# OOPS PRACTICAL FILE

GTBIT

**HARPREET SINGH**

**CSE-3**

**00776802719**

# Index

# Experiment-1

**Aim:-** To find the roots of a given quadratic equation.

## Source Code:-

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
   float a,b,c,discriminant,realpart,imaginarypart,x1,x2;
   cout<<"\nEnter the coefficients of the quadratic equation: ";
   cin>>a>>b>>c;
   discriminant=(b*b)-(4*a*c);
   if(a==0)
   {
      cout<<"\nInvalid inputs\n";
   }
   else if(discriminant>0)
   {
      x1=(-b+sqrt(discriminant))/(2*a);
      x2=(-b-sqrt(discriminant))/(2*a);
      cout<<"\nThe roots are real and distinct: ";
      cout<<"\nx1= "<<x1<<"\nx2= "<<x2;
   }
   else if(discriminant==0)
   {
      x1=-b/(2*a);
      cout<<"\nThe roots are real and equal:";
      cout<<"\n x1=x2= "<<x1;
   }
```

```
    else
    {
    realpart=-b/(2*a);
    imaginarypart=sqrt(-discriminant)/(2*a);
    cout<<"\nThe roots are imaginary and distinct:";
    cout<<"\nx1= "<<realpart<<" + "<<imaginarypart<<"i";
    cout<<"\nx2= "<<realpart<<" - "<<imaginarypart<<"i";
    }
    cout<<"\n";
    return 0;
}
```

## Algorithm:-

**Start**

**Step 1:** Input the value of a, b, c.

**Step 2:** Calculate discriminant=(b*b)-(4*a*c);

**Step 3:** if(a=0)

Output “Invalid Input”.

else If (d > 0)

Output "Roots are real and distinct” and calculate x1=(-b+sqrt(discriminant))/(2*a),

x2=(-b-sqrt(discriminant))/(2*a).

Output x1 and x2.

else if (d = 0)

Display "Roots are real and equal" and calculate x1 = x2= -b/(2*a);

Output x1 and x2.

else

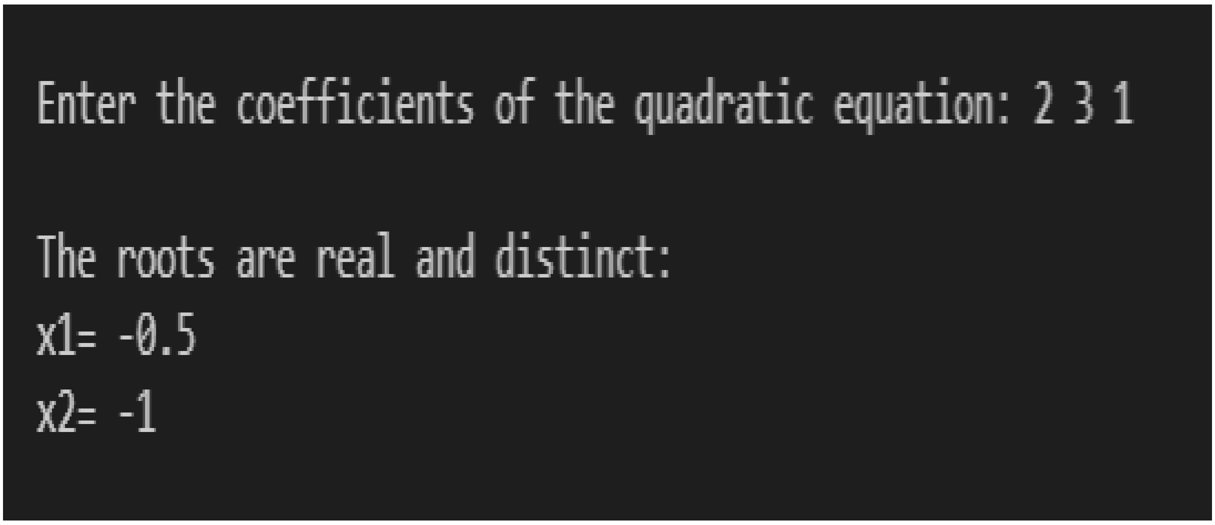
Display "Roots are Imaginary and distinct”, calculate x1=-b/(2*a), x2 = sqrt(-discriminant)/(2*a).

Output x1 and x2.

**Step 4:** End the algorithm.

**Stop**

**Output:-**



```
Enter the coefficients of the quadratic equation: 2 3 1

The roots are real and distinct:
x1= -0.5
x2= -1
```

# Experiment-2

**Aim:-** To find gcd and lcm of 2 nos.

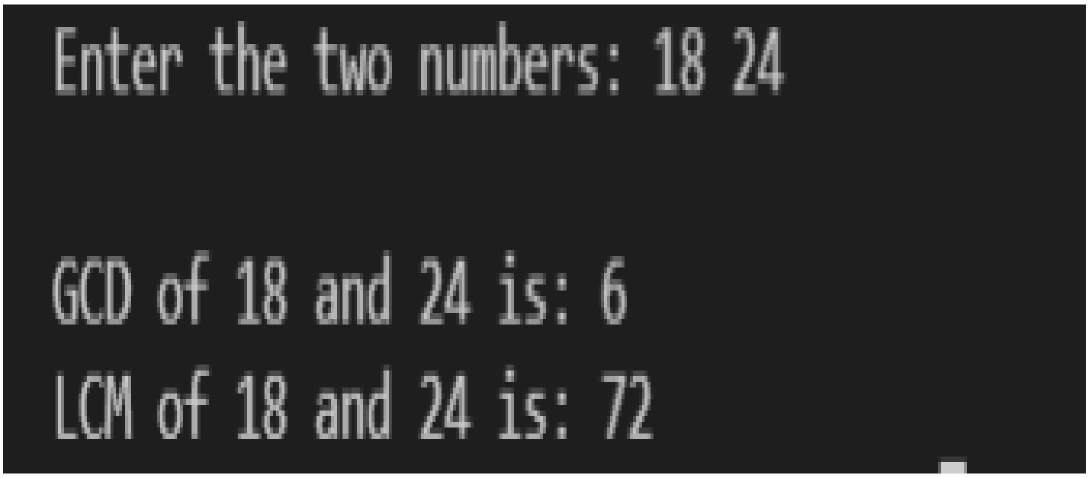
## Source Code:-

```
#include<iostream>
using namespace std;
void printGcdLcm(int num1,int num2)
{
   int gcd,lcm,on1=num1,on2=num2;
   while(num1%num2!=0)
   {
      int rem=num1%num2;
      num1=num2;
      num2=rem;
   }
   gcd=num2;
   lcm=(on1*on2)/gcd;
   cout<<"\nGCD of "<<on1<<" and "<<on2<<" is: "<<gcd;
   cout<<"\nLCM of "<<on1<<" and "<<on2<<" is: "<<lcm;
}
int main()
{
   int num1,num2;
   cout<<"Enter the two numbers: ";
   cin>>num1>>num2;
   printGcdLcm(num1,num2);
   return 0;
}
```

## Algorithm:-

```
Start

Step 1: read num1, num2, on1←num1 , on2←num2

Step 2 while(num1%num2!=0)

    • rem=num1%num2
    • num1=num2
    • num2=rem
End while

step 3: gcd ←num2

                    • Lcm ←(on1 * on2) / gcd

Step 4: print gcd, lcm

Stop
```

## Output:-



```
Enter the two numbers: 18 24


GCD of 18 and 24 is: 6
LCM of 18 and 24 is: 72
```

## Experiment-3

**Aim:-** To multiply two matrices using oop (using classes).

### Source Code:-

```
#include <iostream>
using namespace std;

class Matrix
{
   int a[10][10];
   int b[10][10];
   int c[10][10];
   int i,j,k;
   int m, n, p, q;
public:
   Matrix()
   {
      cout<<"\nEnter the dimensions of matrix 1: ";
      cin>>m>>n;

      cout<<"\nEnter the dimensions of matrix 2: ";
      cin>>p>>q;
   }
   void Mult();
   void InputMatrix();
   void OutputMatrix();
};

void Matrix::InputMatrix()
{
    cout << "\nEnter the values for the first matrix\n";
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            cin >> a[i][j];
        }
    }

    cout << "\nEnter the values for the second matrix\n";
    for (i=0; i<p; i++)
    {
        for (j=0; j<q; j++)
        {
            cin >> b[i][j];
        }
    }
}

void Matrix::Mult()
{
    if(n!=p)
    {
        cout<<"\nThe matrix multiplication is not possible\n";
        return;
    }

    for (i=0; i<m; i++)
    {
        for (j=0; j<q; j++)
        {
            c[i][j]=0;
            for (k=0; k<p; k++)
            {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

void Matrix::OutputMatrix()
{
    cout << "\nThe Resultant Matrix is: \n";
    for (i=0; i<m; i++)
    {
        for (j=0; j<q; j++)
        {
            cout << c[i][j]<<" ";
        }
        cout << endl;
    }
}

