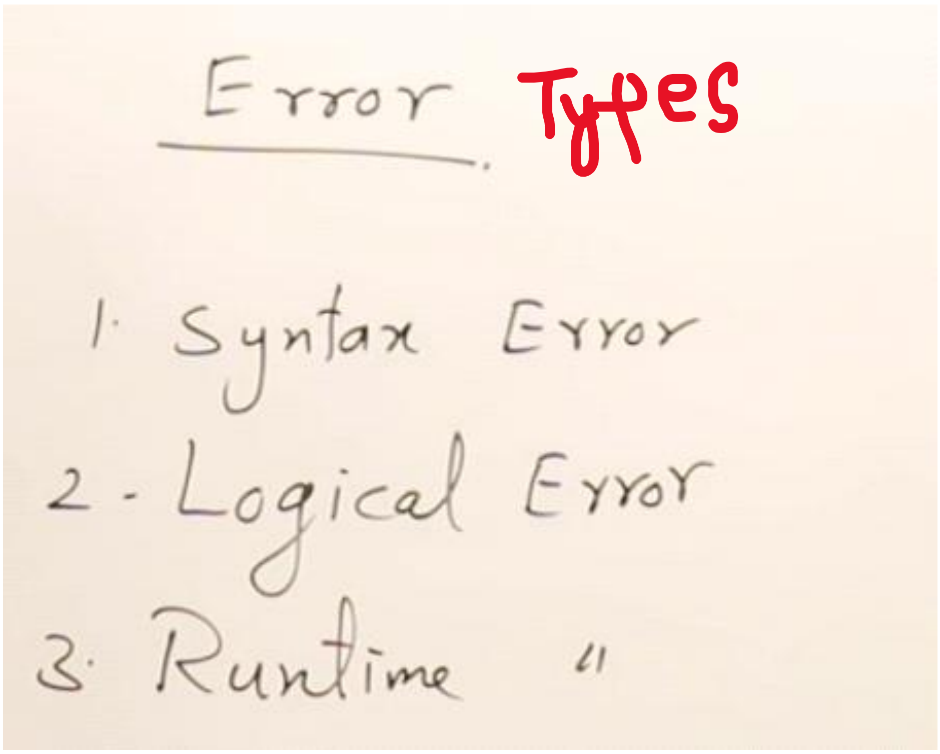
Exception Handling

23 September 2024 19:33

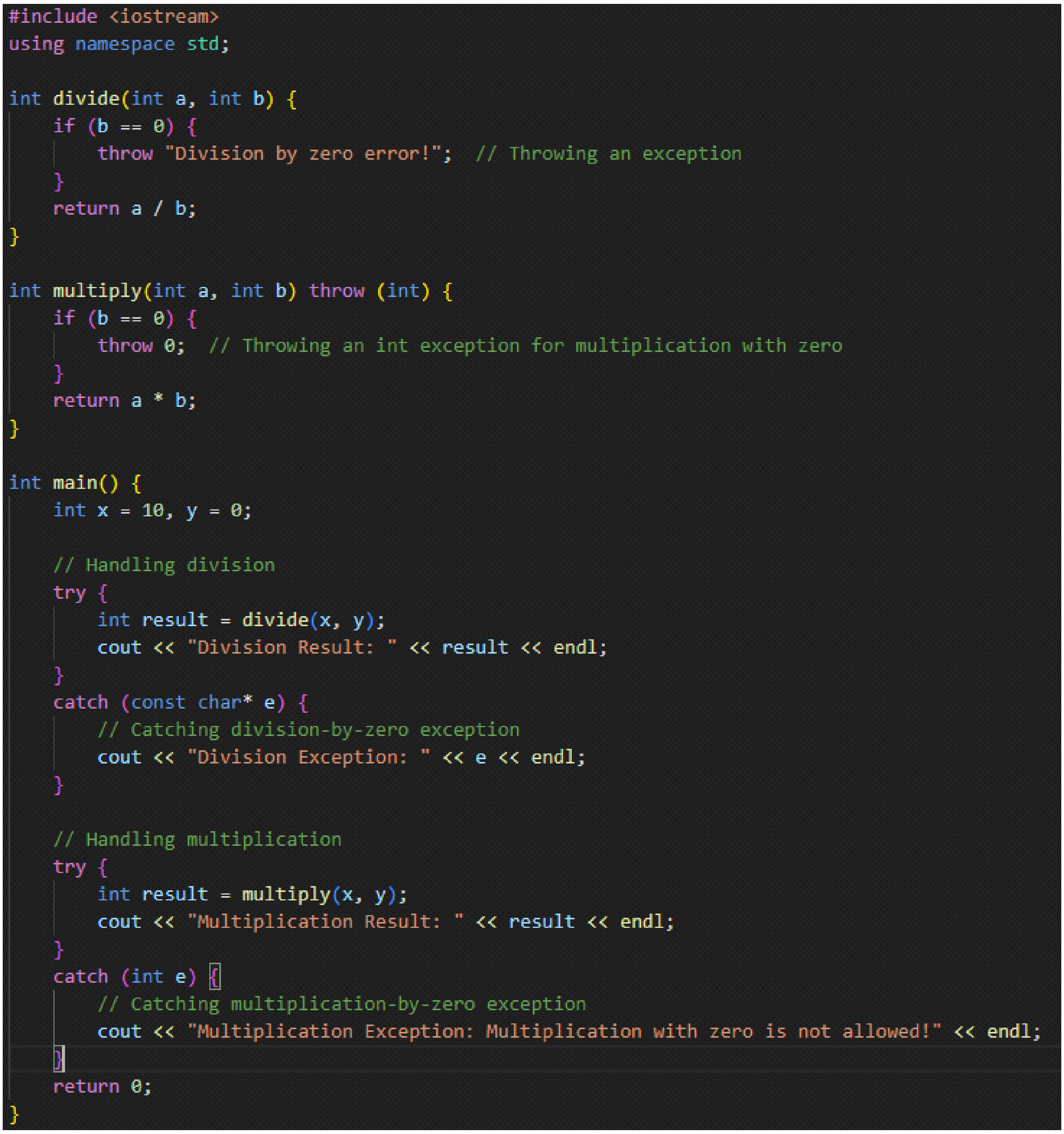

Error Types
1. Syntax Error
2. Logical Error
3. Runtime "

1. **Syntax Error :** it happens when a keyword or defined words are misspelled or mistyped. It is detected by Compiler
2. **Logical Error :** error in logic of program.
3. **Runtime Error :** error faced during the usage of program when user is using. It due to bad input by user like providing integer where string is needed.

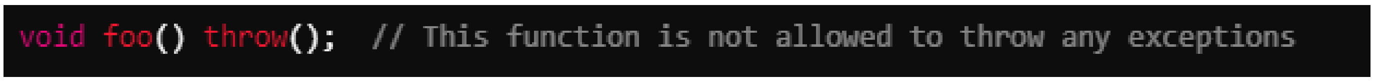
- `try` : Used to define a block of code that will be tested for exceptions.
- `throw` : Used to signal that an exception has occurred.
- `catch` : Used to handle the exception that is thrown.

How Exception Handling Works

1. **Try Block**: This block contains the code that might throw an exception. It is followed by one or more `catch` blocks.
2. **Throw Statement**: When a problem occurs, you use the `throw` keyword to signal an error or exceptional situation.
3. **Catch Block**: This block catches and processes the exception thrown in the `try` block. Each `catch` block can catch a different type of exception.



```cpp
#include <iostream>
using namespace std;

int divide(int a, int b) {
    if (b == 0) {
        throw "Division by zero error!";  // Throwing an exception
    }
    return a / b;
}

int multiply(int a, int b) throw (int) {
    if (b == 0) {
