Your Choice: 3
The Elments in Stack are
 3
 2

 do u want to continue y/n: n
```

Q2. write menu driven program to implement various operations of Queue.

Source Code:

```
#include<stdio.h>
#include<conio.h>
#define max 20
int q[max];
int front=-1,rear=-1;
void ins(int);
int del();
void display();
int isempty();
int isfull();
void main()
{
   int ch,item,x;
   char a;
   printf("\n\t Queue Implementation");
   printf("\n\t --------------------");
   printf("\n\t1.Insert");
   printf("\n\t2.Delete");
   printf("\n\t3.Display");
   printf("\n\t4.IsEmpty");
   printf("\n\t5.IsFull");
   do{
   printf("\n\t Enter Your Choice: ");
   scanf("%d",&ch);
   switch(ch)
   {
   case 1:printf("Enter Element to Insert:\n");
         scanf("%d",&item);
         ins(item);
         break;
   case 2:x=del();
         printf("The Element Deleted from Queue is %d",x);
         break;
   case 3:display();
         break;
   case 4:x=isempty();
         if(x==1)
         printf("Queue is Empty");
         else
         printf("Queue is Not Empty");
         break;
   case 5:x=isfull();
         if(x==1)
         printf("Queue is Full");
         else
         printf("Queue is Not Full");
         break;
   default:printf("INVALID Choice\n");
   }
   printf("\n do u want to continue y/n: ");
   scanf(" %c",&a);
   }while((a=='y')||(a=='Y'));
   getch();
}

void ins(int x)          // Insert function
{
   if(rear==max-1)
   printf("Queue is OVERFLOW \n");
   else if(rear==-1)
    {
      front=0;rear=0;
      q[rear]=x;
     }
   else
     {
   rear++;
   q[rear]=x;
     }
}

int del()        // Delete function
{
   int x;
   if(front==-1)
   printf("Queue is UNDERFLOW \n");
   else if(front==rear)
    {
    x=q[front];
    front=-1;
    rear=-1;
    return(x);
    }
    else
    {
    x=q[front];
    front++;
    return(x);
    }
 }

void display()        // Display function
{
   int i;
   if(rear==-1)
   printf("\n Queue is Empty");
   else
   {
   for(i=front;i<=rear;i++)
   printf("%d\t",q[i]);
   }
}

int isempty()        // isempty function
{
   if(front==-1)
   return 1;
   else
   return 0;
}

int isfull()        // isfull function
{
   if(rear==max-1)
   return 1;
   else
   return 0;
}
```

Output:



```
         Queue Implementation
         --------------------
        1.Insert
        2.Delete
        3.Display
        4.IsEmpty
        5.IsFull
         Enter Your Choice: 1
Enter Element to Insert:
5

 do u want to continue y/n: y

         Enter Your Choice: 1
Enter Element to Insert:
8

 do u want to continue y/n: y

         Enter Your Choice: 2
The Element Deleted from Queue is 5
 do u want to continue y/n: y

         Enter Your Choice: 3
8
 do u want to continue y/n: n
```

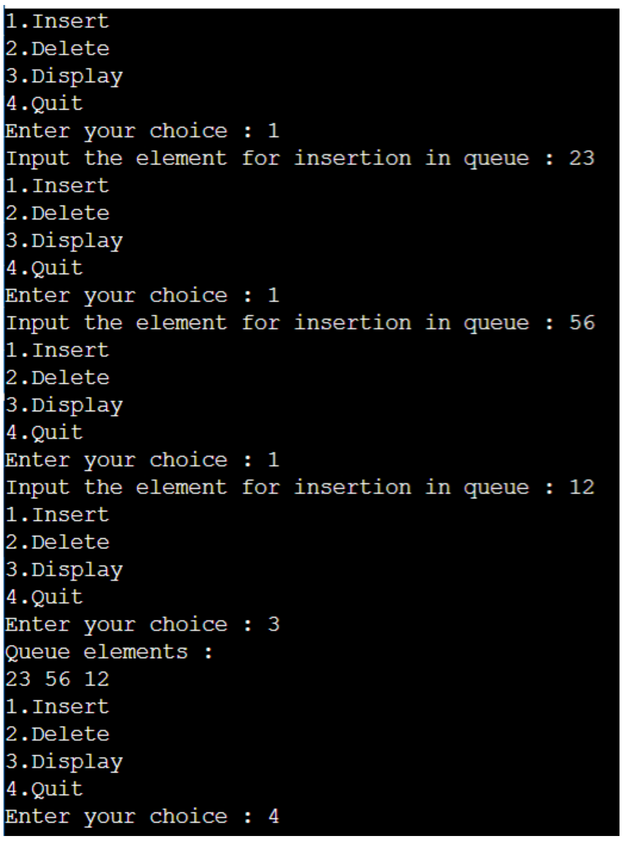
Q3. write menu driven program to implement various operations of Circular Queue.

Source Code:

