# ISA 563: Fundamentals of Systems Programming

Modularity and Information Hiding

Feb. 12, 2013

## Modularity

- Splitting the software in separate modules
    - Separation of concerns
    - Maintainability
    - Code re-use
    - Only modified modules need to be re-compiled
    - Easier to isolate bugs
    - Easier to edit

# Modularity (Cont'd)

- Goal:
  - Minimal dependency between modules
    - Ability to change one module without affecting others
    - Hide information as much as possible
    - Isolate implementation in logical, self-contained units

## Modularity in C

- No strong support for modularity in C
- A file represents a module
- However, we can use existing features to our benefit:
    - Header (.h) files for exporting function prototypes and common declarations
        - Represents a contract
        - Shares common declarations
    - Source (.c) files for implementations

## Demo

stack1.c

## stack1: analysis

- Everything in the same file
    - Hard to re-use implementation
    - Hard to test/debug separately
    - ...
- Suggested improvement:
    - Separate stack-related implementation

## Demo

main2.c, stack2.h, stack2.c

# Stack2: analysis

- Advantages:
  - Separate stack implementation as a module
    - Easier to re-use
    - Easier to test/debug
- Disadvantages:
  - Only one global stack that can be used
- Suggested improvement:
  - Allow multiple stack instances

## Demo

main3.c, stack3.h, stack3.c

## Stack3: analysis

- Advantages:
  - Modular
  - Allows multiple stack instances
- Disadvantages:
  - Still exposes the stack struct (information)
- Suggested improvement:
  - Hide all implementation details

## Demo

main4.c, stack4.h, stack4.c

# Stack4: analysis

- Advantages
  - Modular
  - Supports multiple stack instances
  - Implementation details completely hidden
- Uses “opaque pointers”
  - Hides structures
  - Even if we know the struct declaration, we cannot de-reference its members from outside