Introduction to C++ Programming (PLC144)

UNIT – 5

Text Book: Object-Oriented Programming with C++ , E-Balaguruswamy.

Error Vs Exceptions

- Programs may have errors.
- 2 types of errors. Logical and Syntactical Errors.
- Logical Errors: Occur due to poor understanding of problem and solution procedure.
- Syntactic Errors: Occur due to poor understanding of programming language.
- Can detect errors using exhaustive debugging and testing procedures.

Error Vs Exceptions

- Exceptions are run time anomalies or unusual conditions that a program may encounter during execution.

anomalies or conditions such as

- Division by zero
- Access to an array outside of its bounds
- Running out of memory
- Running out of disk space

Important is to identify and deal exceptions→ called as exception handling.

EXCEPTION HANDLING

- **Exceptions are of 2 kinds**
- Synchronous Exception:
  - Out of range index
  - Overflow
- Asynchronous Exception:
  - Error that are caused by causes beyond the control of the program
  - Keyboard interrupts
- Exception handling in C++ :only synchronous exceptions can be handled.

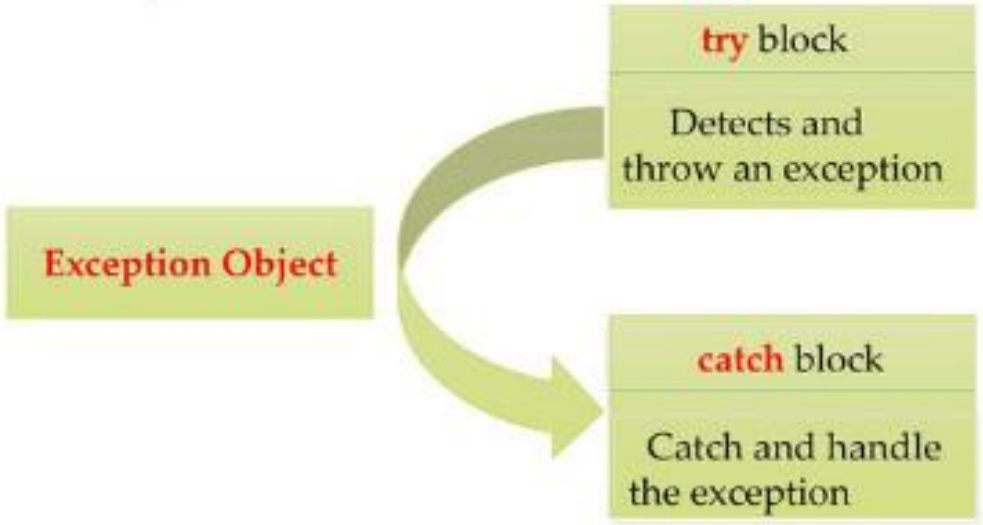
- **EXCEPTION HANDLING (CONT…)**
- Exception handling mechanism: Detect and report exception circumstance so that action can be taken.

A separate error handling code that performs following tasks is required:

- Find the problem (Hit the exception).
- Inform that an error has occurred (Throw the exception).
- Receive the error information (Catch the exception).
- Take corrective action (handle the exception).

EXCEPTION HANDLING MECHANISM

- It is basically build upon three keywords
- try
- throw
- catch



- The keyword try is used to preface a block of statements which may generate exceptions.
- When an exception is detected, it is thrown using a throw statement in the try block.
- A catch block defined by the keyword ‘catch’ catches the exception and handles it appropriately.
- The catch block that catches an exception must immediately follow the try block that throws the exception.

```
.....
.....
try
{
        .....
        throw exception;            // Block of statements which
        .....                       // detects and throws an exception
        .....
}
catch(type arg)                     // Catches exception
{
        .....
        .....                       // Block of statements that
        .....                       // handles the exception
        .....
}
.....
.....
```

- Exceptions are objects used to transmit information about a problem.
- If the type of the object thrown matches the arg type in the catch statement, the catch block is executed.
- If they do not match, the program is aborted.

