



**Himalaya College of Engineering**

## **Advanced C++ Programming Lab Report**

**Lab 8: EXCEPTION HANDLING**

**Prepared By:** Ankit Belbase(HCE081BEI007)

**Subject:** Object-Oriented Programming (OOP)

**Program:** Bachelor of Electronics Engineering

**Institution:** Himalaya College of Engineering

**Date:** July 27,2025

## LAB 8 : EXCEPTION HANDLING

### OBJECTIVE:

To manage runtime errors effectively, ensuring program stability and providing controlled error recovery.

### THEORY:

In the C++ programming language, an exception is defined as an unforeseen or abnormal event that occurs during program execution, disrupting its normal flow. The mechanism of exception handling is provided to systematically detect, signal, and manage these runtime anomalies in a controlled and safe manner. Exception handling enables a program to either continue execution safely or terminate gracefully, thereby preventing abrupt and undesirable program crashes.

### Basic Syntax

```
try {  
    // Code that may cause an exception  
    if (some_error_condition) {  
        throw exception_value;  
    }  
}  
  
catch (exception_type e) {  
    // Code to handle the exception  
}
```

### Mechanism:

#### 1. Throw Statement (throw)

When a throw statement executes, it creates and sends an exception object to be handled elsewhere in the program. Any data type or object can be thrown as an exception, but throwing objects derived from `std::exception` is recommended.

#### 2. Try Block (try)

Contains the code that might generate exceptions. The program monitors this block for any exceptions thrown inside it. If no exception occurs, the try block completes normally and the following code executes.

### **3. Catch Block (catch)**

Defines how to handle exceptions thrown in the associated try block. Each catch block specifies the type of exception it can handle. Multiple catch blocks can follow a single try block to handle different exception types and the first matching catch block handles the exception.

### **4. Exception Propagation**

If no matching catch block is found in the current function, the exception is propagated to the calling function. This propagation continues up the call stack until a suitable handler is found or the program terminates.

### **Exception with arguments**

In C++, an exception is a mechanism that allows a program to detect and respond to runtime errors. When an error occurs, an exception is thrown, and control is transferred to the corresponding exception handler. In C++, exceptions with arguments refer to the practice of throwing additional information, such as an error message, error code, or a custom object, along with the exception to provide more meaningful error reporting and handling. This helps identify what error occurred and why, rather than just signaling that some error happened.

Purpose of Exception with Arguments in C++ :

- To identify the exact cause of an error by providing specific details.

- To facilitate improved debugging and accurate error reporting.

- To enable differentiated handling of various types of errors.

### **Exceptions in Constructors and Destructors**

In C++, constructors and destructors are special member functions responsible for object initialization and cleanup. Exception handling in these functions is critical because improper management can lead to resource leaks or abrupt program termination.

### **Exceptions in Constructors**

If an exception is thrown inside a constructor, the object's construction fails and the object is considered not fully created. The destructor for that object is not called, since the object does not exist completely. However, destructors for any fully constructed base classes and member objects are called automatically to release resources acquired before the exception. Throwing exceptions in constructors is a common method to indicate initialization failure (e.g., invalid parameters, allocation failure).

## Exceptions in Destructors

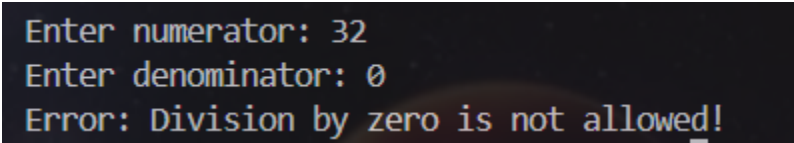
Destructors should never throw exceptions because they are usually called during stack unwinding caused by another exception. Throwing an exception from a destructor while another exception is active causes the program to call `std::terminate()`, ending the program abruptly. To avoid this, destructors must handle all exceptions internally, typically by using a try-catch block and not allowing exceptions to propagate.

## LAB QUESTIONS:

Ques 1: Write a C++ program to handle divide-by-zero exception using try-catch block. Input two numbers. If denominator is zero, throw and catch an exception.

```
#include <iostream>
using namespace std;
int main() {
    double numerator, denominator, result;
    cout << "Enter numerator: ";
    cin >> numerator;
    cout << "Enter denominator: ";
    cin >> denominator;
    try {
        if (denominator == 0) {
            throw "Division by zero is not allowed!";
        }
        result = numerator / denominator;
        cout << "Result: " << result << endl;
    }
    catch (const char* msg) {
        cout << "Error: " << msg << endl;
    }
    return 0;
}
```

## OUTPUT:



```
Enter numerator: 32
Enter denominator: 0
Error: Division by zero is not allowed!
```

Ques 2: Write a program using catch-all handler (catch(...)) to handle any kind of exception.

Illustrate a case where an unexpected data type is thrown and caught generically.

```
#include <iostream>

using namespace std;

int main() {
    int choice;

    cout << "Enter a number (1 to throw double, 2 to throw bool, any other to throw unknown): ";
    cin >> choice;

    try {
        if (choice == 1) {
            throw 3.14;
        }
        else if (choice == 2) {
            throw true;
        }
        else {
            throw nullptr;
        }
    }

    catch (double e) {
        cout << "Caught double exception: " << e << endl;
    }

    catch (bool e) {
        cout << "Caught boolean exception: " << (e ? "true" : "false") << endl;
    }

    catch (...) {
        cout << "Caught unknown exception using catch-all handler" << endl;
    }
}
```

```
return 0;  
  
}
```

## OUTPUT:

```
Enter a number (1 to throw double, 2 to throw bool, any other to throw unknown): 5  
Caught unknown exception using catch-all handler
```

## DISCUSSION:

Exception handling in C++ provides a structured way to handle runtime errors and unexpected situations using try, catch, and throw blocks. It helps in separating error-handling code from regular logic, making the program more readable and reliable. Instead of crashing, a program can gracefully handle errors like division by zero, invalid input, or file read failures. However, improper use of exceptions or catching by value can lead to issues like resource leaks or undefined behavior.

## CONCLUSION:

Exception handling in C++ enhances program stability by providing a clean way to manage errors. It improves code clarity and ensures better control over unexpected events. When used properly, it makes programs more robust, maintainable, and user-friendly.