Chapter 8: Defensive Programming

All about adopting a mindset that you're never sure what other drivers are going to do (with respect to cars)

Take responsibility for protecting yourself even when its the other drivers fault

Protecting Your Program from Invalid Inputs

- Three general ways to handle garbage in:
    - 1) Check the values of all data from EXTERNAL SOURCES
        - Allowable ranges
        - Within tolerances
        - Strings short enough
        - Integer overflows
        - Buffer overflows
    - 2) Check the values of all routine INPUT PARAMETERS
        - Same as 1 but checking internal routine outputs
    - 3) Decide how to HANDLE bad inputs
        - once detected, what do you do with it?

Assertions

- Allows a program to check itself while it runs
- Especially useful in large, complicated and high reliability programs
- Guidelines:
    - Use Error-Handling for conditions you expect to occur
    - Use Assertions for conditions that should NEVER OCCUR
    - Treat assertions as declared assumptions about the data in the code
    - Avoid putting executable code (non value procedures) in assertions
    - Split assertions into
        - Preconditions
        - Postconditions

Error-Handling Techniques

- Return a neutral value
    - return a value that's known to be harmless
- Substitute the next piece of valid data
    - wait for next reading
    - go to next record in database
- Return the same answer as the previous time
- Substitute closest legal value
- Log a warning message to a file
- Return an error code
- Display an error message whenever the error is encountered

- Shut down
- **Be sure to follow same approach consistently through program**

Correctness vs Robustness

- Correctness means never returning an inaccurate result
    - no result is better than an inaccurate one
- Robustness means always trying to do something that will allow the software to keep operating
    - even if the results are inaccurate sometimes
- Safety-critical applications lean towards correctness
- Consumer applications tend to favor robustness

Exceptions

- Exceptions are specific means by which code can pass along errors or exceptional events to the code that called it
- Throw an exception only for conditionals that are truly exceptional
- **Use for events that should NEVER occur**
- Done use an exception to pass the buck --> handle error locally if can
- Avoid empty catch/except blocks --> do something with the error
- Standardize projects use of exceptions --> keep intellectually manageable
    - Standardize data types, object etc
    - Consider project specific error class
        - centralize and standardize logging and reporting
    - Define specific circumstances under which code is allowed to throw an exception that wont be handled locally
    - Consider whether program really needs to handle exceptions period.

Barricade Your Program to Contain the Damage Caused by Errors

- Zones of code that expect incoming data to be raunchy, and outgoing to be perfect for internal use
- Think of like operating room, need to sterilize the data

Debugging Aids

- Don't Automatically Apply Production Constraints in Development
- Be willing to trade speed and resource during dev in exchange for built in tools that can make developments go more smoothly
- Use Offensive Programming
    - "A dead program does a lot less damage than a crippled one"
    - Make sure asserts abort the program
        - Make the problem painful enough it will be fixed
    - Completely fill any memory allocated so you can detect alloc errors
    - Completely fill any files or streams allocated to flush out any file format errors
    - Be sure the code in each case statement default or else clause fails hard (aborts the program) or is otherwise impossible to overlook
    - Fill an object with junk data before its deleted

    - Setup the program to email error log files to yourself

Plan to Remove Debugging Aids

- User version control tools and build tools
- In dev, set build tool to include debug code
- In prod, set build tool to exclude

Determining How Much Defensive Programming to Leave in Production Code

- During dev, want errors to be loud and noticeable
- On prod, want to be as quiet as possible
- Leave in code that checks for important errors
    - Decide which areas of the program can afford to have undetected errors and which cannot
- Remove code that checks for trivial errors (with trivial consequences)
- Remove code that results in hard crashes
- On prod, give users the chance to save work before crashing
- Leave in code that helps programs crash gracefully
- Log errors for tech support personnel
- Make sure error messages are friendly