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
int main()
{
   int a,b;
   cout<<"Enter the values of a and b"<<endl;
   cin>>a;
   cin>>b;
   int x = a-b;
   try
   {
      if(x !=0)
      {
         cout<<"Result (a/x) is \t" <<a/x<<endl;
      }
      else          //There is an Exception
      {
         throw(x);   //Throws int object
      }
   }
   catch(int i)       //catches the exception
   {
      cout<<"Exception caught is in x=" << x <<endl;
   }
cout<<"END";
return 0;
}
```

First Run OUTPUT

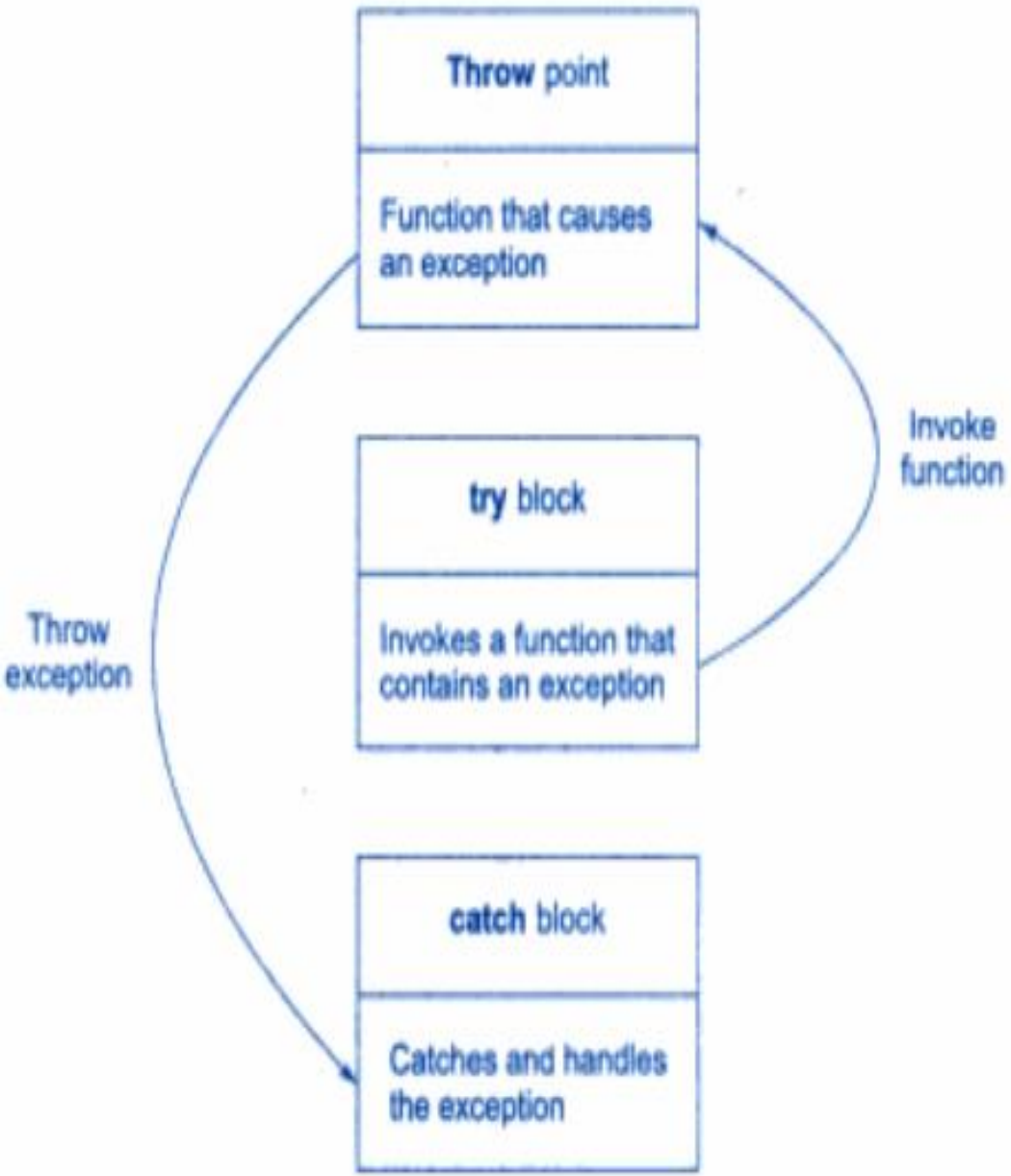
```
Enter the values of a and b
10
5
Result (a/x) is         2
END
```

Second Run OUTPUT

