**Exception Handling in c++**

By Dinesh Thakur

The errors that occur at run-time are known as exceptions. They occur due to different conditions such as division by zero, accessing an element out of bounds of an array, unable to open a file, running out of [memory](https://ecomputernotes.com/fundamental/input-output-and-memory/memory) and many more. All these errors are synchronous exceptions. There is another type of exception known as an asynchronous exception that deals with errors associated with events like disk I/O completion, mouse click, network message arrivals etc. For handling such exceptional situations, an error handling mechanism known as Exception Handling is provided. In C++, exception handling is designed to handle only synchronized exceptions.

Exception handling was not initially the integral part of C++ language and was added later on. The main objective of exception handling is to provide a way to detect and report the exception condition so that necessary action can be taken without troubling the user.

**We’ll be covering the following topics in this tutorial:**

* Exception Handling Mechanism
* [Multiple Catch Blocks](https://ecomputernotes.com/cpp/exception-handling-in-cpp#Multiple_Catch_Blocks)
* [Catch All Exception](https://ecomputernotes.com/cpp/exception-handling-in-cpp#Catch_All_Exception)

**Exception Handling Mechanism**

Whenever an exception occurs in a C++ program, the portion the program that detects the exception can inform that exception has occurred by throwing it On throwing an exception, the program control immediately stops the step by step execution of the code and jumps to the separate block of code known as an exception handler. The exception handler catches the exception and processes it without troubling the user. However, if there is no exception handler, the program terminates abnormally.  
C++ provides three constructs try, throw and catch, for implementing exception handling. To handle an exception in your program, you need to create a try-catch block. In the try block, we define the statement(s) that needs to be tested for an exception(s), and if an exception occurs, we use the throw statement to invoke an exception handler.  
The exception handler in C++ is a catch block which catches the exception and processes it. The general form of the try-catch block for handling an exception is as follow :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | try {    //testing condition    throw excp;  }  catch (datatype arg) {    //code for handling exception  } |

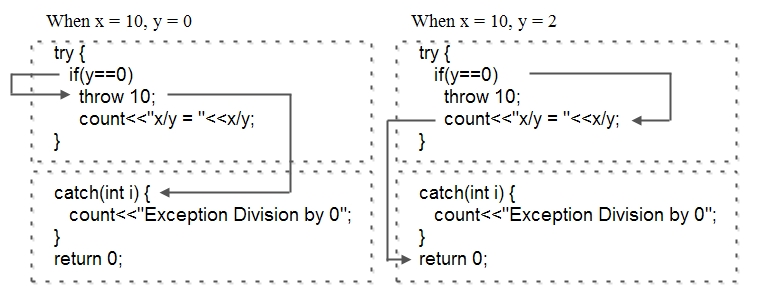
The try block starts with a keyword try followed by a set of statements enclosed in curly braces. The statements in the try block might cause exceptions at run time. On detecting an exception, it is thrown using a throw statement in the try block. The throw is a keyword in C++ and is the only way to throw an exception, and excp represents the type of value to be thrown.

The try block is immediately followed by an exception handler called a catch block. The catch block starts with a keyword catch followed by a set of statements enclosed in curly braces. The catch header specifies in parentheses an exception parameter arg along with its datatype. If the type of value (excp) thrown matches the arg’s [data type](https://ecomputernotes.com/java/data-type-variable-and-array/explain-data-types-in-java), the catch block is executed for processing the exception. Otherwise, the program might terminate abnormally. If no exception occurs in the try block, the program skips the catch block and continues with the next statement following the try-catch block.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #include<iostream.h>  int main() {   int x,y;   cout<<"Enter numerator (x) and denominator (y) =";   cin>>x>>y;   try {     if(y==0)// detects an exception      throw 10;// throw an exception      cout<<"x/y = "<<x/y;      catch(int i)//catches the exception      {        cout<<"Exception Division by 0 not allowed";      }      return 0;  } |

**Explanation**: The program’s purpose is to display the result of the division of two numbers and check for division by 0 exceptions. The program on execution first prompts the user to input numerator x and denominator y, both of type int. In the try block, we test the value of the denominator y. If 0, an exception is detected and thrown using the throw statement with an integer value of 10. The catch block defined in the program serves as an exception handler for divide by 0 error. It catches the exception as the type of value is thrown matches the parameter type of catch statement. In the catch block, we display the appropriate message and gracefully exit from the program.

If the denominator’s value is non zero, then no exception occurs, and division is performed as desired, which is then displayed. After this, the program ignores the catch block and continues with the statement following the catch block (if any).

