Exceptions

Introduction

The most common types of error (also known as bugs) occurred while programming in C++ are Logic error and Syntactic error. The logic errors occur due to poor understanding of the problem and solution procedure. The syntactic errors arise due to poor understanding of the language. These errors are detected by using exhaustive debugging and testing.

There are some problems other than logic or syntax errors. They are known as exceptions.

- These are basically runtime errors.
- Exceptions are runtime anomalies or unusual conditions that a program may encounter while in execution.
- Exceptions might include conditions such as division by zero, access to an array outside of its bounds, running out of memory or disk space, not being able to open a file, trying to initialize an object to an impossible value etc.
- When a program encounters an exceptional condition, it is important that it is identified and dealt with effectively.
- C++ provides built-in language features to detect and handle exceptions.

Why Exception Handling?

The purpose of the exception handling mechanism is to provide means to detect and report an "exceptional circumstance" so that appropriate action can be taken.

Separation of Error Handling code from Normal Code: In traditional error handling codes, there are always if else conditions to handle errors. These conditions and the code to handle errors get mixed up with the normal flow. This makes the code less readable and maintainable. With try, and catch blocks, the code for error handling becomes separate from the normal flow.

Exception Handling Mechanism

The mechanism suggests a separate error handling code that performs the following tasks:

- Find the problem (Hit the exception/try).
- Inform that an error has occurred (throw the exception).
- Receive the error information (catch the exception).
- Take corrective action (Handle the exception).

The error handling code basically consists to two segments, one to detect errors and to throw exceptions, and the other to catch the exceptions and to take appropriate actions.

C++ exception handling is built upon three keywords: try, catch, and throw.

a) try

b) throw

c) catch

- **try:** The keyword "try" is used to refer a block of statements surrounded by braces which may generate exceptions. This block of statement is known as **try** block. A try block identifies a block of code for which particular exceptions will be activated. Try block is intended to throw exceptions, which is followed by catch blocks. Only one try block. It's followed by one or more catch blocks.

- **throw:** A program throws an exception when a problem shows up. This is done using a throw keyword. When an exception is detected, it is thrown using a ***throw*** statement in the try block or in functions that are invoked from within the try block. This is called throwing an exception and the point at which the throw is executed is called the ***throw point***. A throw expression accepts one parameter and that parameter is passed to handler.

❖ **catch:** Represents a block of code that is executed when a particular exception is thrown. A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The ***catch*** keyword indicates the catching of an exception. A ***catch*** block defined by the keyword **catch** catches the exception thrown by the **throw** statement in the try block, and handles it appropriately. Catch block is intended to catch the error and handle the exception condition. We can have multiple catch blocks.

The general syntax for handling exception is:

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

Figure below shows try-catch relationship.

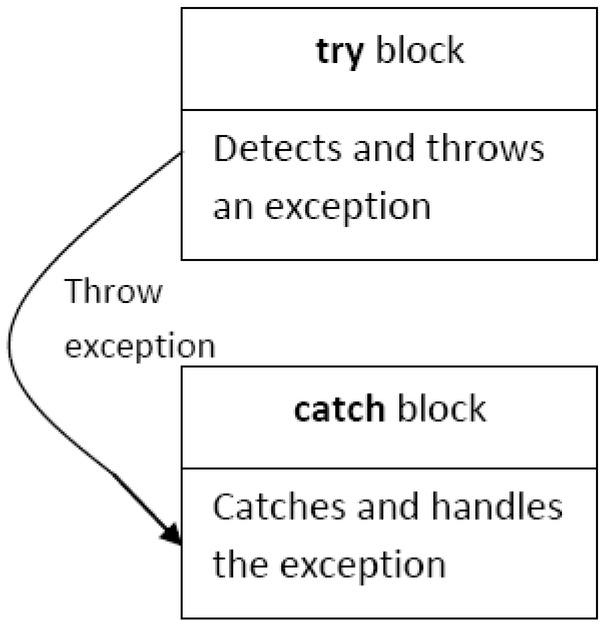

try block
Detects and throws an exception
Throw exception
catch block
Catches and handles the exception

Fig: The block throwing exception

Throwing Mechanism:

The exception is thrown by the use of throw statement in one of the following ways:

```
throw(exception) ;
throw exception ;
```

The object "exception" may be of any type or a constant. We can also throw an object not intended for error handling.