int main()
{
    Matrix x;
    x.InputMatrix();
    x.Mult();
```

```
   x.OutputMatrix();
   return 0;
}
```

## Algorithm:-

**Start**

**Step 1:** Start the Program.

**Step 2:** Enter the row and column of the first (a) matrix.

**Step 3:** Enter the row and column of the second (b) matrix.

**Step 4:** Enter the elements of the first (a) matrix.

**Step 5:** Enter the elements of the second (b) matrix.

**Step 6:** Set a loop up to row.

**Step 7:** Set an inner loop up to the column.

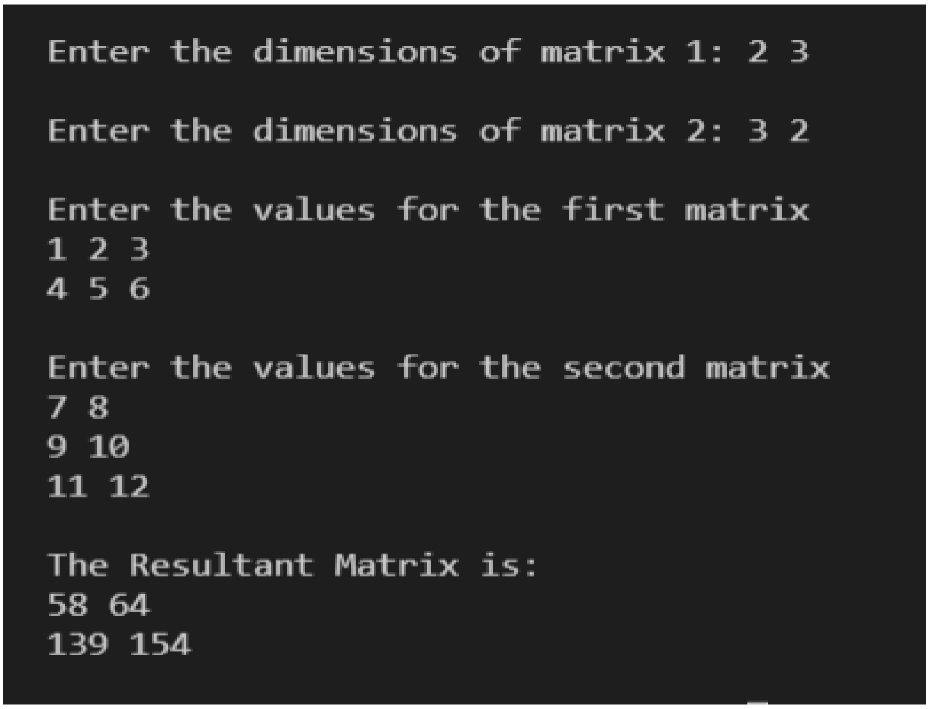
**Step 8:** Set another inner loop up to the column.

**Step 9:** Multiply the first (a) and second (b) matrix and store the element in the third matrix (c)

**Step 10:** Print the final matrix.

**Stop**

**Output:-**



```
Enter the dimensions of matrix 1: 2 3

Enter the dimensions of matrix 2: 3 2

Enter the values for the first matrix
1 2 3
4 5 6

Enter the values for the second matrix
7 8
9 10
11 12

The Resultant Matrix is:
58 64
139 154
```

## Experiment-4

**Aim:-** Write a c++ program to find the greatest of two given numbers in two different classes using friend function.

### Source Code:-

```
#include<iostream>
using namespace std;
class second;
class first
{
int x;
public:
void getx()
{
cout<<"\nEnter the value of x: ";
cin>>x;
}
friend void max(first,second);
};
class second
{
int y;
public:
void gety()
{
cout<<"\nEnter the value of y: ";
cin>>y;
}
friend void max(first,second);
};
```

```
void max(first a,second b)

{

if(a.x>b.y)

{

cout<<"\nGreater value is: "<<a.x;

}

else

{

cout<<"\nGreater value is: "<<b.y;

}

}

int main()

{

first a;

second b;

a.getx();

b.gety();

max(a,b);

return 0;

}
```

## Algorithm:-

**Start**

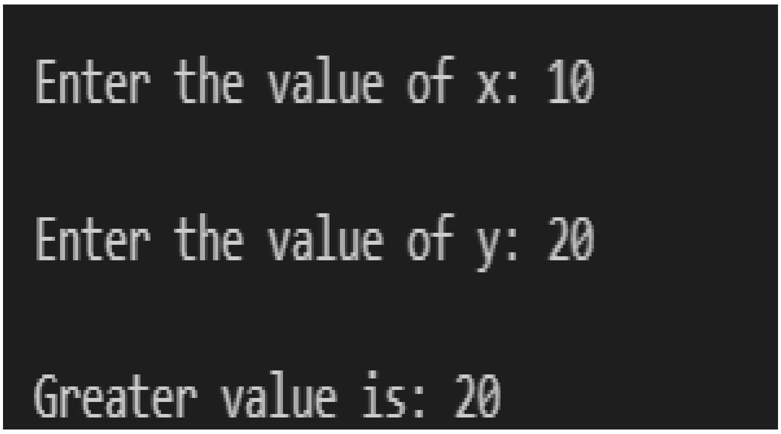
**Step1:** Enter the first number in first class.

**Step2:** Enter the second number in second class.

**Step3:** Compare the both the values in a friend function declared in both classes and print the greatest of two numbers.

**Stop**

**Output:-**



```
Enter the value of x: 10

Enter the value of y: 20

Greater value is: 20
```

## Experiment-5

**Aim:-** Write a c++ program to perform addition of two complex numbers using constructor overloading.

## Source Code:-

```
#include<iostream>
using namespace std;


class Complex
{
        private:
                float real;
                float imag;
        public:
                Complex(){
                        real = 0;
                        imag = 0;
                }
                Complex(float r, float i){
                        real = r;
                        imag = i;
                }
                Complex add(Complex c){
                        return Complex(real + c.real, imag + c.imag);
                }


                void display(){
                        cout<<"\n"<<real<<" + "<<imag<<"i"<<endl;
                }
};
```

```
int main(){
   float r1,i1,r2,i2;
   cout<<"\nEnter the real part of first complex number: ";
   cin>>r1;
   cout<<"\nEnter the imaginary part of first complex number: ";
   cin>>i1;

   cout<<"\nEnter the real part of second complex number: ";
   cin>>r2;
   cout<<"\nEnter the imaginary part of second complex number: ";
   cin>>i2;

          Complex c1(r1, i1), c2(r2, i2), c3;
          c3 = c1.add(c2);
          c1.display();
          c2.display();
          cout<<"\nThe sum is : "<<endl;
          c3.display();
          return 0;
}
```

## Algorithm:-

**Start**

**Step1:** Assign the real and imaginary part of first complex number.

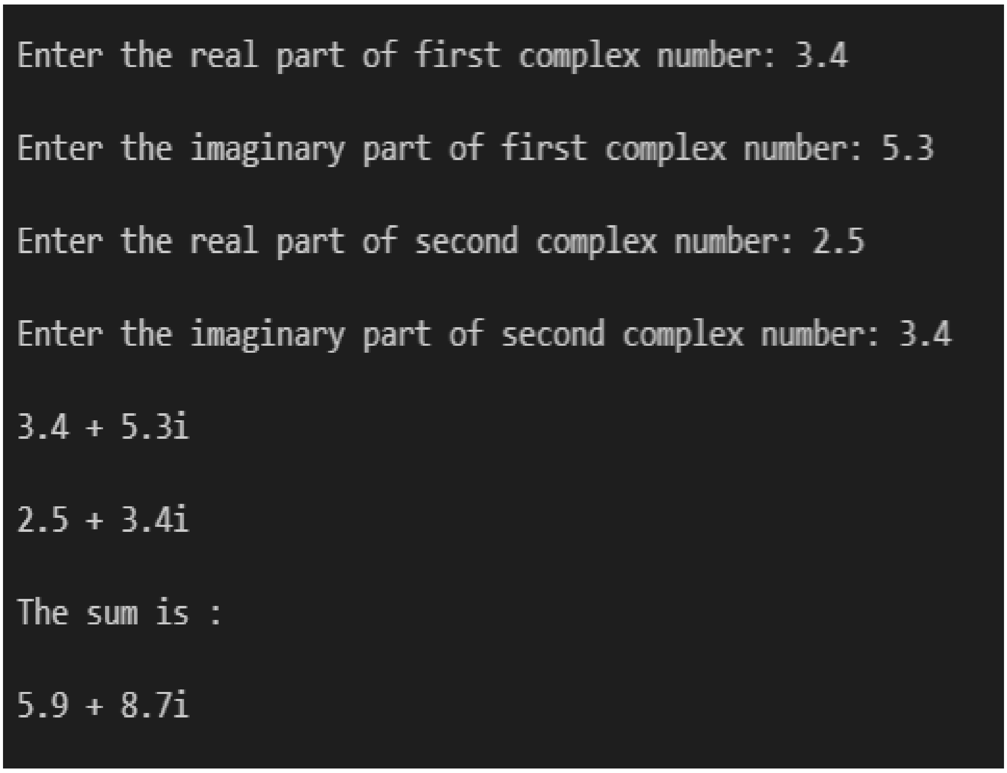
**Step2:** Assign the real and imaginary part of second complex number.

**Step3:** Calculate the sum of the real and imaginary parts of both the complex numbers.

**Step4:** Print the sum of the two complex numbers

**Stop**

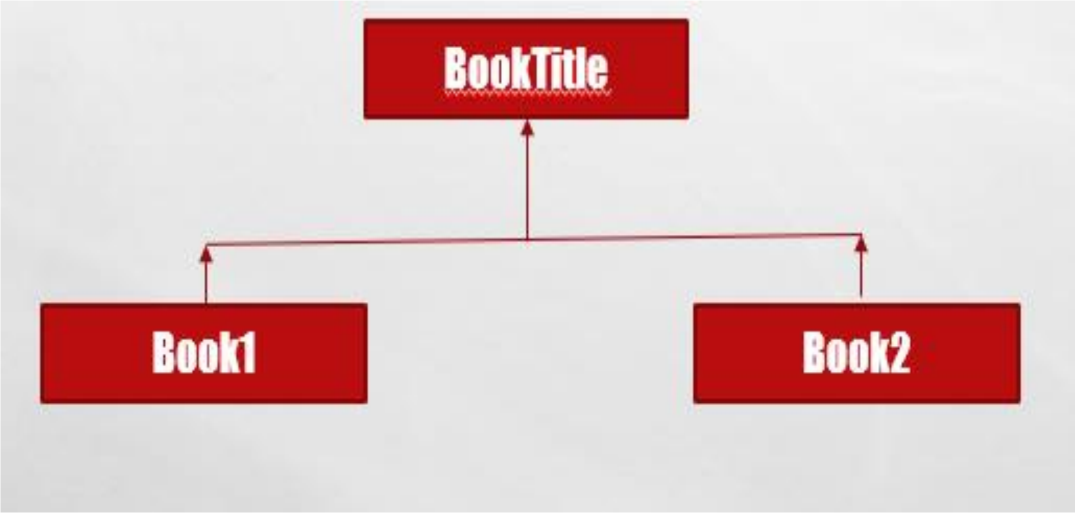
**Output:-**