**Note**: The type of value thrown in the exception must match the parameter type in the catch statement. Otherwise, an exception will not be caught, and an internal function terminate() will be called, which in turns call abort() and results in abnormal program termination.

**Multiple Catch Blocks**

In the preceding section, We have only one catch block to handle an exception. However, different type of exceptions can exist in a program. In order to handle multiple exceptions of different types, we need to define multiple catch blocks associated with a single try block. But how will one come to known that which catch block will be executed for handling an exception. This is determined by the type of the value thrown from the try block. The general form for handling multiple catch blocks is as follows :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | try {    //testing condition    thrown excp1;    //testing condition    throw excp2;    //testing condition    throw excp3;  };  catch (datatype1 arg1) {    //code for handling exception  }  catch (datatype2 arg2) {   //code for handling exception  }  catch(datatype3 arg3) {   //code for handling exception  } |

Each of the multiple catch blocks defined above takes parameters of different data types and each must match a different type of exception. When an exception is thrown from the statement within the try block, the multiple catch blocks that follow the try block is searched in the order they appear in the program. If the type of value thrown matches with the parameter type of a catch statement then the corresponding catch block is selected to handle the exception. Once the matched catch block is found, all other catch blocks are ignored and after executing the selected catch block, the program continues with the next statement following the last catch block.  
If none of the catch blocks are matched, the thrown exception will not be processed and it results is abnormal program tennination.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include<isotream.h>  #include<math.h>  int main() {    int num;    double res;    cout<<"Enter a number :"    cin>>num;    try {     if (num>0xffff)      throw 10;     if(num<0)      throw 'E';      cout<<"Square root of"<< num <<"is="<< sqrt(num);  }  catch(int) {    cout<<"Exception out of range \n";  }  catch (char) {    cout<<"Exception:Square root of negative number doesn't exist";  }  return 0;  } |

**Explanation**: The purpose of this program is to calculate the square root of a given number. Here, we deal with two types of exceptions. The first exception occurs when a number entered by the user is out of range. The second exception occurs when the user enters a negative number. The program contains two catch blocks, one for the exception of type int

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | catch (int i) {  ............  }  and others for the exception of type char  catch(char c) {  ...........  } |

When the user inputs any number out of range, then an exception is detected, and we throw the exception with a value of int type.  
Similarly, when a user inputs a negative number, then again exception is detected, and we throw the exception with a value of char type.

**Catch All Exception**

In the previous section, we provided separate catch blocks to handle different type of exceptions. However, it is not possible to anticipate all the exceptions that might be thrown in a program. It may result in an abnormal termination of the program.

To avoid such situations, C++ supports a feature that catches all the exceptions thrown except those for which separate catch blocks are defined. To do this, we use the following form of catch

|  |  |
| --- | --- |
| 1  2  3 | catch (...) {    //unhandled exceptions  } |

Here, the sequence of three dots in a catch are referred to as an ellipsis that can handle any type of data. The catch (…) must always be placed in the last of exception handlers. If an exception is not handled by any of the catch blocks then the catch (…) { } block is executed.  
A catch (…) can be used along with other catch blocks or it can be specified alone. In the later case, all the exceptions raised in the try block are handled only by the catch (…) block.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #include<iostream.h>  #include<conio.h>  int main() {   int age;   cout<<"Enter age for voting(l8 to 120) ";   cin>>age;   try {    if(age>0 & age<l8)      throw 0;    else if(age>l20)      throw 'v';    else if (age<0)      throw 2.8;    cout<<"Eligible for voting";  }  catch(int i) {    cout<<"Exception Valid age but not eligible for voting";  }  catch( ...)// To catch all unhandled exception  {  cout<<"Exception.: Invalid age for voting";  }  }  Output :  Enter age for voting (18 to 12) : 25  Eligible age for voting |

**Explanation** : The above program checks whether a person is eligible for voting not. We assume that a person is eligible for voting if his age lies between 18 and 120.

On executing the program, if we input the age between 0 and 18, then the program throws a int value 0 as an exception and this exception is handled by a catch block that takes int as a input parameter.

The output of the program is as follows

|  |  |
| --- | --- |
| 1  2 | Enter age for voting (18 to 120) : 5  Exception : Valid age but not eligible for voting |

However, if the user inputs the age to be negative or greater than 120 then the exception thrown in the try block are handled by the last catch block that takes three dots (…) as an input parameter. The output of the program will be as follows.

|  |  |
| --- | --- |
| 1  2 | Enter age for voting (18 to 120) :-5  Exception : Invalid age for voting |