

Computer Programming

Lecture 7

이윤진
서울대학교
2007.1.10.

Slide Credits

- 엄현상 교수님
 - 서울대학교 컴퓨터공학부
 - Computer Programming, 2007 봄학기

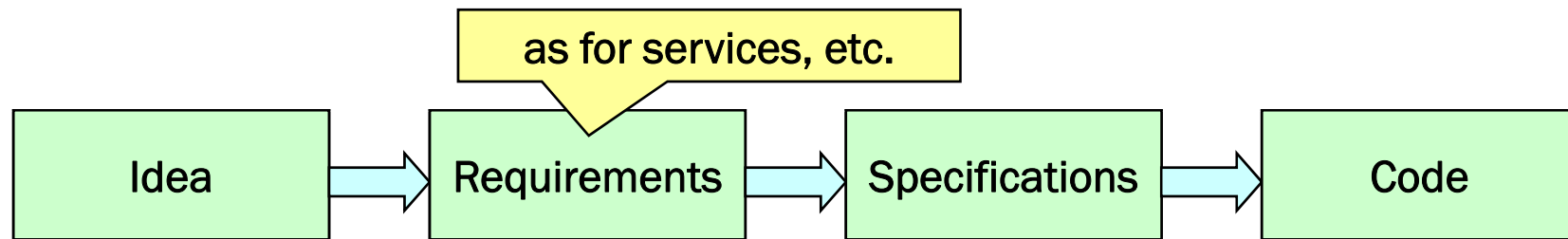
Modularity and Abstraction in C

순서

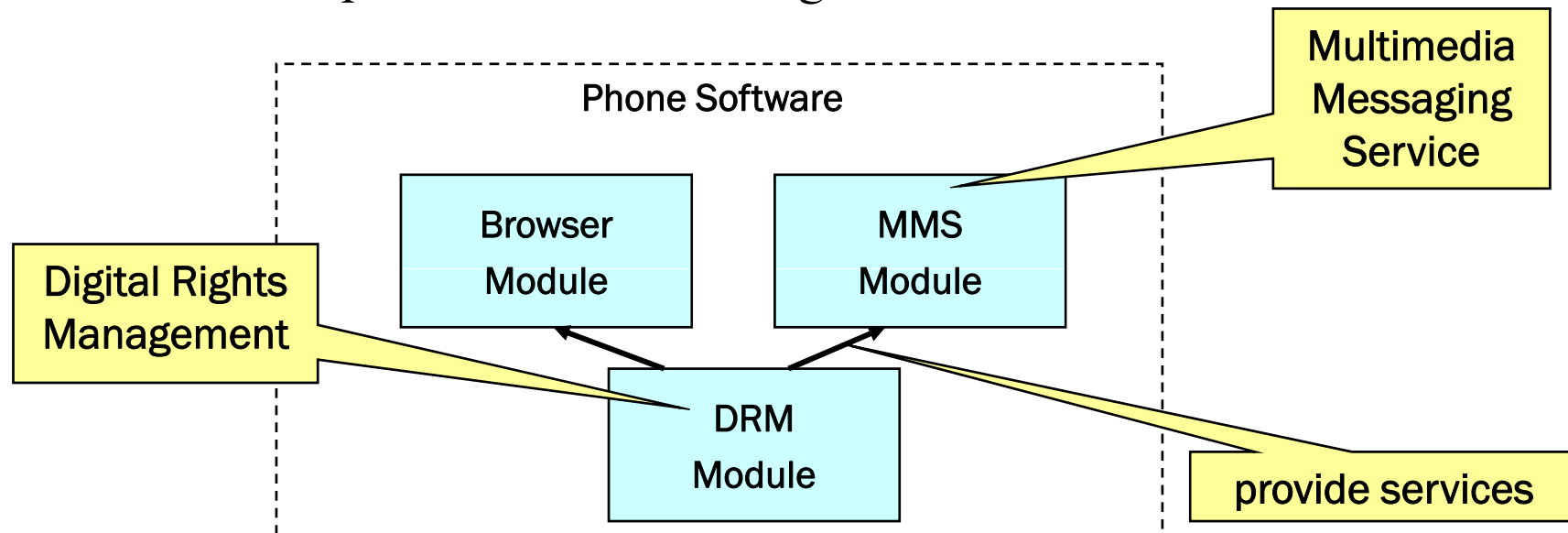
- Modularity & Abstraction w/ C Examples
 - Modules
 - Dividing a Program into Modules
 - Module Types
 - Information Hiding
 - Abstraction
- Q&A