```
Enter the real part of first complex number: 3.4

Enter the imaginary part of first complex number: 5.3

Enter the real part of second complex number: 2.5

Enter the imaginary part of second complex number: 3.4

3.4 + 5.3i

2.5 + 3.4i

The sum is :

5.9 + 8.7i
```

# Experiment-6

**Aim:-** Consider the following class hierarchy, implement it in C++



## Source Code:-

```
#include <iostream>
using namespace std;
class BookTitle
{
   string title;
public:
   void gettitle()
   {
      cout << "\nEnter title of the book: ";
      getline(cin,title);
   }
   void show()
   {
      cout<<"The name of the book is: "<<title<<"\n";
   }
};
class Book1 : public BookTitle
```

```
{
    int price;
    string author;
    public:
    void getdata()
    {
        cout << "\nEnter price of book1: ";
        cin >> price;
        cin.ignore();
        cout <<"\nEnter the author of the book1: ";
        getline(cin,author);
    }
     void display()
    {
        cout<<"The price of the book1 is: "<<price<<"\n";
        cout<<"The author of the book1 is: "<<author<<"\n";
    }
};

class Book2 : public BookTitle
{
    int price;
    string author;
    public:
    void getdata()
    {
        cout << "\nEnter price of book2: ";
        cin >> price;
        cin.ignore();
        cout <<"\nEnter the author of the book2: ";
        getline(cin,author);
    }
     void display()
    {
        cout<<"The price of the book2 is: "<<price<<"\n";
        cout<<"The author of the book2 is: "<<author<<"\n";
    }
};
int main()
{
    Book1 b1;
    Book2 b2;
    b1.gettitle();
    b1.getdata();

    b2.gettitle();
    b2.getdata();
    cout<<"\n\nDetails:-\n";
    b1.show();
    b1.display();
    cout<<"\n";
    b2.show();
    b2.display();
    return 0;
}
```

## Algorithm:-

**Start**

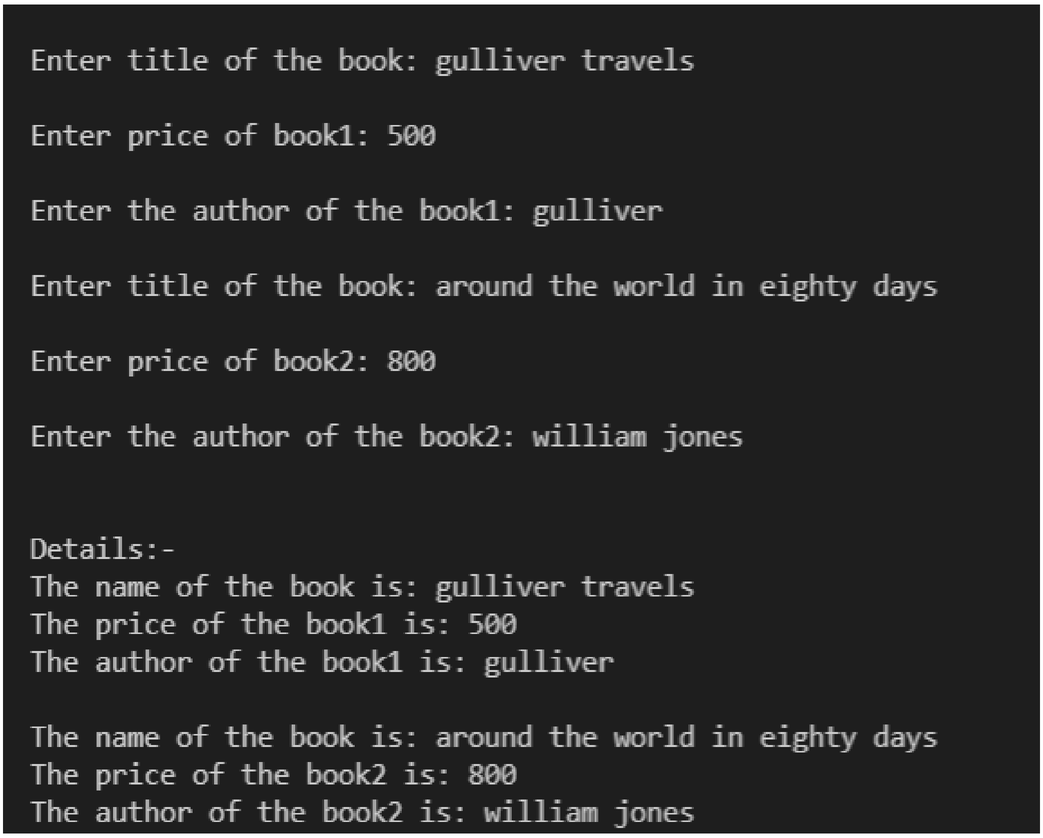
**Step1:** Enter the title of book1 and the other features like price and author.

**Step2:** Enter the title of book2 and the other features like price and author.

**Step3:** The details of the books are shown like the title which is displayed using the parent class and then book's own properties like price and author name are also displayed.

**Stop**

## Output:-



```
Enter title of the book: gulliver travels

Enter price of book1: 500

Enter the author of the book1: gulliver

Enter title of the book: around the world in eighty days

Enter price of book2: 800

Enter the author of the book2: william jones


Details:-
The name of the book is: gulliver travels
The price of the book1 is: 500
The author of the book1 is: gulliver

The name of the book is: around the world in eighty days
The price of the book2 is: 800
The author of the book2 is: william jones
```

# Experiment-7

**Aim:-** Write a c++ program to implement a class string containing the following functions-

- Overload + operator to carry out concatenation of strings.
- Overload = operator to carry out string copy.
- Overload <= operator to carry out string comparisons.
- Function to display the length of string.

## Source Code:-

```
#include <iostream>
#include <cstring>
#include <cctype>
class String
{
 public:
 char s[100];
 String get_string()
 {
 std::cin >> s;
 return *this;
 }
 char* put_string()
 {
 return (char*) s;
 }
 int length()
 {
 return strlen(s);
 }
```

```
String operator = (const String &op)
{
strcpy(s, op.s);
return *this;
}

String operator + (const String &op)
{
String res;
res = *this;
strcat(res.s, op.s);
return res;
}
bool operator <= (String op)
{
if (length() <= op.length())
return true;
else
return false;
}
String tolower()
{
for (int i = 0; i < length(); i++)
{
s[i] = std::tolower(s[i]);
}
return *this;
}
```

```
 String toupper()
 {
 for (int i = 0; i < length(); i++)
 {
 s[i] = std::toupper(s[i]);
 }
 return *this;
 }
};
int main()
{
 String A, B;
 std::cout << "Enter the first string: ";
 A.get_string();
 std::cout << "Enter the second string: ";
 B.get_string();

 String C = A + B;
 std::cout << "\nConcatenated string: " << C.put_string() <<
"\n";

