TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: -08

Title: - Exceptional Handling

Submitted by: -

Name: - Diwas Pokhrel

Roll NO: - HCE081BEI014

Submitted To: -

Department of Electronics and Computer Engineering

Checked by: -

Date of submission: - 2082/04/08

OBJECTIVE

1. To understand the concept and importance of exception handling in C++.
2. To learn how to use try, catch, and throw keywords for managing runtime errors.
3. To write programs that handle common exceptions like divide-by-zero, invalid input, or out-of-range errors.
4. To implement multiple catch blocks and demonstrate user-defined exceptions.
5. To ensure program stability and prevent crashes by handling unexpected situations.

THEORY

Exception handling in C++ is a method used to handle runtime errors, ensuring that the program does not crash unexpectedly. It allows the program to catch errors and take corrective actions without terminating the execution abruptly.

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:

- **Improved Program Stability**: Prevents the program from crashing due to unexpected errors.
- **Error Separation**: Keeps the normal code logic separate from error-handling code.
- **Flexible & Scalable**: Easy to add or modify error-handling without disturbing the main code.
- **Better User Experience**: Allows meaningful error messages instead of program 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

LAB PROGRAMS

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.

```cpp
#include <iostream>
    #include <string>
    using namespace std;
    int main() {
    try {
        int choice;
    cout << "Enter 1 to throw int,2 to throw char,3 to throw
        string: ";
        cin >> choice;
      if (choice == 1)
            throw 100;
        else if (choice == 2)
            throw 'W';
        else if (choice == 3)
            throw string("This is a string exception.");
        else
            cout << "Invalid choice!" << endl;
    }
    catch (int e) {
        cout << "Integer exception: " << e << endl;
    }
    catch (char e) {
        cout << "Char exception: " << e << endl;
    }
    catch (string &e) {
        cout << "String exception: " << e << endl;
    }
    cout << "Program continues after exception handling." << endl;
      return 0;
}4
```

Output

```
Enter 1 to throw int,2 to throw char,3 to throw string: 2
Char exception: W
Program continues after exception handling.


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

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.

```cpp
#include <iostream>
using namespace std;
int main() {
    try {
        int choice;
        cout << "Enter 1 for int, 2 for double, 3 for char, 4 for
            bool: ";
        cin >> choice;
   if (choice == 1)
            throw 100;
        else if (choice == 2)
            throw 3.14;
        else if (choice == 3)
            throw 'Z';
        else if (choice == 4)
            throw true;
        else
            throw "Unknown type";
    }
    catch (int e) {
        cout << " Int exception: " << e << endl;
    }
    catch (...) {
        cout << "Here is an exception of unknown or unexpected
            type!" << endl;
    }
 cout << "Now the program continues." << endl;
 return 0;
}
```

Output

```
Enter 1 for int, 2 for double, 3 for char, 4 for bool: 1
 Int exception: 100
Now the program continues.


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

Discussion:

In this lab, we practiced handling runtime errors using try, throw, and catch blocks in C++. We wrote programs that managed errors like divide-by-zero and invalid input. We also used multiple catch blocks and created custom exceptions. Through this, we learned how to make programs more stable and user-friendly by handling exceptions properly instead of letting the program crash.

Conclusion:

The exception handling lab helped us understand how to manage runtime errors in C++ effectively. By using try, throw, and catch, we learned to catch errors and respond gracefully. We also explored multiple catch blocks and custom exceptions to handle different error types. Overall, the lab showed how exception handling improves program reliability, prevents crashes, and enhances the overall user experience.