Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Module Programming

Module Programming

- Module
    - a collection of functions that perform related tasks
- How to create good modules
    - use an infinite array example to explain
- Modules are divided into two parts
    - public
    - private

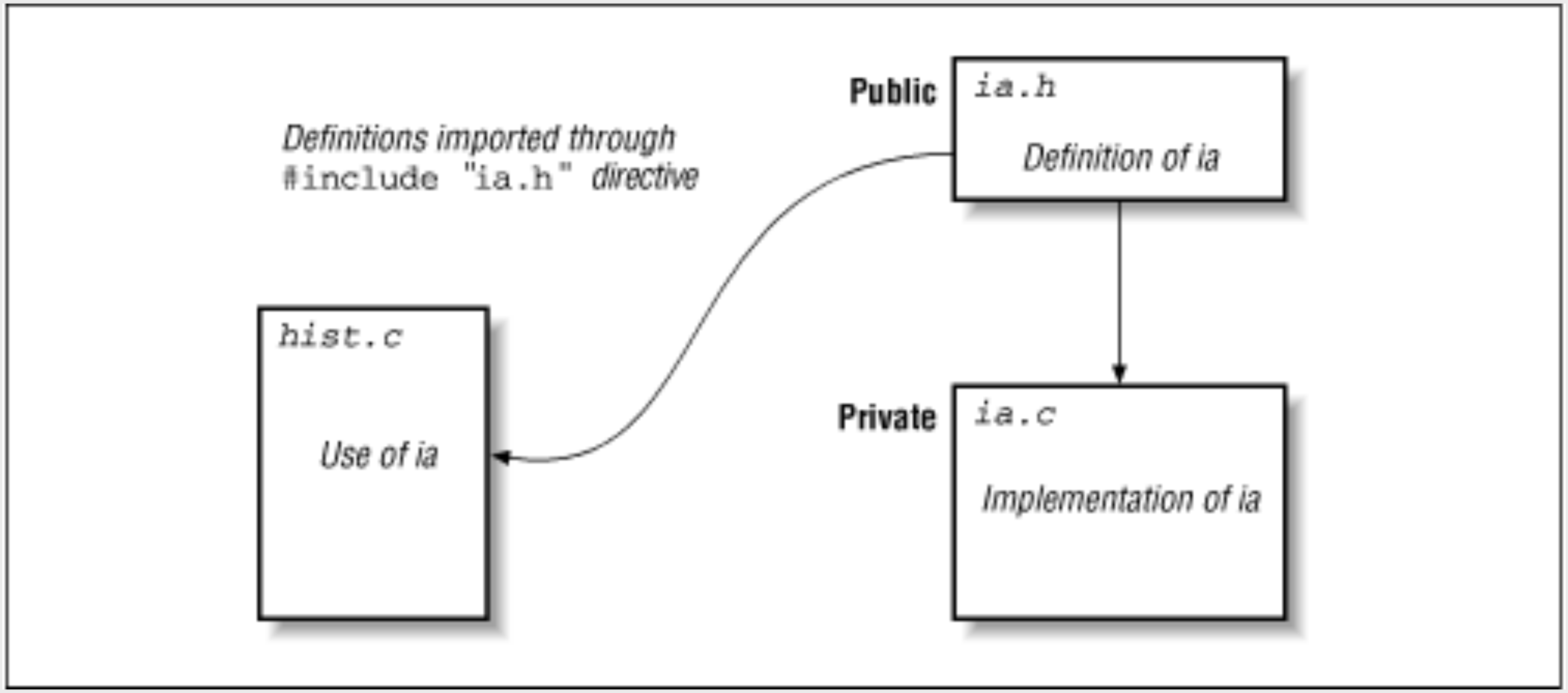
Public and Private

- **`public`**
  - tells users how to call the function in the module
  - contains the definition of data structures and functions that are to be used outside the module
  - puts in a header file (.h)
- **`private`**
  - anything that is internal to the module is private
  - puts in a source file (.c)