```
Enter the values of a and b
10
10
Exception caught is in x=0
END
```

Function throwing exception

- Exceptions can be thrown by functions that are invoked from within the try block.
- The point at which the throw is executed is called throw point.
- Exception is thrown to the catch block. Control cannot return to the throw point.



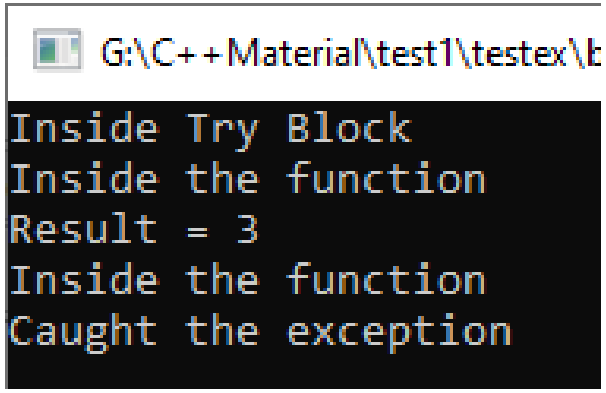

```
type function(arg list)      // Function with exception
{
        ......
        ......
        throw(object);       // Throws exception
        ......
        ......
}
......
......
try
{
        ......
        ...... Invoke function here
        ......
}
catch(type arg)              // Catches exception
{
        ......
        ...... Handles exception here
        ......
}
......
```

Program 2: Invoking Function that generates exception

```
void divide(int x, int y ,int z)
{
   cout<<"Inside the function "<<endl;
   if((x-y)!=0)
   {
      int R = z / (x-y);
      cout<<"Result = "<<R<<endl;
   }
   else
   {
      throw(x-y);
   }
}
```

```
int main()
{
   try
   {
      cout<<"Inside Try Block"<<endl;
      divide(20,10,30);
      divide(10,10,30);
   }
   catch(int i)
   {
      cout<<"Caught the exception"<<endl;
   }
   return 0;
}
```

OUTPUT



THROWING MECHANISM

- The throw statement can have one of the following 3 forms
- throw(exception)
- throw exception
- throw //used to re-throw a exception
- The operand object exception can be of any type, including constant.
- It is also possible to throw an object not intended for error handling.
- Throw point can be in a deeply nested scope within a try block or in a deeply nested function call. In any case, control is transferred to the catch statement.

CATCHING MECHANISM catch(type arg) { … }

- The type indicates the type of exception the catch block handles.
- The parameter arg is an optional parameter name.
- The catch statement catches an exception whose type matches with the type of the catch argument.

```
catch(type arg)
{
 …// Statements for managing exceptions
}
```

- If the parameter in the catch statement is named, then the parameter can be used in the exception handling code.
- If a catch statement does not match the exception it is skipped.
- More than one catch statement can be associated with a try block.

```
try
 {
throw exception;
}
catch(type1 arg)    // catch block 1
{
Statements;
}
catch(type2 arg)   // catch block 2
{
Statements;
}
…
catch(typeN arg)  // catch block N
{
Statements;
}
```

When an exception is thrown, the exception handlers are searched in order for a match.

The first handler that yields a match is executed.

If several catch statement matches the type of an exception the first handler that matches the exception type is executed.

Catch all exception

```
catch (…)
{
// statement for processing all exceptions
}
```

- A handler may decide to rethrow the exception caught without processing it.
- In such a case we have to invoke throw without any arguments as shown below

```
throw;
```

- This causes the current exception to be thrown to the next enclosing try/catch sequence and is caught by a catch statement listed after the enclosing try block

Program 3: Multiple catch statements:

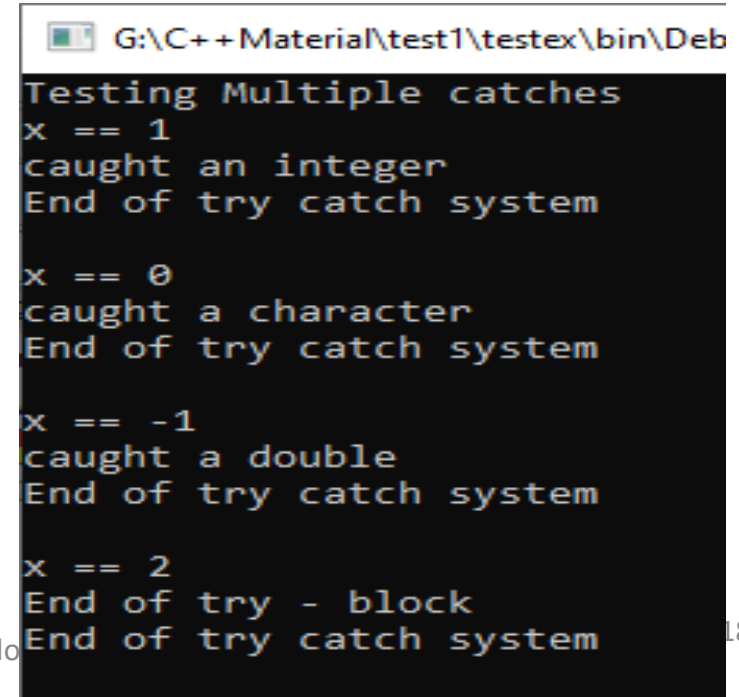
```
void test(int x)
{
   try
   {
      if(x == 1) throw x;
      else if(x == 0) throw 'x';
      else if(x == -1) throw  1.0;