Modular Software Design

- Standardized Software Design & Development

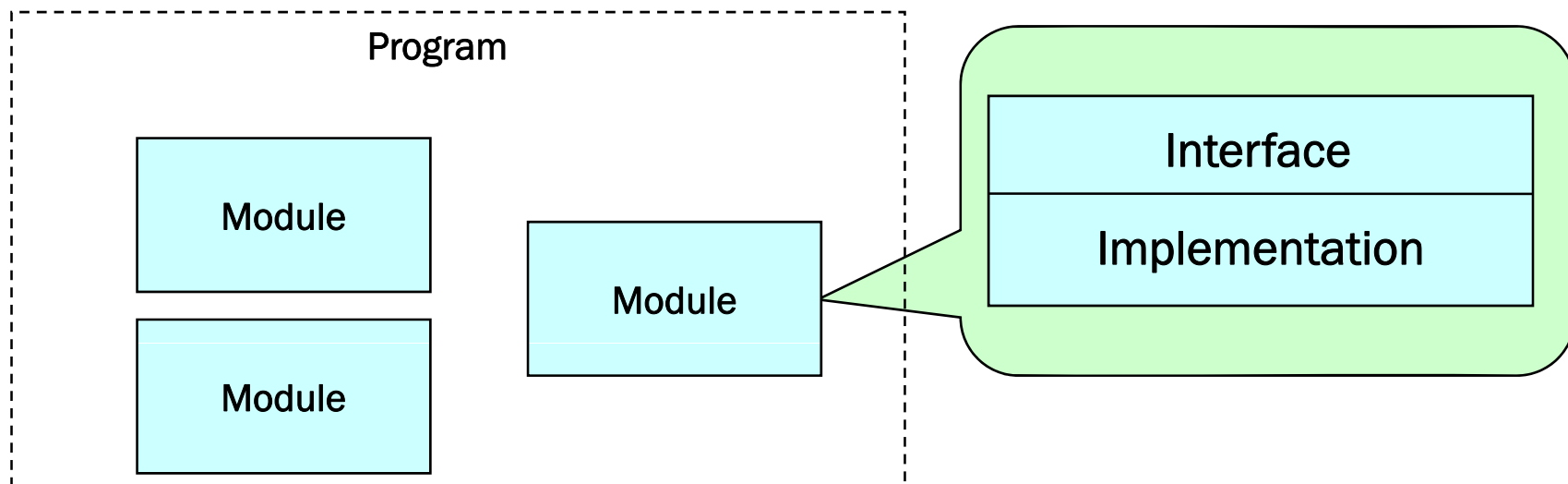


- Example of Modular Design



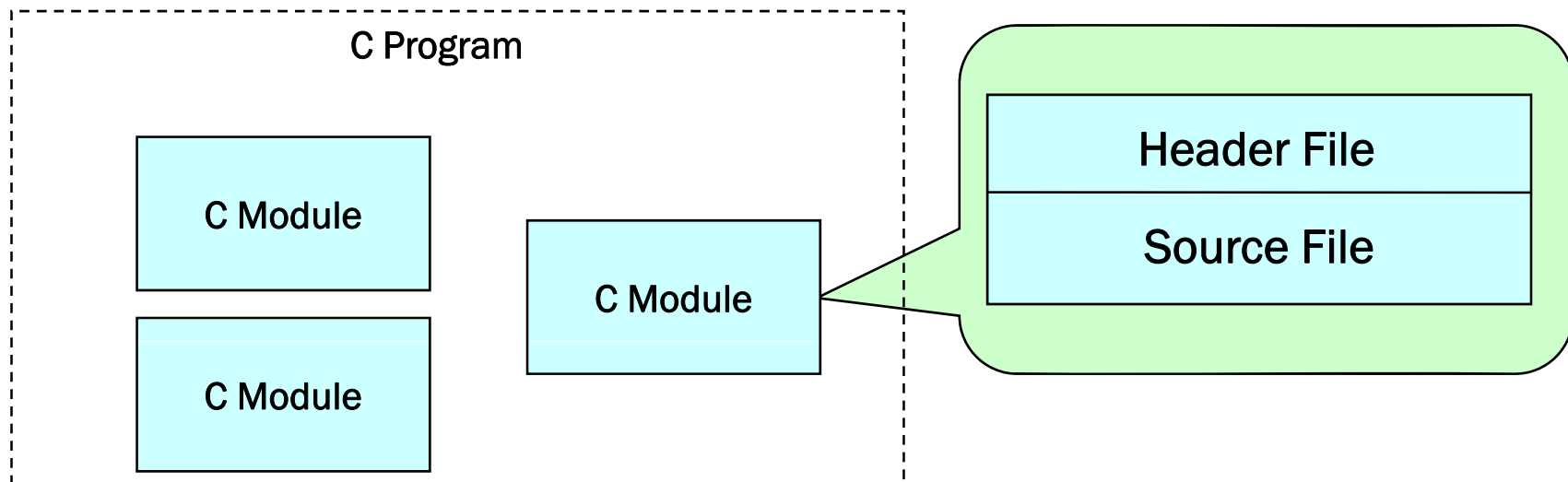
Modules

- Program
 - *Independent* Modules
- Module
 - Collection of Services
 - Interface: description of available services
 - Implementation: detailed definitions of services