```
#include<stdlib.h>
#include<conio.h>
# include<stdio.h>
# define MAX 5
int cqueue_arr[MAX];
int front = -1;
int rear = -1;
void insert(int item)
{
  if((front == 0 && rear == MAX-1) || (front == rear+1))
  {
    printf("Queue Overflow \n");
    return;
  }
  if (front == -1)  /*If queue is empty */
  {
    front = 0;
    rear = 0;
  }
  else
  {
    if(rear == MAX-1) /*rear is at last position of queue */
    rear = 0;
    else
    rear = rear+1;
  }
  cqueue_arr[rear] = item ;
}
```

```
void del()
{
  if (front == -1)
  {
    printf("Queue Underflow\n");
    return ;
  }
  printf("Element deleted from queue is : %d\n",cqueue_arr[front]);
  if(front == rear) /* queue has only one element */
  {
    front = -1;
    rear=-1;
  }
  else
  {
    if(front == MAX-1)
    front = 0;
    else
    front = front+1;
  }
}
void display()
{
  int front_pos = front,rear_pos = rear;
  if(front == -1)
  {
    printf("Queue is empty\n");
    return;
  }
  printf("Queue elements :\n");
  if( front_pos <= rear_pos )
  while(front_pos <= rear_pos)
  {
    printf("%d ",cqueue_arr[front_pos]);
    front_pos++;
  }
  else
  {
    while(front_pos <= MAX-1)
  {
    printf("%d ",cqueue_arr[front_pos]);
    front_pos++;
  }
  front_pos = 0;
  while(front_pos <= rear_pos)
  {
    printf("%d ",cqueue_arr[front_pos]);
    front_pos++;
    }
  }
  printf("\n");
}
int main()
{
int choice,item;
do
{
  printf("1.Insert\n");
  printf("2.Delete\n");
  printf("3.Display\n");
  printf("4.Quit\n");
  printf("Enter your choice : ");
  scanf("%d",&choice);
  switch(choice)
  {
  case 1 :
  printf("Input the element for insertion in queue : ");
  scanf("%d", &item);
  insert(item);
  break;
  case 2 :
  del();
  break;
  case 3:
  display();
  break;
  case 4:
  break;
  default:
  printf("Wrong choice\n");
  }
}
while(choice!=4);
  return 0;
}
```

Output:



```
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 23
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 56
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 12
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 3
Queue elements :
23 56 12
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 4
```

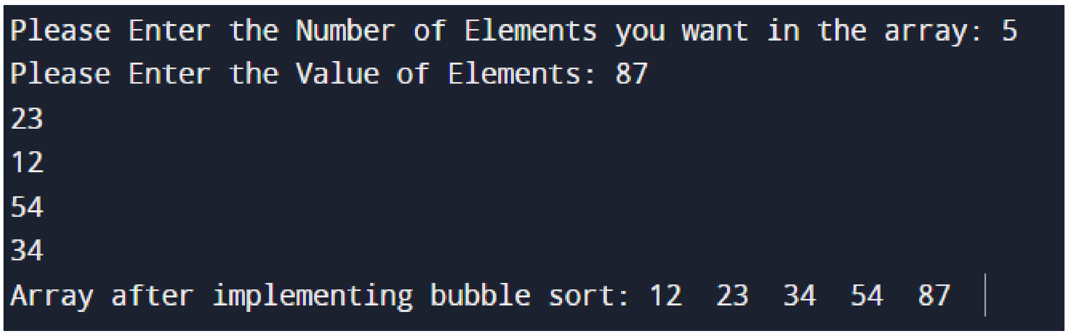
LAB 11: - Write Programs On Searching And Sorting Algorithm

Q1. Write program to implement Bubble Sort.

Source Code:

```
#include <stdio.h>

int main(){
    int arr[50], num, x, y, temp;
    printf("Please Enter the Number of Elements you want in the array: ");
    scanf("%d", &num);
    printf("Please Enter the Value of Elements: ");
    for(x = 0; x < num; x++)
        scanf("%d", &arr[x]);
    for(x = 0; x < num - 1; x++){
        for(y = 0; y < num - x - 1; y++){
            if(arr[y] > arr[y + 1]){
                temp = arr[y];
                arr[y] = arr[y + 1];
                arr[y + 1] = temp;
            }
        }
    }
    printf("Array after implementing bubble sort: ");
    for(x = 0; x < num; x++){
        printf("%d  ", arr[x]);
    }
    return 0;
}
```

Output:



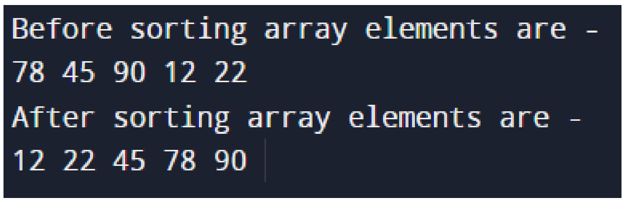
```
Please Enter the Number of Elements you want in the array: 5
Please Enter the Value of Elements: 87
23
12
54
34
Array after implementing bubble sort: 12  23  34  54  87
```

Q2. write a program to implement selection sort.

Source Code:

```
#include <stdio.h>
void selection(int arr[], int n)
{
    int i, j, small;
    for (i = 0; i < n-1; i++)
    {
        small = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[small])
            small = j;
    int temp = arr[small];
    arr[small] = arr[i];
    arr[i] = temp;
    }
}
void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}
int main()
{
    int a[] = { 78,45,90,12,22 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    selection(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);
    return 0;
}
```

Output:



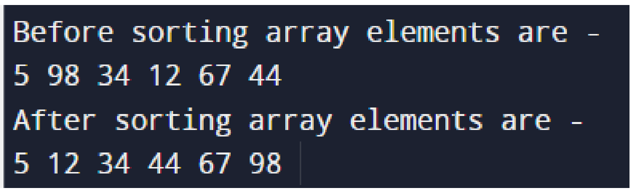
```
Before sorting array elements are -
78 45 90 12 22
After sorting array elements are -
12 22 45 78 90
```

Q3. Write program to implement Insertion Sort.

Source Code:

```
#include <stdio.h>

void insert(int a[], int n)
{
    int i, j, temp;
    for (i = 1; i < n; i++) {
        temp = a[i];
        j = i - 1;

        while(j>=0 && temp <= a[j])
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = temp;
    }
}

void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}

int main()
{
    int a[] = { 5,98,34,12,67,44 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    insert(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);

    return 0;
}
```

Output:



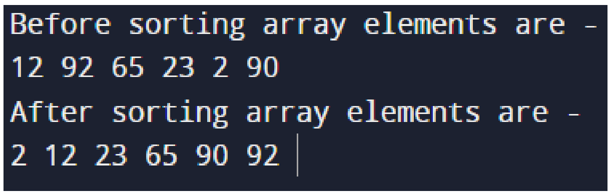
```
Before sorting array elements are -
5 98 34 12 67 44
After sorting array elements are -
5 12 34 44 67 98
```

Q4. Write program to implement Quick Sort.

Source Code:

```
#include <stdio.h>

int partition (int a[], int start, int end)
{
    int pivot = a[end];
    int i = (start - 1);

    for (int j = start; j <= end - 1; j++)
    {

        if (a[j] < pivot)
        {
            i++;
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    int t = a[i+1];
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}

void quick(int a[], int start, int end)
{
    if (start < end)
    {
        int p = partition(a, start, end);
        quick(a, start, p - 1);
        quick(a, p + 1, end);
    }
}

void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}
int main()
{
    int a[] = { 12,92,65,23,2,90 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    quick(a, 0, n - 1);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);

    return 0;
}
```

Output:



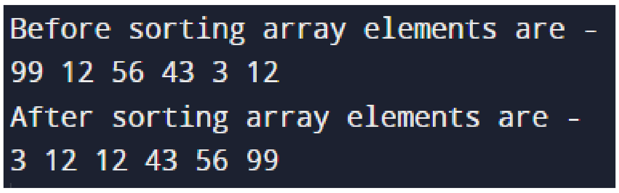
```
Before sorting array elements are -
12 92 65 23 2 90
After sorting array elements are -
2 12 23 65 90 92
```

Q5. Write program to implement Merge Sort.

Source Code:

```
#include <stdio.h>
void merge(int a[], int beg, int mid, int end)
{
   int i, j, k;
   int n1 = mid - beg + 1;
   int n2 = end - mid;
   int LeftArray[n1], RightArray[n2];
   for (int i = 0; i < n1; i++)
   LeftArray[i] = a[beg + i];
   for (int j = 0; j < n2; j++)
   RightArray[j] = a[mid + 1 + j];
   i = 0;
   j = 0;
   k = beg;
   while (i < n1 && j < n2)
   {
      if(LeftArray[i] <= RightArray[j])
      {
         a[k] = LeftArray[i];
         i++;
      }
      else
      {
         a[k] = RightArray[j];
         j++;
      }
      k++;
   }
   while (i<n1)
   {
      a[k] = LeftArray[i];
      i++;
      k++;
   }
   while (j<n2)
   {
      a[k] = RightArray[j];
      j++;
      k++;
   }
}
void mergeSort(int a[], int beg, int end)
{
  if (beg < end)
  {
    int mid = (beg + end) / 2;
    mergeSort(a, beg, mid);
    mergeSort(a, mid + 1, end);
    merge(a, beg, mid, end);
  }
}
void printArray(int a[], int n)
{
  int i;
  for (i = 0; i < n; i++)
    printf("%d ", a[i]);
  printf("\n");
}
int main()
{
  int a[] = { 99,12,56,43,3,12 };
  int n = sizeof(a) / sizeof(a[0]);
  printf("Before sorting array elements are - \n");
  printArray(a, n);
  mergeSort(a, 0, n - 1);
  printf("After sorting array elements are - \n");
  printArray(a, n);
  return 0;
}
```

Output:



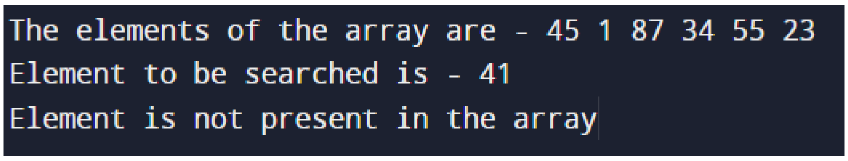
```
Before sorting array elements are -
99 12 56 43 3 12
After sorting array elements are -
3 12 12 43 56 99
```

Q6. Write program to implement Linear Search.

Source Code:

```
#include <stdio.h>
int linearSearch(int a[], int n, int val) {
 for (int i = 0; i < n; i++)
  {
    if (a[i] == val)
    return i+1;
  }
 return -1;
}
int main() {
 int a[] = {45,1,87,34,55,23};
 int val = 41;
 int n = sizeof(a) / sizeof(a[0]);
 int res = linearSearch(a, n, val);
 printf("The elements of the array are - ");
 for (int i = 0; i < n; i++)
 printf("%d ", a[i]);
 printf("\nElement to be searched is - %d", val);
 if (res == -1)
 printf("\nElement is not present in the array");
 else
 printf("\nElement is present at %d position of array", res);
 return 0;
}
```

Output:



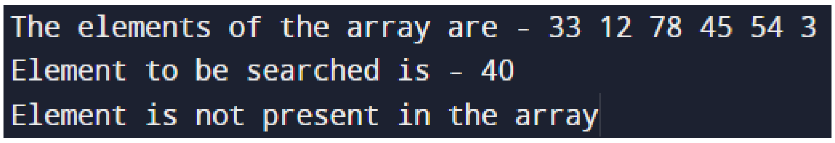
```
The elements of the array are - 45 1 87 34 55 23
Element to be searched is - 41
Element is not present in the array
```

Q7. Write program to implement Binary Search.

Source Code:

```
#include <stdio.h>
int binarySearch(int a[], int beg, int end, int val)
{
   int mid;
   if(end >= beg)
   {       mid = (beg + end)/2;

      if(a[mid] == val)
      {
         return mid+1;
      }
      else if(a[mid] < val)
      {
         return binarySearch(a, mid+1, end, val);
      }
      else
      {
         return binarySearch(a, beg, mid-1, val);
      }
   }
   return -1;
}
int main() {
 int a[] = {33,12,78,45,54,3};
 int val = 40;
 int n = sizeof(a) / sizeof(a[0]);
 int res = binarySearch(a, 0, n-1, val);
 printf("The elements of the array are - ");
 for (int i = 0; i < n; i++)
 printf("%d ", a[i]);
 printf("\nElement to be searched is - %d", val);
 if (res == -1)
 printf("\nElement is not present in the array");
 else
 printf("\nElement is present at %d position of array", res);
 return 0;
}
```

Output:



```
The elements of the array are - 33 12 78 45 54 3
Element to be searched is - 40
Element is not present in the array
```

LAB 12: - Write programs on tree and graph

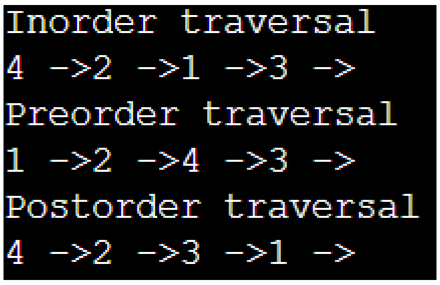
Q1. Write program to implement Binary Search Tree and its Traversals (Preorder, Inorder, Postorder)

CODE:

```
// Tree traversal in C
#include <stdio.h>
#include <stdlib.h>
struct node {
 int item;
 struct node* left;
 struct node* right;
};
// Inorder traversal
void inorderTraversal(struct node* root) {
 if (root == NULL) return;
 inorderTraversal(root->left);
 printf("%d ->", root->item);
 inorderTraversal(root->right);
}
// Preorder traversal
void preorderTraversal(struct node* root) {
 if (root == NULL) return;
 printf("%d ->", root->item);
 preorderTraversal(root->left);
 preorderTraversal(root->right);
}
// Postorder traversal
void postorderTraversal(struct node* root) {
 if (root == NULL) return;
 postorderTraversal(root->left);
```

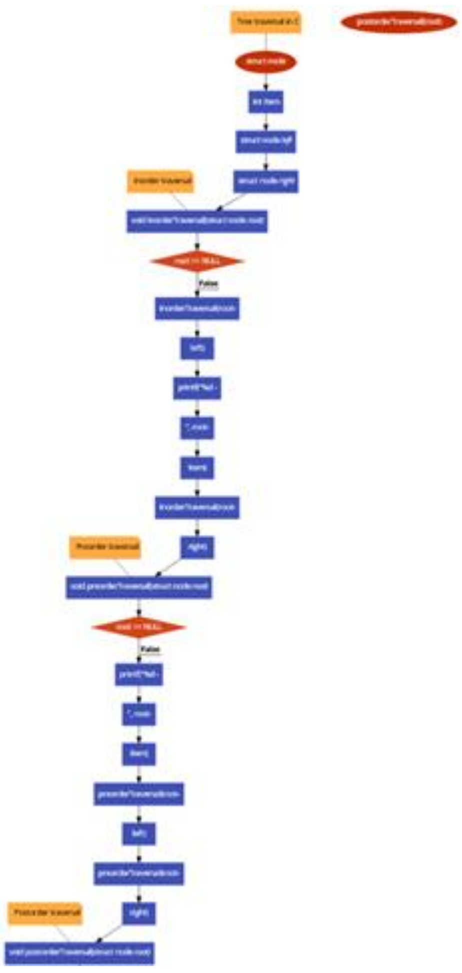
```
  postorderTraversal(root->right);
  printf("%d ->", root->item);
}
// Create a new Node
struct node* createNode(value) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->item = value;
  newNode->left = NULL;
  newNode->right = NULL;
  return newNode;
}
// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
  root->left = createNode(value);
  return root->left;
}
// Insert on the right of the node
struct node* insertRight(struct node* root, int value) {
  root->right = createNode(value);
  return root->right;
}
int main() {
  struct node* root = createNode(1);
  insertLeft(root, 2);
  insertRight(root, 3);
  insertLeft(root->left, 4);
  printf("Inorder traversal \n");
  inorderTraversal(root);
  printf("\nPreorder traversal \n");
  preorderTraversal(root);
  printf("\nPostorder traversal \n");
  postorderTraversal(root);
}
```

OUTPUT:



```
Inorder traversal
4 ->2 ->1 ->3 ->
Preorder traversal
1 ->2 ->4 ->3 ->
Postorder traversal
4 ->2 ->3 ->1 ->
```

FLOWCHART:



Q2. Program to implement Shortest Path Algorithm: Dijkstra's Algorithm

CODE:

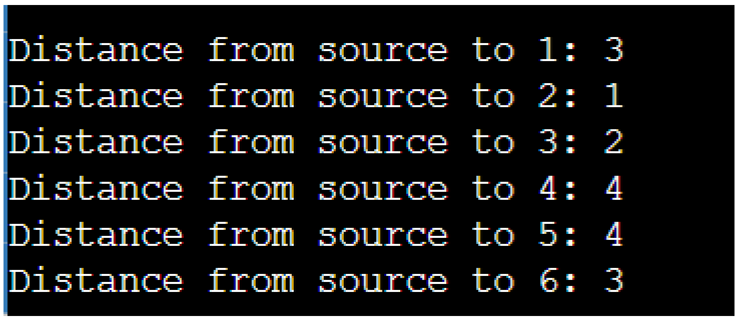
```
// Dijkstra's Algorithm in C

