Exception Handling

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

We know that it is very rare that a program work correctly first time. It might have bugs. The two most common types of bugs are logic errors & syntactic errors.

Logic error occurs due of poor understanding of the problem and solution procedure.

Syntactic error arise due to poor understanding of the language itself.

We can detect these errors by using debugging and testing procedures.

Some times some peculiar problems arise other than logic or syntax errors. They are known as exceptions.

What are exceptions:

- Exception are run time anomalies or unusual conditions that a program may encounter while excetuing.
- Anomalies conditions i.e. division by zero, access to an array outside of its bounds or running out of memory or disk space.

- When a program encounters an exception condition, it is important that it is identified and deal with properly.

- Note:
- Exception Handling are not the part of the original C++. It is a new feature added to ANSI C++. Today almost all compilers support this fetaure.
- C++ exception handling provides type-saf, integrated approach for coping with the unusual predictable problems that aries while executing a program.

Types of Exceptions:

- Exception are of two kinds:
- 1) Synchronous Exceptions
- 2) Asynchronous Exceptions

- Errors such as “Out-of-range index” and “Overflow” belong to the Synchronous type exceptions.

- The errors that are caused by events beyond the control of the program (such as keyboard interrupts) are called Asynchronous Exceptions.

Purpose of Error Handling :

- The purpose of error handling mechanism is to detect & report an “exceptional circumstances” , so that appropriate action can takes place.

- error handling mechanism do the foollowing tasks:
- 1) Find the problem. (Hit the exception)
- 2) Inform that problem has encountered (Throw the exception)
- 3) Receive the error information (Catch the Exception)
- 4) Take Action (Handle the Exception)

Exception Handling Mechanism:

- C++ exception handling mechamnism basically built upon three keywords (try, throw and catch).

- Try keyword is used to contains a block of statements that may generate the exceptions. This block of statements is called Try block.
- When an exception is detected, it is thrown using a throw atatenment in the try block.
- A catch block defined by the catch keyword. It catched the exception which is thrown by the throw statement from the Try block and handles it appropriately .

-

General form of Exception Handling:

```
……
……
Try
{
…….
Throw exception ;
…….
…….
}
Catch (type arg)
{
……
……
}
……
……
```

Flow of Exception Handling Mechanism:

- When the try block throws an exception, the progran control leaves the try block and enters the catch block.
- Note that exceptions are basically objects which is used to transmit informationabout a problem. If the type of object thrown matched the argument in the catch statement, then catch block is excuted for ha dling the xeception. If they do not amy=tch, the problem is aborted with the help of abort() function which is invoked bydefault.

- When no exception is detected and thrown, then the control goes to the statement immediately after the catch block means catch block is skipped.

Program:

```
int main()
{
   int a , b ;
   cout << "Enter the value of a & b \n";
   cin>>a>>b;

   int x = a-b;
   try
   {
   if(x != 0)
      {
       cout << "Result (a/x)" << a/x <<endl;
      }
      else
         {
            throw (x);
         }
   }
```

```
catch (int i)
   {
      cout<< "Exception caught : Divide By Zero
     "<< endl;
   }
}
```

Output:

Enter the value of a & b :
20 15
Result 4

Enter the value of a & b :
10 10
Exception caught : Divide By Zero

General format of Function invoked by Try Block throwing Exception:

```
type function (arg list)
{
.....
.....
throw (object);
.....
.....
}
.....
.....
try
{
.....
.....
}
```

```
catch (arg list)
{
.....
.....
}
```

Program:

```
Void divide ( int x , int y , int z )
{
if ( (x-y) != 0)
{
Int result = z/ (x-y);
Cout << " Result is << result<<endl;
}
Else
{
Throw (x-y);
}
}
int main()
{
Try
{
Divide(10 , 20 , 30 );
Divide(10 , 10 , 20 );
}
 catch ( int i)
{
cout <<" Caught an exception \n" ;
}
}
```

