**Source:**

[**http://www.cplusplus.com/doc/tutorial/exceptions/**](http://www.cplusplus.com/doc/tutorial/exceptions/)

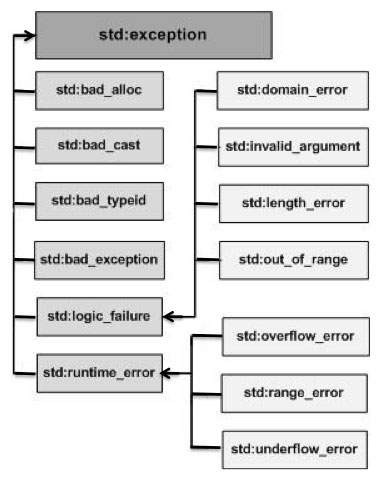
**Source:** [**http://www.tutorialspoint.com/cplusplus/cpp\_exceptions\_handling.htm**](http://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm)

**Source:**

[**http://www.battersea-locksmith.co.uk/briefings/divByZeroInCpp.html**](http://www.battersea-locksmith.co.uk/briefings/divByZeroInCpp.html)

C++ Standard Exceptions:

C++ provides a list of standard exceptions defined in **<exception>** which we can use in our programs. These are arranged in a parent-child class hierarchy shown below:



Description of each exception mentioned in the above hierarchy:

|  |  |
| --- | --- |
| **Exception** | **Description** |
| **std::exception** | An exception and parent class of all the standard C++ exceptions. |
| std::bad\_alloc | This can be thrown by **new**. |
| std::bad\_cast | This can be thrown by **dynamic\_cast**. |
| std::bad\_exception | This is useful device to handle unexpected exceptions in a C++ program |
| std::bad\_typeid | This can be thrown by **typeid**. |
| **std::logic\_error** | An exception that theoretically can be detected by reading the code. |
| std::domain\_error | This is an exception thrown when a mathematically invalid domain is used |
| std::invalid\_argument | This is thrown due to invalid arguments. |
| std::length\_error | This is thrown when a too big std::string is created |
| std::out\_of\_range | This can be thrown by the at method from for example a std::vector and std::bitset<>::operator[](). |
| **std::runtime\_error** | An exception that theoretically can not be detected by reading the code. |
| std::overflow\_error | This is thrown if a mathematical overflow occurs. |
| std::range\_error | This is occured when you try to store a value which is out of range. |
| std::underflow\_error | This is thrown if a mathematical underflow occurs. |

**Divide by zero in C++**

Why doesn't a divide-by-zero cause an exception in C++?

It was a specific design decision of C++ not to handle divide-by-zero; whereas Java and Ada, for example, take the opposite view. Why? The usual answers -- efficiency and an assumption that C++ will tend to be used with more of an awareness of the hardware.

Stroustrup says, in "The Design and Evolution of C++" (Addison Wesley, 1994), "low-level events, such as arithmetic overflows and divide by zero, are assumed to be handled by a dedicated lower-level mechanism rather than by exceptions. This enables C++ to match the behavior of other languages when it comes to arithmetic. It also avoids the problems that occur on heavily pipelined architectures where events such as divide by zero are asynchronous."

So you must check your divisors yourself, or discover if your hardware maps divide-by-zero onto some other kind of exception and catch that. It's fairly easy to discover the latter. Just put a try {} catch (...) {} around a divide by zero