Headers

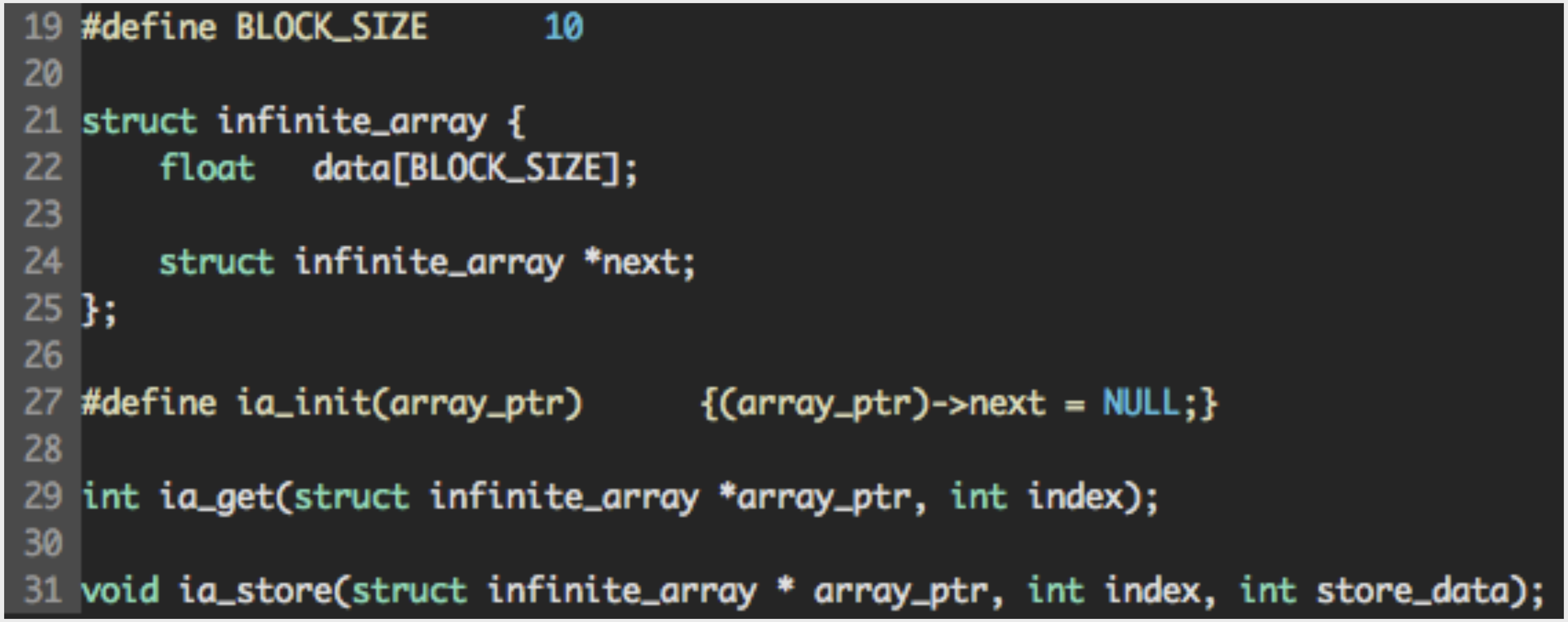
- Information that is shared between modules should be put in a header file
- The header should contain all the public information
  - A comment section
  - Common constants
  - Common structures
  - Prototypes of all the public functions
  - extern declarations for public variables

