Exceptional Handling

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

One of the advantages of C++ over C is Exception Handling. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. There are two types of exceptions: a)Synchronous, b)Asynchronous(Ex:which are beyond the program's control, Disc failure etc). C++ provides following specialized keywords for this purpose.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch,** and **throw**.

- **try** − A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.
- **throw** − A program throws an exception when a problem shows up. This is done using a **throw** keyword.
- **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.

Assuming a block will raise an exception, a method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch as follows –

```
try {
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
} catch( ExceptionName eN ) {
   // catch block
}
```

You can list down multiple **catch** statements to catch different type of exceptions in case your **try** block raises more than one exception in different situations.

Throwing Exceptions

- Exceptions can be thrown anywhere within a code block using **throw** statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.
- Following is an example of throwing an exception when dividing by zero condition occurs −

```
double division(int a, int b) {
   if( b == 0 ) {
      throw "Division by zero condition!";
   }
   return (a/b);
}
```

Catching Exceptions

The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

Catching Exceptions

The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

```
try {
   // protected code
} catch( ExceptionName e ) {
  // code to handle ExceptionName exception
}
```

Above code will catch an exception of **ExceptionName** type. If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the exception declaration as follows −

```
try {
   // protected code
} catch(...) {
  // code to handle any exception
}
```

Example

```
try {
  int age = 15;
  if (age > 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw (age);
  }
}
catch (int myNum) {
  cout << "Access denied - You must be at least 18 years old.\n";
  cout << "Age is: " << myNum;
}
```

- We use the try block to test some code: If the age variable is less than 18, we will throw an exception, and handle it in our catch block.
- In the catch block, we catch the error and do something about it. The catch statement takes a **parameter**: in our example we use an int variable (myNum) (because we are throwing an exception of int type in the try block (age)), to output the value of age.
- If no error occurs (e.g. if age is 20 instead of 15, meaning it will be be greater than 18), the catch block is skipped:

Let's take a simple example to understand the usage of try, catch and throw.

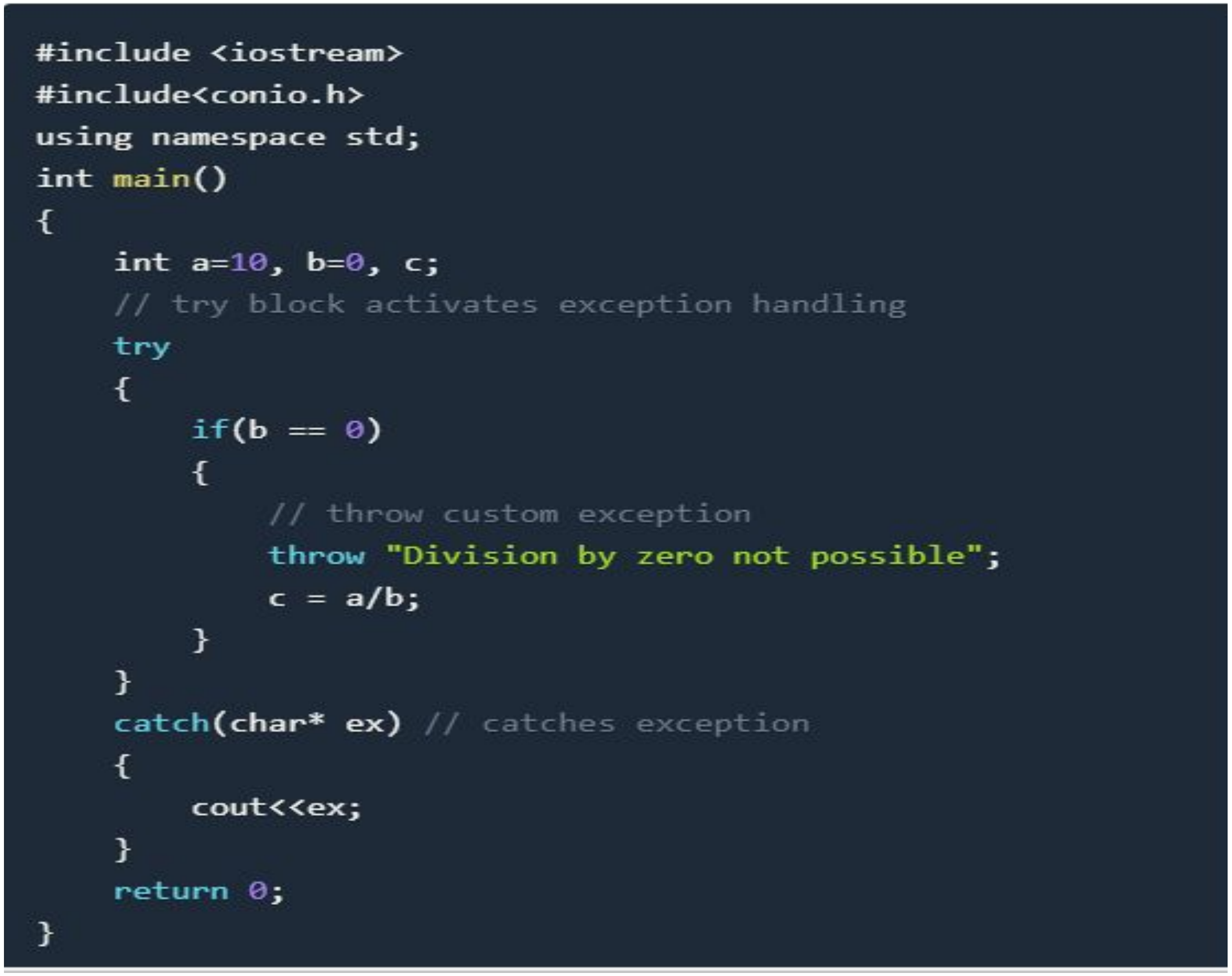
Below program compiles successfully but the program fails at runtime, leading to an exception.

```
#include <iostream>#include<conio.h>
using namespace std;
int main()
{
    int a=10,b=0,c;
    c=a/b;
    return 0;
}
```