        throw 0;  // Throwing an int exception for multiplication with zero
    }
    return a * b;
}

int main() {
    int x = 10, y = 0;

    // Handling division
    try {
        int result = divide(x, y);
        cout << "Division Result: " << result << endl;
    }
    catch (const char* e) {
        // Catching division-by-zero exception
        cout << "Division Exception: " << e << endl;
    }

    // Handling multiplication
    try {
        int result = multiply(x, y);
        cout << "Multiplication Result: " << result << endl;
    }
    catch (int e) {
        // Catching multiplication-by-zero exception
        cout << "Multiplication Exception: Multiplication with zero is not allowed!" << endl;
    }
    return 0;
}
```



```cpp
void foo() throw();  // This function is not allowed to throw any exceptions
```

Modern Replacement: `noexcept`

Instead of using `throw()`, modern C++ uses the `noexcept` specifier to indicate that a function does not throw exceptions. It has two forms:

- `noexcept`: The function is guaranteed not to throw any exceptions.
- `noexcept(expression)`: The function is conditionally noexcept, based on the truth value of the expression.

Example with `noexcept`:

cpp Copy code

```cpp
void foo() noexcept {  // This function will not throw any exceptions
    // Function body
}

void bar() noexcept(false) {  // This function can throw exceptions
    throw "Some error";  // This is allowed because noexcept(false) allows exceptions
}
```