      cout<<"End of try - block "<<endl;
   }
   catch(char c)
   {
      cout<<"caught a character"<<endl;
   }
   catch(int m)
   {
      cout<<"caught an integer"<<endl;
   }
   catch(double c)
   {
      cout<<"caught a double"<<endl;
   }
   cout<<"End of try catch system"<<endl<<endl;
}
```

```
int main()
{
   cout<<"Testing Multiple catches"<<endl;
   cout<<"x == 1"<<endl;
   test(1);
   cout<<"x == 0"<<endl;
   test(0);
   cout<<"x == -1"<<endl;
   test(-1);
   cout<<"x == 2"<<endl;
   test(2);

   return 0;
}
```

OUTPUT:



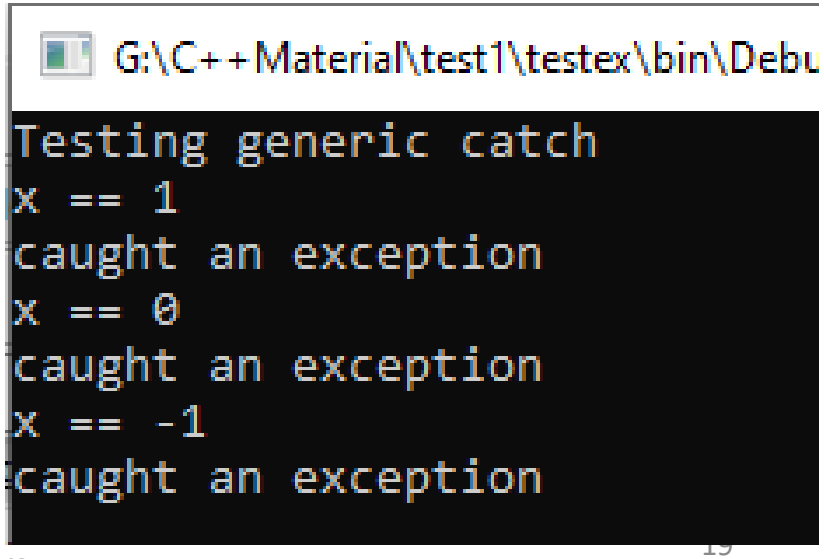
Program 4: catching all exceptions

```
void test(int x)
{
    try
    {
        if(x == 1) throw x;
        else if(x == 0) throw 'x';
        else if(x == -1) throw  1.0;

        cout<<"End of try - block "<<endl;
    }
    catch(...)
    {
        cout<<"caught an exception"<<endl;
    }
}
```

```
int main()
{
    cout<<"Testing generic catch"<<endl;
    cout<<"x == 1"<<endl;
    test(1);
    cout<<"x == 0"<<endl;
    test(0);
    cout<<"x == -1"<<endl;
    test(-1);
    return 0;
}
```

OUTPUT

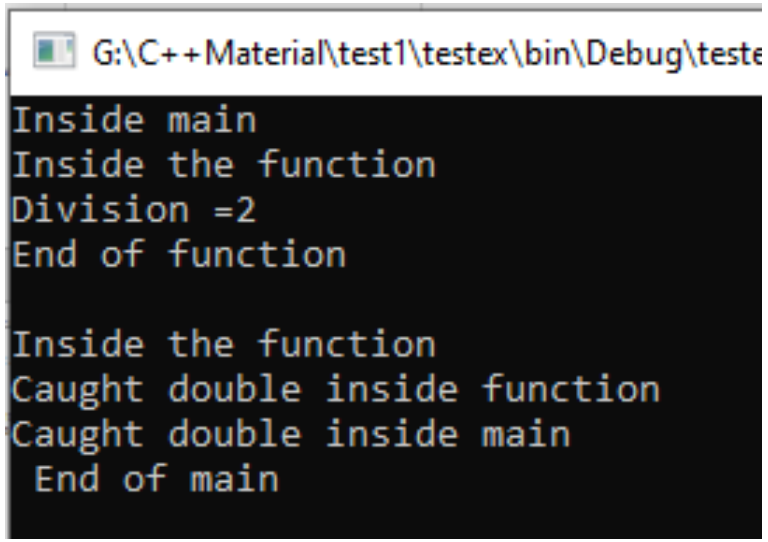


Program 5: Rethrowing an exception

