TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: - 08

Title: - Exception Handling

Submitted by: -

Name: - Gaurab Pandey

Roll NO: - 15

Submitted To: -

Department Of

Checked by: -

Date of submission: -

Objectives:

- To gain an understanding of Exception handling in C++.

Tools and Libraries Used:

- Programming Language: C++
- IDE: G++
- Libraries: include <iostream>, include <string>

Theory:

Exception handling in C++ is a mechanism that allows a program to detect and manage runtime errors gracefully, without crashing. It separates error-handling code from the main logic, improving readability and robustness.

C++ uses three main keywords:

- try: Defines a block of code to test errors.
- throw: Signals the occurrence of an exception.
- catch: Defines a block of code that handles the error.

Syntax:

```
try {
    throw "Error occurred";
}
catch (const char* msg) {
    cout << msg;
}
```

You can throw and catch different types: integers, strings, objects, or even custom exception classes.

Benefits:

- Handles errors without terminating the program.
- Promotes clean, maintainable code.

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.

Code:

```
#include <iostream>
#include <string>
using namespace std;
void testException(int x) {
    try {
        if (x == 0)
            throw 10;
        else if (x == 1)
            throw 'a';
        else if (x == 2)
            throw string("Error");
    }
    catch (int e) {
        cout << "Caught integer exception: " << e << endl;
    }
    catch (char e) {
        cout << "Caught character exception: " << e << endl;
    }
    catch (string e) {
        cout << "Caught string exception: " << e << endl;
    }
    catch (...) {
        cout << "Caught unknown exception type" << endl;
    }


int main() {
    cout << "Testing multiple catch blocks:\n";

    cout << "\nThrowing int exception: ";
    testException(0);

    cout << "\nThrowing char exception: ";
    testException(1);

    cout << "\nThrowing string exception: ";
    testException(2);

    return 0;
```

Output:

```
Testing multiple catch blocks:

Throwing int exception: Caught integer exception: 10

Throwing char exception: Caught character exception: a

Throwing string exception: Caught string exception: Error
```

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.

Code:

```
#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 10;
        else if (choice == 2)
            throw 3.14;
        else if (choice == 3)
            throw 'X';
        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: 10
Now the program continues.
```

Discussion

In this lab, we explored exception handling in C++ using multiple catch blocks to handle different data types, including int, char, and string. The experiment demonstrated how C++ allows programmers to throw and catch exceptions of varying types, enabling more precise error management. By designing a function that conditionally throws different exceptions based on input, we observed how each catch block selectively processes its corresponding exception type, ensuring that errors are handled appropriately without unintended side effects.

Conclusion

The lab successfully demonstrated the effectiveness of multiple catch blocks in handling diverse exception types in C++. By implementing int, char, and string exceptions, we verified that C++'s exception handling mechanism provides flexibility in error management, allowing developers to tailor responses to specific failure scenarios.