UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

Throwing and catching exceptions

Douglas Wilhelm Harder, M.Math. LEL
Prof. Hiren Patel, Ph.D., P.Eng.
Prof. Werner Dietl, Ph.D.

1

Outline

- In this lesson, we will:
  - Review assertions
  - Be introduced to exception classes
  - See that exceptions can be thrown,
    and may terminate the execution of the program
  - Learn how to catch exceptions and deal with them

2

Assertions

- To this point, we have discussed assertions
  - Assertions are powerful and useful because:
    - They terminate the program immediately indicating the problem
    - They can all be turned *off*,
      meaning you can turn all assertions on during code development,
      but turn them off when you build your releases
  - Turning assertions off is as simple as including:

    ```
    #define NDEBUG
    ```
    - Not even the condition is performed,
      turning off assertions,
      it is equivalent to erasing all assertions before compilation

3

Assertions

- The issue with assertions, however, is in released code,
  they would automatically terminate the user's program
  - This does not make for happy clients or users…

"Nurse, turn on the artificial lung."
"Yes, doctor."

"Nurse, turn the artificial lung back on!"
"Doctor, it says something about "disk full"."

```
assertion "disk_full == false" failed: file "record_data.cpp",
line 666, fuction: int write_data()
Aborted (core dumped)
```

4

Assertions

- We need an approach to assertions that can more safely be used in deployed code
  - First, we will present how we can signal that there is a problem
    - We will *throw an exception*
  - Next, we will see how we can correctly deal with problems
    - We will also *catch exceptions*

The New Indian Express

The Cricket Monthly

Calling member functions

- When something unexpected happens, you can *throw* an exception
  - There are numerous classes that are exceptions all defined in the `stdexcept` library:

Logical errors	Run-time errors
`std::logic_error`	`std::runtime_error`
`std::domain_error`	`std::range_error`
`std::invalid_argument`	`std::overflow_error`
`std::length_error`	`std::underflow_error`
`std::out_of_range`	

Throwing exceptions

- Just like we return a value,
  we can throw an instance of one of these classes:

```
int isqrt( int n ) {
    if ( n < 0 ) {
        throw std::domain_error{
            "Cannot compute the square root of "
              + std::to_string( n )
        };
    } else {
        // Calculate the integer square root of 'n'
        //      ...
    }
}
```

Throwing exceptions

```
int isqrt( int n ) {
    if ( n < 0 ) {
        throw std::domain_error{
            "Cannot compute the square root of " + std::to_string( n )
        };
    } else {
        int result{0};
        int rem{0};

        for ( int k{15}; k >= 0; --k ) {
            rem <<= 2;
            rem += ( n >> (k << 1) ) & 3;
            result <<= 1;
            int sub{ (result << 1) | 1 };

            if ( sub <= rem ) {
               result |= 1;
               rem -= sub;
            }
        }

        return result;
    }
}
```

Throwing exceptions

- If you call this function with an invalid argument,
  like an assertion, the program terminates:

```
#include <iostream>

// Function declarations
int main();
int isqrt( int n );

int main() {
    std::cout << isqrt( -10 ) << std::endl;
    return 0;
}
```

Output:

```
terminate called after throwing an instance of 'std::domain_error'
  what():  Cannot compute the square root of -10
Aborted (core dumped)
```

Catching exceptions

- If all you do is throw exceptions,
  exceptions are no different than assertions that fail
  - The difference, however, is that we can *catch* exceptions
    - If an exception is caught, the program continues executing

Catching exceptions

```
int main() {
    try {
        std::cout << isqrt(  53209825 ) << std::endl;
        std::cout << isqrt(  95810392 ) << std::endl;
        std::cout << isqrt( 139502589 ) << std::endl;
    } catch ( std::domain_error &e ) {
        std::cout << "An exception was caught" << std::endl;
        std::cout << " - the error string was \"" << e.what()
                  << "\"" << std::endl;
    }

    std::cout << "Please try again..." << std::endl;

    return 0;
}
```

Output:

```
7294
9788
11811
Please try again...
```

Catching exceptions

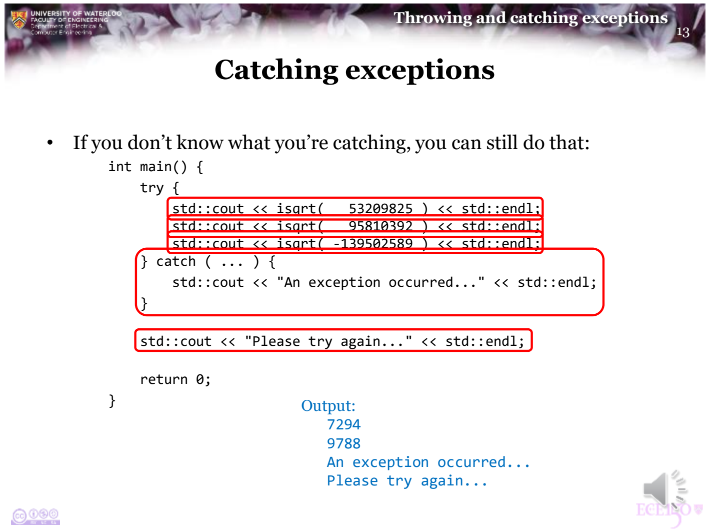
```
int main() {
    try {
        std::cout << isqrt(  53209825 ) << std::endl;
        std::cout << isqrt( -95810392 ) << std::endl;
        std::cout << isqrt( 139502589 ) << std::endl;
    } catch ( std::domain_error &e ) {
        std::cout << "An exception was caught" << std::endl;
        std::cout << " - the error string was \"" << e.what()
                  << "\"" << std::endl;
    }

    std::cout << "Please try again..." << std::endl;

    return 0;
}
```