Public and Private



Headers

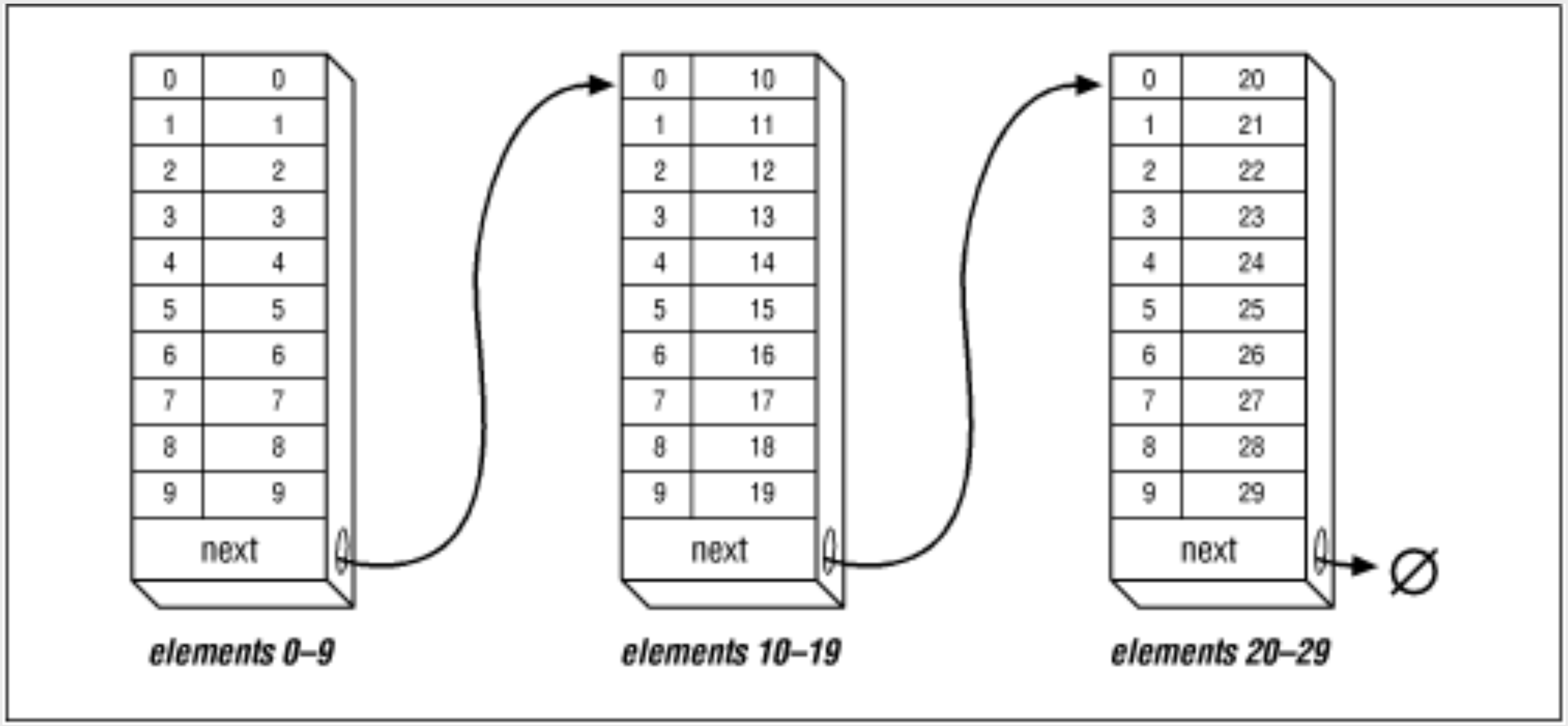
- Example: mod_prog/ia.h



```
#define BLOCK_SIZE      10

struct infinite_array {
    float   data[BLOCK_SIZE];

    struct infinite_array *next;
};

#define ia_init(array_ptr)      {(array_ptr)->next = NULL;}

int ia_get(struct infinite_array *array_ptr, int index);

void ia_store(struct infinite_array * array_ptr, int index, int store_data);
```

