Exception handling (Exception Handling) in C ++

An Exception is an issue that occurs during the execution of a program. An Exception in C ++ is a response to an exception situation that occurs while a program is running, such as dividing by zero.

An **Exception** is an issue that occurs during the execution of a program. An Exception in C ++ is a response to an exception situation that occurs while a program is running, such as dividing by zero.

Exception provides a way to pass control from one part of a program to another. Exception Handling in C ++ is built on 3 keywords: **try, catch, and throw** .

1. **throw** : A program throws an Exception when a problem occurs. This is done using the **throw** keyword in C ++.
2. **catch** : A program catches an Exception with an Exception Handler in place in a program where you want to handle that problem. C ++ **catch** keyword leads to catching an exception.
3. **try** : A **try** block can be captured by a specific number of exceptions. It is followed by one or more catch blocks.

Suppose a block will create an Exeption, a method of catching an exception by using a combination of try and catch keywords. A try / catch block is placed around the code that can create an exception. The code inside a try / catch block is treated as protected code, and the syntax to use try / catch in C ++ is as follows:

```
try { // phan code duoc bao ve } catch (  ten_Exception e1  ) { // day la khoi
```

You can list many **catch** commands to **catch** different exception types in case your **try** block appears more than one exception in different situations.

Throw Exception in C ++

Exception can be thrown anywhere within a code block using C ++ throw commands. The command of the throw command determines the type of exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

The following example illustrates throwing an exception when divided by zero in C ++:

```
double  phepchia ( int  a , int  b ) { if (  b  == 0 ) { throw "Chu y: Ban dang
```

Getting Exception in C ++

The **catch** block following the try block in C ++ will catch any exception. You can determine the type of exception you want to catch and this is determined by the exception declaration that appears in parentheses

following the catch keyword in C ++.

```
try { // phan code duoc bao ve } catch (  ten_Exception e  ) { // phan code de x
```

The above code will catch an exception of type **ten_Exception** . If you want to determine that a catch candle block handles any type of exception thrown in a try block, you must place a dot (.) in parentheses following the catch keyword, as follows:

```
try { // phan code duoc bao ve } catch (.) { // phan code de xu ly bat ky kieu n
```

The following example throws an exception when divided by zero and we catch it in the catch block.

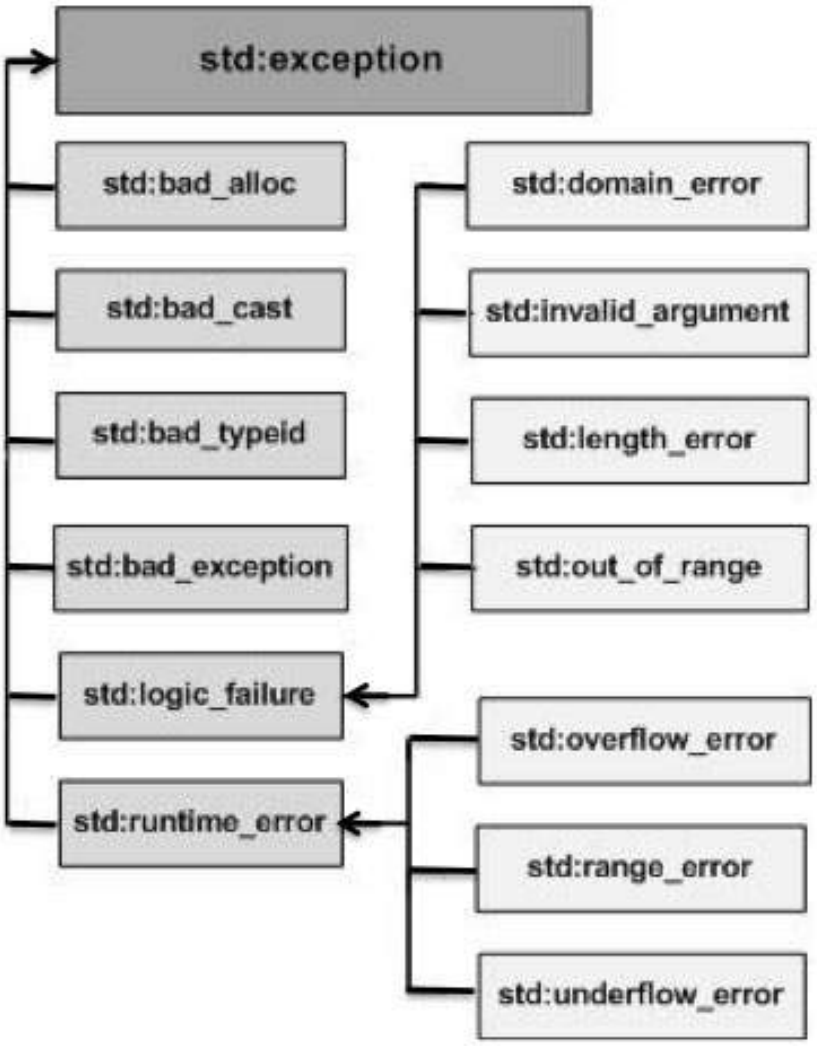
```
#include using namespace  std ; double  phepchia ( int  a , int  b ) { if (  b
```

Because we're creating an exception of type **const char *** , so while catching this exception, we have to use const char * in the catch block. Compiling and running the above C ++ program will produce the following results:

```
Chu  y : Ban  dang chi cho so  0 !!!
```

Standard Exception in C ++

C ++ provides a list of defined Standard Exception that we can use in programs. These exceptions are arranged in the parent-child structure as follows:



The following table is a brief description of each **exception** mentioned in the diagram above:

Exception Description **std :: exception** An exception and parent of all Standard Exception in C ++ std :: bad_alloc Can be thrown by **new** std :: bad_cast Can be thrown by **dynamic_cast** std :: bad_exception This is a useful device to handle Unexpected Exception logic in a C ++ program std :: bad_typeid Can be thrown by **typeid std :: logic_error** An exception that can theoretically be detected by reading the std :: domain_error code This is an exception thrown when a domain invalid math used std :: invalid_argument Thrown by invalid parameters std :: length_error Thrown when a std :: string is too large to be created std :: out_of_range Can be thrown by a method , for example std :: vector and std :: bitset > :: operator [] (). **std :: runtime_error** An exception that cannot theoretically be detected by reading std :: overflow_error code Is thrown if a mathematical overflow occurs std :: range_error Appears when you try to save an external value of the range of std :: underflow_error is thrown if a mathematical underflow appears

Defining new Exception in C ++

You can define **exceptions** for yourself by inheriting and overwriting the exception class feature in C ++. The following example illustrates how you can use the std :: exception class to deploy your own exception in a standard way in C ++:

```
#include #include using namespace  std ; struct MyException : public  exception
```

It will produce the following result:

```
MyException  da duoc bat ! Exception  trong C ++
```

Here, what () is a public method provided by the exception class in C ++ and it has been overwritten by all exception subclasses. This example returns the cause of an exception in C ++.

According to Tutorialspoint

Previous article: Read / write File in C ++ | fstream in c ++

Next article: Dynamic memory in C / C ++