

Himalaya College of Engineering

**Advanced C++ Programming Lab Report**

Lab 9: Exception Handling in C++

**Prepared By :** Monsoon Sapkota (HCE081BEI022)

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

**Program :** Bachelor of Electronics, Communication and Information Engineering

**Institution :** Himalaya College of Engineering

**Date :** July 17, 2025

**OBJECTIVE**

➢ To understand the concept of Exception handling in C++.

# BACKGROUND THEORY

## Definition

When an issue occurs (like division by zero), an "exception" is **thrown**. This exception is then **caught** by dedicated code, allowing your program to recover or exit gracefully.

It helps keep your main code clean and easily propagate error information.

## Syntax

Exception handling in C++ uses three keywords: **try**: Encloses code that might cause a problem.

try {

// Code that could throw an exception

}

**throw**: Signals an exceptional condition. You can throw numbers, text, or custom error objects.

if (value < 0) { throw "Negative value not allowed!";

}

**catch**: Handles exceptions thrown from a try block. Each catch block handles a specific type of exception.

catch (const char\* errorMessage) {

// Handle text message error

} catch (int errorCode) {

// Handle integer error code

} catch (...) {

// Catch any other exception

}

## 3. Types of Exceptions

You can throw:

Custom Exceptions: Error types you create using C++ classes for specific details. class MyError : public std::exception { public:

std::string message;

MyError(const std::string& msg) : message(msg) {}

};

Standard Exceptions: Built-in C++ exceptions for common problems (e.g., std::runtime\_error, std::logic\_error, std::out\_of\_range, std::bad\_alloc).

## 4. Simple Example

#include <iostream>

#include <string>

double divide(double numerator, double denominator) { if (denominator == 0) { throw "Error: Cannot divide by zero!";

}

return numerator / denominator;

}

int main() {

try { double result1 = divide(10.0, 2.0); std::cout << "10 / 2 = " << result1 << std::endl;

double result2 = divide(5.0, 0.0); std::cout << "This line will not be executed." << std::endl;

}

catch (const char\* errorMessage) {

std::cerr << "Caught an error: " << errorMessage << std::endl;

}

std::cout << "Program continues after error handling." << std::endl;

return 0;

}

**Explanation of the Example:**

1. divide throws an error if the denominator is zero.
2. main calls divide within a try block.
3. When divide(5.0, 0.0) throws, the try block stops. 4. The catch block for const char\* handles the error.

5. Program execution resumes after the catch block.