#include <stdio.h>

#define INFINITY 9999

#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {

  int cost[MAX][MAX], distance[MAX], pred[MAX];

  int visited[MAX], count, mindistance, nextnode, i, j;

  // Creating cost matrix

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

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

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

        cost[i][j] = INFINITY;
```

```
   else
    cost[i][j] = Graph[i][j];
 for (i = 0; i < n; i++) {
  distance[i] = cost[start][i];
  pred[i] = start;
  visited[i] = 0;
 }
 distance[start] = 0;
 visited[start] = 1;
 count = 1;
 while (count < n - 1) {
  mindistance = INFINITY;
  for (i = 0; i < n; i++)
   if (distance[i] < mindistance && !visited[i]) {
    mindistance = distance[i];
    nextnode = i;
   }
  visited[nextnode] = 1;
  for (i = 0; i < n; i++)
   if (!visited[i])
    if (mindistance + cost[nextnode][i] < distance[i]) {
     distance[i] = mindistance + cost[nextnode][i];
     pred[i] = nextnode;
    }
  count++;
 }
 // Printing the distance
 for (i = 0; i < n; i++)
  if (i != start) {
   printf("\nDistance from source to %d: %d", i, distance[i]);
  }
}
int main() {
```

```
int Graph[MAX][MAX], i, j, n, u;

n = 7;

Graph[0][0] = 0;

Graph[0][1] = 0;

Graph[0][2] = 1;

Graph[0][3] = 2;

Graph[0][4] = 0;

Graph[0][5] = 0;

Graph[0][6] = 0;


Graph[1][0] = 0;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[1][4] = 0;

Graph[1][5] = 3;

Graph[1][6] = 0;


Graph[2][0] = 1;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 1;

Graph[2][4] = 3;

Graph[2][5] = 0;

Graph[2][6] = 0;


Graph[3][0] = 2;

Graph[3][1] = 0;

Graph[3][2] = 1;

Graph[3][3] = 0;

Graph[3][4] = 0;

Graph[3][5] = 0;

Graph[3][6] = 1;
```

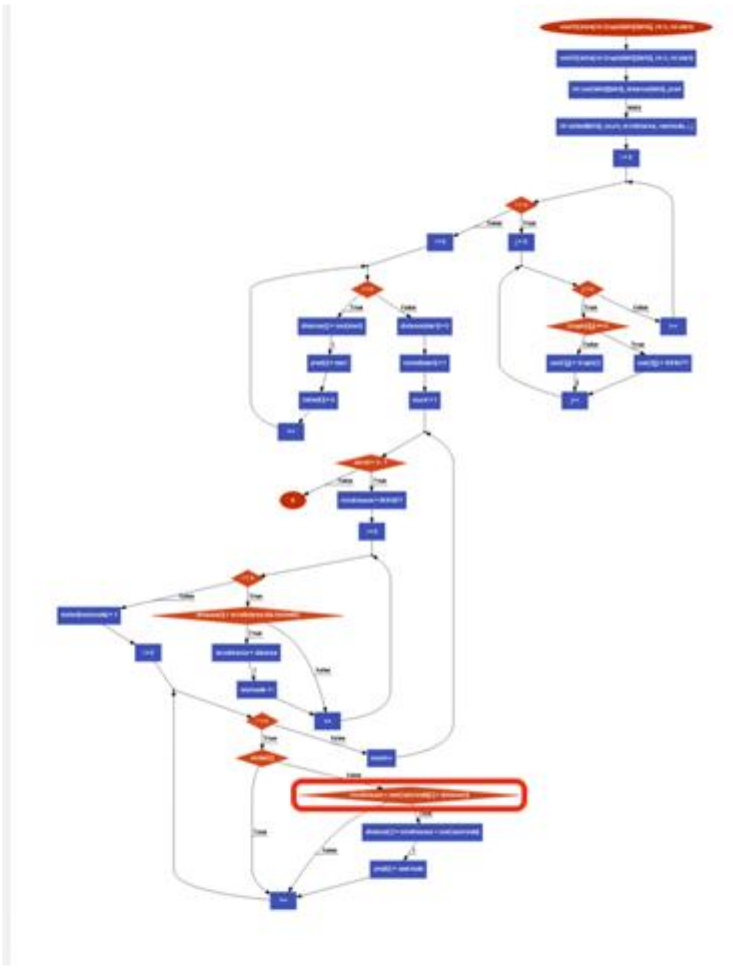
```
  Graph[4][0] = 0;

