Lecture 19 — Exception Handling

J. Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

July 23, 2016

Acknowledgments: W.D. Bishop

Kinds of Exceptions

An exception is an instance of a detected run-time error.

Exceptions can be thrown by a line of code that indicates it.

This is when the keyword `throw` appears in the code. A programmer has explicitly `thrown` an exception.

There are also errors that occur without the appearance of the `throw` keyword. An example is a division by zero.

The line of code is

```
double result = input / divisor;
```

but an error, not an exception, will occur here if divisor is zero.

Exception Handling

The process of detecting an exception and attempting to correct the situation is known as exception handling.

Exceptions can be handled or unhandled:

- A handled exception is caught by a portion of the program and corrective action is taken to fix the situation
- An unhandled exception is not caught by a portion of the program and it results in the termination of the program

Undhandled Exception

Here is an example of an unhandled exception:

```
int main( )
{
    cout << "Beginning program..." << endl;
    throw "An unhandled exception has occurred.";
    cout << "This code is unreachable." << endl;
    return 0;

}
```

Handling an Exception

The keyword for handling an exception is `catch`.
(The opposite of `throw`.)

Much like someone who is trying to catch a ball in the air, the `catch` statement needs to be looking in the right direction.

A `catch` statement therefore is explicitly applied to a block of code.

Try-Catch

The syntax for exception handling is the `try-catch` statement.

```
try
{
    // Statement Block
}
catch( const char* e )
{
    // Exception Handling Block
}
```

The keyword `try` indicates what a `catch` is looking at.

The `catch` applies to the statements inside the `try` block.
It will catch an exception only if it comes from within the `try` block.

ExceptionType

In the syntax for `catch` there is: `const char* e`.

This is the general form, if we use just a simple text message to indicate what is wrong.

There are more specific exception types, to allow us to tell different sorts of errors apart.

In addition to saying where we are looking for an exception to be thrown, we also specify what kind of exception we want to `catch`.

Example: we could try to catch `runtime_error`.

C++ Important Exception Types

Some common Exception Types in C++:

Exception Name	Description
`exception`	General exception
`bad_alloc`	Error in allocating memory
`bad_cast`	Casting one type to another failed
`overflow_error`	A mathematical overflow occurred
`underflow_error`	A mathematical underflow occurred
`invalid_argument`	An argument to a function is invalid
`runtime_error`	Some other run-time error

Don't memorize this list, but you may use it for reference.

Try-Catch Example

Let's fill in the try-catch example from before.

```
int main() {
    try {
        int f = factorial( -1 );
        cout << f << endl;
    } catch ( const char* e ) {
        cout << e << endl;
    }
    cout << "Program Completed." << endl;
}
```

Assume that Factorial throws the exception string we used earlier.

Try-Catch Example Alternative

An alternative: we use the `invalid_argument` exception.

```
int main() {
    try {
        int f = factorial( -1 );
        cout << f << endl;
    } catch ( invalid_argument e ) {
        cout << e << endl;
    }
    cout << "Program Completed." << endl;
}
```

Try-Catch Execution

The only output was the error message.
    The statements to output `f` did not occur.

Like `return` or `break`, when an exception is encountered, execution of the current block is stopped at the point of the exception.

If an exception occurs, no further statements of the `try` block run.

The system will then look for a `catch` block matching the type of the exception that was encountered.

In the example above, there is a matching `catch` block.
    What happens if none is found?

Looking for Someone to Catch

The system will continue looking by going up a level to the function that called the function where the exception was encountered.

In the above example, an exception happened in `factorial()` but the `catch` block is found in `main`.

Looking for Someone to Catch

If the calling function lacks a matching `catch` block, the system goes up another level and repeats this procedure until it gets to `Main`.

If `Main` does not have one either, it's an unhandled exception.
The program terminates with an error message.

In both of the preceding examples, we found a `catch` block matching the type of the exception, so it is handled.

Try-Catch Example

eption occurs, control goes to the `catch` block and the statement there executes.

Then the next statement executed is after the end of the `catch` block.
Not the console writes.

Variable Scope in Try-Catch

Note that our usual rules of variable scope apply in a `try-catch`.

In the previous example `int f` is only in scope within the `try` block and not after the `try-catch` is done.

Like a function parameter, `invalid_argument e` is a parameter available in the `catch` block.

Right now, we won't make much use of `e`, but if you are feeling ambitious you can play around with what it contains.

Multiple Types of Exceptions

It is possible to `catch` more than one type of exceptions from a single `try` block.

The syntax and execution for this works like an `if-else` statement.

When an exception is encountered, the exceptions are checked in order from top to bottom, like the `if-else`.

The first `catch` that matches executes.

Only one of the `catch` blocks will execute (mutual exclusivity).

Multiple Types of Exceptions

```
try
{
   complicated_function( );
}
catch( invalid_argument e )
{
  cout << "Invalid Argument Detected!" << endl;
}
catch ( bad_cast e2 )
{
  cout << "Bad Cast Detected!" << endl;
}
catch ( runtime_error e3 )
{
  cout << "Run time error detected!" << endl;
}
```

Handling All Exceptions

It's possible to catch all exceptions using `catch (... )`

```
try
{
    // Try block
}
catch( ... )
{
    // All exceptions caught
}
```

This is sometimes referred to as "Pokémon Exception Handling" ("Gotta Catch 'Em All!").

What to Do with Exceptions

Sometimes, when an exception is encountered, the program can take some corrective action and deal with the problem in the `catch` block.

In this lecture we have mostly handled errors by simply reporting them and carrying on.

If the program is a calculator and it takes as input two numbers and an operation, one exception should not end the program.

In other cases, carrying on is harmful and we should stop execution of the program before anything else goes wrong.