

OBJECTIVE

- To understand the concept and importance of exception handling in C++ programming.
- To learn how to use the try, throw, and catch blocks for managing runtime errors.
- To implement programs that detect and handle various types of exceptions.

BACKGROUND THEORY

Exception handling in C++ is a powerful mechanism designed to detect, manage, and respond to errors or abnormal situations that occur during the execution of a program. Instead of allowing the program to crash or behave unpredictably, exceptions provide a structured way to gracefully handle runtime anomalies.

The core idea is to isolate the error-prone code using a try block, and to "throw" an exception when an error is encountered. This exception is then "caught" by a corresponding catch block, where appropriate corrective action can be taken.

Basic Components:

- try block: Contains the code that might cause an exception.
- throw statement: Used to signal the occurrence of an exception.
- catch block: Handles the exception thrown by the throw statement.

Types of Exceptions:

1. Built-in types: int, float, char, const char*, etc.
2. Standard exceptions: Provided by the C++ Standard Library (like `std::out_of_range`, `std::runtime_error`)
3. User-defined exceptions: Created using custom classes for specific error handling needs.

Advantages of Exception Handling:

- Prevents program crashes during runtime errors.
- Enhances program robustness and user experience.
- Simplifies error-handling code.
- Encourages separation of logic and error-handling.

1. Write a C++ program to demonstrate multiple catch blocks handling different data types throw and handle int, character, and string type exception 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;
    }
}
```

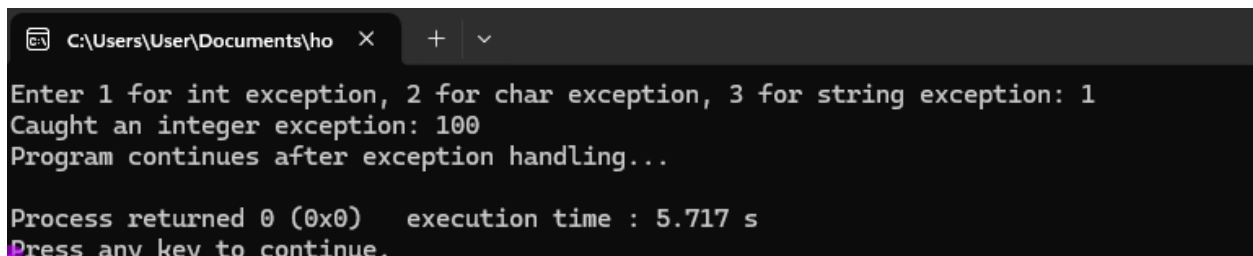
```

    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;
}

```

Output



```

C:\Users\User\Documents\ho
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 CatchAll Handler (catch(...)) to handle any kind of exception. Illustrate a case where an unexpected data type is shown 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;

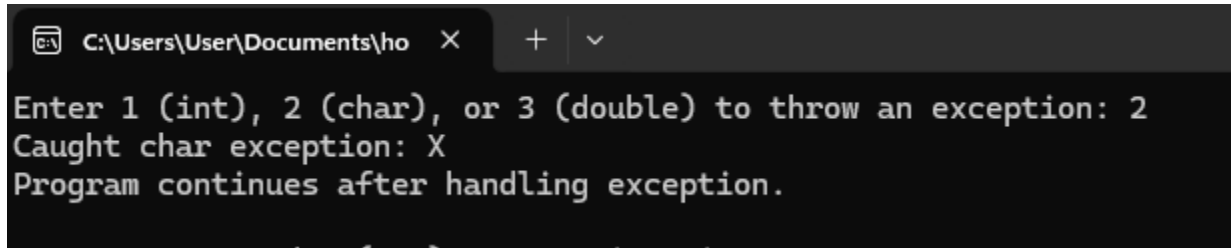
    try {
        testException(choice);
    }
    catch (int e) {
        cout << "Caught int exception: " << e << endl;
    }
    catch (char e) {
        cout << "Caught char exception: " << e << endl;
    }
    catch (...) {
        cout << "Caught an unknown or unexpected exception!" << endl;
    }

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

```

```
    return 0;  
}
```

Output

A screenshot of a terminal window with a dark background. The window title bar shows the file path 'C:\Users\User\Documents\ho' and standard window controls. The terminal text reads: 'Enter 1 (int), 2 (char), or 3 (double) to throw an exception: 2', 'Caught char exception: X', and 'Program continues after handling exception.'

```
C:\Users\User\Documents\ho X + v  
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 explored the concept of exception handling in C++ using the try, throw, and catch mechanisms. The focus was on handling different types of runtime errors gracefully without crashing the program. Various types of exceptions, such as int, char, and string, were thrown and caught in separate catch blocks to demonstrate type-specific handling. Additionally, a catch-all handler (catch(...)) was used to illustrate how unknown or unexpected exceptions can be caught safely.

Conclusion:

Exception handling in C++ provides a structured and effective way to detect and manage runtime errors. By using try, throw, and catch blocks, programmers can ensure that their applications remain stable and user-friendly even under unexpected conditions. This lab demonstrated the use of multiple catch blocks and catch-all handlers to cover both specific and generic exceptions. Overall, exception handling is an essential tool for building reliable and fault-tolerant software systems.