Output:

```
7294
An exception was caught
 - the error string was "Cannot compute the square root of -95810392"
Please try again...
```



Catching exceptions

- If you don't know what you're catching, you can still do that:

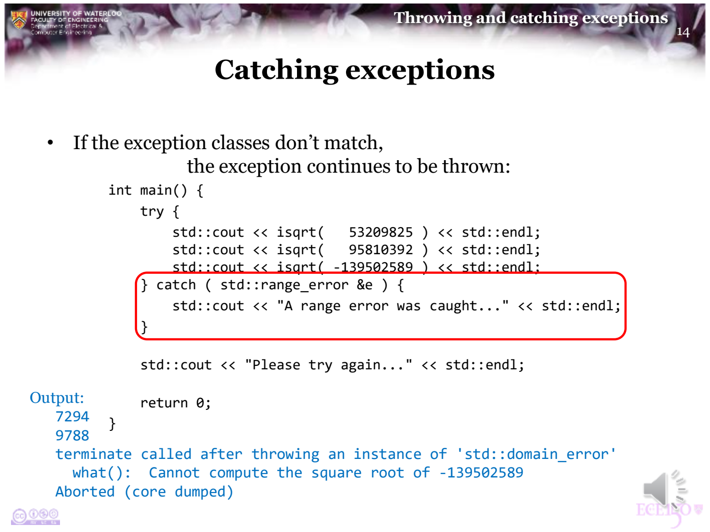
```
int main() {
    try {
        std::cout << isqrt(   53209825 ) << std::endl;
        std::cout << isqrt(   95810392 ) << std::endl;
        std::cout << isqrt( -139502589 ) << std::endl;
    } catch ( ... ) {
        std::cout << "An exception occurred..." << std::endl;
    }

    std::cout << "Please try again..." << std::endl;

    return 0;
}
```

Output:
```
7294
9788
An exception occurred...
Please try again...
```

13



Catching exceptions

- If the exception classes don't match, the exception continues to be thrown:

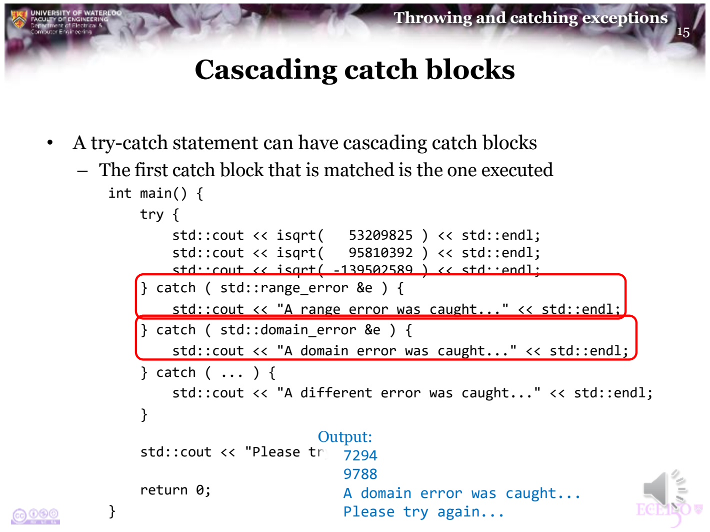
```
int main() {
    try {
        std::cout << isqrt(   53209825 ) << std::endl;
        std::cout << isqrt(   95810392 ) << std::endl;
        std::cout << isqrt( -139502589 ) << std::endl;
    } catch ( std::range_error &e ) {
        std::cout << "A range error was caught..." << std::endl;
    }

    std::cout << "Please try again..." << std::endl;

    return 0;
}
```

Output:
```
7294
9788
terminate called after throwing an instance of 'std::domain_error'
  what():  Cannot compute the square root of -139502589
Aborted (core dumped)
```

14



Cascading catch blocks

- A try-catch statement can have cascading catch blocks
  - The first catch block that is matched is the one executed