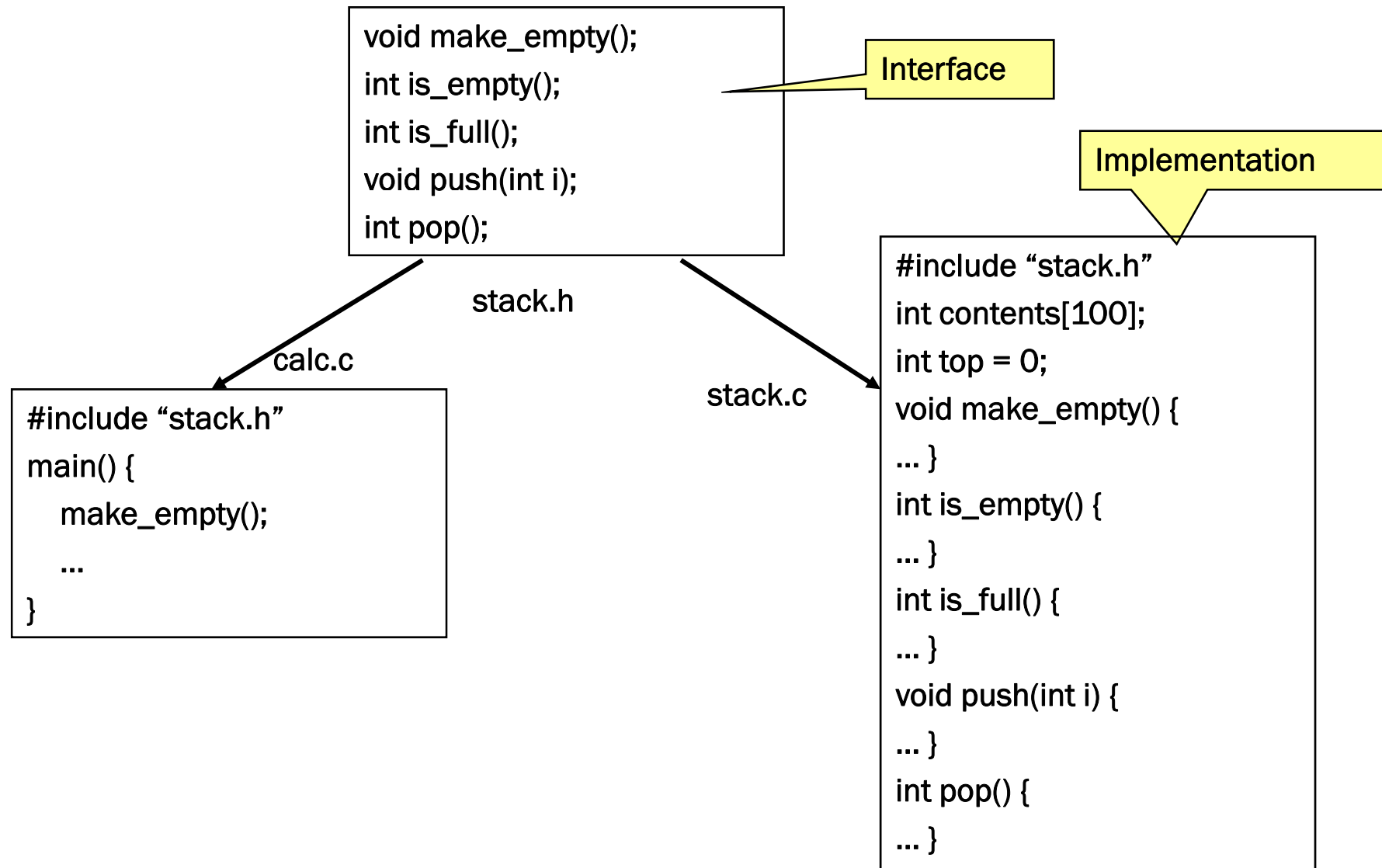


C Modules

- C Program
 - *Independent C Modules*
- C Module
 - Collection of Functions
 - Interface: a header file containing prototypes
 - Implementation: a source file containing definitions



Module Example



Dividing a Program into Modules

- Advantages
 - Abstraction
 - What they do; interface
 - Reusability
 - Reusable services
 - Maintainability
 - Module-wise maintenance
- Considerations
 - High Cohesion
 - Cooperating towards a common goal
 - Low Coupling
 - Indenpendence