  Graph[4][1] = 0;

  Graph[4][2] = 3;

  Graph[4][3] = 0;

  Graph[4][4] = 0;

  Graph[4][5] = 2;

  Graph[4][6] = 0;


  Graph[5][0] = 0;

  Graph[5][1] = 3;

  Graph[5][2] = 0;

  Graph[5][3] = 0;

  Graph[5][4] = 2;

  Graph[5][5] = 0;

  Graph[5][6] = 1;


  Graph[6][0] = 0;

  Graph[6][1] = 0;

  Graph[6][2] = 0;

  Graph[6][3] = 1;

  Graph[6][4] = 0;

  Graph[6][5] = 1;

  Graph[6][6] = 0;

  u = 0;

  Dijkstra(Graph, n, u);

  return 0;

}
```

OUTPUT:



```
Distance from source to 1: 3
Distance from source to 2: 1
Distance from source to 3: 2
Distance from source to 4: 4
Distance from source to 5: 4
Distance from source to 6: 3
```

FLOWCHART:



Q3. Program to implement Minimum Spanning Trees: Kruskal's and Prim's Algorithm

CODE:

Kruskal's

```
// Kruskal's algorithm in C

#include <stdio.h>

#define MAX 30

typedef struct edge {

  int u, v, w;

} edge;
```

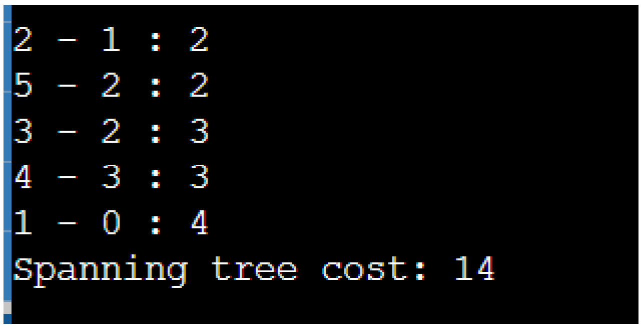
```
typedef struct edge_list {
  edge data[MAX];
  int n;
} edge_list;
edge_list elist;
int Graph[MAX][MAX], n;
edge_list spanlist;
void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();
// Applying Krushkal Algo
void kruskalAlgo() {
  int belongs[MAX], i, j, cno1, cno2;
  elist.n = 0;
  for (i = 1; i < n; i++)
    for (j = 0; j < i; j++) {
      if (Graph[i][j] != 0) {
        elist.data[elist.n].u = i;
        elist.data[elist.n].v = j;
        elist.data[elist.n].w = Graph[i][j];
        elist.n++;
      }
    }
  sort();
  for (i = 0; i < n; i++)
    belongs[i] = i;
  spanlist.n = 0;
  for (i = 0; i < elist.n; i++) {
    cno1 = find(belongs, elist.data[i].u);
    cno2 = find(belongs, elist.data[i].v);
    if (cno1 != cno2) {
      spanlist.data[spanlist.n] = elist.data[i];
      spanlist.n = spanlist.n + 1;
      applyUnion(belongs, cno1, cno2);
    }
  }
}
int find(int belongs[], int vertexno) {
  return (belongs[vertexno]);
}
void applyUnion(int belongs[], int c1, int c2) {
  int i;
  for (i = 0; i < n; i++)
    if (belongs[i] == c2)
      belongs[i] = c1;
}
// Sorting algo
void sort() {
  int i, j;
  edge temp;
  for (i = 1; i < elist.n; i++)
    for (j = 0; j < elist.n - 1; j++)
      if (elist.data[j].w > elist.data[j + 1].w) {
        temp = elist.data[j];
        elist.data[j] = elist.data[j + 1];
        elist.data[j + 1] = temp;
      }
}
// Printing the result
void print() {
  int i, cost = 0;
  for (i = 0; i < spanlist.n; i++) {
    printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
    cost = cost + spanlist.data[i].w;
  }
  printf("\nSpanning tree cost: %d", cost);
}
int main() {
  int i, j, total_cost;
  n = 6;
  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

  Graph[1][0] = 4;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 0;
  Graph[1][6] = 0;

  Graph[2][0] = 4;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
  Graph[2][3] = 3;
  Graph[2][4] = 4;
  Graph[2][5] = 0;
  Graph[2][6] = 0;

  Graph[3][0] = 0;
  Graph[3][1] = 0;
  Graph[3][2] = 3;
  Graph[3][3] = 0;
  Graph[3][4] = 3;
  Graph[3][5] = 0;
  Graph[3][6] = 0;

