Handling Exceptions

Exception Handling

- Deals with unusual circumstances that may require different reactions
- THIS IS NOT YOUR TYPICAL ERROR HANDLING
  - Handling bad input that you can predict will be common should not use Exception Handling
  - Save Exception Handling for edge cases
  - Some examples:
    - No permissions to access requested file
    - Input file is corrupted
    - Computer is low on memory and cannot allocate dynamic memory
    - Hard drive suddenly ran out of free space while writing a file

Terminology

- Code that encounters unexpected problem is said to “**throw**” an exception
- The user’s code is broken into two parts…
  - **try** block:

    This is where you put the code that could possibly throw an exception
  - **catch** block:

    Here is where you place the code that should run when an exception is encountered
- The try/catch structure can be used multiple times in your code
- A single **try** block can have multiple **catch** blocks to deal with different types of exceptions

try-throw-catch syntax

```
try {
     // this is where you place the code that might
throw an exception
     // if an exception is thrown, the try block will
immediately halt execution
     // and the catch block will begin to execute
} catch (type_of_exception) {
     // code to handle the exception
} catch (some_other_type_of_exception) {
     // code to handle the exception
}
```

Details

- If no exception is thrown, catch is ignored
- Can catch multiple exceptions, just have more catch blocks
  - **catch** blocks get executed in order of appearance (put more specific first)
  - **catch(…)** catches any exception and is considered a good default
- Common to define specialized exception class
  - Can create your own or use the generic C++ <exception> class
  - See demo code for an example of a custom exception class named **Swim_Exception**

Throwing Exceptions in Functions

- Usually **throw** in one function and **catch** in a different one
- Functions can have Exception Specification List
  - Tells the compiler which exceptions the function is expected to throw
  - Should appear in function declaration and definition
  - If more than one exception may be thrown, separate via comma
  - If an exception is thrown in the function but not listed in the Exception Specification List, unexpected() is called (terminates the program by default)
- Examples

  // Treat specified exceptions normally, all others unexpected()
  ```
  void someFunctionA() throw(NegativeNumber, DivideByZero, Swim_Exception);
  ```
  // List empty, treat all exceptions as unexpected();
  ```
  void someFunctionB() throw();
  ```
  // Treat all exceptions as expected
  ```
  void someFunctionC();
  ```

Example Usage w/ Exception Specification List

```
void functionA() throw (MyException) {
      …
      throw MyException(<Maybe an argument>)
      …
}
void functionB() {
      try {
            functionA();
      }
      catch(MyException e) {
            <Handle exception>
      }
}
```