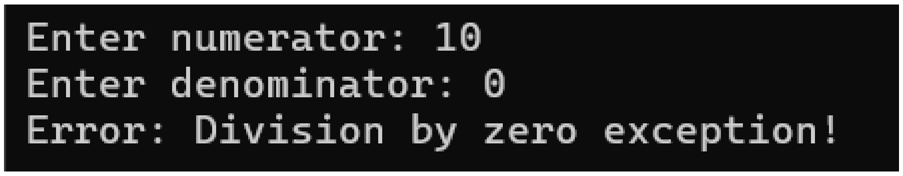
LAB ASSIGNMENTS

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

Source Code:

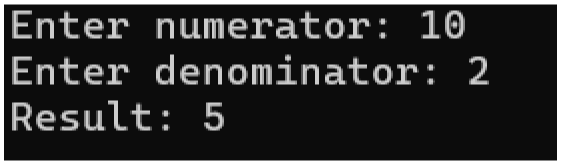
```
#include <iostream>
using namespace std;
int main() {
    float numerator, denominator;
    cout << "Enter numerator: ";
    cin >> numerator;
    cout << "Enter denominator: ";
    cin >> denominator;
    try {
        if (denominator == 0) {
            throw "Division by zero exception!";
        }
        float result = numerator / denominator;
        cout << "Result: " << result << endl;
    } catch (const char* msg) {
        cout << "Error: " << msg << endl;
    }
    return 0;
}
```

Output 1:



```
Enter numerator: 10
Enter denominator: 0
Error: Division by zero exception!
```

Output 2:



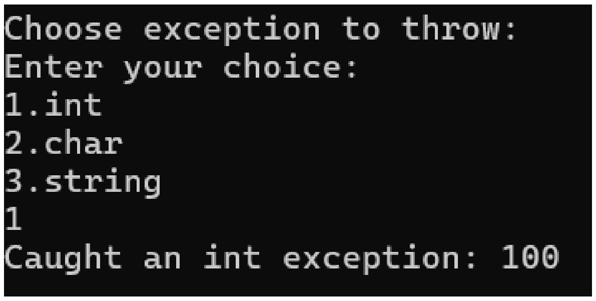
```
Enter numerator: 10
Enter denominator: 2
Result: 5
```

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.

Source Code:

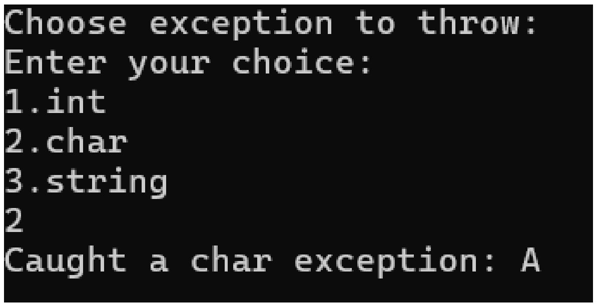
```cpp
#include <iostream>

#include <string>

using namespace std;


int main() {

    int choice;

    cout << "Choose exception to throw:"<<endl;

    cout << "Enter your choice:\n1.int\n2.char\n3.string"<<endl;

    cin >> choice;

    try {

        if (choice == 1) {

            throw 100;

        } else if (choice == 2) {

            throw 'A';

        } else if (choice == 3) {

            throw string("This is a string exception!");

        } else {

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

        }


    } catch (int e) {

        cout << "Caught an int exception: " << e << endl;

    } catch (char e) {

        cout << "Caught a char exception: " << e << endl;
```

Output 1:



```
Choose exception to throw:
Enter your choice:
1.int
2.char
3.string
1
Caught an int exception: 100
```

