Goals of this Tutorial

The goal of this tutorial is to reinforce the following material:

- Declarations vs. Definitions
- Modular Code Design
- Debugging
Declarations vs. Definitions

// Declaration:
bool is_impossible(int power_level);

// Definition:
bool is_impossible(int power_level) {
    return (power_level > 9000);
}
Why?

- Reusability
- Maintainability
- Abstraction
Clients

- Client
  - requires
  - provides
    - interface
    - implementation
  - module
Interface vs. Implementation

We can provide the **interface** to the client. We can also hide the **implementation**.

Why?

- Keep code safe
- Ease of Use
- Hide sensitive information
- Flexibility to change implementation
/* header.h */

// This is a description of the file.

#include <stdio.h>
#include <stdbool.h>

bool predicate_fn(int n);
int side_effects_fn(int *p);

/* modules don't contain the main function! */
Example Program

Interface: impossible.h

#include <stdbool.h>

// is_impossible(power_level) determines whether the given
// power level is plausible.
bool is_impossible(int power_level);

Implementation: impossible.c

#include "impossible.h"

bool is_impossible(int power_level) {
    return (power_level > 9000);
}
#include "impossible.h"
#include <stdio.h>

int main(void) {
    if( is_impossible(9001) ) {
        printf("Over 9000?! That's impossible!\n");
    } else {
        printf("Oh yeah, that's possible.\n");
    }
}
Practice!-Modularization

We’ve provided a program that finds the posns within a certain range.

• In Seashell, separate the program into its modular parts

• Write a new I/O testing client
Practice!-Stack Use and Debugging

- Reimplement A2Q2b using the `stack.h` module given on A5 and iteration.

- This means you will implement a function that reads in an arbitrary amount of numbers and prints them in reverse order, adding the total count to each element.