POINTER SUPPLEMENT POINTER SUPPLEMENT POINTER SUPPLEMENT

엄현상(Eom, Hyeonsang) School of Computer Science and Engineering Seoul National University

©COPYRIGHTS 2019 EOM, HYEONSANG ALL RIGHTS RESERVED

Pointer

- Pointer Operators
 - &: 'Address of' Operator
 - Get the address of the variable
 - *: 'Indirect' Operator
 - Get the value at the address

```
p=&var;
*p = 3;
```

• Result
var
p

Pointer Example

```
int i, j, *p1, *p2;
i = 7;
p1 = &i;
printf("%d \ n",*p1);
printf("%d \ n",*&i);
j = 10;
p2 = &j;
*p1 = 3;
*p2 = *p1;
printf("%d, %d \ n", i, j);
printf("%x, %x \ n", &i, p1);
```

Result

```
7
7
3, 3
bffffc98
```

Self-Referential Structure

 Structure w/ a Pointer Pointing to a Structure of the Same Type

```
struct node {
  int data;
  struct node *next;
};
```

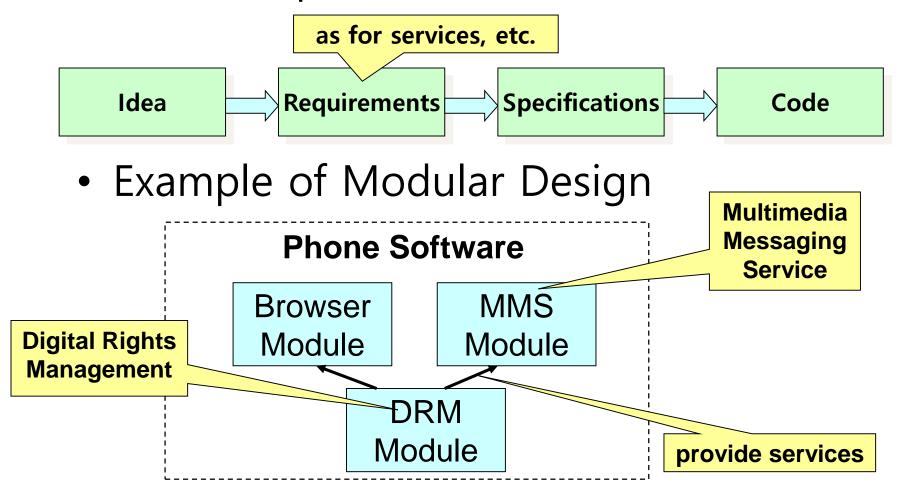
Example

```
data next
```

```
struct node a, b;
a.data = 1;
b.data = 2;
a.next = b.next = NULL; /* Pointing to nothing */
a.next = &b;
printf("%d \ n", a.next->data);
```

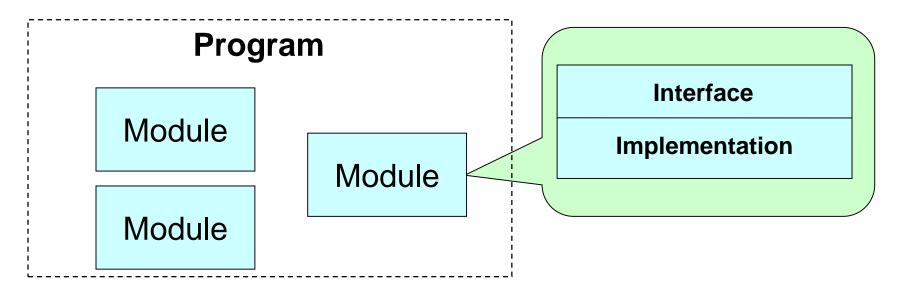
Modular Software Design

 Standardized Software Design & Development



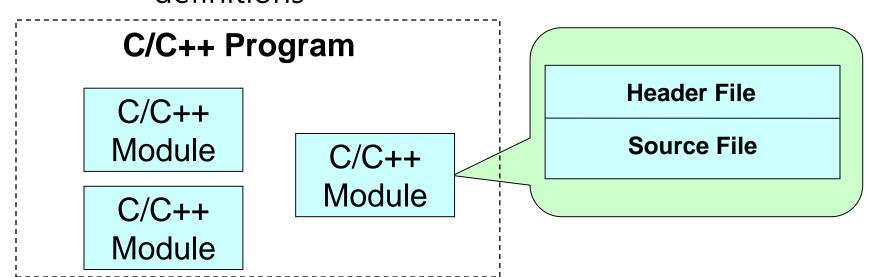
Modules

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



C/C++ Modules

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



Module Example

```
void make_empty();
                                                  Interface
                  int is_empty();
                   int is_full();
                                                               Implementation
                   void push(int i);
                   int pop();
                                                   #include "stack.h"
                          stack.h
                                                    int contents[100];
                                                    int top = 0;
              calc.c
                                        stack.c
                                                    void make_empty() {
#include "stack.h"
main() {
                                                    int is_empty() {
  make_empty();
                                                    int is_full() {
                                                    void push(int i) {
                                                   int pop() {
```

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
 - Independence

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/C++ Tool
 - Static
 - Static functions callable within the file
 - Static variable accessible within the file/function

Module as an Abstract Object

```
void make_empty();
                                                  Interface
                   int is_empty();
                   int is_full();
                                                             Implementation
                   void push(int i);
                   int pop();
                                                    #include "stack.h"
                          stack.h
                                                    static int contents[100];
                                                    static int top = 0;
               calc.c
                                        stack.c
                                                    void make_empty() {
#include "stack.h"
main() {
                                                    int is_empty() {
  make_empty();
                                                    int is_full() {
                                                    void push(int i) {
                                                    int pop() {
```

Module as an ADT

```
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);
```

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

calcADT.c

stackADT.c

Interface

Implementation

```
#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/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/C++ data types

Abstraction Cont'd

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