The above program will not run, and will show **runtime error** on screen, because we are trying to divide a number with **0**, which is not possible.

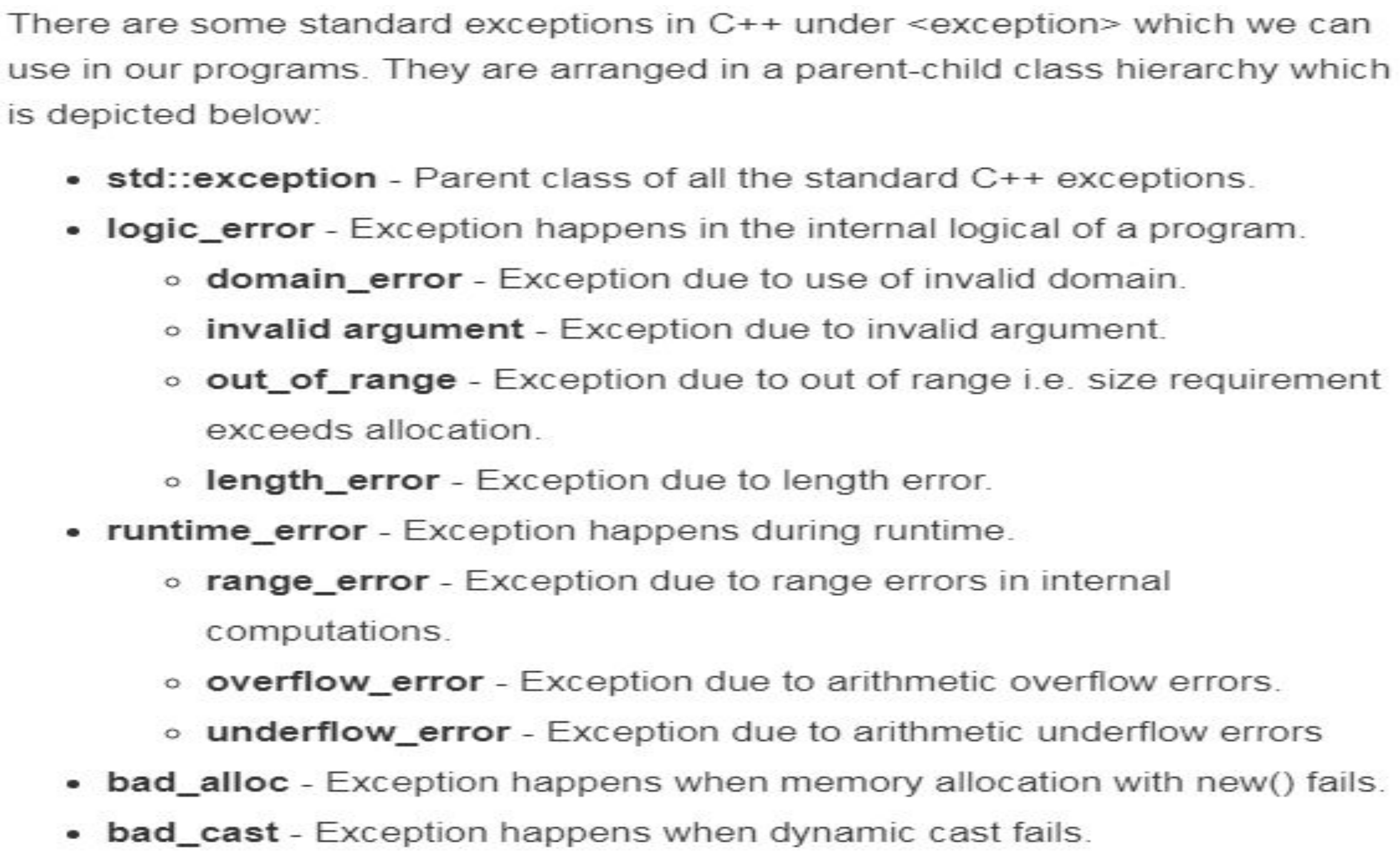
How to handle this situation? We can handle such situations using exception handling and can inform the user that you cannot divide a number by zero, by displaying a message.

Exceptional Handling



```
#include <iostream>
#include<conio.h>
using namespace std;
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;
}
```

Standard Exceptions



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.