 std::cout << "String A <= String B: " << (A <= B ? "true\n"
: "false\n");
String D;
std::cout<<"\nThe copied string: "<<(D=C).put_string()<<"\n";
 std::cout << "Length of string A: " << A.length() << "\n";
 std::cout << "Length of string B: " << B.length() << "\n";
 return 0;
}
```

## Algorithm:-

**Start**

**Step1:** Declare a class with a string variable and an operator function '+' that accepts an instance of the class and concatenates it's variable with the string variable of the current instance and also a '=' operator function for copied string that accepts the instance of the class and copies instance's string into the variable and a '<=' operator function for one string length less than equal to another one that accepts an instance of the class and compare variable's length with the instance's string.

**Step2:** Create two instances of the class and initialize their class variables with the two input strings respectively.

**Step3:** Now, use the overloaded operator(+) function to concatenate the class variable of the two instances.

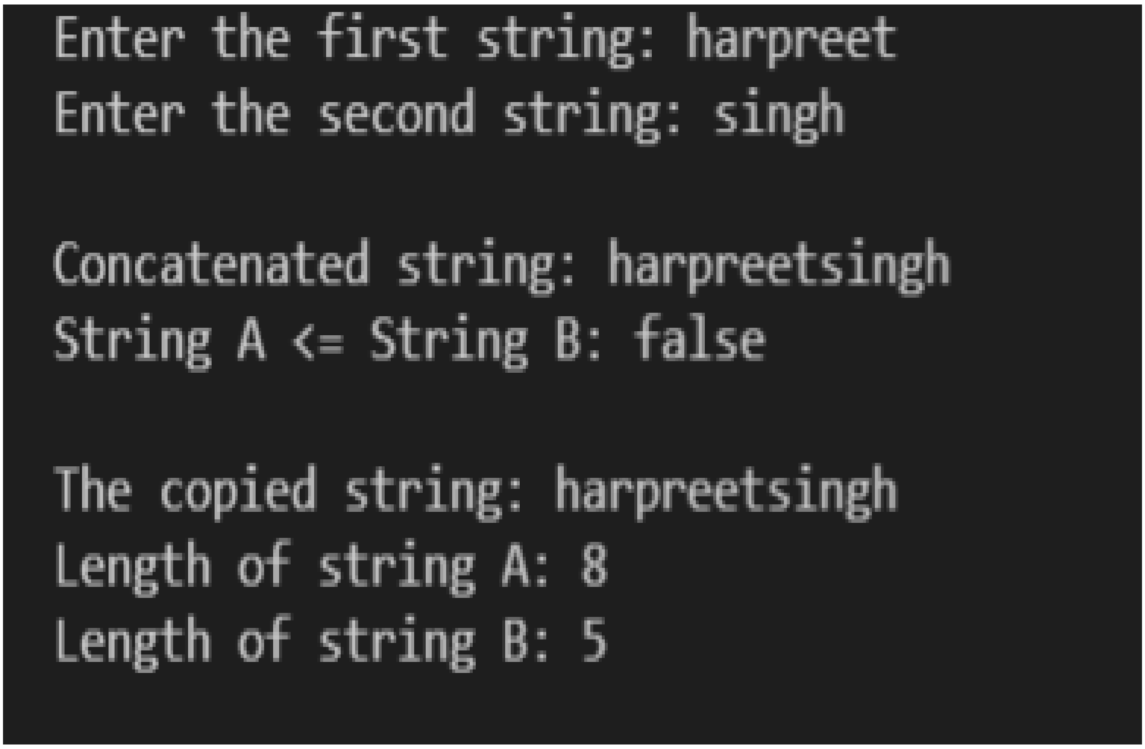
**Step4**: Use overloaded operator(=) function to copy the class variable of one instance to other.

**Step5:** Use overloaded operator(<=) function to check whether the length of class variable of one instance is less than or equal to the other.

**Step6:** Display the length of the class variable of one instance.

**Stop**

## Output:-



```
Enter the first string: harpreet
Enter the second string: singh

Concatenated string: harpreetsingh
String A <= String B: false

The copied string: harpreetsingh
Length of string A: 8
Length of string B: 5
```

## Experiment-8

**Aim:-** Create a class called list with two pure virtual functions store( ) and retrieve( ), To store a value call store and to retrieve. Derive two classes stack and queue from it and override store and retrieve.

**Source Code:-**

```
#include<iostream>

#include<stdlib.h>

#include<conio.h>

using namespace std;


struct node

{

   int data;

   node *next;

};


node *head=NULL,*tail=NULL;


class List

{

   public:

   void view()

   {

      node *n = head;

      if(head==NULL)

      {

         cout<<"\n No elements found...";

      }

      else

      {
            cout<<" ";
            while(n!=NULL)
            {
                if(n->next==NULL)
                {
                    cout<<n->data;
                }
                else
                {
                    cout<<n->data<<"->";
                }
                n = n->next;
            }
        }
    }
    virtual void store(int n)=0;
    virtual int retrive()=0;
};

class Stack :public List
{
    public:
    void store(int n)
    {
        node *n1 = new node();
        n1->data = n;
        n1->next=NULL;
        if((head==NULL)&&(tail==NULL))
        {
```

```
        head = n1;
        tail = n1;
    }
    else
    {
        tail->next = n1;
        tail = n1;
    }
}
int retrive()
{
    if((tail==NULL)&&(head==NULL))
    {
        return -1;
    }
    else
    {
        int n = tail->data;
        node *n1 = head;
        while((n1->next!=tail)&&(head!=tail))
        {
            n1 = n1->next;
        }
        n1->next = NULL;
        free(tail);
        if(head!=tail)
        {
            tail = n1;
        }
```

```
            else
            {
                tail=NULL;
                head=NULL;
            }
            return n;
        }
    }
};

class Queue:public List
{
    public:
    void store(int n)
    {
        node *n1 = new node();
        n1->data = n;
        n1->next=NULL;
        if((head==NULL)&&(tail==NULL))
        {
            head = n1;
            tail = n1;
        }
        else
        {
            tail->next = n1;
            tail = n1;
        }
    }
    int retrive()
    {
        if((tail==NULL)&&(head==NULL))
        {
            return -1;
        }
        else
        {
            int n = head->data;
            if(head==tail)
            {
                head = tail = NULL;
            }
            else
            {
                head = head->next;
            }
            return n;
        }
    }
};

int main()
{
    Stack s1;
    int ch;
    while(1)
    {
        cout<<"\n\n Program to implement stack and queue using pure virtual functions store
and retrieve";
```

```
cout<<"\n\n Menu";
cout<<"\n\n 1. Stack";
cout<<"\n 2. Queue";
cout<<"\n 3. Exit";
cout<<"\n\n Enter your choice - ";
cin>>ch;
if(ch==1)
{
  Stack s1;
  int ch1;
  while(1)
  {
    cout<<"\n\n Stack Menu";
    cout<<"\n 1. Push Element";
    cout<<"\n 2. Pop Element";
    cout<<"\n 3. View Stack";
    cout<<"\n 4. Exit";
    cout<<"\n\n Enter your choice - ";
    cin>>ch1;
    if(ch1==1)
    {
      int element;
      cout<<"\n Enter the element you want to push - ";
      cin>>element;
      s1.store(element);
      cout<<"\n Element Pushed";
    }
    else if(ch1==2)
    {
         int element=0;
         element = s1.retrive();
         if(element==-1)
         {
            cout<<"\n Stack is Empty";
         }
         else
         {
            cout<<"\n Element Popped = "<<element;
         }
      }
      else if(ch1==3)
      {
         cout<<"\n Elements in stack from bottom to top:- ";
         s1.view();
      }
      else if(ch1==4)
      {
         break;
      }
      else
      {
         cout<<"\n\n Wrong choice";
      }
      getch();
   }
}
else if(ch==2)
{
```

```
Queue q1;
int ch1;
while(1)
{
   cout<<"\n\n Queue Menu";
   cout<<"\n 1. Push Element";
   cout<<"\n 2. Pop Element";
   cout<<"\n 3. View Queue";
   cout<<"\n 4. Exit";
   cout<<"\n\n Enter your choice - ";
   cin>>ch1;
   if(ch1==1)
   {
      int element;
      cout<<"\n Enter the element you want to push - ";
      cin>>element;
      q1.store(element);
      cout<<"\n Element Pushed";
   }
   else if(ch1==2)
   {
      int element=0;
      element = q1.retrive();
      if(element==-1)
      {
         cout<<"\n Queue is Empty";
      }
      else
      {
```

```
                    cout<<"\n Element Popped = "<<element;
                }
            }
            else if(ch1==3)
            {
                cout<<"\n Elements in queue from front to rear:- ";
                q1.view();
            }
            else if(ch1==4)
            {
                break;
            }
            else
            {
                cout<<"\n\n Wrong choice";
            }
            getch();
        }
    }
    else if(ch==3)
    {
        exit(0);
    }
    else
    {
        cout<<"\n\n Wrong Choice";
    }
    getch();
}
```

