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

#include <iostream>

using namespace std;

int main() {

double numerator, denominator, result;

cout << "Enter numerator: ";

cin >> numerator;

cout << "Enter denominator: ";

cin >> denominator;

try {

if (denominator == 0) {

throw "Division by zero is not allowed!";

}

result = numerator / denominator;

cout << "Result: " << result << endl;

}

catch (const char\* msg) {

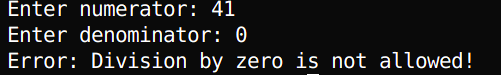
cout << "Error: " << msg << endl;

}

return 0;

}

**OUTPUT**



**2. 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() {

int choice;

cout << "Enter a number (1 for int, 2 for char, 3 for string): ";

cin >> choice;

try {

if (choice == 1) {

throw 42;

}

else if (choice == 2) {

throw 'X';

}

else if (choice == 3) {

throw string("String exception");

}

else {

cout << "No exception thrown. Invalid choice." << endl;

}

}

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 (...) {

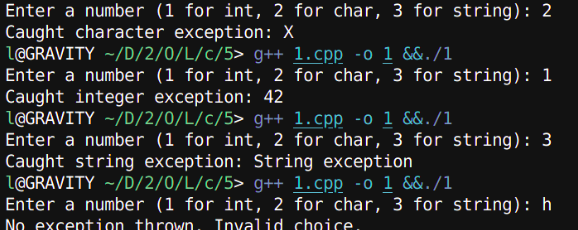
cout << "Caught unknown exception" << endl;

}

return 0;

}

**OUTPUT**



**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.**

#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 if (choice == 3) {

throw "String exception"; // Throwing string

}

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 (const char\* e) {

cout << "Caught string exception: " << e << endl;

}

catch (...) {

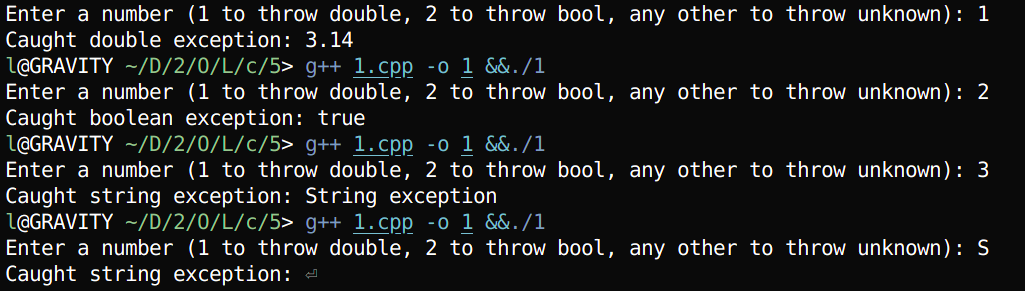
cout << "Caught unknown exception using catch-all handler" << endl;

}

return 0;

}

**OUTPUT**



**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.**

#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 the exception

}

}

catch (const char\* msg) {

cout << "Outer catch: " << msg << endl;

}

catch (...) {

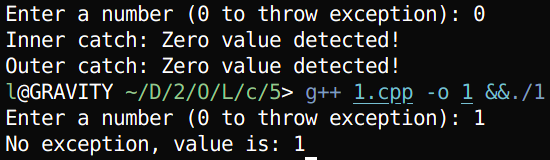
cout << "Outer catch: Unknown exception caught" << endl;

}

return 0;

}

**OUTPUT**



**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.**

#include <iostream>

#include <string>

using namespace std;

// User-defined exception class

class MyException {

private:

string message;

public:

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 (...) {

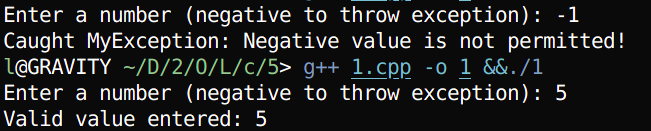
cout << "Caught unknown exception" << endl;

}

return 0;

}

**OUTPUT**



**6. Write a program that sets a custom terminate handler using set\_terminate() and demonstrates uncaught exception handling.Throw an exception with no matching catch block.**

#include <iostream>

#include <exception>

#include <string>

using namespace std;

// Custom terminate handler

void customTerminate() {

cout << "Custom terminate handler called: Uncaught exception detected!" << endl;

exit(1);

}

int main() {

// Set custom terminate handler

set\_terminate(customTerminate);

int value;

cout << "Enter a number (non-zero to throw uncaught exception): ";

cin >> value;

// Try block with no matching catch for string exception

try {

if (value != 0) {

throw string("Uncaught string exception thrown!");

}

cout << "No exception thrown, value is: " << value << endl;

}

catch (int e) {

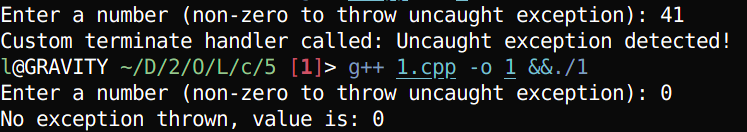
cout << "This catch block won't be used: " << e << endl;

