TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: - 07

Title: - Exception Handling

Submitted by: -

Name: - Bikram Panthi

Roll NO: - 13

Submitted To: -

Department Of Electronics Engineering

Checked by: -

Date of submission: -

Objectives

- To understand the concept of exception handling in C++ using try, catch, and throw blocks.
- To handle different types of exceptions including int, char, string, and user-defined types using multiple catch blocks.
- To implement catch-all handlers (catch(...)) for catching unknown or unexpected exceptions.
- To demonstrate how exceptions can be rethrown and handled at different levels using nested try-catch blocks.
- To learn how to create and use custom exception classes that carry error messages.
- To safely handle exceptions in class constructors and understand why destructors should not throw exceptions.

Theory

In C++, exception handling is a powerful mechanism that allows a program to detect and manage runtime errors in a controlled way. When an unexpected event or error occurs (such as division by zero, file not found, or invalid input), the program can throw an exception, which can then be caught and handled gracefully — without crashing the whole program.

C++ provides three main keywords for exception handling:

- try – defines a block of code to be tested for errors.
- catch – defines a block of code to handle the error.
- throw – is used to signal (or "throw") an exception.

Using this structure, we can separate error-handling logic from the regular logic of our program. This helps improve program readability, reliability, and security. We can also throw and catch user-defined exceptions using custom types.

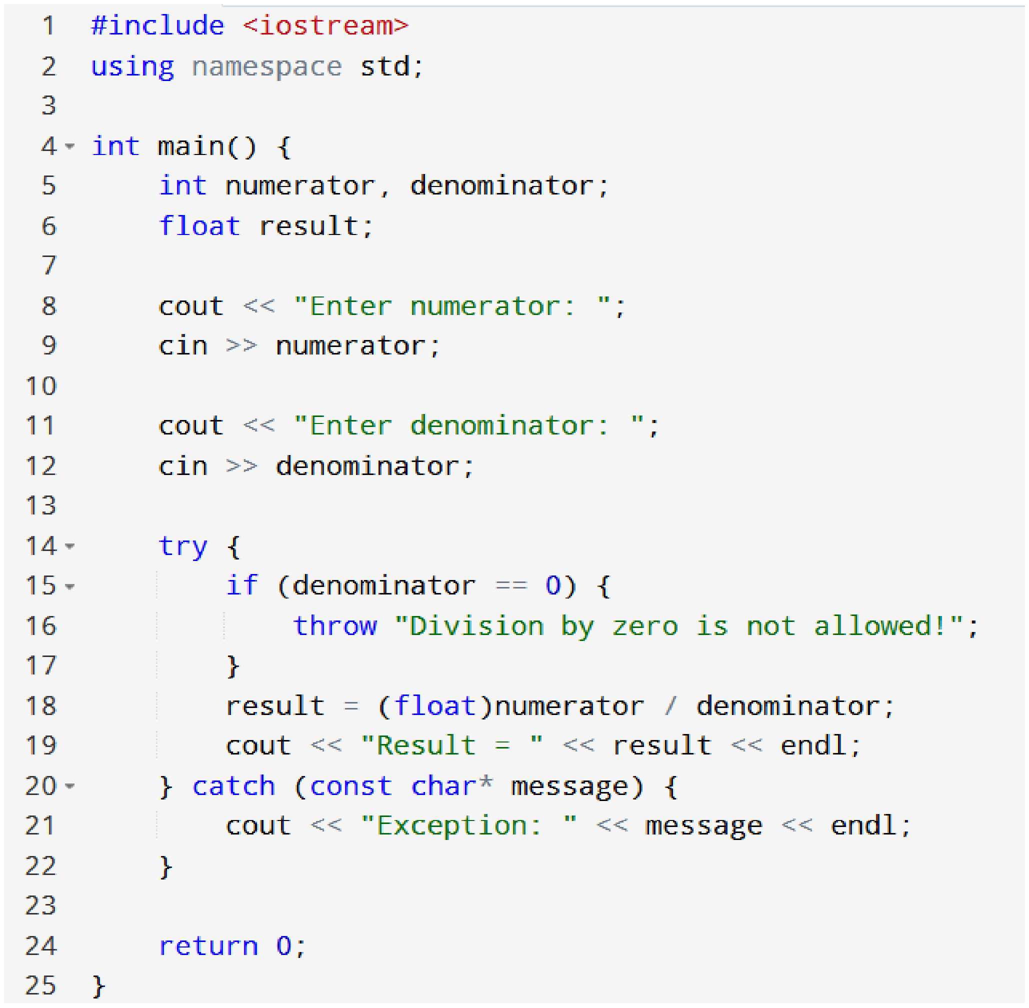
Basic Syntax:

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

Questions

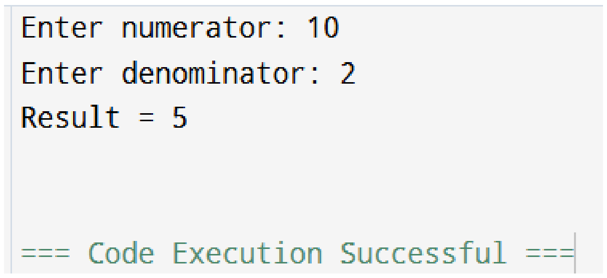
1.Write a C++ program to handle divide-by-zero exception using try-catch block. Input two numbers. If the denominator is zero, throw and catch an exception.

Code:

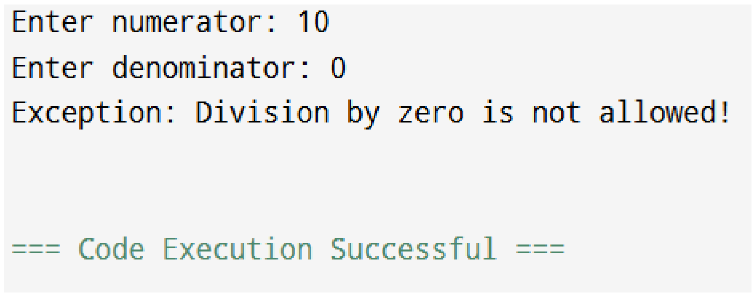


```cpp
#include <iostream>
using namespace std;

int main() {
    int numerator, denominator;
    float result;

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

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

    try {
        if (denominator == 0) {
            throw "Division by zero is not allowed!";
        }
        result = (float)numerator / denominator;
        cout << "Result = " << result << endl;
    } catch (const char* message) {
        cout << "Exception: " << message << endl;
    }

    return 0;
}
```

Output:



```
Enter numerator: 10
Enter denominator: 2
Result = 5


=== Code Execution Successful ===
```



```
Enter numerator: 10
Enter denominator: 0
Exception: Division by zero is not allowed!


=== Code Execution Successful ===
```

2. Write a C++ program to demonstrate multiple catch blocks handling different data types. Throw and handle int, char, string, type exceptions in separate catch blocks.

Code:

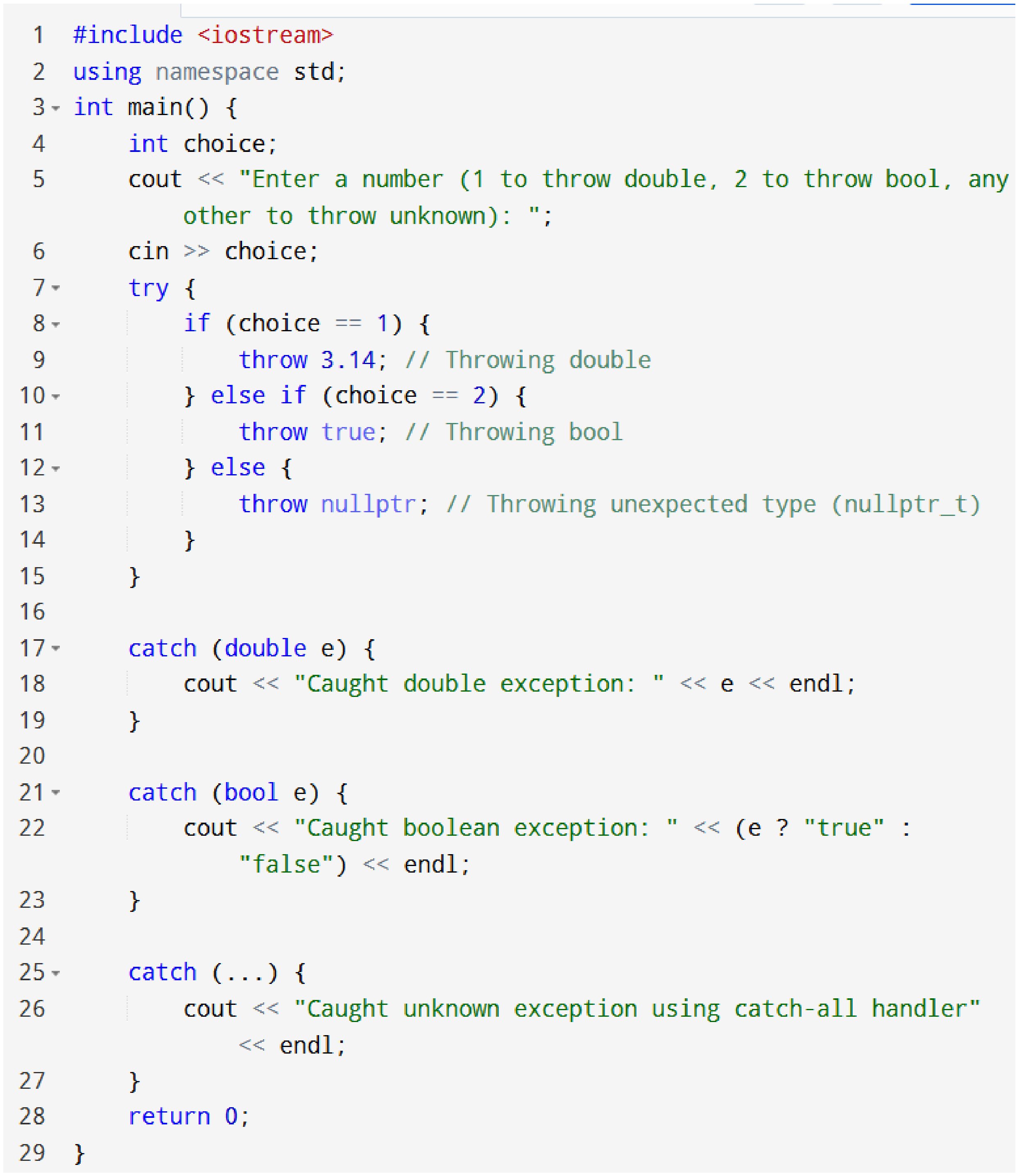
```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    try {
        int choice;
        cout << "Choose exception to throw:" << endl;
        cout << "1. int\n2. char\n3. string\n4. float\n";
        cin >> choice;

        switch (choice) {
            case 1: throw 100;                  // throwing int
            case 2: throw 'A';                  // throwing char
            case 3: throw string("Error!");     // throwing string
            case 4: throw 3.14f;                // throwing float
            default: cout << "Invalid choice!" << endl;
        }
    }
    catch (int e) {
        cout << "Caught an integer exception: " << e << endl;
    }
    catch (char e) {
        cout << "Caught a character exception: " << e << endl;
    }
    catch (string e) {
        cout << "Caught a string exception: " << e << endl;
    }
    catch (...) {
        cout << "Caught an unknown exception!" << endl;
    }
  return 0;
}
```

Output:

```
Choose exception to throw:
1. int
2. char
3. string
4. float
2
Caught a character exception: A
```