```
    return 0;
}
```

## Algorithm:-

**Start**

**Step1:** A class called list with two pure virtual functions store( ) and retrieve( ) is created

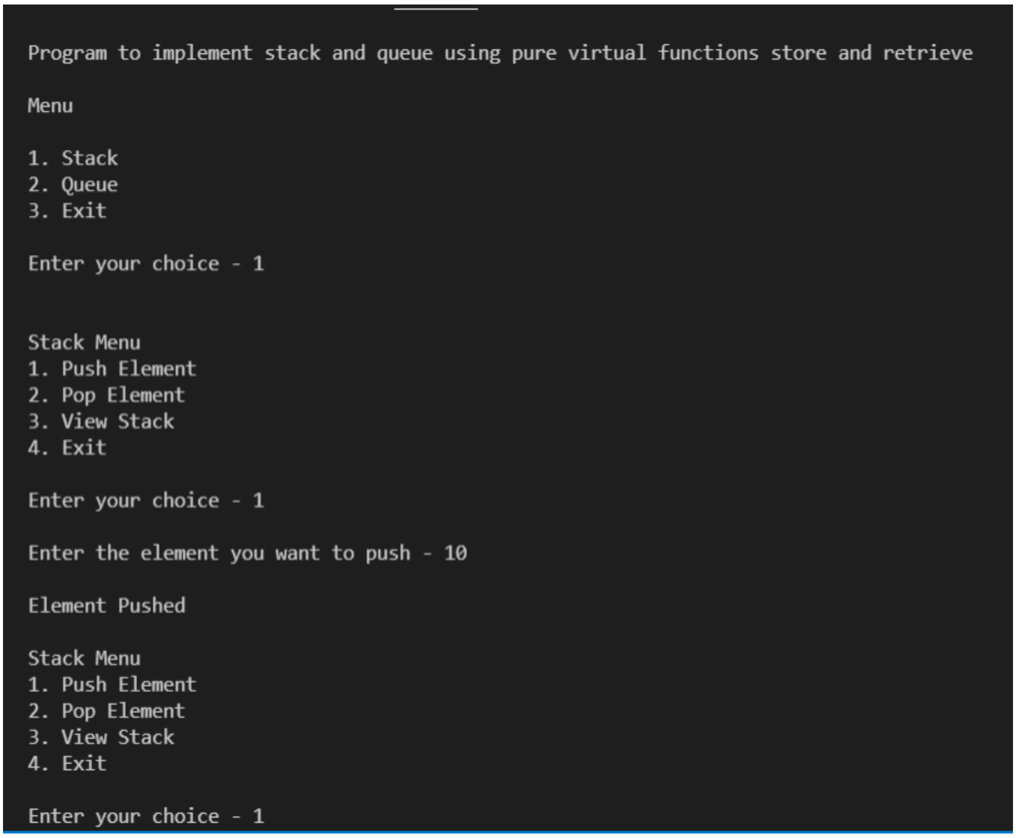
**Step2:** To store a value, store function is called and to retrieve a value, retrieve function is called

**Step3:** Two classes stack and queue are derived from it and store and retrieve functions are overridden.

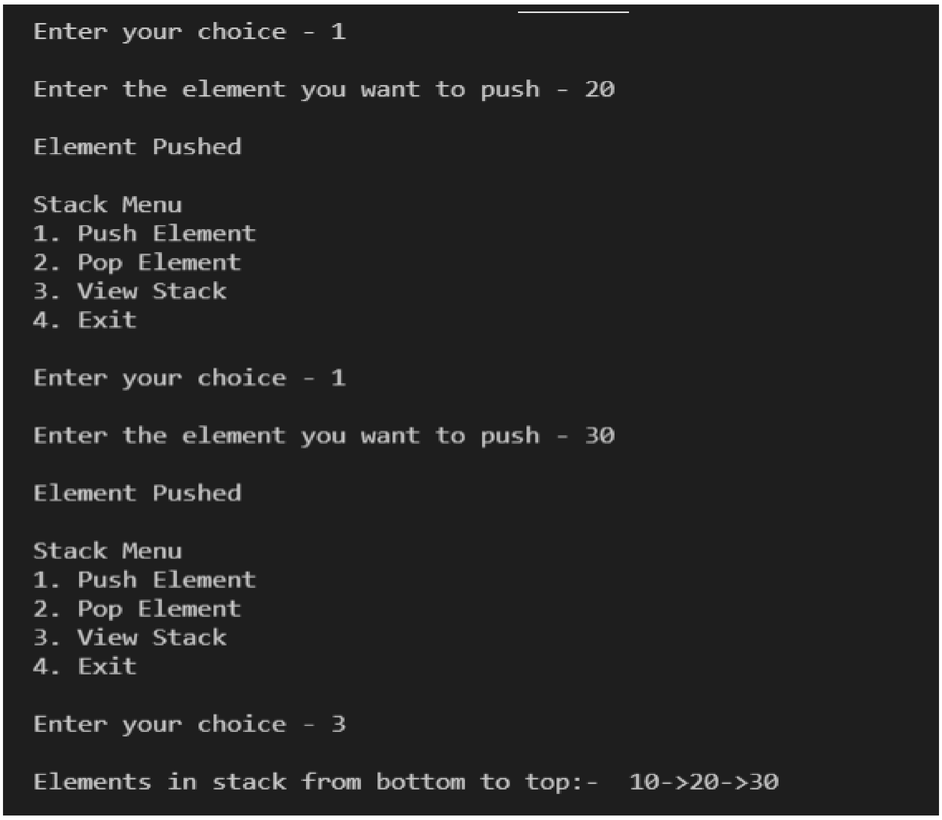
**Step4:** Normal stack and queue push and pop operations are performed on a single list.

**Stop**

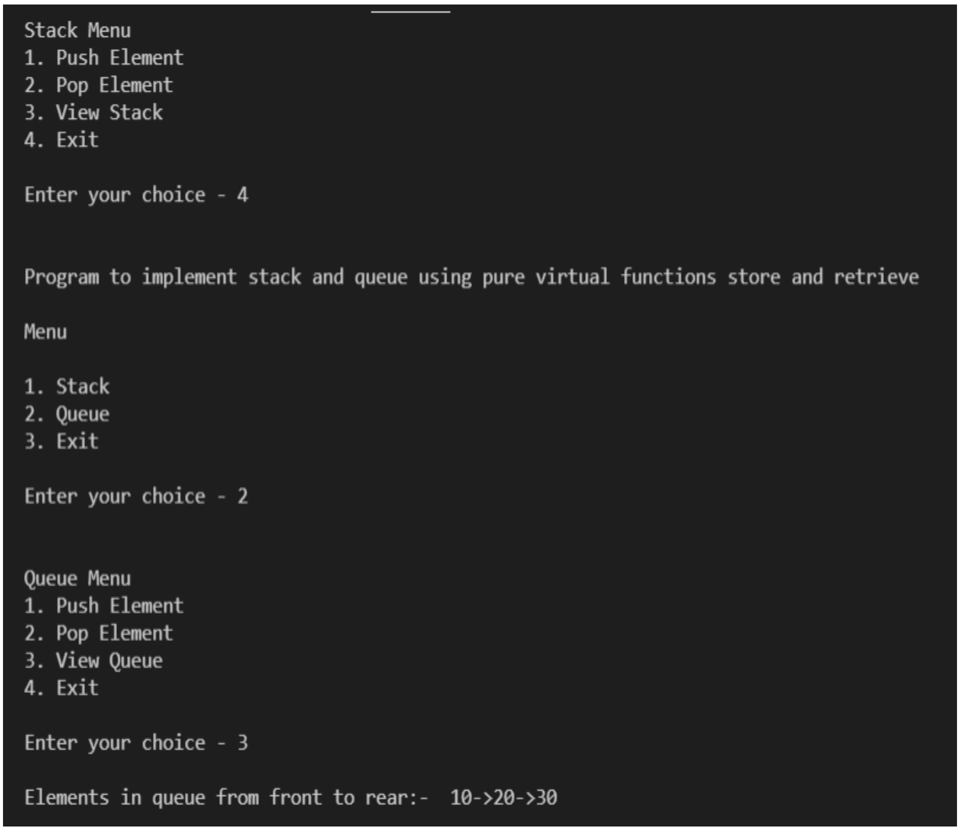
## Output:-



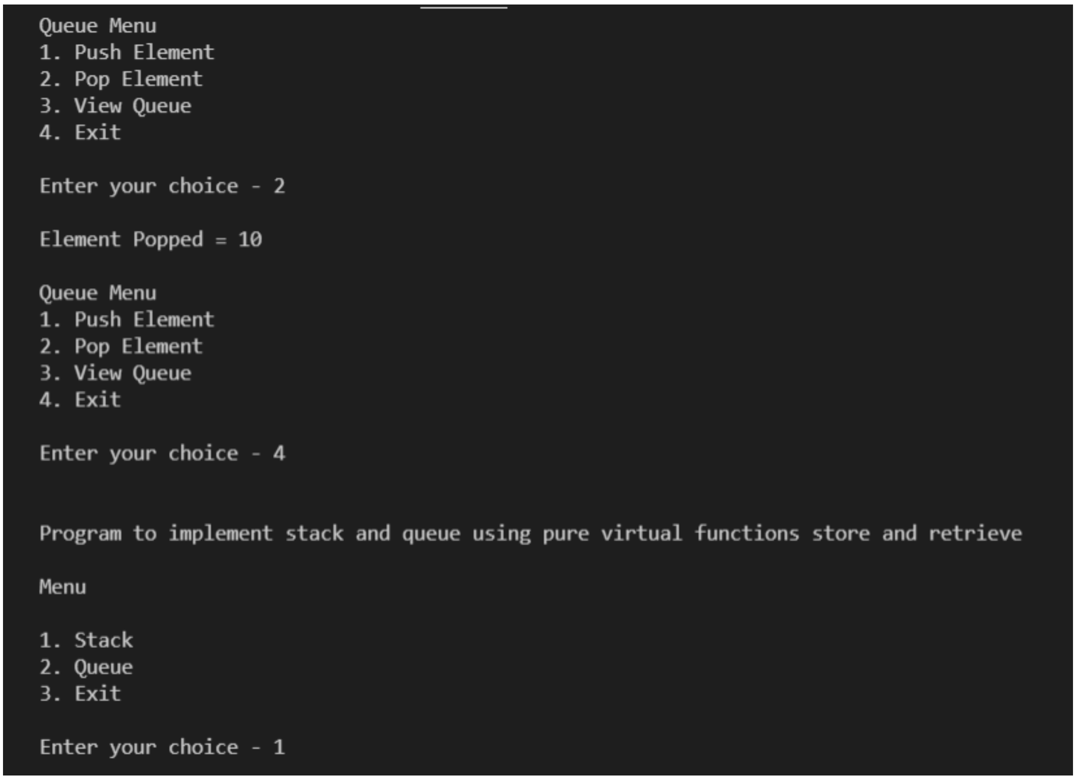
```
Program to implement stack and queue using pure virtual functions store and retrieve