Catching Mechanism:

The catch block must immediately follow the try block that throws the exception. Code for handling exceptions is included in catch blocks. A catch block looks like a function definition and is of the form:

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

The “type” indicates the type of exception that **catch** block handles. The parameter arg is an optional parameter name. The exception_handling code is placed between two braces. The catch statement catches an exception whose type matches with the type of catch argument. When it is caught, the code in the catch block is executed.

The figure below shows the exception handling mechanism, here try block invokes the function that contains the throw statement.

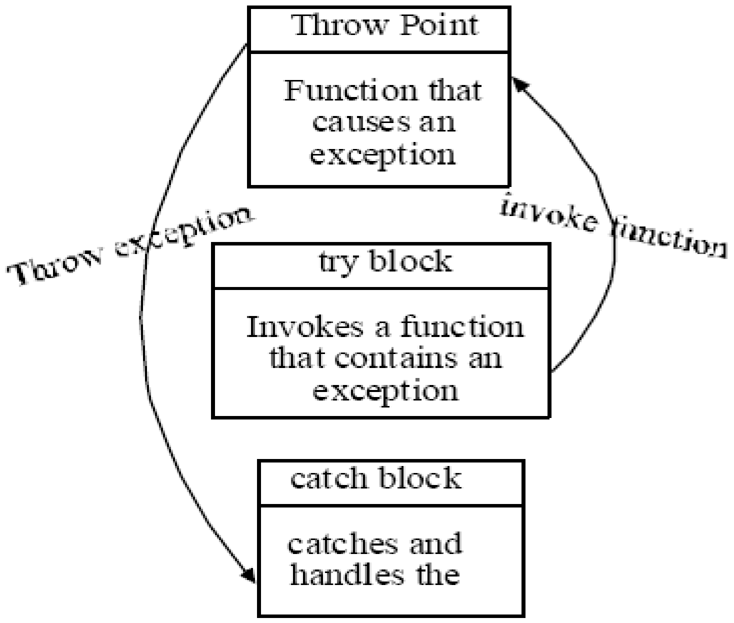


Fig : Function invoked by try block throwing exception

When the **try** block throws an exception, the program control leaves the **try** block and enters the **catch** statement of the **catch** block. Exceptions are objects, which are used to transmit information about a problem. If the type of object *thrown* matches the *arg type* in the **catch** statement, then catch block is executed for handling the execution. If they do not match, the program is aborted with the help of *abort()* function which is invoked by default.
When no exception is detected and thrown, catch block is skipped.

Example 1: Try block throwing exception

```
#include <iostream>
using namespace std;
int main()
{
    float a, b;
    cout<<"Enter values of a & b:\n";
    cin>>a>>b;
try
{
            if(b == 0)
                        throw b;
             else
                        cout<<"Result = "<<a/b;
}
catch(float)
   {
        cout<<"Divide by zero exception:= "<<endl;
   }
  return 0;
}
```

