Data Structures

Part I: Software Engineering Principles

BeomSeok Kim
Department of Computer Engineering
KyungHee University
passion0822@khu.ac.kr

Programming Life Cycle Activities

■ 문제 분석 (Problem Analysis)	■ 문제의 이해
■ 요구사항 정의 (Requirements Definition)	■ 프로그램이 수행할 내용 정의
■ 설계 (High- and Low-level Design)	■ 요구사항을 만족시킬 방법
■ 구현 (Implementation of Design)	■ 코딩
■ 검사 및 검증 (Testing and Verification)	■ 오류 발견 및 해결
■ 배포 (Delivery)	■ 고객에게 배포
■ 실행 (Operation)	■ 프로그램 사용
■ 유지보수 (Maintenance)	■ 프로그램 변경

Software Engineering

- 컴퓨터 프로그램의 설계, 생산, 유지보수를 위한 규율적 접근 방법
  A disciplined approach to the design, production, and maintenance of computer programs

  - ✓ 정해진 시간 안에 예측되는 비용을 통해 개발하는 방법
    A developing method on time within cost estimation

  - ✓ 소프트웨어 제품의 크기와 복잡도를 관리하는데 도움을 주는 도구
    Tools that help to manage the size and complexity of the resulting software productions

Algorithm

- 한정적 시간 안에 주어진 문제를 연산할 수 있는 완벽한 대책을 나타내는 비연속 단위의 논리적인 순서

  A logical sequence of discrete steps that describes a complete solution to a given problem computable in a finite amount of time

Goals of Quality software

- 동작되어야 함 (It works)

- 쉽고 효과적이게 수정이 가능해야함 (It can be modified without excessive time and effort)
  - ✓ 가독성이 뛰어나야하며, 이해 가능해야함
    It is readable and understandable

- 재사용 가능해야함 (It is reusable)

- 정해진 시간 안에 정해진 비용으로 완성되어야 함
  (It is completed on time and within budget)

Detailed Program Specification

- 프로그램이 무엇을 하는지 (Tells what the program must do, but not how it does it)
  - ✓ 명세에서는 어떻게 하는지에 대해서는 무시 (Ignores how the program do in the program specification)

- 프로그램에 대한 문서를 작성 (Is written documentation about the program)

Detailed Program Specification Includes

- Inputs
- Outputs
- Processing requirements
- Assumptions

Example:
Scenarios to understand the requirement: ATM

- 고객의 카드 삽입 (The customer inserts a bank card)
- 카드의 계좌정보 read (Reads the account # on the card)
- PIN 번호 요구 (Requests a PIN #)
- 고객이 555를 입력 (The customer enters 555)
- 계좌번호와 PIN의 조합을 검증 (Verifies the account # and PIN combination)
- 고객에게 다음 동작을 요구 (Asks the customer to select a transaction type)
- 고객이 선택 (The customer select)
- …..

Abstraction

- 복잡한 시스템의 모델은 시스템을 바라보는 관점에 초점을 두었을 때 시스템의 핵심적 특성을 포함
  (A model of a complex system that includes only the details essential to the perspective of the viewer of the system)

  ✓ 전체적인 시스템 개념을 쉽게 파악하여 구현할 수 있고 테스트할 수 있음
  (Easy to understand, implement and test the overall system concept)

Information Hiding

- 모듈이나 구조의 상세에 접근을 제어할 목적으로 함수나 자료구조의 상세를 은닉
  (Hiding the details of a function or data structure with the goal of controlling access to the details of a module or structure)

  - ✓ 저단계 설계의 상세 사항에 의존성이 있는 고단계 설계가 변경되는 경우를 피하기 위함
    (To prevent high-level designs from depending on low-level design details that may be changed)

  - ✓ 사용자 입장에서는 모듈과 구조의 접근이 중요
    (For users, access to modules and structures is important)

Two Approaches to Building Manageable Modules

절차지향적 팀 객체지향적 / 전력이 많이 필요

상단점

FUNCTIONAL DECOMPOSITION

Divides the problem into more easily handled subtasks, until the functional modules (subproblems) can be coded.