Menu

1. Stack
2. Queue
3. Exit

Enter your choice - 1


Stack Menu
1. Push Element
2. Pop Element
3. View Stack
4. Exit

Enter your choice - 1

Enter the element you want to push - 10

Element Pushed

Stack Menu
1. Push Element
2. Pop Element
3. View Stack
4. Exit

Enter your choice - 1
```



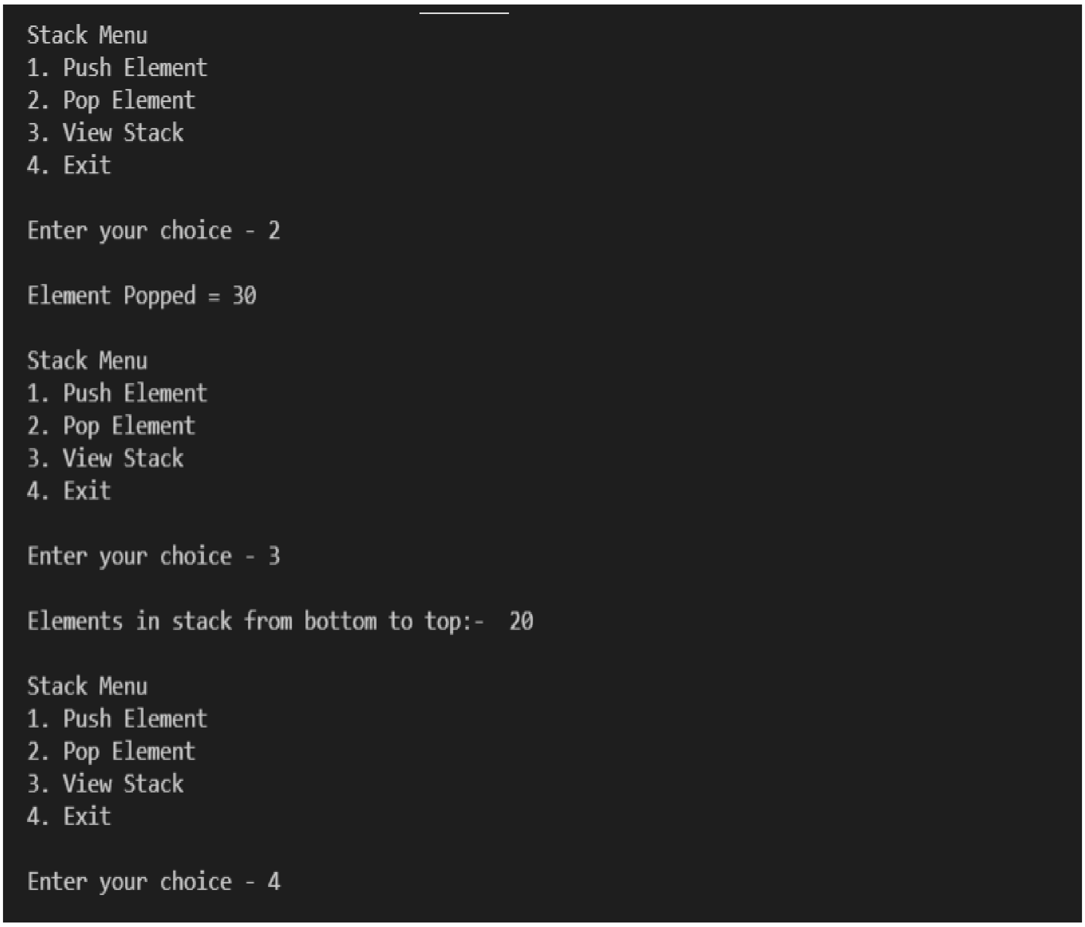
```
Enter your choice - 1

Enter the element you want to push - 20

Element Pushed

Stack Menu
1. Push Element
2. Pop Element
3. View Stack
4. Exit

Enter your choice - 1

Enter the element you want to push - 30

Element Pushed

Stack Menu
1. Push Element
2. Pop Element
3. View Stack
4. Exit

Enter your choice - 3

Elements in stack from bottom to top:-  10->20->30
```



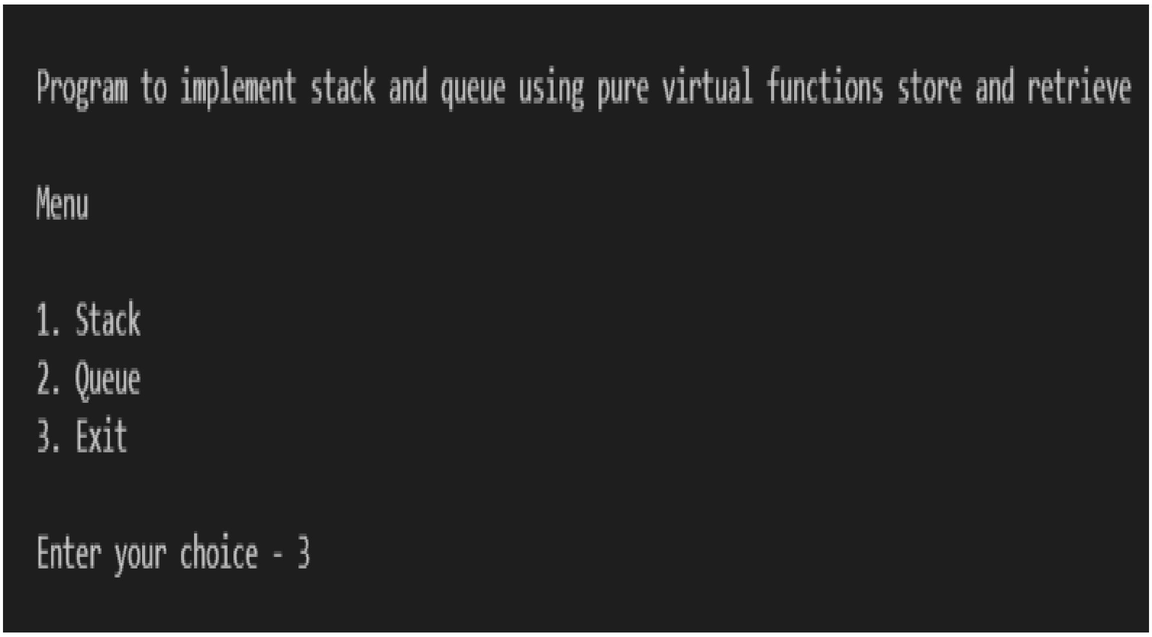
```
Stack Menu
1. Push Element
2. Pop Element
3. View Stack
4. Exit

Enter your choice - 4


Program to implement stack and queue using pure virtual functions store and retrieve

Menu

1. Stack
2. Queue
3. Exit

Enter your choice - 2


Queue Menu
1. Push Element
2. Pop Element
3. View Queue
4. Exit

Enter your choice - 3

Elements in queue from front to rear:-  10->20->30
```



```
Queue Menu
1. Push Element
2. Pop Element
3. View Queue
4. Exit

Enter your choice - 2

Element Popped = 10

Queue Menu
1. Push Element
2. Pop Element
3. View Queue
4. Exit

Enter your choice - 4


Program to implement stack and queue using pure virtual functions store and retrieve

Menu

1. Stack
2. Queue
3. Exit

Enter your choice - 1
```



```
Stack Menu
1. Push Element
2. Pop Element
3. View Stack
4. Exit

Enter your choice - 2

Element Popped = 30

Stack Menu
1. Push Element
2. Pop Element
3. View Stack
4. Exit

Enter your choice - 3

Elements in stack from bottom to top:-  20

Stack Menu
1. Push Element
2. Pop Element
3. View Stack
4. Exit

Enter your choice - 4
```



```
Program to implement stack and queue using pure virtual functions store and retrieve

Menu

1. Stack
2. Queue
3. Exit

Enter your choice - 3
```

# Experiment-9

**Aim:-** Write a c++ program to find the absolute value of an int, float and double using templates only.

## Source Code:-

```
#include<iostream>
using namespace std;
template <class T>
class absoluteValue{
   public:
T AbsoluteValue( T nNumber)
{
  return (nNumber>0)? nNumber:-nNumber;
}
};
int main(){
   absoluteValue<int> a;
   int n1;
   float n2;
   double n3;
   cout<<"\nEnter a integer: ";
   cin>>n1;

   cout<<"\nEnter a float type number: ";
   cin>>n2;

   cout<<"\nEnter a double type number: ";
   cin>>n3;
   cout<<"\nAbsolute value of "<<n1<<" is: "<<a.AbsoluteValue(n1);
   cout<<"\nAbsolute value of "<<-n1<<" is: "<<a.AbsoluteValue(-n1);
```

```
    absoluteValue<float> b;



    cout<<"\nAbsolute value of "<<n2<<" is: "<<b.AbsoluteValue(n2);
    cout<<"\nAbsolute value of "<<-n2<<" is: "<<b.AbsoluteValue(-n2);

    absoluteValue<double> c;

    cout<<"\nAbsolute value of "<<n3<<" is: "<<c.AbsoluteValue(n3);
    cout<<"\nAbsolute value of "<<-n3<<" is: "<<c.AbsoluteValue(-n3);
    return 0;
}
```

## Algorithm:-

**Start**

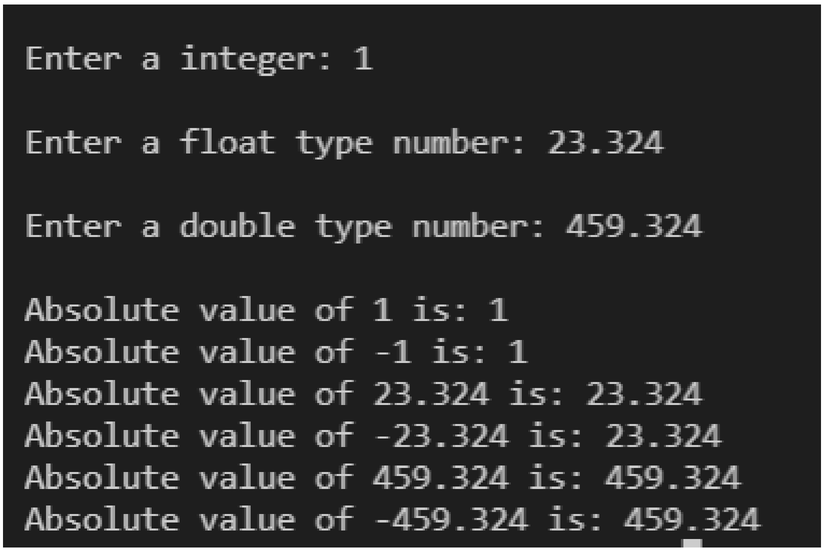
**Step1:** Enter the integer,float and double number.

**Step2:** According to the input given of a particular datatype ,the corresponding template do the work of calculating the absolute value of the input and return the absolute value.

**Step3:** The returned absolute values of the number given of the particular datatype are printed accordingly.

**Stop.**

## Output:-