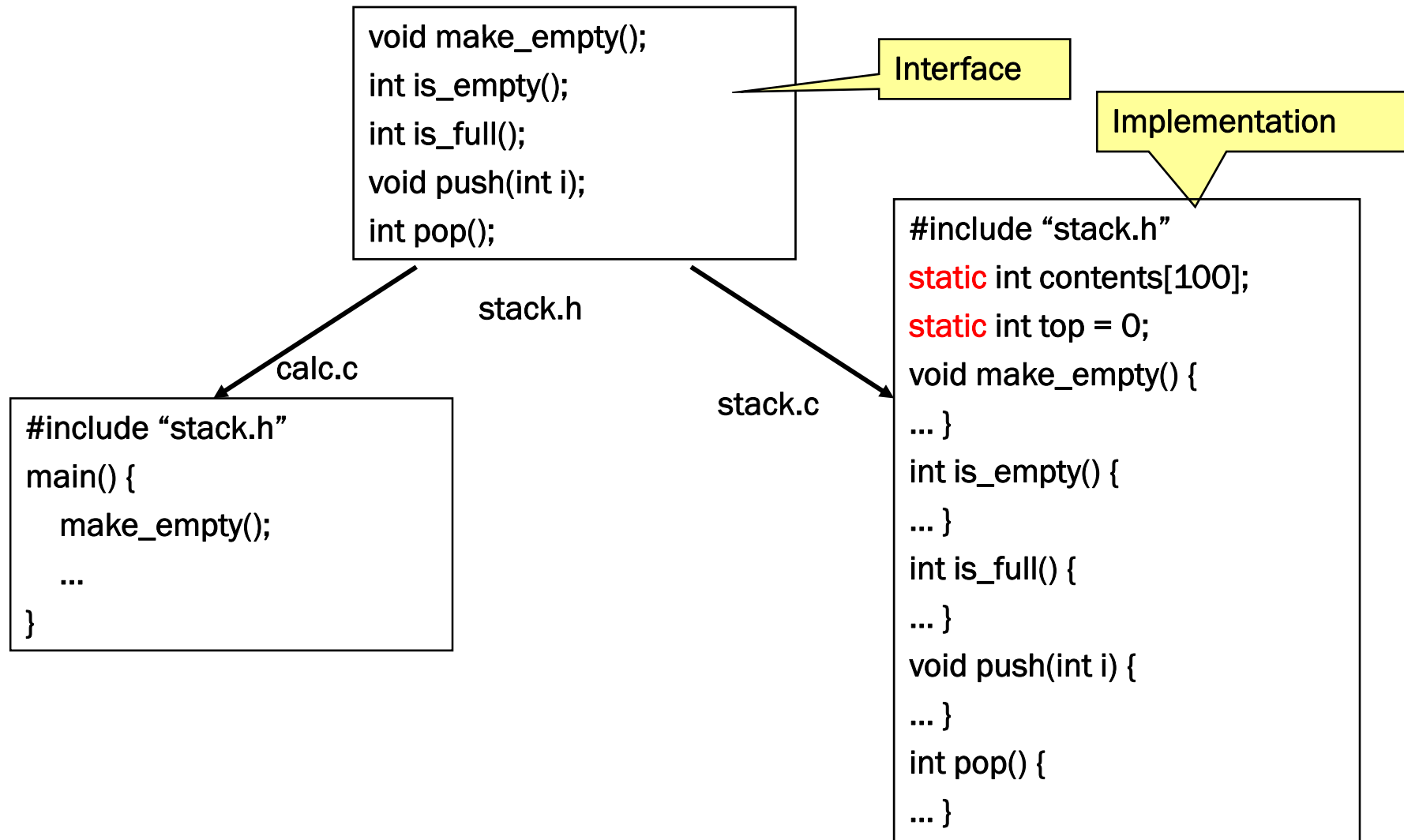
Module Types

- Data Pool
 - Collection of Related Variables and/or Constants
 - e.g., <limits.h>
- Library
 - Collection of Related Functions
 - e.g., <string.h>
- Abstract Object
 - Collection of Functions That Operate on a *Hidden* Data Structure
 - e.g., stack module
- Abstract Data Type (ADT)
 - Type with Its Representation *Hidden*

Information Hiding

- Advantages
 - Security
 - Only access to public information possible
 - Flexibility (Due to Abstraction)
 - Separation of interface from implementation
- C Tool
 - **Static**
 - Static functions callable within the file
 - Static variable accessible within the file/function

Module as an Abstract Object



Module as an ADT

Interface

Implementation

```
typedef struct _stack *pStack;  
pStack create();  
void make_empty(pStack pS);  
int is_empty(const pStack pS);  
int is_full(const pStack pS);  
void push(pStack pS, int i);  
int pop(pStack pS);
```

stackADT.h

calcADT.c

```
#include "stackADT.h"  
main() {  
    pStack pS = create();  
    make_empty(pS);  
    ...  
}
```

stackADT.c

```
#include "stackADT.h"  
struct _stack {  
    int contents[100];  
    int top;  
};  
pStack create() {  
    ...  
}  
void make_empty(pStack pS) {  
    ...  
}  
int is_empty(pStack pS) {  
    ...  
}  
int is_full(pStack pS) {  
    ...  
}  
void push(pStack pS, int i) {  
    ...  
}  
int pop(pStack pS) {  
    ...  
}
```

Abstraction

- Definition
 - Process of Separating the Qualities of Something from the Object That They Belong to
 - Separation of *what* from *how*
 - e.g., C variables
- Major Types
 - Procedural Abstraction
 - Separation of *what* a function does from *how*
 - e.g., function outline & algorithm
 - Data Abstraction
 - Separation of *what* is stored (data object and its operators) from *how*
 - e.g., C data types

Abstraction (계속)

- Advantages
 - Reduced Complexity
 - Information Hiding
 - Flexibility
 - Reusability
- Level
 - How to Determine It?