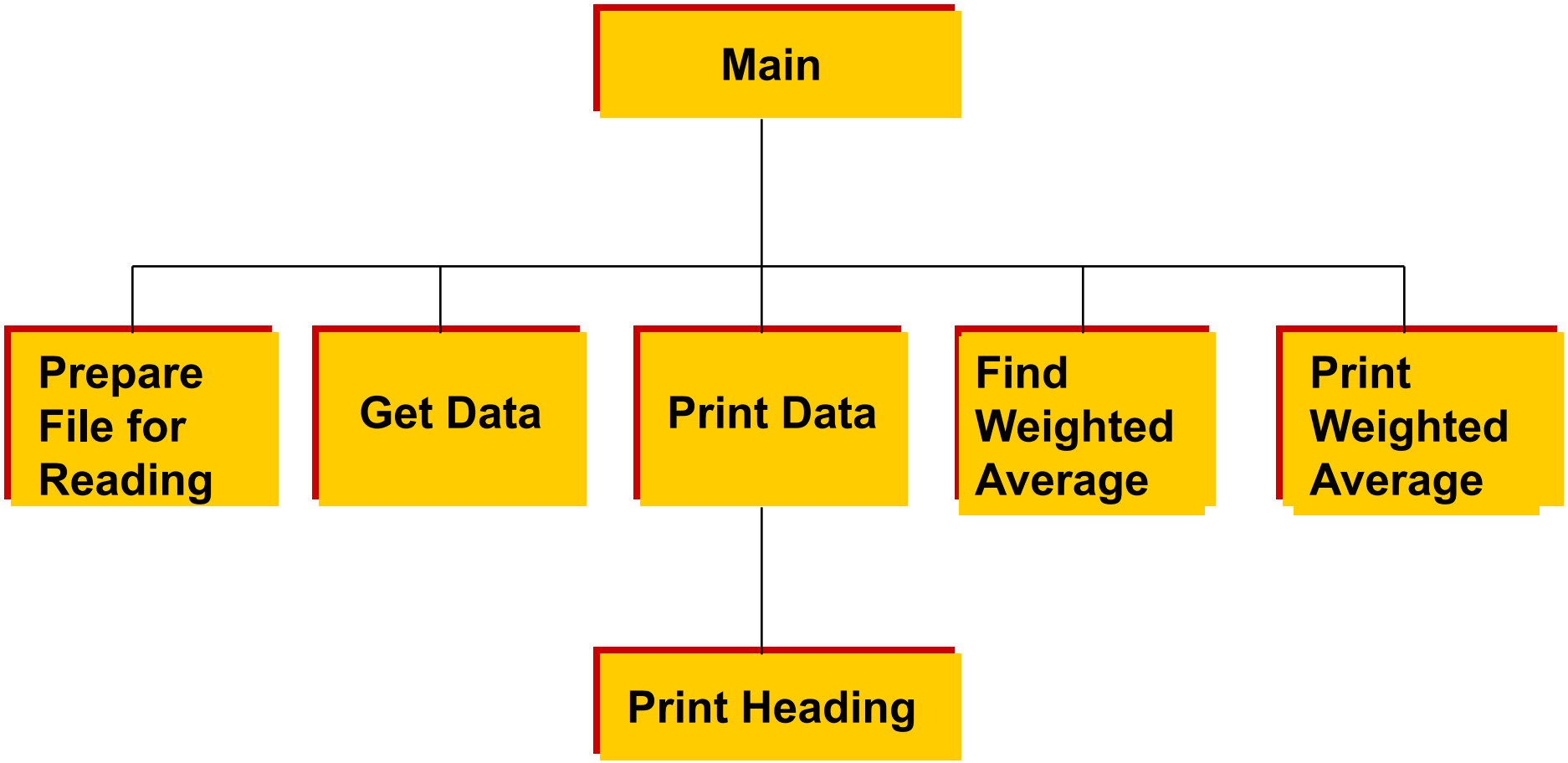
FOCUS ON: processes

OBJECT-ORIENTED DESIGN

Identifies various objects composed of data and operations, that can be used together to solve the problem.