  Graph[4][0] = 0;
  Graph[4][1] = 0;
  Graph[4][2] = 4;
  Graph[4][3] = 3;
  Graph[4][4] = 0;
  Graph[4][5] = 0;
  Graph[4][6] = 0;

  Graph[5][0] = 0;
  Graph[5][1] = 0;
  Graph[5][2] = 2;
  Graph[5][3] = 0;
  Graph[5][4] = 3;
  Graph[5][5] = 0;
  Graph[5][6] = 0;
  kruskalAlgo();
  print();
}
```

Prim's

```
// Prim's Algorithm in C
#include<stdio.h>
#include<stdbool.h>
#define INF 9999999
// number of vertices in graph
#define V 5
// create a 2d array of size 5x5
//for adjacency matrix to represent graph
int G[V][V] = {
 {0, 9, 75, 0, 0},
 {9, 0, 95, 19, 42},
 {75, 95, 0, 51, 66},
 {0, 19, 51, 0, 31},
 {0, 42, 66, 31, 0}};
int main() {
 int no_edge;  // number of edge
 // create a array to track selected vertex
 // selected will become true otherwise false
 int selected[V];
 // set selected false initially
 memset(selected, false, sizeof(selected));
 // set number of edge to 0
 no_edge = 0;
 // the number of egde in minimum spanning tree will be
 // always less than (V -1), where V is number of vertices in
 //graph
 // choose 0th vertex and make it true
 selected[0] = true;
 int x;  //  row number
 int y;  //  col number
 // print for edge and weight
 printf("Edge : Weight\n");
 while (no_edge < V - 1) {
  //For every vertex in the set S, find the all adjacent vertices
  // , calculate the distance from the vertex selected at step 1.
  // if the vertex is already in the set S, discard it otherwise
  //choose another vertex nearest to selected vertex  at step 1.
  int min = INF;
  x = 0;
  y = 0;
  for (int i = 0; i < V; i++) {
   if (selected[i]) {
        for (int j = 0; j < V; j++) {
          if (!selected[j] && G[i][j]) {  // not in selected and there is an edge
            if (min > G[i][j]) {
              min = G[i][j];
              x = i;
              y = j;
            }
          }
        }
      }
    }
    printf("%d - %d : %d\n", x, y, G[x][y]);
    selected[y] = true;
    no_edge++;
  }
  return 0;
}
```

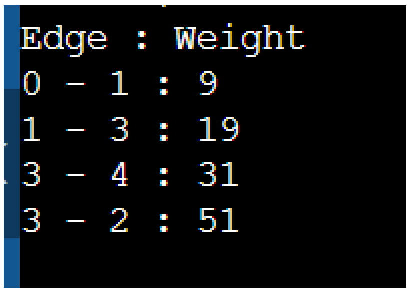
OUTPUT:

Kruskal's



```
2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14
```

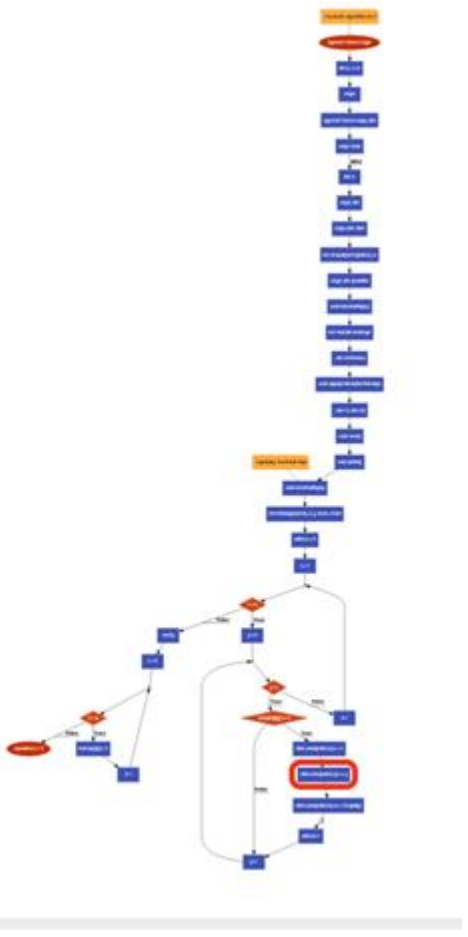
Prim's



```
Edge : Weight
0 - 1 : 9
1 - 3 : 19
3 - 4 : 31
3 - 2 : 51
```

FLOWCHART:

Kruskal’s



Prim’s