```
Enter a integer: 1

Enter a float type number: 23.324

Enter a double type number: 459.324

Absolute value of 1 is: 1
Absolute value of -1 is: 1
Absolute value of 23.324 is: 23.324
Absolute value of -23.324 is: 23.324
Absolute value of 459.324 is: 459.324
Absolute value of -459.324 is: 459.324
```

## Experiment-10

**Aim:-** Consider a data structure QUEUE. It can INSERT and DELETE data. Using exception handling, simulate a QUEUE. Throw exceptions when QUEUE is full or is empty.

### Source Code:-

```
#include<iostream>

using namespace std;

class queue

{

private:

int *q;

int max, front, rear, cnt;

public:

class FULL{};  //for exception handling

class EMPTY{}; //for exception handling

queue(int);

void enqueue(int);

int dequeue(void);

void display(void);

};

queue::queue(int m)

{

q=new int[m];

rear=0;

front=0;

cnt=0;

max=m;

}

void queue::enqueue(int item)

{
```

```
if(cnt<max)
{
front = front%max;
q[front++]=item;
cnt++;
}
else
throw FULL(); //FULL object is thrown
}
int queue::dequeue(void)
{
if(cnt>0)
{
cnt=cnt-1;
rear = rear %max;
return q[rear++];
}
else
throw EMPTY(); //EMPTY object is thrown
}
void queue::display(void)
{
if(cnt>0)
{
for(int i=0, j=front; i<cnt;i++,j++)
cout<<q[j%max]<<" ";
cout<<endl;
}
else
```

```
throw EMPTY();

}

int main()

{

int item, size;

int ch=1;

cout<<"\nEnter the size of the queue: ";

cin>>size;

queue q(size);

cout<<"\nQueue Operations using Exception Handling";

cout<<"\n\nMENU\n1.ENQUEUE\n2.DEQUEUE\n3.SHOW QUEUE\n4.EXIT";

cout<<"\nEnter your choice: ";

cin>>ch;

do

{

switch(ch)

{

case 1:

cout<<"\nEnter the item to insert in to the queue: ";

cin>>item;

try

{

q.enqueue(item);

}

catch(queue::FULL)  //FULL object is caught

{

cout<<"\n***Queue Full***\n";

}

break;
case 2:

try

{

cout<<"\nRemoved Item from the Q is: "<<q.dequeue();

}

catch(queue::EMPTY) //EMPTY object is caught

{

cout<<"\n***Queue Empty***\n";

}

break;

case 3:

cout<<"\nThe Queue is \n";

try

{

q.display();

}

catch(queue::EMPTY)

{

cout<<"\n***Queue Empty***\n";

}

break;

case 4:

exit(0);

}

cout<<"\nEnter your choice: ";

cin>>ch;

}while(ch<5);

return 0;

}
```

## Algorithm:-

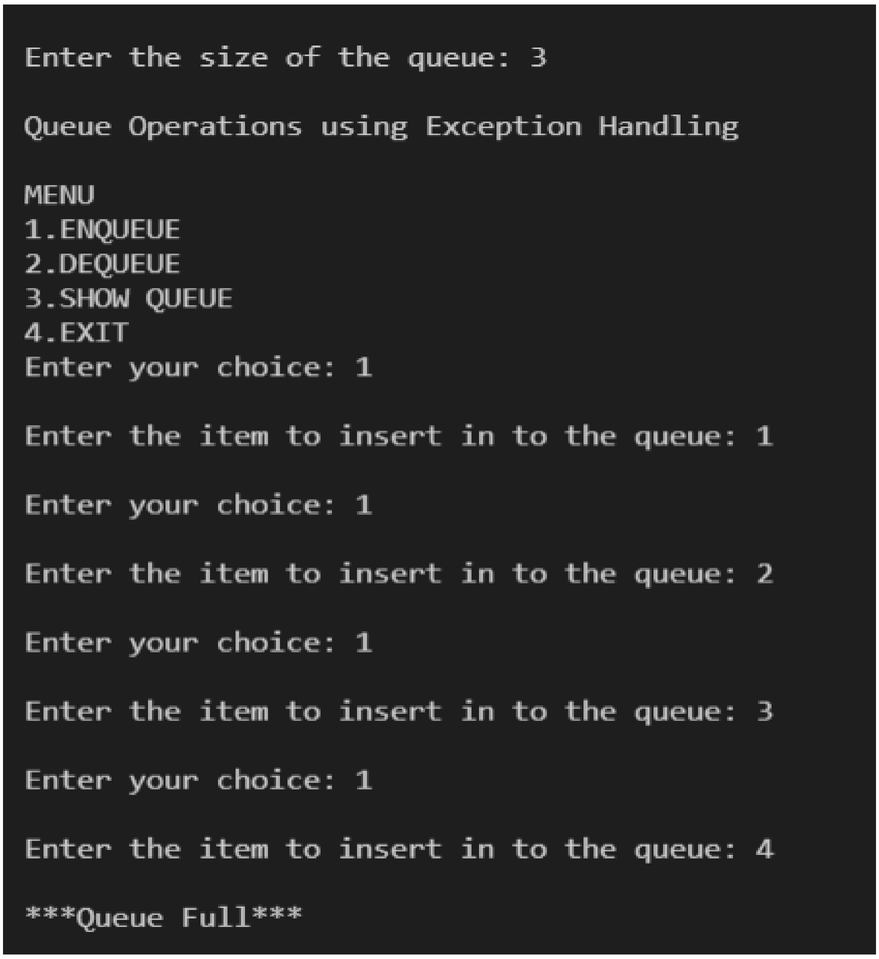
**Start**

**Step1:** Class queue is created with the operations of enqueue for inserting an element into the queue and dequeue for deleting an element from the queue.

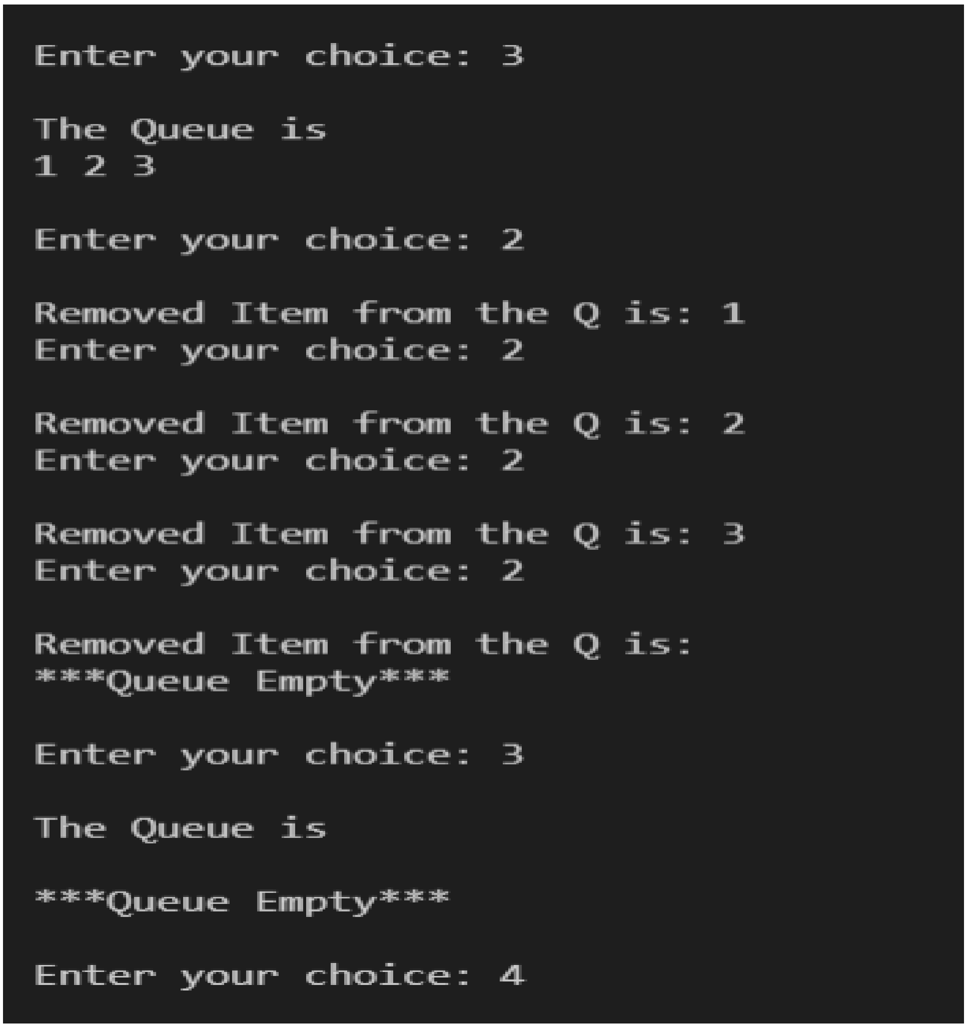
**Step2:** Exception handling is used to show messages when the queue is full while enqueue and when the queue is empty while dequeue or display.

**Stop**

## Output:-