Example2: Function invoked by try block throwing exception

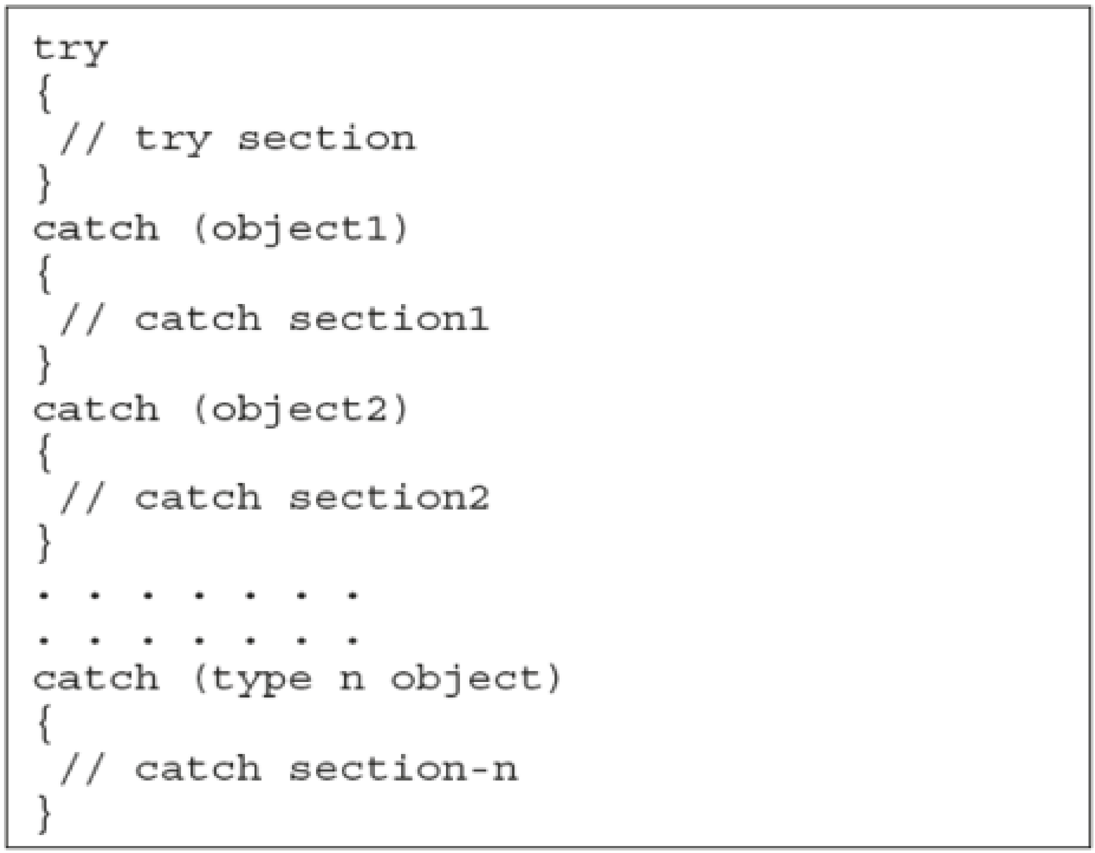
```
#include <iostream>
using namespace std;
void divide(int a, int b)
{
        if(b == 0)
                throw b;
        else
                cout<<"Result = "<<(float)a/b;
}
int main()
{
        int a, b;
        cout<<"Enter values of a & b:\n";
        cin>>a>>b;
        try
        {
                divide(a, b);           //invokes a function.
        }
        catch(int i)
        {
                cout<<"Divide by zero exception: b = "<<i;
        }
        return 0;
}
```

In the first example, if exception occurs in the try block, it is thrown and the program control leaves from the try block and enters the catch block. In the second example, try block invokes the function divide(). If exception occurs in this function, it is thrown and control leaves this function and enters the catch block.

