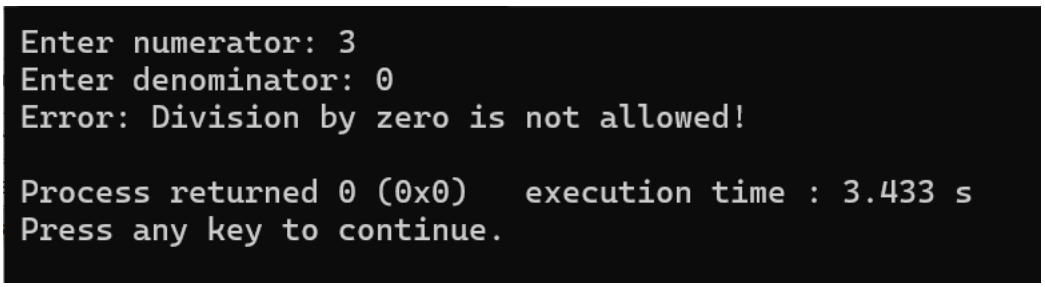
Lab Questions

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;

}
```



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;
}
```

```
Enter a number (1 for int, 2 for char, 3 for string): 3
Caught string exception: String exception

Process returned 0 (0x0)   execution time : 4.386 s
Press any key to continue.
```

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.

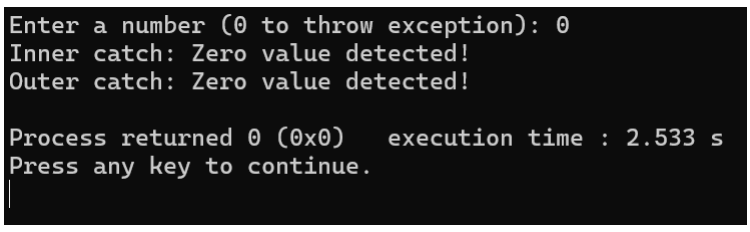
```
#include <iostream>
#include<stdexcept>
#include<string>
using namespace std;
int main() {
int choice;
cout << "Enter a number (1 to throw double, 2 to throw bool, any other to throw unknown): ";
cin >> choice;
try {
if (choice == 1) {
throw 3.14;
}
else if (choice == 2) {
throw true;
}
else {
```

```
Enter a number (1 to throw double, 2 to throw bool, any other to throw unknown): 2
Caught boolean exception: true

Process returned 0 (0x0)   execution time : 3.573 s
Press any key to continue.
```

```
throw nullptr;
}
}
catch (double e) {
cout << "Caught double exception: " << e << endl;
}
catch (bool e) {
cout << "Caught boolean exception: " << (e ? "true" : "false") << endl;
}
catch (...) {
cout << "Caught unknown exception using catch-all handler" << endl;
}
return 0;
}
```

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;
}
}
catch (const char* msg) {
cout << "Outer catch: " << msg << endl;
}
catch (...) {
cout << "Outer catch: Unknown exception caught" << endl;
}
return 0;
}
```

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;
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;
}
```

```
Enter numerator: 5
Enter denominator: 0
Error: Division by zero is not allowed!

Process returned 0 (0x0)   execution time : 4.837 s
Press any key to continue.
```

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;
void customTerminate() {
cout << "Mat Kar Lala Mat Kar:" << endl;
exit(1);
}
int main() {
set_terminate(customTerminate);
```

```
Enter a number (non-zero to throw uncaught exception): 2
Custom terminate handler called: Uncaught exception detected!

Process returned 1 (0x1)   execution time : 1.898 s
Press any key to continue.
```

```
int value;
cout << "Enter a number (non-zero to throw uncaught exception): ";
cin >> value;
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;
}
```

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;
void customUnexpected() {
cout << "Custom unexpected handler called: Exception specification violated!" << endl;
exit(1);
}
void riskyFunction(int value) throw(double) {
if (value < 0) {
throw 42;
}
cout << "Function executed successfully with value: " << value << endl;
}
```

```
int main() {
set_unexpected(customUnexpected);
int value;
cout << "Enter a number (negative to throw int exception): ";
cin >> value;
try {
riskyFunction(value);
}
catch (double e) {
cout << "Caught double exception: " << e << endl;
}
catch (...) {
cout << "Caught unexpected exception in main" << endl;
}
return 0;
}
```

```
Enter a number (negative to throw int exception): -6
Custom unexpected handler called: Exception specification violated!

Process returned 1 (0x1)   execution time : 5.523 s
Press any key to continue.
```

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<stdexcept>
using namespace std;
class TestClass {
private:
int value;
public:
TestClass(int val) {
try {
if (val < 0) {
throw string("Negative value not allowed in constructor!");
}
```

```
Enter a number for object creation: 3
Constructor: Object created with value 3
Current value: 3
Destructor: Cleaning up object with value 3

Process returned 0 (0x0)   execution time : 3.322 s
Press any key to continue.
```

```
value = val;
cout << "Constructor: Object created with value " << value << endl;
}
catch (const string& e) {
cout << "Constructor exception: " << e << endl;
throw;
}
}
~TestClass() {
cout << "Destructor: Cleaning up object with value " << value << endl;
}
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;
}
```

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;
int getValidAge() {
string input;
int age;
cout << "Enter your age: ";
cin >> input;
for (char ch : input) {
if (!isdigit(ch)) {
throw invalid_argument("K xa re bhai age negative halxas. Ta pagal ho:");
}
}
age = stoi(input);
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) {
```

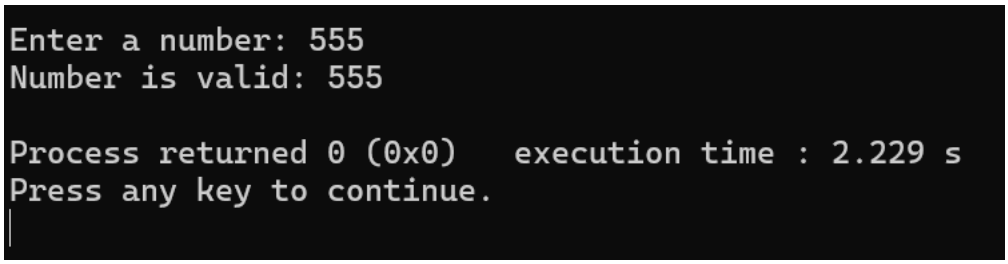
```
cerr << "Range Error: " << e.what() << endl;
}
catch (...) {
cerr << "An unknown error occurred." << endl;
}
return 0;
}
```

```
Enter your age: -8
Input Error: Invalid input: Age must be a positive integer.

Process returned 0 (0x0)   execution time : 4.162 s
Press any key to continue.
```

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;
void functionC(int num) {
if (num < 0) {
throw runtime_error("Negative number is not allowed!");
}
cout << "Number is valid: " << num << endl;
}
void functionB(int num) {
functionC(num);
}
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);
return 0;
}
```



Discussion

While doing the lab assignments, we found out the significances of the use of exception handling cases in C++. We have to be careful about the use of syntax while writing the code. We also got the idea of how to handle errors that come unexpectedly during the execution of program.

Conclusions:
And hence we successfully implemented exception handling cases in C++.