```
Enter the size of the queue: 3

Queue Operations using Exception Handling

MENU
1.ENQUEUE
2.DEQUEUE
3.SHOW QUEUE
4.EXIT
Enter your choice: 1

Enter the item to insert in to the queue: 1

Enter your choice: 1

Enter the item to insert in to the queue: 2

Enter your choice: 1

Enter the item to insert in to the queue: 3

Enter your choice: 1

Enter the item to insert in to the queue: 4

***Queue Full***
```



```
Enter your choice: 3

The Queue is
1 2 3

Enter your choice: 2

Removed Item from the Q is: 1
Enter your choice: 2

Removed Item from the Q is: 2
Enter your choice: 2

Removed Item from the Q is: 3
Enter your choice: 2

Removed Item from the Q is:
***Queue Empty***

Enter your choice: 3

The Queue is

***Queue Empty***

Enter your choice: 4
```

# Experiment-11

**Aim:-** Write a c++ program that reads a file and counts the number of sentences, words and characters present in it.

## Source Code:-

```
#include<iostream>

#include<fstream>

#include<string.h>

#include<cstdlib>

using namespace std;

int main()

{

    int noc=0,now=0,nol=0;

    FILE *fr;

    char fname[20],ch;


    cout<<"\n Enter Source File Name : ";

    gets(fname);

    fr=fopen(fname,"r");

    if(fr==NULL)

    {

        cout<<"\n Invalid File Name. \n No such File or Directory ";

        exit(0);

    }

    ch=fgetc(fr);

    while(ch!=EOF)

    {

        if(ch!=' ' && ch!='\n')

            noc++;

        if(ch==' ')
```

```
            now++;
         if(ch=='\n')
         {
            nol++;
            now++;
         }
         ch=fgetc(fr);
     }
     fclose(fr);
     cout<<"\n Total No. of Characters  : "<<noc;
     cout<<"\n Total No. of Words       : "<<now;
     cout<<"\n Total No. of Sentences   : "<<nol;


     return 0;
}
```

## Algorithm:-

**Start**

**Step1:** Open source file in r (read) mode.

**Step2:** Initialize three variables noc=0,now=0 and nol=0 to store counts.

**Step3:** Read a character from file and store it to some variable say ch.

**Step4:** Increment the characters count if(ch!='  ' && ch!='\n')

Increment the words count if(ch=='  ')

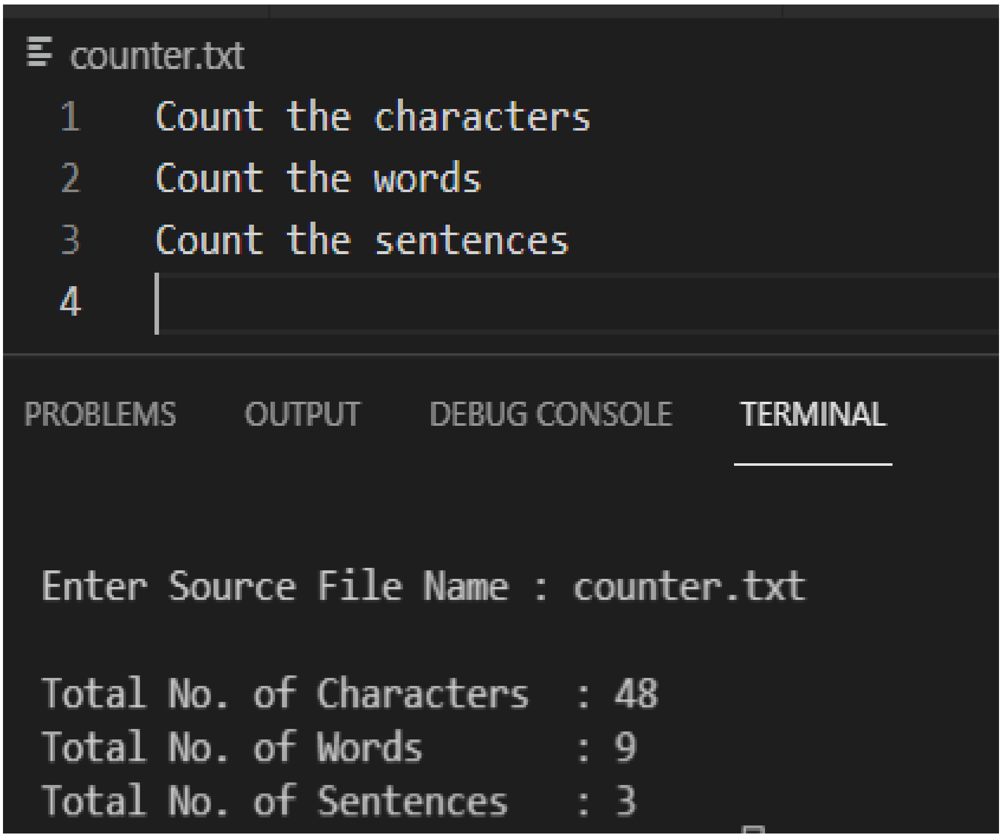
Increment the words count as well as the sentences count if(ch=='\n')

**Step5:** Repeat step 3-4 till file has reached end.

**Step6:** Display the character count, words count and sentences count.

**Stop**

**Output:-**



```
Count the characters
Count the words
Count the sentences

```

```
Enter Source File Name : counter.txt

Total No. of Characters  : 48
Total No. of Words       : 9
Total No. of Sentences   : 3
```

# Experiment-12

**Aim:-** Write a c++ program that reads an array of numbers from file and creates another two files to store the odd numbers in one file and even numbers in another file.

## Source Code:-

```
#include<iostream>
using namespace std;
int main()
{
  FILE *fptr1, *fptr2, *fptr3;
  int n, i, num;
  cout<<"Enter number of values : ";
  cin>>n;
  cout<<"\nEnter the values : ";
  fptr1 = fopen("NUMBERS.txt", "w");
  for(i = 0 ; i < n ; i++)
  {
    cin>>num;
    putw(num, fptr1);
  }
  fclose(fptr1);
  fptr1 = fopen("NUMBERS.txt", "r");
  fptr2 = fopen("ODD.txt", "w");
  fptr3 = fopen("EVEN.txt", "w");
  while((num = getw(fptr1)) != EOF)
  {
    if(num % 2 == 0){
      putw(num, fptr3) ;
    } else{
      putw(num, fptr2) ;
    }
  }
  fclose(fptr1);
  fclose(fptr2);
  fclose(fptr3);
  fptr2 = fopen("ODD.txt", "r");
  fptr3 = fopen("EVEN.txt", "r");
  cout<<"\nContents of ODD file is : ";
  while((num = getw(fptr2)) != EOF){
    cout<<num<<" ";
  }
  cout<<"\n\nContents of EVEN file is : ";
  while((num = getw(fptr3)) != EOF){
    cout<<num<<" ";
  }
  fclose(fptr2);
  fclose(fptr3);
}
```

## Algorithm:-

**Start**

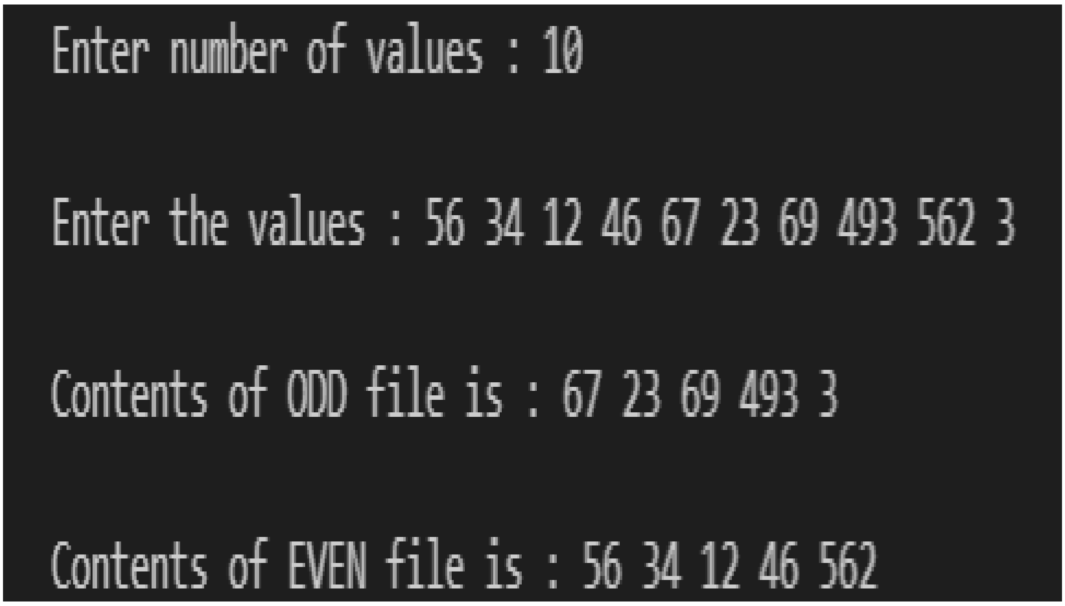
**Step1:** Write the number of values to be stored in the file and then create a file “NUMBERS.txt” and then store the numbers in this “NUMBERS.txt” file taking it from the terminal.

**Step2:** Now read the file “NUMBERS.txt” and write into “ODD.txt” and “EVEN.txt” and assign the numbers from “NUMBERS.txt” file to “ODD.txt” and “EVEN.txt” depending upon the condition  if(num % 2 == 0) then put it in “EVEN.txt” otherwise put it in “ODD.txt”.

**Step3:** Read the data from the files “EVEN.txt” and “ODD.txt” and display it on the terminal.

**Stop**

**Output:-**



```
Enter number of values : 10


Enter the values : 56 34 12 46 67 23 69 493 562 3


Contents of ODD file is : 67 23 69 493 3


Contents of EVEN file is : 56 34 12 46 562
```