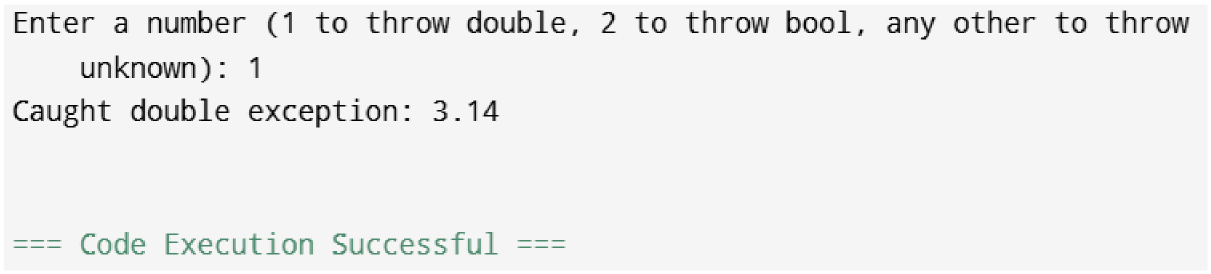
3. Write a program using catch-all handler (catch(...)) to handle any kind of exception. Illustrate a case where an unexpected data type is thrown and caught generically.

Code:

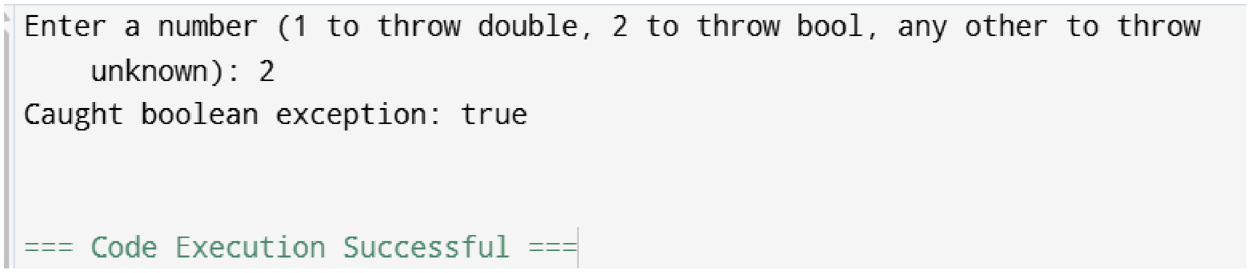


```cpp
#include <iostream>
using namespace std;
int main() {
    int choice;
    cout << "Enter a number (1 to throw double, 2 to throw bool, any other to throw unknown): ";
    cin >> choice;
    try {
        if (choice == 1) {
            throw 3.14; // Throwing double
        } else if (choice == 2) {
            throw true; // Throwing bool
        } else {
            throw nullptr; // Throwing unexpected type (nullptr_t)
        }
    }

    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;
}
```

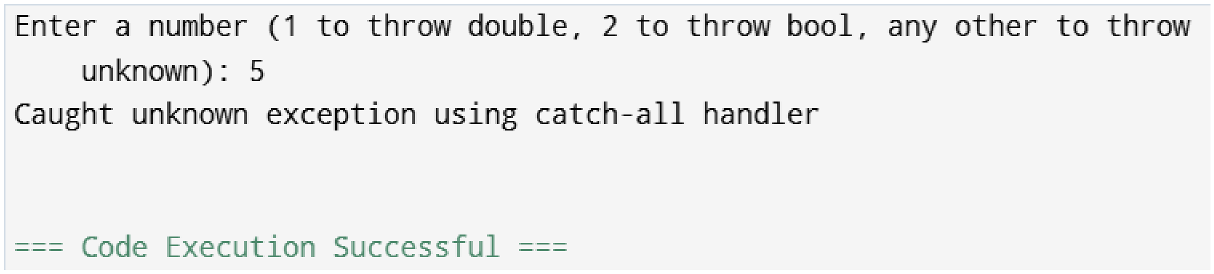
Output:



```
Enter a number (1 to throw double, 2 to throw bool, any other to throw
    unknown): 1
Caught double exception: 3.14


=== Code Execution Successful ===
```



```
Enter a number (1 to throw double, 2 to throw bool, any other to throw
    unknown): 2
Caught boolean exception: true


=== Code Execution Successful ===
```



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


=== Code Execution Successful ===
```

4. Write a C++ program that rethrows an exception after catching it once. Use a nested try-catch where the inner catch block rethrows the exception to be handled by the outer block.

Code:

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

int main() {
    int value;

    cout << "Enter a number (0 to throw exception): ";
    cin >> value;

    try {
        try {
            if (value == 0) {
                throw "Zero value detected!";
            }
            cout << "No exception, value is: " << value << endl;
        }
        catch (const char* msg) {
            cout << "Inner catch: " << msg << endl;
            throw; // Rethrow to outer catch
        }
    }
    catch (const char* msg) {
        cout << "Outer catch: " << msg << endl;
    }
    catch (...) {
        cout << "Outer catch: Unknown exception caught" << endl;
    }

    return 0;
}
```