}

return 0;

}

**OUTPUT**



**7. Write a C++ program with a function that violates its exception specification and handles it using set\_unexpected().Use throw(double) in the function declaration but throw an int.**

#include <iostream>

#include <exception>

using namespace std;

// Custom terminate handler

void customTerminate() {

cout << "Custom terminate handler called: noexcept violated!" << endl;

exit(1); // Or use abort();

}

// Function marked noexcept but throws an exception (violation)

void riskyFunction(int value) noexcept {

if (value < 0) {

throw 42; // Throws int, violates noexcept

}

cout << "Function executed successfully with value: " << value << endl;

}

int main() {

// Set custom terminate handler

set\_terminate(customTerminate);

int value;

cout << "Enter a number (negative to throw exception): ";

cin >> value;

riskyFunction(value); // No try-catch; noexcept violation goes to terminate

return 0;

}

**OUTPUT**





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

#include<iostream>

#include<exception>

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 caller to handle

}

}

// Destructor (avoids throwing exceptions)

~TestClass() {

cout << "Destructor: Cleaning up object with value " << value << endl;

// No throw in destructor to prevent program termination

// Resource cleanup happens safely

}

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);

obj.display();

}

catch (const string& e) {

cout << "Main caught exception: " << e << endl;

}

catch (...) {

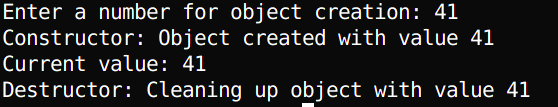
cout << "Main caught unknown exception" << endl;

}

return 0;

}

**OUTPUT**



**9. Write a function that accepts user input and throws an exception if input is invalid (e.g., non- integer for age).Use exception handling to validate data entry.**

#include <iostream>

#include <string>

#include <stdexcept>

using namespace std;

// Function to validate age input

int getValidAge() {

string input;

int age;

cout << "Enter your age: ";

cin >> input;

// Check if all characters in input are digits

for (char ch : input) {

if (!isdigit(ch)) {

throw invalid\_argument("Invalid input: Age must be a positive integer.");

}

}

// Convert string to integer

age = stoi(input);

// Check logical age constraint

if (age <= 0 || age > 120) {

throw out\_of\_range("Age must be between 1 and 120.");

}

return age;

}

int main() {

try {

int age = getValidAge();

cout << "Valid age entered: " << age << endl;

}

catch (const invalid\_argument& e) {

cerr << "Input Error: " << e.what() << endl;

}

catch (const out\_of\_range& e) {

cerr << "Range Error: " << e.what() << endl;

}

catch (...) {

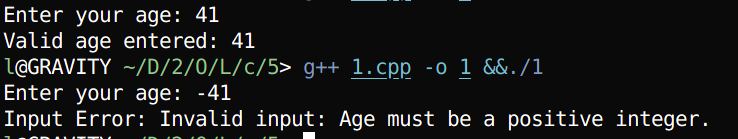
cerr << "An unknown error occurred." << endl;

}

return 0;

}

**OUTPUT**



**10. Write a C++ program to demonstrate exception propagation across multiple function calls.Function A calls B, which calls C. C throws an exception. Handle it in A.**

#include <iostream>

#include <stdexcept>

using namespace std;

// Function C – Validates input and throws exception if negative

void functionC(int num) {

if (num < 0) {

throw runtime\_error("Negative number is not allowed!");

}

cout << "Number is valid: " << num << endl;

}

// Function B – calls function C

void functionB(int num) {

functionC(num); // Exception propagates if thrown

}

// Function A – calls function B and handles exception

void functionA(int num) {

try {

functionB(num);

}

catch (const runtime\_error& e) {

cout << "Exception caught in function A: " << e.what() << endl;

}

}

int main() {

int input;

cout << "Enter a number: ";

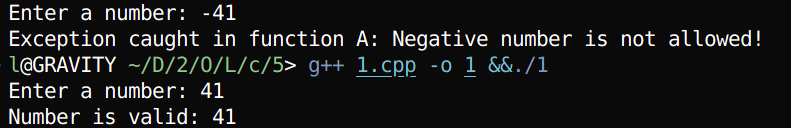
cin >> input;

functionA(input); // Starts the function call chain

return 0;

}

**OUTPUT**



# DISCUSSION

This lab demonstrated how C++ handles runtime errors using try, throw, and catch. We explored multiple catch blocks, catch-all handlers, rethrowing, custom exceptions, and exception propagation. It also covered handling exceptions in constructors, destructors, and uncaught cases using set\_terminate().

# CONCLUSION

The lab successfully demonstrated that exception handling in C++ improves program reliability and robustness by separating error handling from core logic. It enables developers to gracefully manage errors without complex conditional checks. Features like rethrowing, custom exceptions, and propagation across functions enhance flexibility and modularity. Mastery of exception handling is crucial for building safe, maintainable software, especially when dealing with user input, resource management, and system-level errors.