TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: -08

Title: - Exceptional Handling

Submitted by: -	**Submitted To: -**
Name: -Aastha Gaire	**Department of Electronics and Computer Engineering**
Roll NO: - HCE081BEI003	**Checked by: -**

Date of submission: - 2082/04/08

OBJECTIVE

1. To comprehend the idea of exception handling in C++ and its significance.

2. To become proficient in handling runtime problems by using the try, catch, and throw keywords.

3. To create programs that manage typical exceptions, such as out-of-range errors, divide-by-zero, and incorrect input.

4. To show off user-defined exceptions and construct several catch blocks.

5. To manage unforeseen circumstances in order to guarantee program stability and avoid crashes.

THEORY

In C++, exception handling is a technique for managing runtime problems that prevents unexpected software crashes. Without stopping the program's execution abruptly, it enables it to detect faults and take corrective action.

The main components of exception handling are:

1. try Block:

- The code that might generate an error is placed inside the try block.

2. throw Statement:

- When an error is detected, the throw keyword is used to raise (throw) an exception.

3. catch Block:

- The catch block is used to catch and handle the exception thrown by the try block.

Example:

```
try {

   int a = 10, b = 0;
   if (b == 0)
      throw "Division by zero error!";
   cout << a / b;
}
catch (const char* msg) {
   cout << "Exception caught: " << msg;
}
```

4. Multiple Catch Blocks:

- A program can have multiple catch blocks to handle different types of exceptions separately.

5. User-Defined Exceptions:

- Programmers can define their own exception classes for more specific error handling.

Benefits of Exception Handling:

• **Better Program Stability:** Prevents unanticipated mistakes from causing the program to crash.
• **Error Separation**: This keeps error-handling code and regular code logic apart.
• Easily add or change error-handling without affecting the core code because to its flexibility and scalability.
• **Improved User Experience**: This feature permits insightful error messages rather than application termination.

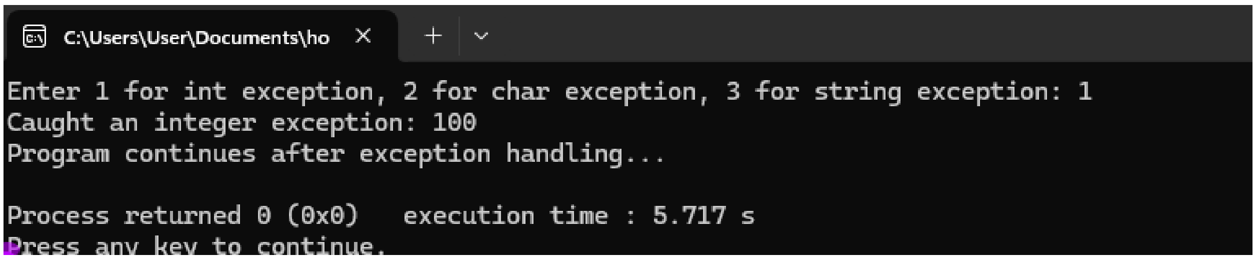
Real-World Applications:

- File I/O operations (e.g., file not found)
- User input validation
- Network programming (e.g., connection timeout)
- Memory allocation failures
- Banking or transaction systems

<u>LAB PROGRAMS</u>

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

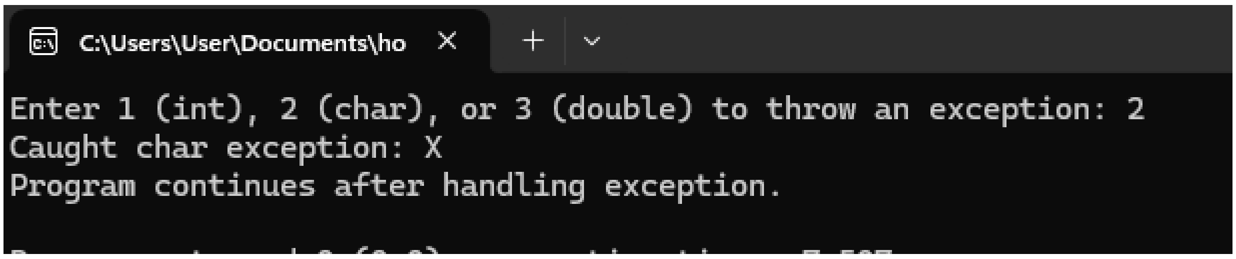
```
#include <iostream>

#include <string>

using namespace std;


int main() {

    try {

        int choice;

        cout << "Enter 1 for int exception, 2 for char exception, 3 for string exception: ";

        cin >> choice;


        if (choice == 1) {

            throw 100;              // Throwing an int

        }

        else if (choice == 2) {

            throw 'E';              // Throwing a char

        }

        else if (choice == 3) {

            throw string("String type exception"); // Throwing a string

        }

        else {
            cout << "No exception thrown." << 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;

    }


    cout << "Program continues after exception handling..." << endl;

    return 0;

}
```



```
Enter 1 for int exception, 2 for char exception, 3 for string exception: 1
Caught an integer exception: 100
Program continues after exception handling...

Process returned 0 (0x0)   execution time : 5.717 s
Press any key to continue.
```

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;
void testException(int code) {
    if (code == 1) {
        throw 42; // int exception
    }
    else if (code == 2) {
        throw 'X'; // char exception
    }
    else if (code == 3) {
        throw 3.14; // double (not explicitly caught)
    }
    else {
        cout << "No exception thrown." << endl;
    }
}

int main() {
    int choice;
    cout << "Enter 1 (int), 2 (char), or 3 (double) to throw an exception: ";
    cin >> choice;

    try {
        testException(choice);
    }
    catch (int e) {
        cout << "Caught int exception: " << e << endl;
    }
    catch (char e) {
        cout << "Caught char exception: " << e << endl;
    }
    catch (...) {
        cout << "Caught an unknown or unexpected exception!" << endl;
    }

    cout << "Program continues after handling exception." << endl;
    return 0;
}
```



```
Enter 1 (int), 2 (char), or 3 (double) to throw an exception: 2
Caught char exception: X
Program continues after handling exception.
```

Discussion:

In this lab, we learned how to use the try, throw, and catch blocks in C++ to handle runtime issues. We created programs to handle issues such as incorrect input and divide-by-zero. Additionally, we developed bespoke exceptions and employed numerous catch blocks. This taught us how to handle exceptions correctly rather than allowing the program to crash, which makes programs more reliable and user-friendly.

Conclusion:

We learned how to properly handle runtime problems in C++ thanks to the exception handling lab. Try, throw, and catch taught us how to recognize mistakes and react appropriately. In order to handle various error kinds, we also investigated using custom exceptions and multiple catch blocks. All things considered, the experiment demonstrated how exception handling increases software dependability, avoids crashes, and improves user experience.