Output:

Result is -3
Caught an Exception

Throwing Mechanism :

When an exception is detected, it is thrown using the throw statement in one of the following forms:

```
throw (exception );
throw exception;
throw ;          //  Used for rethrowing an Exception
```

Catching Mechanism :

An error is handled in catch block. A catch block looks like a function definition and is of the form :

```
Catch ( type arg )
{
.....
...../     Statement for managing exceptions
.....
}
```

Multiple Catch Statements :

It is possible that a program segment has more than one condition to throw an exception. In such cases, we can associate more than one catch statement with a Try block (Like the conditions in a Switch statements) as shown below :

```
try
{
.....
}
catch (type1 arg )
{
.....
}
catch (type2 arg )
{
.....
}
.
.
catch (typeN arg )
{
.....
}
```

Program:

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

      catch(char c)
      {
         cout<<"Caught a character";
      }
```

```
      catch(int a)
            {
               cout<<"Caught an integer";
            }
      catch(double d)
            {
               cout<<"Caught a double";
            }
      }

      int main()
            {
               int x;
               cout<<"Enter the value of x: \n";
               cin>>x;
               test (x);
            }
```

Output:

Enter the value of x:
1
Caught an integer

Catch All Exceptions :

Sometimes we may not able to define all possible types of exceptions & therefor may not be able to design indpendent catch handlers to catch them. In such circumstances to catch all exceptions instead of a certain type alone we use following catch statement .

Syntax :

```
Catch ( … )
{
…..
…..//     Statement for managing all types of exceptions
…..
}
```

Note : Catch(…) should be placed last in the list of hadlers.

Program:

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

       catch(...)
       {
          cout<<"Caught an exception";
       }

    }
```

```
int main()
    {
       int x;

       cout<<"Enter the value of x: \n";
       cin>>x;

       test (x);

       return 0;
    }
```

Output:

Enter the value of x:
1
Caught an Exception

Rethrowing an Exception :

Sometimes user may decide to rethrow the exception caught without procrssing it. In such situations, we may simply invoke throw without any arguments as shown below:

Syntax : 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 that enclosing try block.

Program:

```
void test(int x)
    {
       try
       {
          if (x == 1)
           throw x;
        if (x == 0)
           throw 'x';
        if (x == -1)
           throw 1.0;
       }
       catch(int i)
       {
          cout<<"Caught an int"<<endl;
          throw;
       }
       catch(char j)
       {
          cout<<"Caught a char";
       }
       catch(double d)
          {
             cout<<"Caught double";
          }
       }
       int main()
       {
          int x;

          try
          {
            cout<<"Enter the value of x: \n";
            cin>>x;
           test (x);
          }
          catch (int)
          {
             cout<< "Caught an int in main" <<endl;
        }
}
```

Output:

Enter the value of x:
1
Caught an int
Caught an int in main

Specifying Exceptions :

It is possible to restrict a function to throw only certain specified exceptions. This is achieved by adding a throw list clause to the function definition .

Syntax : type function (arg - list) throw (type – list)

```
{
        .....
        .....
}
```

The type-list specifies the type of exceptions that maybe thrown.

Throwing any other type of exception will cause abnormal program termintion.

If we wish to prevent function from throwing any exception , we may do it by making the type-list empty.

Syntax : throw () ; // Empty List

Program:

```
void test(int x) throw ( int ,double)
    {
        if (x == 1)
         throw x;
      if (x == 0)
         throw 'x';
      if (x == -1)
         throw 1.0;
     }

    int main()
    {
       int x;
     try
     {
       test (0);
       test (1);
       test(-1);
     }
    catch(int i)
        {
          cout<<"Caught an int";
        }
        catch(char j)
        {
          cout<<"Caught a char";
        }
        catch(double d)
        {
          cout<<"Caught double";
        }
}
```

Output:

Terminate called after throwing an instance of 'char'