Exception Handling

Faculty: SREEDIVYA I
Branch: S6 ECE

Exceptions in C++

- When executing a C++ code, different errors can occur:
  - coding errors made by the programmer,
  - errors due to wrong input, or
  - others
- When an error occurs, compiler will normally stop and generate an error message. The technical term for this is: C++ will throw an **exception** (throw an error).
- An exception is an unexpected event that occurs during program execution.
- For example,

  float divide = 7 / 0;
- The above code causes an exception as it is not possible to divide a number by 0.

Exception Handling

- The process of handling these types of errors is known as exception handling.
- Exception Handling in C++ is a process to handle runtime errors.
- Using the exception handling mechanism, the control from one part of the program where the exception occurred can be transferred to another part of the code.
- We perform exception handling so the normal flow of the application can be maintained even after runtime errors.

C++ Standard Exceptions

- C++ has provided us with a number of standard exceptions that we can use in our exception handling. Some of them are shown in the table below.

Exception	Description
std::exception	The parent class of all C++ exceptions.
std::bad_alloc	Thrown when a dynamic memory allocation fails.
std::bad_cast	Thrown by C++ when an attempt is made to perform a dynamic_cast to an invalid type.
std::bad_exception	Typically thrown when an exception is thrown and it cannot be rethrown.

- In C++, exception is an event or object which is thrown at runtime.
- All exceptions are derived from std::exception class. It is a runtime error which can be handled.
- If we don't handle the exception, it prints exception message and terminates the program.
- In C++, we handle exceptions with the help of the try and catch blocks along with the throw keyword.
- **try -** code that may raise an exception.
- **throw -** throws an exception when an error is detected.
- **catch -** code that handles the exception thrown by the throw keyword.

Syntax:

```
try {
        // Code that might throw an exception
        throw SomeExceptionType("Error message");
    }
catch( ExceptionName e1 )  {
        // catch block catches the exception
       // that is thrown from try block
       }
```

1. try

The try keyword represents a block of code that may throw an exception placed inside the try block. It is followed by one or more catch blocks. If an exception occurs, try block throws that exception.

2. catch

The catch statement represents a block of code that is executed when a particular exception is thrown from the try block. The code to handle the exception is written inside the catch block.

3. throw

An exception in C++ can be thrown using the throw keyword. When a program encounters a throw statement, then it immediately terminates the current function and starts finding a matching catch block to handle the thrown exception.

Program to divide two numbers which throws an exception when the divisor is 0

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

int main()
{
    double numerator, denominator, divide;
    cout << "Enter numerator: ";
    cin >> numerator;
    cout << "Enter denominator: ";
    cin >> denominator;
```

