Error Handling

Lecture Notes

Overview

- Errors occur (malformed input, missing resources, etc.)
- Errors can be:
    - Completely unexpected (fatal)
    - Anticipated as a possibility which may be dealt with in a specific manner
- May or may not be able to recover or *handle* the error
    - May not be able to handle it: code cannot reasonably be expected to fix a bad connection, find a missing file, etc.
    - By design we may *not want* to handle it
- Error Handling
    - Exception Handling
        - Exceptions are exceptions to the normal control flow of a program that may require special processing
        - Many programming languages support explicit exception handling:
            - Exceptions are a defined type (may have multiple types)
            - A try-catch-finally control structure may be supported
    - Defensive programming:
        - Prevent errors by making checks before executing dangerous code (division by zero
        - Allow errors to occur, but then report them using some mechanism
            - Encapsulate the behavior and error communication in a function
            - Calling function is responsible for checking if an error occurred and making the decision on what needs to be done (if anything)
            - Allows for a more flexible design (decision is on the user, not the language or the library)

Error Handling in C

1. No exceptions in C
    - C++ supports exception handling, but not C
    - Defensive programming : by convention, the user (programmer) is expected to prevent errors from occurring in the first place
    - Check for potential errors before you execute an unsafe statement (check for denominator being zero); handle the alternatives appropriately
    - Some functions return special flags upon failure (-1, 0, NULL) that should be checked

- In general: little that can be done to recover from general errors
- Assertions in C
    i. You can use an assert to catch bad input to a function
    ii. http://ptolemy.eecs.berkeley.edu/~johnr/tutorials/assertions.html
    iii. More of a debugging tool, cannot be caught

2. Error handling by design
    a. Standard Library
        - Standard C library provides a error number header file: errno.h
        - When included, the error result of any standard library call(s) are stored in a globally available variable called `errno`
        - But: only 3 errors are required to be defined:
            1. `EDOM` – results from a bad parameter (ex: sqrt(-1))
            2. `ERANGE` – results from a bad result (outside a function's range, ex: log(0.0))
            3. `EILSEQ` – results from an illegal byte sequence
        - An expanded example: (POSIX compliance) http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/errno.h.html
        - Errors can be converted to human-readable messages: `char * strerror(errno)`
        - Code example: `errorDemo.c`
    b. User defined errors
        i. Common convention: all function results are communicated via parameters
        ii. Return type is an `int` (or enumerated ERROR code)
        iii. User is responsible for checking for errors & handling
3. Exercises
    - Modify the quadratic root function to communicate errors to the calling function (define errors for division by zero, complex roots, null pointers, no error)
    - Write a temperature conversion function that simultaneously converts an input temperature to several different scales; identify potential errors and define and return error codes appropriately
    - Write a weight conversion function that simultaneously converts an input weight to several different scales; identify potential errors and define and return error codes appropriately

Error Handling in Java

1. Overview
    - Defensive programming is an option
    - Disadvantages:
        - Encapsulating error handling in a function means we are deciding for the client (calling code) how to handle errors; would be better to let them decide

    - No semantic meaning to using enumerated types for error messaging
    - A sequence of instructions could be predicated on the success of each other; leading to a huge nesting of checks (example: open a file, determine its size, allocate memory, read file, close file)
- Better strategy: communicate errors via *exceptions* and let client code *handle* them

2. Exceptions
    - An exception is an event that occurs that disrupts the normal control flow
    - In Java: `Throwable` -> {`Error` (fatal: should not be caught), `Exception`}
    - All exceptions must be explicitly thrown, caught, handled except `RuntimeExceptions` (and their subclasses)
    - Methods can be specified to throw certain exceptions
        - throws XXX, YYY
        - The caller is then *forced* to try-catch it
        - Explicit throws become part of the method signature!
    - Throwing exceptions
    - Creating exceptions (extending Exception or `RuntimeException`)
    - Use: Try-catch-finally
    - Java 7: try-with-resources (see http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html)
3. Best Practices
    - Java Tutorial puts it best:
      "Here's the bottom line guideline: If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception."
      However, this is not always true in the JDK! (`FileNotFoundException`, `SQLException`)
    - Personal opinion: never explicitly throw an exception; just document which may be thrown and when/why
    - Catch, tag, rethrow:
        - Often exceptions are fatal but some APIs and methods explicitly throw exceptions
        - Nothing can be done, so catch; log, and rethrow them wrapped in a runtime exception:

```
catch(Exception e) {
  System.err.println("An exception occurred: ");
  e.printStackTrace();
  throw new RuntimeException(e);
}
```

4. Exercises
    - Modify the quadratic root function to communicate errors to the calling function (define errors for division by zero, complex roots, null pointers, no error)

- Write a temperature conversion function that simultaneously converts an input temperature to several different scales; identify potential errors and define and return error codes appropriately
- Write a weight conversion function that simultaneously converts an input weight to several different scales; identify potential errors and define and return error codes appropriately