FOCUS ON: data objects

Functional Design Modules

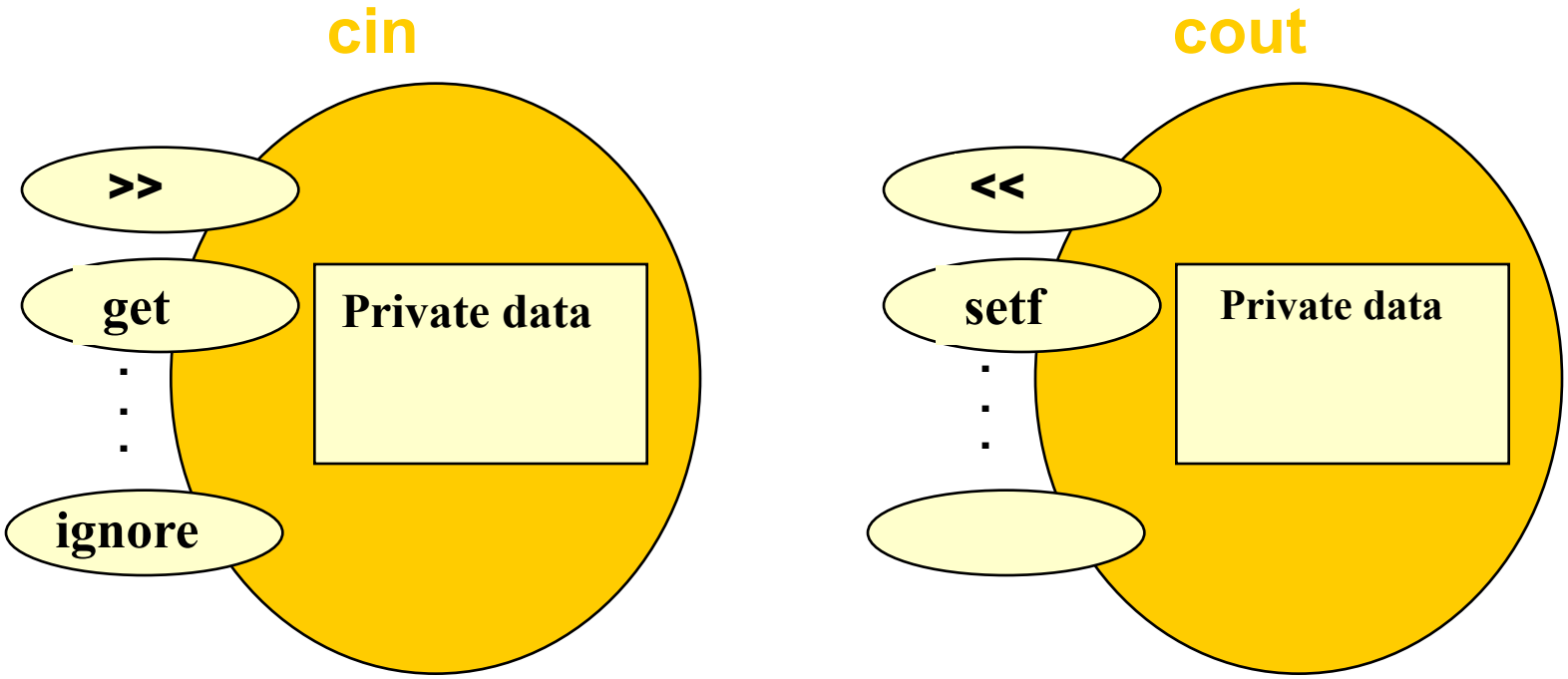


Object-Oriented Design

- 자료와 자료에 대한 연산들로 구성된 독립적인 요소인 객체로 표현하는 솔루션인 프로그램을을 개발하는 기술
  (A technique for developing a program in which the solution is expressed in terms of objects -- self-contained entities composed of data and operations on that data)

