**Exception Handling in C++**

Errors can be broadly categorized into two types. We will discuss them one by one.

* Compile Time Errors
* Run Time Errors
* **Compile Time Errors –** Errors caught during compiled time is called Compile time errors. Compile time errors include library reference, syntax error or incorrect class import.
* **Run Time Errors -** They are also known as exceptions. An exception caught during run time creates serious issues.

Errors hinder normal execution of program. Exception handling is the process of handling errors and exceptions in such a way that they do not hinder normal execution of the system. For example, User divides a number by zero, this will compile successfully but an exception or run time error will occur due to which our applications will be crashed. In order to avoid this we'll introduce exception handling techniques in our code.

Error handling is done using three keywords -

* try
* catch
* throw

**try**

**{**

**//code**

**throw parameter;**

**}**

**catch(exceptionname ex)**

**{**

**//code to handle exception**

**}**

* **try block**

The code which can throw any exception is kept inside(or enclosed in) a try block. Then, when the code will lead to any error, that error/exception will get caught inside the catch block.

* **catch block**

catch block is intended to catch the error and handle the exception condition. We can have multiple catch blocks to handle different types of exception and perform different actions when the exceptions occur. For example, we can display descriptive messages to explain why any particular exception occurred.

* **throw statement**

It is used to throw exceptions to exception handler i.e. it is used to communicate information about error. A throw expression accepts one parameter and that parameter are passed to handler. throw statement is used when we explicitly want an exception to occur, then we can use throw statement to throw or generate that exception.

**Understanding Need of Exception Handling -**

**#include <iostream.h>**

**#include<conio.h>**

**int main()**

**{**

**int a=10,b=0,c;**

**c=a/b;**

**return 0;**

**}**

**Using try, catch and throw Statement -**

**#include <iostream.h>**

**#include<conio.h>**

**int main()**

**{**

**int a=10, b=0, c;**

**// try block activates exception handling**

**try**

**{**

**if(b == 0)**

**{**

**// throw custom exception**

**throw "Division by zero not possible";**

**c = a/b;**

**}**

**}**

**catch(char\* ex) // catches exception**

**{**

**cout<<ex;**

**}**

**return 0;**

**}**

**Using Multiple catch blocks**

**#include <iostream.h>**

**#include<conio.h>**

**int main()**

**{**

**int x[3] = {-1,2};**

**for(int i=0; i<2; i++)**

**{**

**int ex = x[i];**

**try**

**{**

**if (ex > 0)**

**// throwing numeric value as exception**

**throw ex;**

**else**

**// throwing a character as exception**

**throw 'ex';**

**}**

**catch (int ex) // to catch numeric exceptions**

**{**

**cout << "Integer exception\n";**

**}**

**catch (char ex) // to catch character/string exceptions**

**{**

**cout << "Character exception\n";**

**}**

**}**

**}**

**Generalized catch block in C++ -**

Below program contains a generalized catch block to catch any uncaught errors/exceptions. catch(...) block takes care of all type of exceptions.

**#include <iostream.h>**

**#include<conio.h>**

**int main()**

**{**

**int x[3] = {-1,2};**

**for(int i=0; i<2; i++)**

**{**

**int ex=x[i];**

**try**

**{**

**if (ex > 0)**

**throw ex;**

**else**

**throw 'ex';**

**}**

**// generalised catch block**

**catch (...)**

**{**

**cout << "Special exception\n";**

**}**

**}**

**return 0;**

**}**

**Standard Exceptions in C++ -**

There are some standard exceptions in C++ under <exception> which we can use in our programs. They are arranged in a parent-child class hierarchy which is depicted below:

* **std::exception** - Parent class of all the standard C++ exceptions.
* **logic\_error** - Exception happens in the internal logical of a program.
* **domain\_error** - Exception due to use of invalid domain.
* **invalid argument** - Exception due to invalid argument.
* **out\_of\_range** - Exception due to out of range i.e. size requirement exceeds allocation.
* **length\_error** - Exception due to length error.
* **runtime\_error** - Exception happens during runtime.
* **range\_error** - Exception due to range errors in internal computations.
* **overflow\_error** - Exception due to arithmetic overflow errors.
* **underflow\_error** - Exception due to arithmetic underflow errors
* **bad\_alloc** - Exception happens when memory allocation with new() fails.
* **bad\_cast** - Exception happens when dynamic cast fails.
* **bad\_exception** - Exception is specially designed to be listed in the dynamic-exception-specifier.
* **bad\_typeid** - Exception thrown by typeid.