```
try {
        if (denominator == 0)                          // throw an exception if denominator is 0
            throw 0;
       divide = numerator / denominator;               // not executed if denominator is 0
       cout << numerator << " / " << denominator << " = " << divide << endl;
    }
    catch (int num_exception) {
        cout << "Error: Cannot divide by " << num_exception << endl;
    }
    return 0;
}
```

OUTPUT 1

Enter numerator: 72

Enter denominator: 0

Error: Cannot divide by 0

OUTPUT 2

Enter numerator: 72

Enter denominator: 3

72 / 3 = 24

- To handle the exception, we have put the code

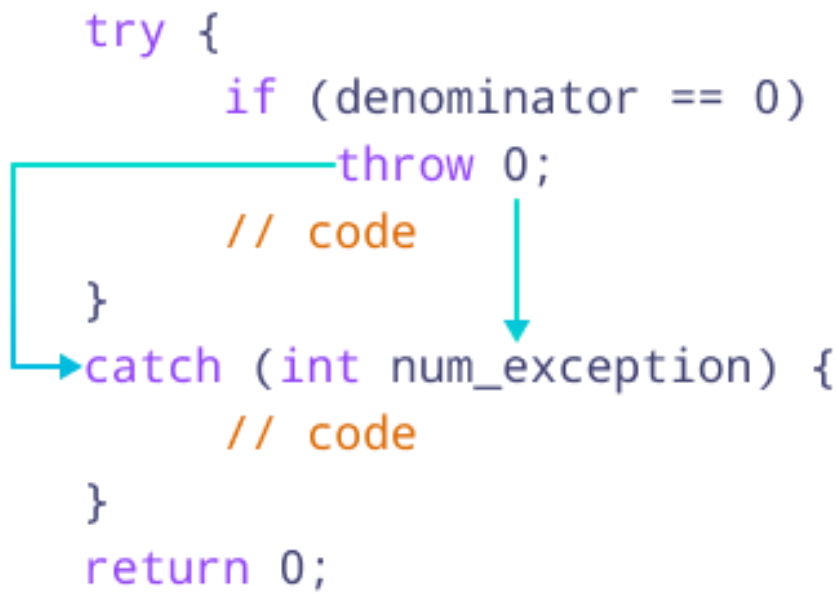
  divide = numerator / denominator;

  inside the try block. Now, when an exception occurs, the rest of the code inside the try block is skipped.

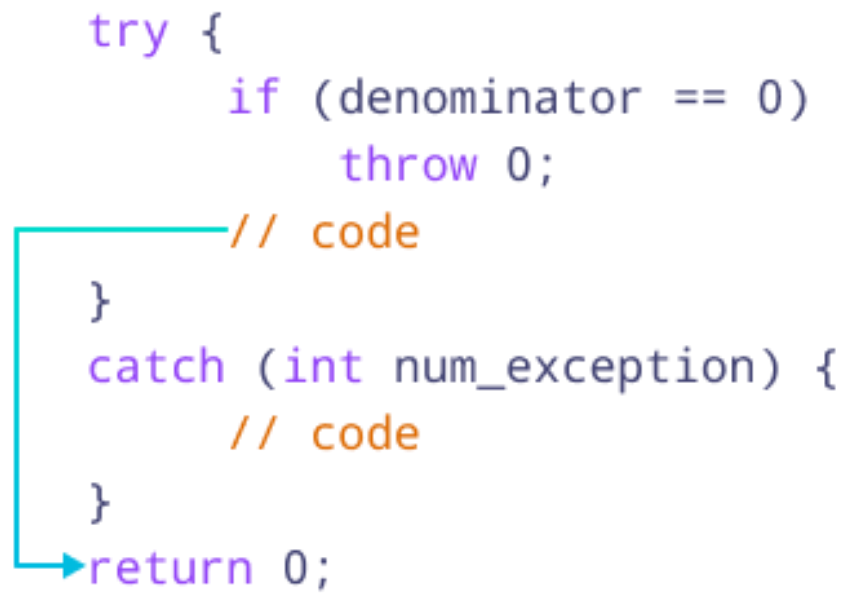
- The catch block catches the thrown exception and executes the statements inside it.

- If none of the statements in the try block generates an exception, the catch block is skipped.

denominator == 0



denominator != 0



- If we do not know the types of exceptions that can occur in our try block, then we can use the ellipsis symbol ... as our catch parameter.

```
try {
    // code
}
catch (...) {
    // code
}
```

C++ Multiple catch Statements

In C++, we can use multiple catch statements for different kinds of exceptions that can result from a single block of code.

Syntax

```
try {
    // code
}
catch (exception1) {
    // code
}
catch (exception2) {
    // code
}
catch (...) {
    // code
}
```

- Here, our program catches exception1 if that exception occurs. If not, it will catch exception2 if it occurs.

- If there is an error that is neither exception1 nor exception2, then the code inside of catch (...) {} is executed.

Notes:

- catch (...) {} should always be the final block in our try...catch statement. This is because this block catches all possible exceptions and acts as the default catch block.
- It is not compulsory to include the default catch block in our code.

Example 2: C++ Multiple catch Statements

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

int main() {

    double numerator, denominator, arr[4] = {0.0, 0.0, 0.0, 0.0};
    int index;

    cout << "Enter array index: ";
    cin >> index;

    try {
        if (index >= 4)                         // throw exception if array out of bounds
        throw "Error: Array out of bounds!";
```

```
    cout << "Enter numerator: ";       // not executed if array is out of bounds
    cin >> numerator;

    cout << "Enter denominator: ";
    cin >> denominator;

     if (denominator == 0)                 // throw exception if denominator is 0
        throw 0;

     arr[index] = numerator / denominator; // not executed if denominator is 0
     cout << arr[index] << endl;
}
```

```
    catch (const char* msg) {           // catch "Array out of bounds" exception
        cout << msg << endl;
    }

    catch (int num) {                    // catch "Divide by 0" exception
       cout << "Error: Cannot divide by " << num << endl;
    }

    catch (...) {                        // catch any other exception
       cout << "Unexpected exception!" << endl;
    }

   return 0;
}
```

OUTPUT 1

Enter array index: 5

Error: Array out of bounds!

Explanation

Here, the array arr[] only has 4 elements. So, index cannot be greater than 3.

In this case, index is 5. So we throw a string literal "Error: Array out of bounds!". This exception is caught by the first catch block.

Notice the catch parameter const char* msg. This indicates that the catch statement takes a string literal as an argument.

OUTPUT 2

Enter array index: 2

Enter numerator: 5

Enter denominator: 0

Error: Cannot divide by 0

Explanation

Here, the denominator is 0. So we throw the int literal 0. This exception is caught by the second catch block.

If any other exception occurs, it is caught by the default catch block.

OUTPUT 3

Enter array index: 2

Enter numerator: 5

Enter denominator: 2

2.5

Explanation

Here, the program runs without any problem as no exception occurs.