절차 지향에 비해

  - ✓ 독립적인 객체들로 구성되어 가독성, 재사용성, 확장성이 좋아진다.
    (For better readability, reusability, and extensibility)



More about OOD

- OOD를 지원하는 언어: C++, Java, Smaltalk, Object-Pascal, etc.
  (Languages supporting OOD include: C++, Java, Smalltalk, and Object-Pascal)

- 클래스는 프로그래머 정의 데이터 형식이며, 객체는 해당 형식의 변수
  (A class is a programmer-defined data type and objects are variables of that type)

- C++에서 cin은 istream이라는 데이터 유형 (class)의 객체이며, cout은 ostream 클래스의 객체임
  (In C++, cin is an object of a data type (class) named istream, and cout is an object of a class ostream)
  - ✓ 헤더파일 iostream 및 fstream에는 stream 클래스 정의가 포함됨
    (Header files iostream and fstream contain definitions of stream classes)

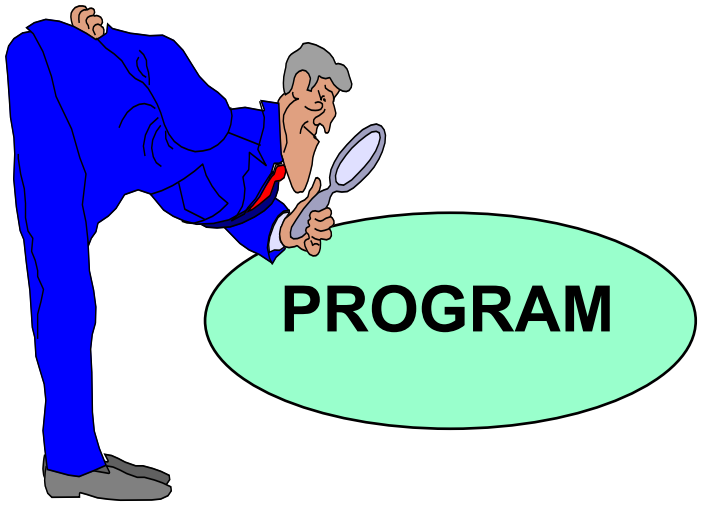
Procedural vs. Object-Oriented Code

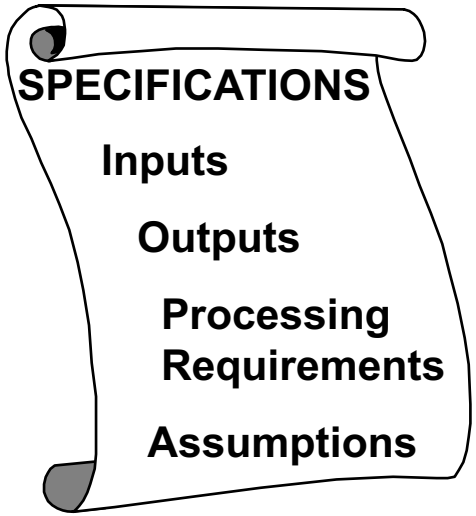
"Read the specification of the software you want to build. Underline the verbs if you are after procedural code, the nouns if you aim for an object-oriented program."

Grady Booch, "What is and Isn't Object Oriented Design," 1989.

Program Verification

- 프로그램 검증은 소프트웨어 제품이 사양을 충족시키는 정보를 결정하는 프로세스
  (Program Verification is the process of determining the degree to which a software product fulfills its specifications)





Verification vs. Validation

프로그램 구현검증

Program verification asks,

"Are we doing the job right?"