```
void divide(double x, double y)
{
   cout<<"Inside the function "<<endl;
   try
   {
      if(y == 0.0)
         throw y;
      else
         cout<<"Division ="<<x/y<<endl;
   }
   catch(double)
   {
      cout<<"Caught double inside function"<<endl;
      throw;
   }
   cout<<"End of function"<<endl<<endl;
}
```

```
int main()
{
   cout<<"Inside main"<<endl;
   try
   {
      divide(20,10);
      divide(10,0);
   }
   catch(double)
   {
      cout<<"Caught double inside main"<<endl;
   }
   cout<<" End of main"<<endl;
   return 0;
}
```

OUTPUT

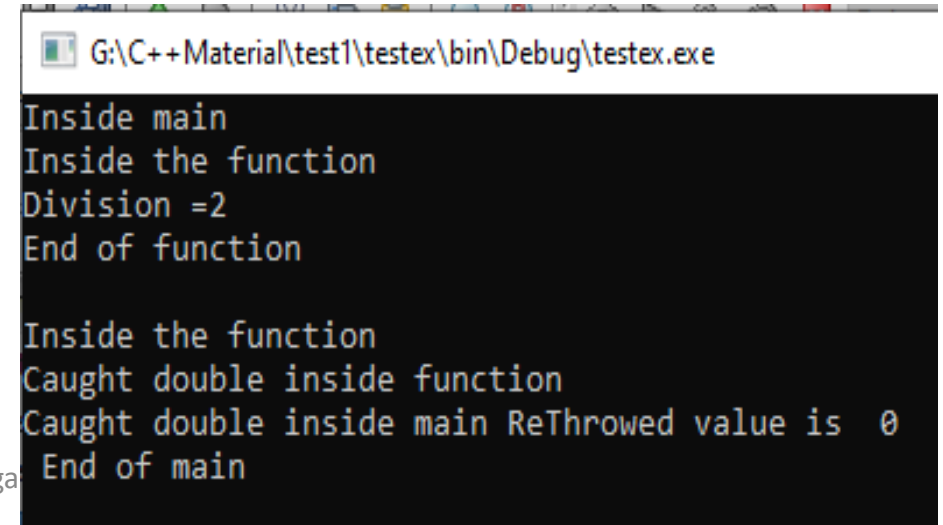


Program 6: Rethrowing an exception

```
void divide(double x, double y)
{
   cout<<"Inside the function "<<endl;
   try
   {
      if(y == 0.0)
         throw y;
      else
         cout<<"Division ="<<x/y<<endl;
   }
   catch(double i)
   {
      cout<<"Caught double inside function"<<endl;
      throw;
   }
   cout<<"End of function"<<endl<<endl;
}
```

```
int main()
{
   cout<<"Inside main"<<endl;
   try
   {
      divide(20,10);
      divide(10,0);
   }
   catch(double j)
   {
      cout<<"Caught double inside main ReThrowed value is
"<<j<<endl;
   }
   cout<<" End of main"<<endl;
   return 0;
}
```

OUTPUT



Specifying Exceptions

- Can restrict a function to throw only specified exceptions.
- Use throw list clause to the function definition.

Syntax:

```
type function(arg-list) throw (type-list)
{
----
---- function body
-----
}
```

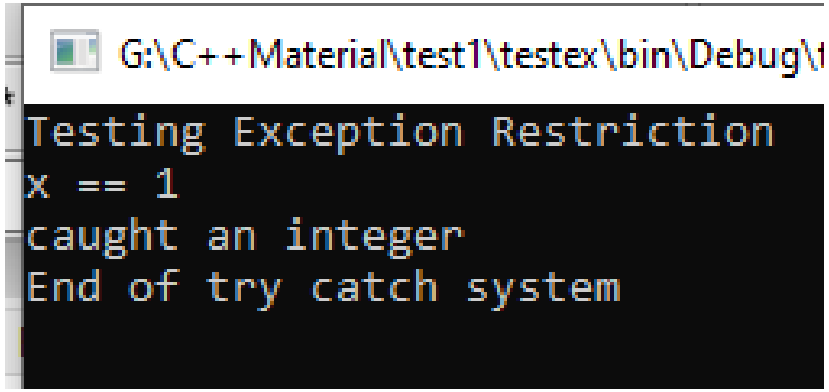
- type-list specifies type of exceptions that may be thrown.
- Throwing any other type of exception will cause abnormal program termination.
- Can prevent function from throwing any exception using throw() // empty list