Multiple catch statements

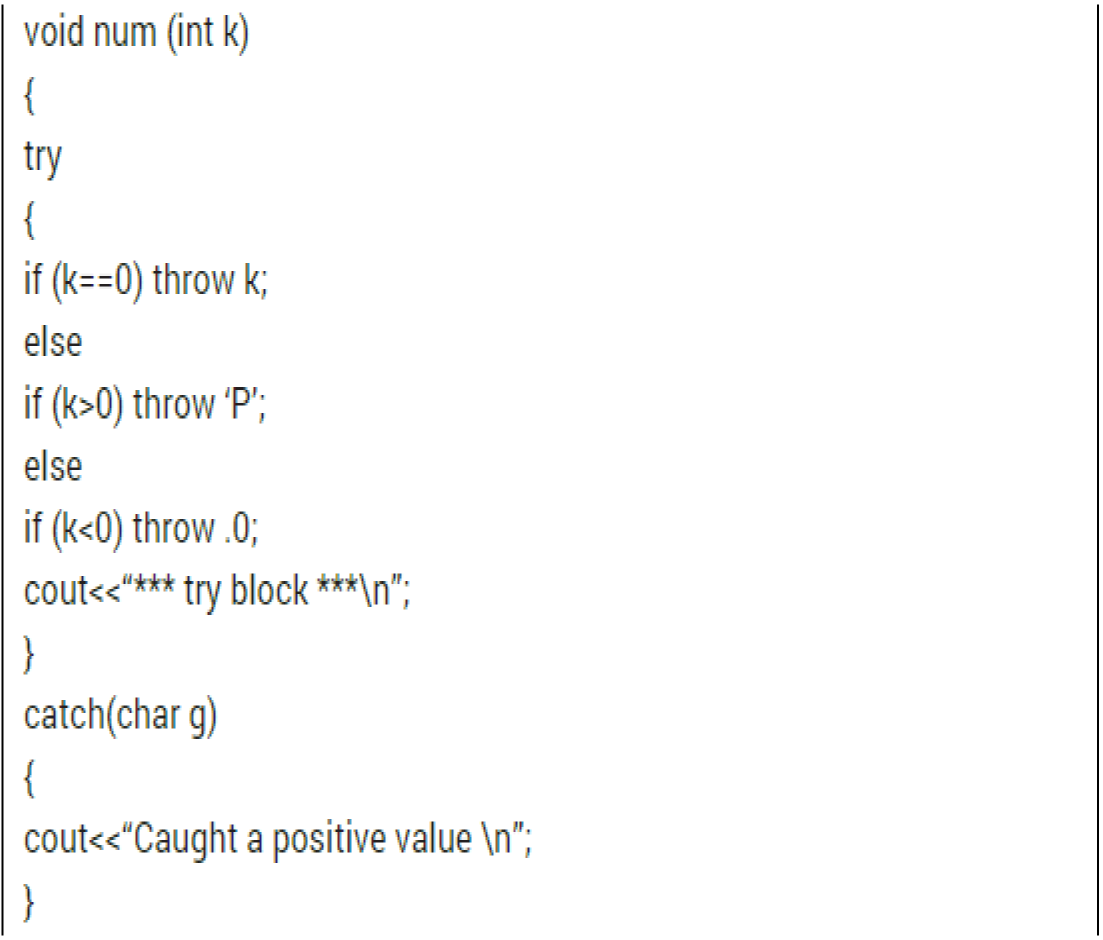
As soon as an exception is thrown, the compiler searches for an appropriate matching catch block. The matching catch block is executed, and control passes to the successive statement after the last catch block. In case no match is found, the program is terminated. In a multiplecatch statement, if objects of many catch statements are similar to the type of an exception, in such a situation, the first catch block that matches is executed.

We can also define multiple catch blocks; in the try block, such programs also contain multiple throw statements based on certain conditions. The format of multiple catch statements is as follows:

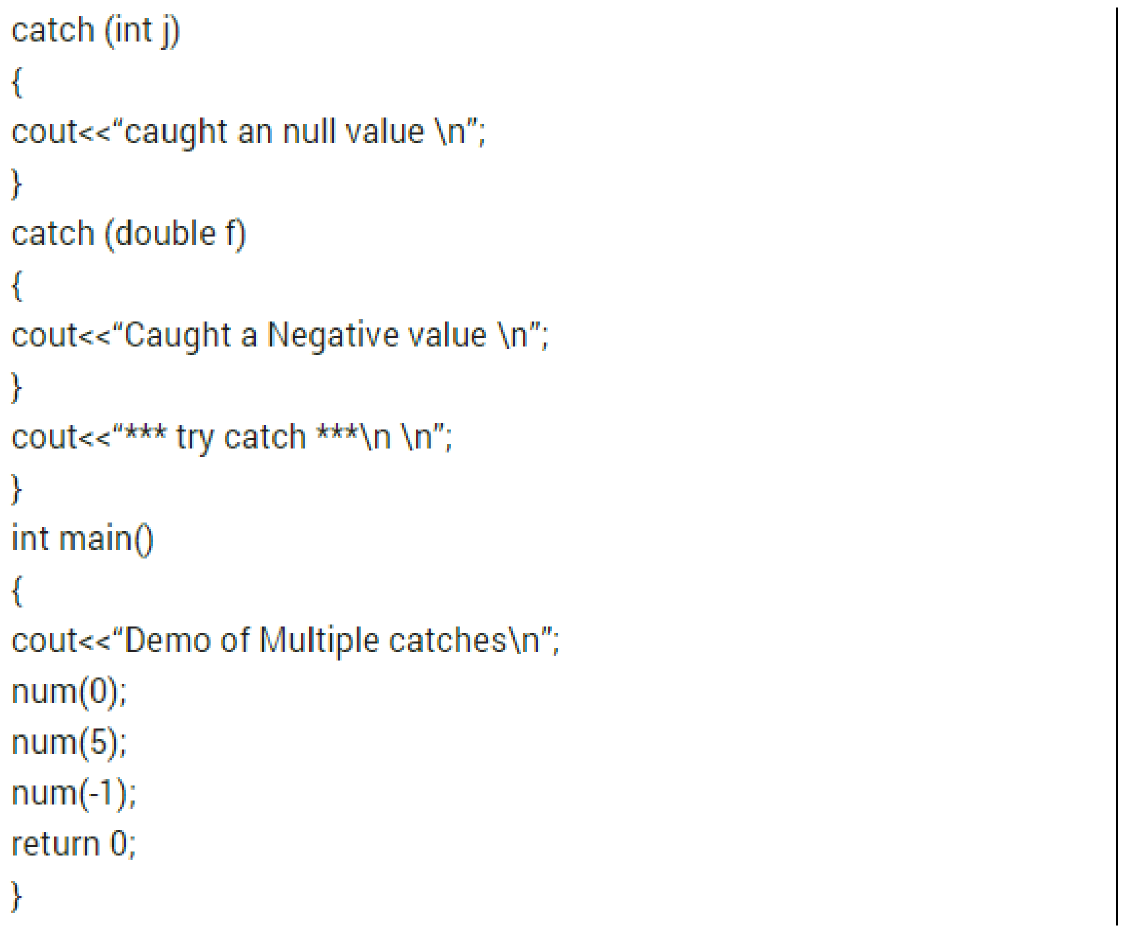


```
try
{
 // try section
}
catch (object1)
{
 // catch section1
}
catch (object2)
{
 // catch section2
}
. . . . . . .
. . . . . . .
catch (type n object)
{
 // catch section-n
}
```

For example:



```
void num (int k)
{
try
{
if (k==0) throw k;
else
if (k>0) throw ‘P’;
else
if (k<0) throw .0;
cout<<“*** try block ***\n”;
}
catch(char g)
{
cout<<“Caught a positive value \n”;
}
```



```
catch (int j)
{
cout<<"caught an null value \n";
}
catch (double f)
{
cout<<"Caught a Negative value \n";
}
cout<<"*** try catch ***\n \n";
}
int main()
{
cout<<"Demo of Multiple catches\n";
num(0);
num(5);
num(-1);
return 0;
}
```

Catching all exceptions

We can also define a catch block that captures all the exceptions independently of the type used in the call to throw. For that we have to write three points instead of the parameter type and name accepted by catch.

```
try {

……}

catch (…)

{

cout<<"Exception occurred";}
```