결과

Program validation asks,

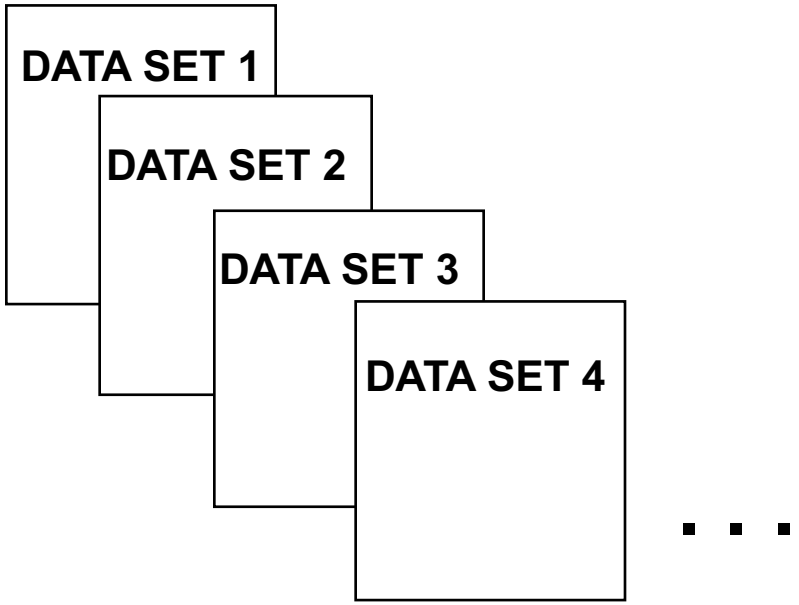
"Are we doing the right job?"

과정

B. W. Boehm, Software Engineering Economics, 1981.

Program Testing

- 테스트는 오류를 발견하도록 설계된 다양한 데이터 셋으로 프로그램을 실행하는 프로세스
  (Testing is the process of executing a program with various data sets designed to discover errors)



Various Types of Errors

- **설계오류:** 명세가 잘못된 경우
  (**Design errors** occur when specifications are wrong)

- **컴파일오류:** 문법이 잘못된 경우
  (**Compile errors** occur when syntax is wrong)

- **런타임오류:** 잘못된 가정, 프로그래밍 언어에 대한 불완전한 이해 또는 예기치않은 사용자 오류로 인한 결과
  (**Run-time errors** result from incorrect assumptions, incomplete understanding of the programming language, or unanticipated user errors)

Robustness

- 견고성은 프로그램이 오류 발생 후 복구하는 능력이며, 프로그램이 동작하는 환경 내에서 프로그램을 영속적으로 운영할 수 있는 능력
  (Robustness is the ability of a program to recover following an error; the ability of a program to continue to operate within its environment)

Preconditions and Postconditions

- **전제조건**은 실행을 시작하기 전에 함수가 참이어야 하는 명제를 설정하는 가정
  (The **precondition** is an assertion describing what a function requires to be true before beginning execution)

- **사후조건**은 함수가 실행을 완료하는 순간에 무엇이 참이어야 하는지를 설명
  (The **postcondition** describes what must be true at the moment the function finishes execution)

- **호출자**는 전제조건을 보장할 책임이 있으며, 함수 코드는 사후조건을 보장해야 함
  (The **caller** is responsible for ensuring the precondition, and the function code must ensure the postcondition)

Preconditions and Postconditions

```
void  GetRoots (float  a,  float  b,  float  c,
                        float&  Root1,  float&  Root2 )
//  Pre:   a, b, and c are assigned.
//           a is non-zero,  b*b - 4*a*c is non-zero.
//  Post:    Root1 and Root2 are assigned
//         Root1 and Root2 are roots of quadratic with coefficients a, b, c
{
      using namespace std;
      float    temp;
      temp = b * b - 4.0 * a * c;
   Root1 = (-b + sqrt(temp) ) / ( 2.0 * a );

      Root2 = (-b - sqrt(temp) ) / ( 2.0 * a );
      return;
}
```

A Walk-Through and Inspection

- 대략적 검사: 팀을 구성하여 시험 입력을 가지고 칠판 및 종이에 손으로 쓰면서 자료를 추적함으로써 프로그램의 요구사항, 설계, 구현방법 등을 대략적으로 검사하는 방법
  (Walking-through: A verification method using a team to perform a manual simulation of the program or design, using sample test inputs, and keeping track of the program's data by hand)

