Exceptions and Exception Handling in C++

ANURAG SINGH

Dr.APJ Abdul Kalam Technical University Uttar Pradesh
Department of Computer Science and engineering, IEC College of Engineering & Technology, Uttar Pradesh, India.

---***---

***Abstract** - Exception handling is the process of responding to the occurrence, during computation, of exceptions or exceptional conditions requiring special processing .An exception is a problem that arises during the execution of the program. A C++ exceptions a type of response for an exceptional situation that arises while a program is running, in such types of situation as an attempt to divide by zero. The errors occurred in program may be logical errors or syntactic errors. The logical errors remains in the program due to an unsatisfactory understanding of the program. The goal of exception handling is to create a routine that detect and sends an exceptional condition in order to execute suitable actions.*

Key Words*: *Exceptions, try, catch, throw, re-throwing.

Introduction:

Exceptions are errors that occur at run time. They are caused by a wide variety of exceptional circumstance, such as running out of memory, not being able to open a file, trying to initialize an object to an impossible value, or using an out - of - bounds index to a vector. An exception is an error or an expected event. The exception handler is a set of codes that executes when an exception occurs. Exception handling is one of the most recently added features in C++.
Exception handling in C++ provides a better method by which the caller of a function can be informed that some error condition has occurred. The following keywords are used for error handling in C++.

- Try
- Catch
- Throw

An exception also known as run time errors because these are occurs at runtime. For example there may be an abnormal condition or unexpected behavior when we try to divide a number by zero or an array is accessed outside of it's bounds, or when required memory is not available, etc.

1.1 Program for Demo of Exception:

```
#include<iostream.h>
void main()
{
        Int a, b, x;
        cout<<"Enter two numbers\n";
         cin>>a>>b;
        x=a/b;
        cout<<a<<"/"<<b<<"="<<x;
}
```

Output 1

```
Enter two numbers
20
5
20/5 = 4
```

Output 2

```
Enter two numbers
13
0
```

Not produce any output but will terminate abnormally and produce an exception divide- error.

2. Process:

Sometimes the application makes a mistake, causing an error to be detected in a member function. This member function then informs the application that an error has occurred. When exceptions are used, this is called throwing an exception. In this application we install a separate section of code to handle the error. This code is called an exception handler or catch block, It catches the exceptions thrown by the number function. Any code in application that uses object of the class is enclosed in a try block. Error generate in the block will be caught in the catch block.

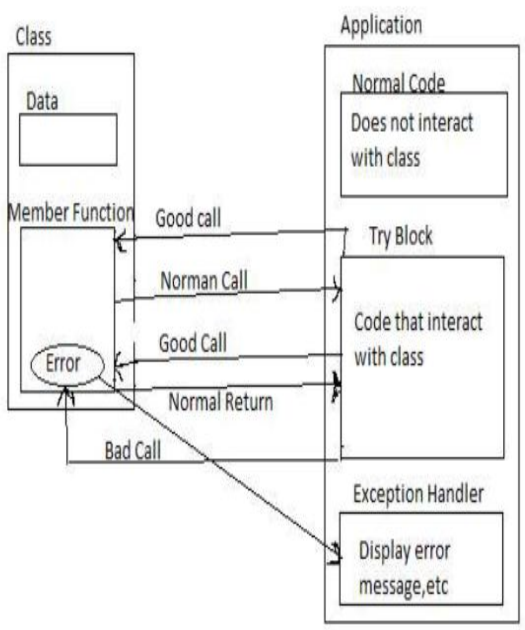

Fig. The Exception mechanism.

The general for of the try and catch block is as follow:
Try mechanism:

```
try
{
      …………
      …………
throw exception ;  // block of statements
}// which detect and throws an exception.
catch(arg) // catches exception
{
      ………// block of statement
      ……… // handle the exception
}
```

When the try block detect an exception that throws it then the control is transferred from the try block to the catch block. When n exception is detected and thrown then the control goes to the statements immediately after the catch block i.e. the catch block is skipped.
The exception that is thrown by catch block is an object which gives information about exception. The type of object thrown must match to the type of argument arg inside the catch parenthesis otherwise catch block does not execute.

2.1 Throwing mechanism:

When an exception that is desired to be handled to be detected. It is throw using the throw statement in one of the following forms.

```
throw(exception);

throw ;  //used for re-throwing an exception.
```

The operand object exception may be of any type, including constants. It is also possible to throw objects not intended for error handling. When an exception is throw, It will be caught by the catch statement associated with the try block. That is the control exits the current try block, and is transferred to the catch block after that try block. Throw point can be in deeply nested scope with a try block or in a deeply nested function call. In any case, control is transferred to the catch statement.