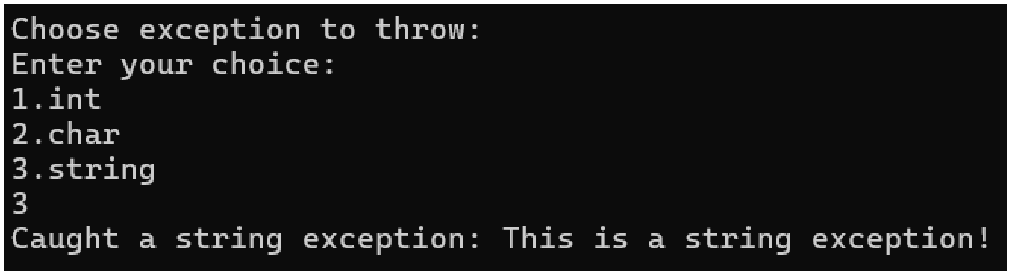
Output 2:



```
Choose exception to throw:
Enter your choice:
1.int
2.char
3.string
2
Caught a char exception: A
```

```
    } catch (string& e) {
        cout << "Caught a string exception: " << e << endl;
    }
    return 0;
}
```

Output 3:



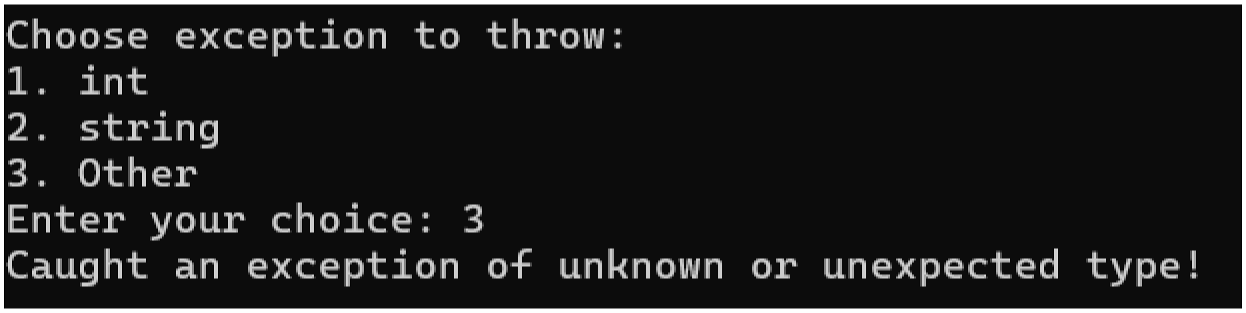
```
Choose exception to throw:
Enter your choice:
1.int
2.char
3.string
3
Caught a string exception: This is a string exception!
```

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.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    int choice;
    cout << "Choose exception to throw:" << endl;
    cout << "1. int" << endl;
    cout << "2. string" << endl;
    cout << "3. Other" << endl;
    cout << "Enter your choice: ";
    cin >> choice;

    try {
        if (choice == 1) {
            throw 42;
        } else if (choice == 2) {
            throw string("A string exception!");
        } else if (choice == 3) {
            throw 3.14159;
        } else {
            cout << "Invalid choice. No exception thrown." << endl;
        }
    } catch (int e) {
        cout << "Caught int exception: " << e << endl;
    } catch (const string& e) {
        cout << "Caught string exception: " << e << endl;
    } catch (...) {
        cout << "Caught an exception of unknown or unexpected type!" << endl;
    }

    return 0;
}
```

Output:



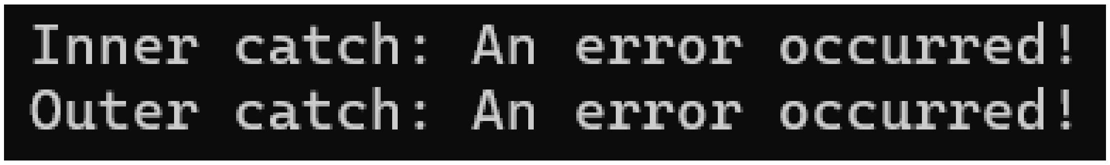
```
Choose exception to throw:
1. int
2. string
3. Other
Enter your choice: 3
Caught an exception of unknown or unexpected type!
```

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.

Source Code:

```
#include <iostream>
using namespace std;
int main() {
   try {
      try {
         throw runtime_error("An error occurred!");
      }
      catch (const runtime_error& e) {
         cout << "Inner catch: " << e.what() << endl;
         throw;
      }
   }
   catch (const runtime_error& e) {
      cout << "Outer catch: " << e.what() << endl;
   }
   return 0;
}
```

Output:



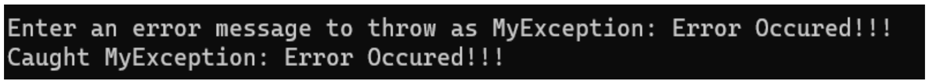
```
Inner catch: An error occurred!
Outer catch: An error occurred!
```

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.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;

class MyException {
   string message;
public:
   MyException(const string& msg) : message(msg) {}
   string getMessage() const { return message; }
};

int main() {
   try {
      string msg;
      cout << "Enter an error message to throw as MyException: ";
      getline(cin, msg);
      throw MyException(msg);
   } catch (const MyException& e) {
      cout << "Caught MyException: " << e.getMessage() << endl;
   }
   return 0;
}
```

Output:



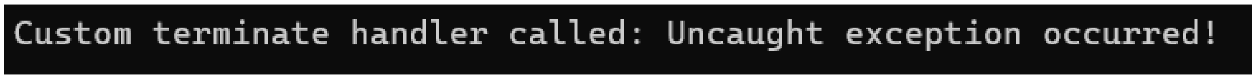
```
Enter an error message to throw as MyException: Error Occured!!!
Caught MyException: Error Occured!!!
```

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.

Source Code:

```
#include <iostream>
#include <exception>
using namespace std;

void myTerminateHandler() {
    cout << "Custom terminate handler called: Uncaught exception occurred!" << endl;
    exit(1);
}

int main() {
    set_terminate(myTerminateHandler);

    try {
        throw 3.14;
    } catch (int e) {
        cout << "Caught int exception: " << e << endl;
    }
    return 0;
}
```

Output:



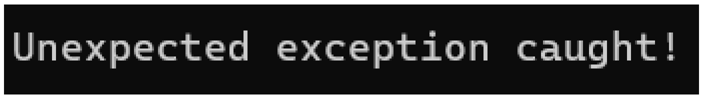
```
Custom terminate handler called: Uncaught exception occurred!
```

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.

Source Code:

```
#include <iostream>

#include <exception>

using namespace std;


void myUnexpected() {

   cout << "Unexpected exception caught!" << endl;

   exit(1);

}


void test() throw(double) {

   throw 42;

}

int main() {

   set_unexpected(myUnexpected);

   try {

      test();

   } catch(double d) {

      cout << "Caught double: " << d << endl;

   }

   return 0;

}
```

Output:



```
Unexpected exception caught!
```

8. Write a program to show exception handling inside a class constructor and destructor.

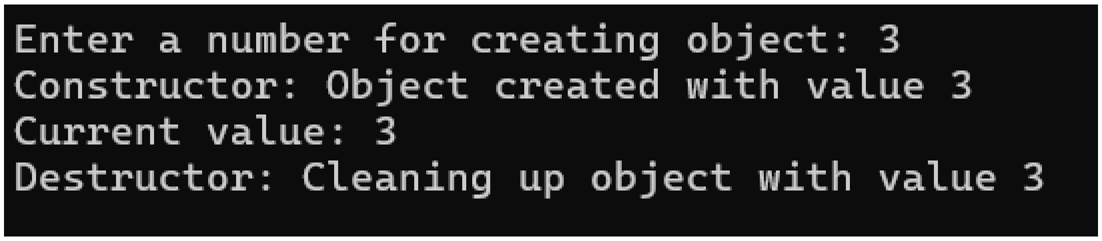
Handle constructor exceptions properly; avoid throwing from destructor.