- 정밀검사: 한 멤버가 프로그램이나 디자인을 한 줄씩 읽고 다른 멤버가 오류를 지적하여 확인하는 방법
  (Inspection: A verification method in which one member reads the program or design line by line and the other members point out error)

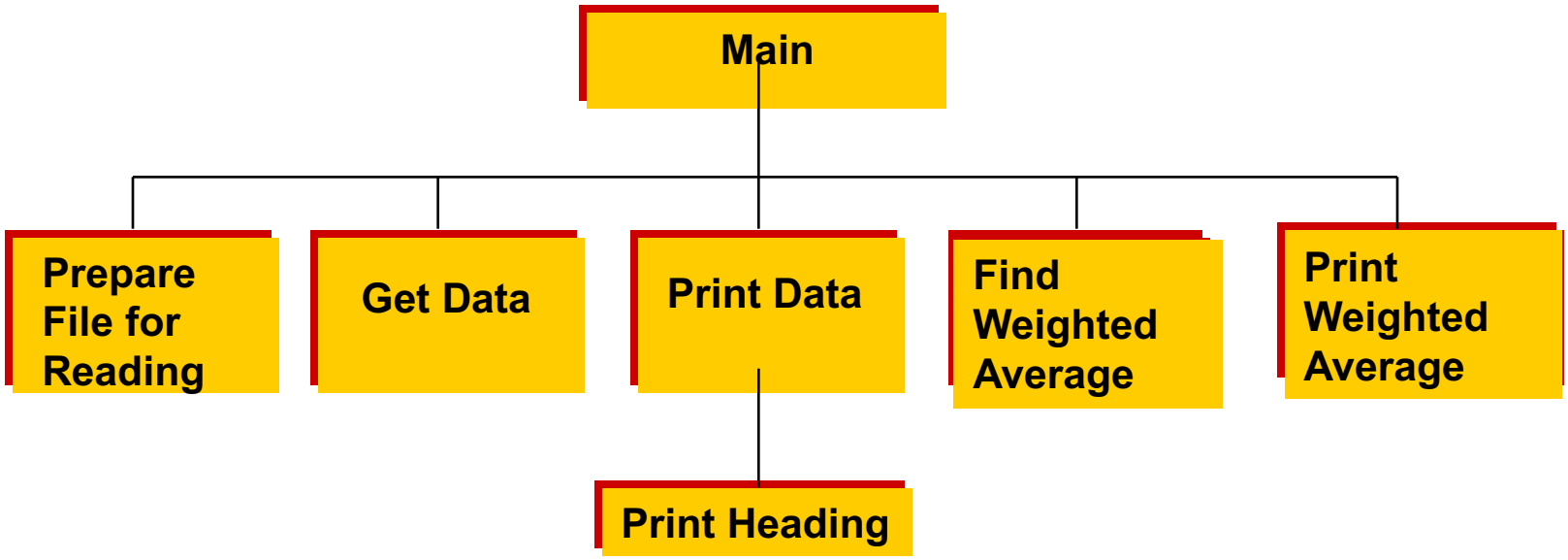
- 검사의 목적은 프로그래머의 디자인 또는 구현에 대한 토론을 유도하는 것
  (Its purpose is to stimulate discussion about the programmer's design or implementation)

Tasks within each test case:

- determine *inputs* that demonstrate the goal.
- determine the *expected behavior* for the input.
- run the program and *observe results*.
- *compare* expected behavior and actual behavior. If they differ, we begin debugging.

Integration Testing

- 독립적으로 단위 테스트를 거친 프로그램 모듈을 통합하기 위해 수행
  (Is performed to integrate program modules that have already been independently unit tested)



Integration Testing Approaches

큰거부터

TOP-DOWN

Ensures correct overall design logic.

USES: placeholder module "stubs" to test the order of calls.

작은거부터

BOTTOM-UP

Ensures individual modules work together correctly, beginning with the lowest level.

USES: a test driver to call the functions being tested.

Thank You!
Q&A