Output:

```
Enter a number (0 to throw exception): 0
Inner catch: Zero value detected!
Outer catch: Zero value detected!


=== Code Execution Successful ===
```

```
Enter a number (0 to throw exception): 5
No exception, value is: 5


=== Code Execution Successful ===
```

5. Write a program to demonstrate throwing and catching a user-defined exception class with message argument. Define a custom class MyException and pass an error message to its constructor.

Code:

```cpp
#include <iostream>
#include <string>
using namespace std;
class MyException {
private:
    string message;
public:
    // Constructor with message
    MyException(const string& msg) : message(msg) {}

    string getMessage() const {
        return message;
    }
};
int main() {
    int value;
    cout << "Enter a number (negative to throw exception): ";
    cin >> value;
    try {
        if (value < 0) {
            throw MyException("Negative value is not permitted!");
        }
        cout << "Valid value entered: " << value << endl;
    }
    catch (const MyException& e) {
        cout << "Caught MyException: " << e.getMessage() << endl;
    }
    catch (...) {
        cout << "Caught unknown exception" << endl;
    }
    return 0;
}
```

Output:

```
Enter a number (negative to throw exception): -3
Caught MyException: Negative value is not permitted!
```

```
Enter a number (negative to throw exception): 5
Valid value entered: 5
```

6. Write a program to show exception handling inside a class constructor and destructor. Handle constructor exceptions properly; avoid throwing from destructor.

Code:

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

class TestClass {
private:
    int value;

public:
    // Constructor with exception handling
    TestClass(int val) {
        try {
            if (val < 0) {
                throw string("Negative value not allowed in constructor!");
            }
            value = val;
            cout << "Constructor: Object created with value " << value << endl;
        }
        catch (const string& e) {
            cout << "Constructor exception: " << e << endl;
            throw; // Rethrow to allow main to handle it
        }
    }

    // Destructor (never throws)
    ~TestClass() {
        cout << "Destructor: Cleaning up object with value " << value << endl;
        // Never throw in destructor – it may cause termination
    }

    void display() const {
        cout << "Current value: " << value << endl;
    }
};

int main() {
    try {
        int input;
        cout << "Enter a number for object creation: ";
        cin >> input;

        TestClass obj(input); // Object creation
        obj.display();
    }
    catch (const string& e) {
        cout << "Main caught exception: " << e << endl;
    }
    catch (...) {
        cout << "Main caught unknown exception" << endl;
    }

    return 0;
}
```

Output:

```
Enter a number for object creation: 10
Constructor: Object created with value 10
Current value: 10
Destructor: Cleaning up object with value 10


=== Code Execution Successful ===
```

```
Enter a number for object creation: -5
Constructor exception: Negative value not allowed in constructor!
Main caught exception: Negative value not allowed in constructor!


=== Code Execution Successful ===
```

Discussion:

In this lab, we explored how exception handling makes C++ programs more reliable and safe. We learned to use try, catch, and throw statements to handle different kinds of runtime errors gracefully, such as divide-by-zero, invalid input, or unexpected data types. By using multiple catch blocks, we were able to handle specific exception types like int, char, and string, while also using the catch-all handler (catch(...)) to catch unknown exceptions. We also worked with user-defined exception classes, which allowed us to pass and display custom error messages. Another important part of the lab was learning how to handle exceptions inside class constructors and why we should avoid throwing exceptions from destructors.

Conclusion:

This lab helped us understand how exception handling is essential for writing robust and crash-free programs. We saw that instead of terminating the program on errors, we can handle problems gracefully and inform the user. Using exception handling techniques, we can isolate error-handling logic from regular code, making programs more organized and easy to debug. We also learned safe practices like rethrowing exceptions and using custom exception classes. Overall, the lab improved our confidence in writing safer, more reliable, and maintainable C++ programs.