```
using namespace std;
void test(int x) throw(int,double)
{

      if(x == 1) throw x;
      else if(x == 0) throw 'x';
      else if(x == -1) throw  1.0;

      cout<<"End of function block "<<endl;
}

int main()
{
   try
   {
   cout<<"Testing Exception Restriction"<<endl;
   cout<<"x == 1"<<endl;
   test(1);
   cout<<"x == 0"<<endl;
   test(0);
   cout<<"x == -1"<<endl;
   test(-1);
  cout<<"x == 2"<<endl;
  test(2);
   }
catch(char c)
   {
      cout<<"caught a character"<<endl;
   }
   catch(int m)
   {
      cout<<"caught an integer"<<endl;
   }
   catch(double c)
   {
      cout<<"caught a double"<<endl;
   }
   cout<<"End of try catch system"<<endl<<endl;
   return 0;
}
```



```
int add(int a, int b) {
if(( a<0 ) || (b<0))  {
throw "Enter positive number";
}
return (a+b);
}

int sub(int a, int b) {
if(( a<0 ) || (b<0))  {
throw " Enter positive number ";
}
 return (a-b);
}
```

```
int division(int a, int b) {
if( (b == 0 ) || (a==0) ) {
throw "Enter a number greater than zero”;
}
return (a/b);
}
int product(int a, int b) {
if(( a == 0 ) && (b==0))  {
throw " Enter a number greater than zero!";
}
return (a*b);
}

int main () {
int x = 50;
int y = 0;
int  z = 0;
```

```
try {
z = division(x, y);
cout << z << endl;
} catch (const char* msg) {
cerr << msg << endl;
}
try {
z = product(x, y);
cout << z << endl;
} catch (const char* msg) {
cerr << msg << endl;
}
```

```
try {
z = add(x, y);
cout << z << endl;
} catch (const char* msg) {
cerr << msg << endl;
}
try {
z = sub(x, y);
cout << z << endl;
} catch (const char* msg) {
cerr << msg << endl;
}

return 0;
}
```

- a) Write a C++ program that creates a Calculator class. The class contains two variables of integer type. Design a constructor that accepts two values as parameter and set those values.
- Design four methods named Add (), Subtract (), multiply (), Division ( ) for performing addition, subtraction, multiplication and division of two numbers.
- For addition and subtraction, two numbers should be positive. If any negative number is entered then throw an exception in respective methods. So design an exception handler (ArithmeticException) in Add () and Subtract () methods respectively to check whether any number is negative or not.
- For division and multiplication two numbers should not be zero. If zero is entered for any number then throw an exception in respective methods. So design an exception handler (ArithmeticException) in multiply () and Division () methods respectively to check whether any number is zero or not.

```
class Calculator
{
   int a,b;
public:
   Calculator()
   {
      a=0;b=0;
   }
   Calculator(int i,int j)
   {
      a=i;b=j;
   }
   int add()
   {
   if(( a<0 ) || (b<0))
   {
   throw "Enter positive number";
   }
   return (a+b);
   }
int sub()
   {
   if(( a<0 ) || (b<0))
   {
   throw " Enter positive number ";
   }

   return (a-b);
   }
   int division()
   {
   if( (b == 0 ) || (a==0) )
   {
   throw "Enter a number greater than zero" ;
   }
   return (a/b);
   }
int product()
   {
   if(( a == 0 ) && (b==0))
   {
   throw " Enter a number greater
than zero!";
   }
   return (a*b);
   }

};
```

```
int main()
{
   Calculator C1(5,6),C2;
   try
   {
      cout<<C1.add()<<endl;
   }
   catch(const char*msg)
   {
      cout<<msg<<endl;
   }
   try
   {
      cout<<C1.sub()<<endl;
   }
   catch(const char*msg)
   {
      cout<<msg<<endl;
   }
   try
   {
      cout<<C2.division()<<endl;
   }
   catch(const char*msg)
   {
      cout<<msg<<endl;
   }
   try
   {
      cout<<C2.product()<<endl;
   }
   catch(const char*msg)
   {
      cout<<msg<<endl;
   }
   return 0;
}
```