Source Code:

```
#include <iostream>
#include <exception>
using namespace std;

class TestClass {
   int value;
public:
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;
}
}
~TestClass() {
cout << "Destructor: Cleaning up object with value " << value << endl;
}
void display() const {
```

```
cout << "Current value: " << value << endl;
}
};
int main() {
try {
int number;
cout << "Enter a number for creating object: ";
cin >> number;
TestClass object(number);
object.display();
}
catch (const string& e) {
cout << "Main caught exception: " << e << endl;
}
catch (...) {
cout << "Main caught unknown exception" << endl;
}
return 0;
}
```

Output:



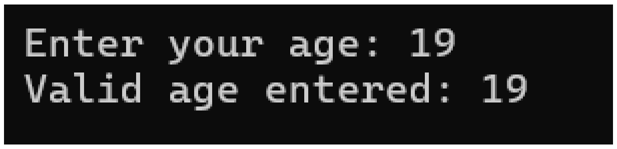
```
Enter a number for creating object: 3
Constructor: Object created with value 3
Current value: 3
Destructor: Cleaning up object with value 3
```

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.

Source Code:

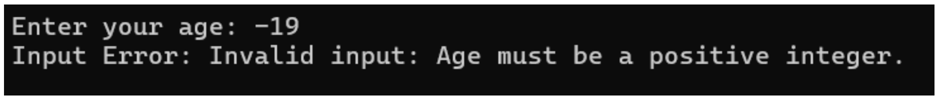
```
#include <iostream>
#include <string>
#include <stdexcept>
using namespace std;

int getValidAge() {
string input;
int age;
cout << "Enter your age: ";
cin >> input;
for (char ch : input) {
if (!isdigit(ch)) {
throw invalid_argument("Invalid input: Age must be a positive integer.");
}
}
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();
```

Output 1:



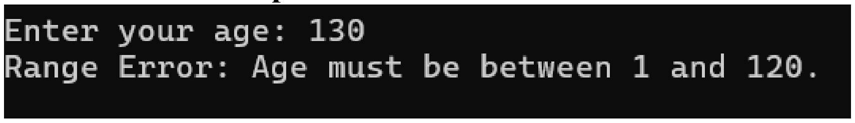
```
Enter your age: 19
Valid age entered: 19
```

Output 2:



```
Enter your age: -19
Input Error: Invalid input: Age must be a positive integer.
```

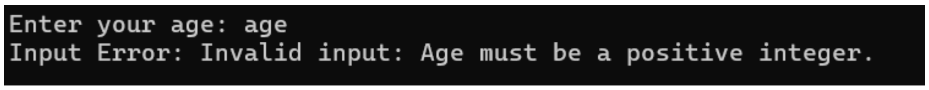
Output 3:



```
Enter your age: 130
Range Error: Age must be between 1 and 120.
```

```
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 4:



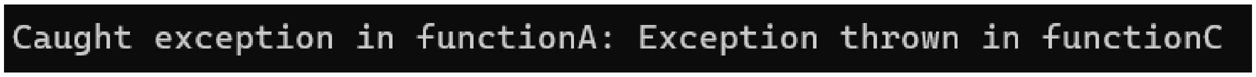
```
Enter your age: age
Input Error: Invalid input: Age must be a positive integer.
```

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

Source Code:

```
#include <iostream>
#include <string>
using namespace std;

void functionC() {
   throw string("Exception thrown in functionC");
}

void functionB() {
   functionC();
}
void functionA() {
    try {
        functionB();
    } catch (const string& e) {
        cout << "Caught exception in functionA: " << e << endl;
    }
}
int main() {
    functionA();
    return 0;
}
```

Output:



```
Caught exception in functionA: Exception thrown in functionC
```

DISCUSSION

In this lab, we learned about **Exception Handling** in C++. It's a special method for our programs to deal with problems that might while they are running, like trying to do something other than regular action. We learned to check if any action can trigger an error within the program using **try** keyword, how the program will **throw** an output if there's a problem, and how we can receive the output signal using **catch** keyword and handle it safely. This helps our programs avoid crashes and keeps them running smoothly, even when unexpected things happen.

CONCLUSION

In this lab, we learned how to make our C++ programs handle unexpected problems simply using **Exception Handling**. We practiced using special keywords like try to watch for errors, throw signals when a problem occurs, and trying to deal with that problem without stopping the program. By doing this, we saw how it helps our programs avoid crashes and makes them much more reliable and stronger, which is a key part of writing good programs.