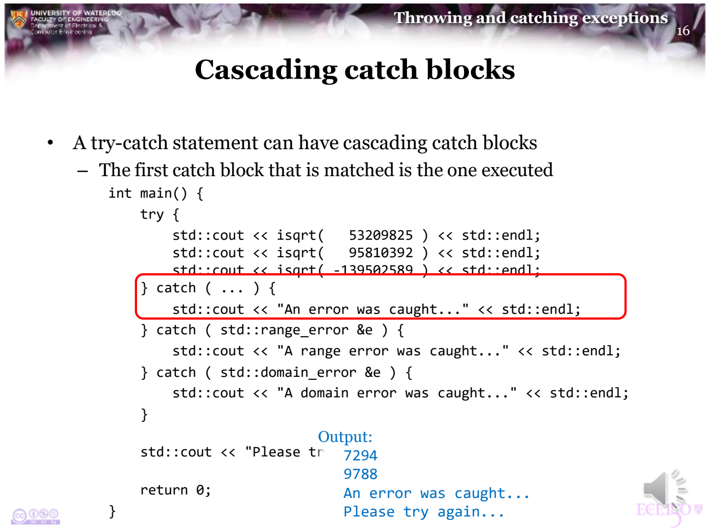
```
int main() {
    try {
        std::cout << isqrt(   53209825 ) << std::endl;
        std::cout << isqrt(   95810392 ) << std::endl;
        std::cout << isqrt( -139502589 ) << std::endl;
    } catch ( std::range_error &e ) {
        std::cout << "A range error was caught..." << std::endl;
    } catch ( std::domain_error &e ) {
        std::cout << "A domain error was caught..." << std::endl;
    } catch ( ... ) {
        std::cout << "A different error was caught..." << std::endl;
    }

    std::cout << "Please tr

    return 0;
}
```

Output:
```
7294
9788
A domain error was caught...
Please try again...
```

15



Cascading catch blocks

- A try-catch statement can have cascading catch blocks
  - The first catch block that is matched is the one executed

```
int main() {
    try {
        std::cout << isqrt(   53209825 ) << std::endl;
        std::cout << isqrt(   95810392 ) << std::endl;
        std::cout << isqrt( -139502589 ) << std::endl;
    } catch ( ... ) {
        std::cout << "An error was caught..." << std::endl;
    } catch ( std::range_error &e ) {
        std::cout << "A range error was caught..." << std::endl;
    } catch ( std::domain_error &e ) {
        std::cout << "A domain error was caught..." << std::endl;
    }

    std::cout << "Please tr

    return 0;
}
```

Output:
```
7294
9788
An error was caught...
Please try again...
```

16

Assertions or throwing exceptions?

- In this course, we will continue to use both assertions and throws
  - Assertions will be used to check that the code is correct:
    - For example, the conditions we expect to be true are actually true
  - Exceptions will be thrown if there are potential errors during the execution of the code that must be dealt with
    - For example, invalid arguments, or a result that is invalid
- Failure to deal with exceptions have led to numerous problems
  - The first Arian 5 rocket launch resulted in the destruction of the rocket a minute after launch due to an uncaught exception
  - The Clementine satellite simply allowed division-by-zero errors to cause the system to reboot

17

Why not use exceptions everywhere?

- Why not use exceptions everywhere?
  - Exceptions are expensive and require significant infrastructure
  - Suppose you have a try-catch statement:
    - You call a function in the try block
    - That function calls another function
    - That function calls another function, which calls another, which calls another,
      - Which finally throws an exception you are catching
    - Each called function has parameters and local variables on the call stack, and all this must be undone without executing any more statements in those functions
      - We must jump all the way back to the appropriate catch block…
- That is a lot of overhead…
  - Fine for programs running on general purpose desktop, laptop, etc.
  - Not so great for embedded or real-time systems

18

An application of a try-catch statement

- An example of using try-catch statement:
  - You set up a network connection
  - You then execute code that expects that network connection to be up
    - There may be a significant amount of code that is being executed
  - You don't want every single function called to deal with network failures…
  - Instead, if ever the network connection fails, the first function to detect this failure can throw an appropriate exception back to the top level where the corresponding catch block can either:
    - Try to re-establish the network connection, or
    - Signal that the network connection failed
  - If the network connection fails to be re-established, a user may be prompted what to do next, or the failure may simply be entered into an error log

19

Summary

- Following this lesson, you now
  - Have been exposed to the various exception classes
    - If an exception is thrown and not caught, the program terminates
  - Know that each exception class has a `what()` member function
  - Know that thrown exceptions inside try blocks can be caught with corresponding catch blocks
    - Like cascading conditionals, you can have cascading catch blocks
    - The `catch( ... )` is similar to an `if ( true )` condition
  - Know that once the first matching catch block is finished executing, the program keeps on executing as per normal
    - No evidence remains that something has occurred

20

References

[1] https://en.wikipedia.org/wiki/C++_classes
[2] https://en.wikipedia.org/wiki/Exception_handling
[3] https://www.cplusplus.com/reference/stdexcept/

21

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

22

Disclaimer

23