2.1.1 Program for throw exception

```
#include<iostream.h>

#include<conio.h>

Void divide()
{
   Int  a, b, x;
   cout<<"Enter two numbers\n";
   cin>>a>>b;
   if(b==0)
   {
        throw(a);
        }
        Else
        {
        x = a/b;
        cout<<a<<"/"<<b<<"=<<x";
        }
  }
Void main()
{
try
{
        divide()
        }
        catch(int  i)
       {
cout<<"Divide by zero";
        }
}
```

Output 1

```
Enter two numbers
16
4
16/4= 4
```

Output 2

```
Enter two numbers
17
0
Divide by zero error.
```

2.2 Catch mechanism:

The catch statement catches an exception whose type match with the type of catch argument. When it is caught, The code in the catch block executed.
Code for handling exception is included in catch blocks, A catch block looks a function definition and is of the form

```
catch(arg)
{
// statement for
// managing exceptions
}
```

The type indicates the type of exception that catch block handles. The parameter arg is an optional parameter name.
If the parameter in the catch statement is named, then the parameter can be used in the exception handling code. After executing the handler the control goes to the statement immediately following catch block. Due to mismatch if any exception is not caught, abnormal program termination will occur.

2.2.1 Multiple catch statements:

It is possible that a program can generate more than one type of exceptions. In such cases, we can have more than one catch statement with a single try block.

```
    try
    {
    // try block
    }
    catch(type1 arg)
    {
        // catch block 1
     }
catch(type2 arg)
{
        // catch block 2
}
……………………….
……………………….
catch(typeN arg)
{
        // catch block N
}
```

It is possible that arguments of several catch statements match the type of an exception, In such cases, The first handler that matches the exception type is executed. The type of argument inside the parenthesis of catch block indicates the type of exception that catch block handles and argument is an optional parameter name.

2.2.2 Program for multiple catch block:

```
#include< iostream.h >
void test()
{
  try
  {
If(x= =0)
            throw x;
//throwing an int  type of exception.
else if(x= =1)
            throw 'x';
// throwing a char  type of exception.
else if(x= = -1)
            throw 1.0;
// throwing a double type of exception.
}
catch(int  i)
{
Cout<<"Character type of exception caught\n";
}
catch(int  i)
{
cout<<"Integer type of exception caught\n";
}
catch(double i)
{
cout<<"Double type of exception caught\n";
}
void main()
{
Int  a;
cout<<"Enter a number\n";
cin>>a;
test(a);
}
```

Output 1

```
Enter a number
1
Character type of exception caught.
End of multiple try – catch block.
```

Output 2

```
Enter a number
0
Integer type of exception caught.
End of multiple try – catch block.
```

Output 3

```
Enter a number
-1
Double type of exception caught.
End of multiple try – catch block.
```

Output 4

```
Enter a number
2
End of multiple try – catch block.
```

3. Re-throwing an exception:

A catch block itself may detect and throw an exception. When an exception is thrown in the catch block then this is known as re-throwing an exception. To re-throw an exception we may simple use throw without any arguments.
throw;

The above statement will re-throw the current exception and it will not be caught by the same catch in the parallel try-catch block. Rather it will be caught by an appropriate catch of the outer try-catch block only.

3.1 Program for Re-throwing an exception:

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

int  a, b;
cout<<"Enter two numbers\n";
cin>>a>>b;
try
{
        Try
        {
                If(b= = 0)
                throw b;
//throwing int  type of exception
                else
cout<<a<<"/"<<b<<"="<<a/b<<endl\n;
        }
catch(int)
{
cout<<"Integer type exception caught \n";
throw;
// re-throwing int type of exception
}
}
catch(int)
{
Cout<<"Integer type exception caught again\n";
}
}
```

Output 1

```
Enter two numbers
15
3
15/3=5
```

Output 2

```
Enter two numbers
15
0
Integer type exception caught
Integer type exception caught again.
```

4. Specifying exception:

It is possible to restrict a function to throw only specified type of exceptions. This is done by the following syntax:

```
return_typefunction_name(argument
list)throw(exception_type_list)
{
……….
……… //function body
}
```

The exception_type_list specify the type of exceptions that can be thrown by the functions. Throwing any other type of exception will cause abnormal program termination.
A function can only be restricted in what types of exceptions it throws back to try block that called it and not within a function.

5. References:

1. Advanced Topics in Exception handling techniques "ISBN 3540374435".
2. Exception Handling Resumption v/s termination.
3. Optimizing away C++ exception handling bye Schilling Jonathan L.
4. Lecture notes by Praveen Kumar from Incapp Infotech Pvt. Ltd.
5. Uncaught exception bye "Mac Developer Library".