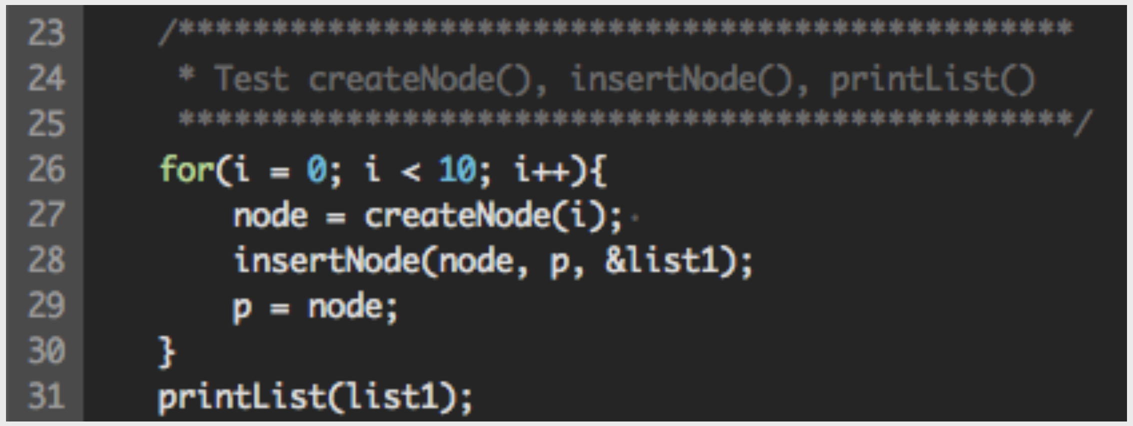
Headers

- Example: mod_prog/ia.c

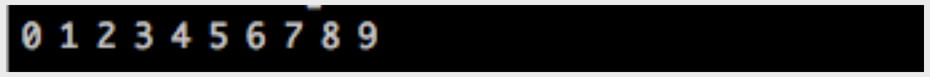


ModList

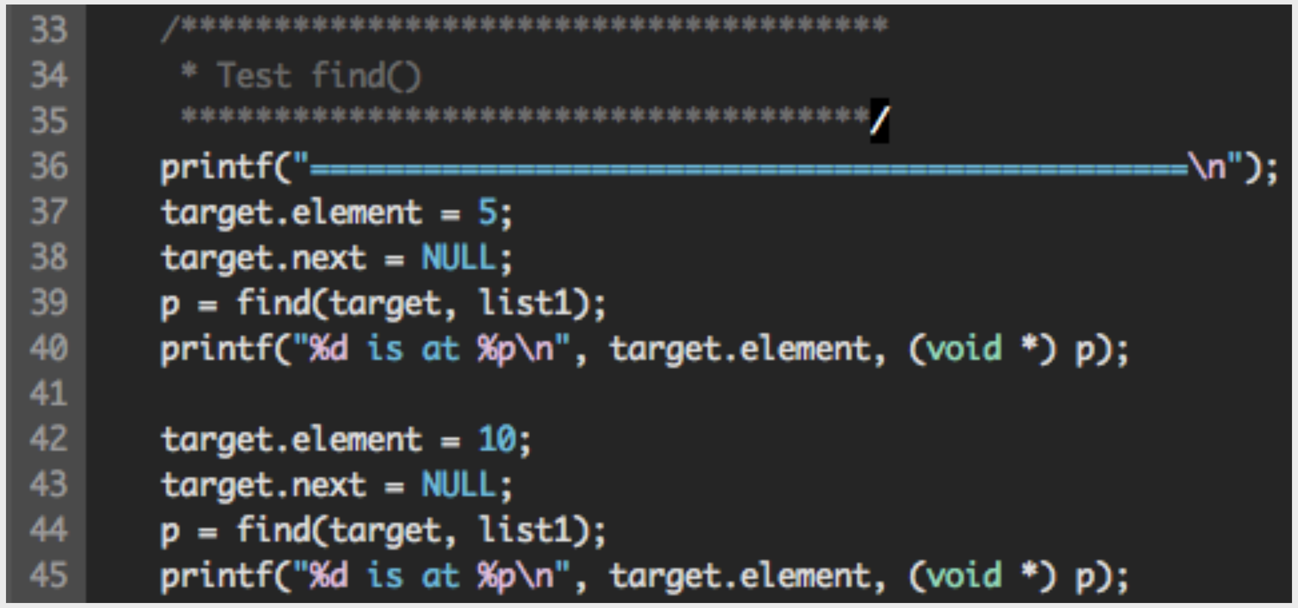
- Example: modList/Makefile, list.h, list.c, test.c



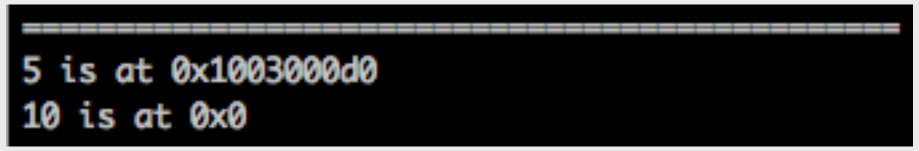
```c
/*********************************************
 * Test createNode(), insertNode(), printList()
 *********************************************/
for(i = 0; i < 10; i++){
    node = createNode(i);
    insertNode(node, p, &list1);
    p = node;
}
printList(list1);
```



```
0 1 2 3 4 5 6 7 8 9
```



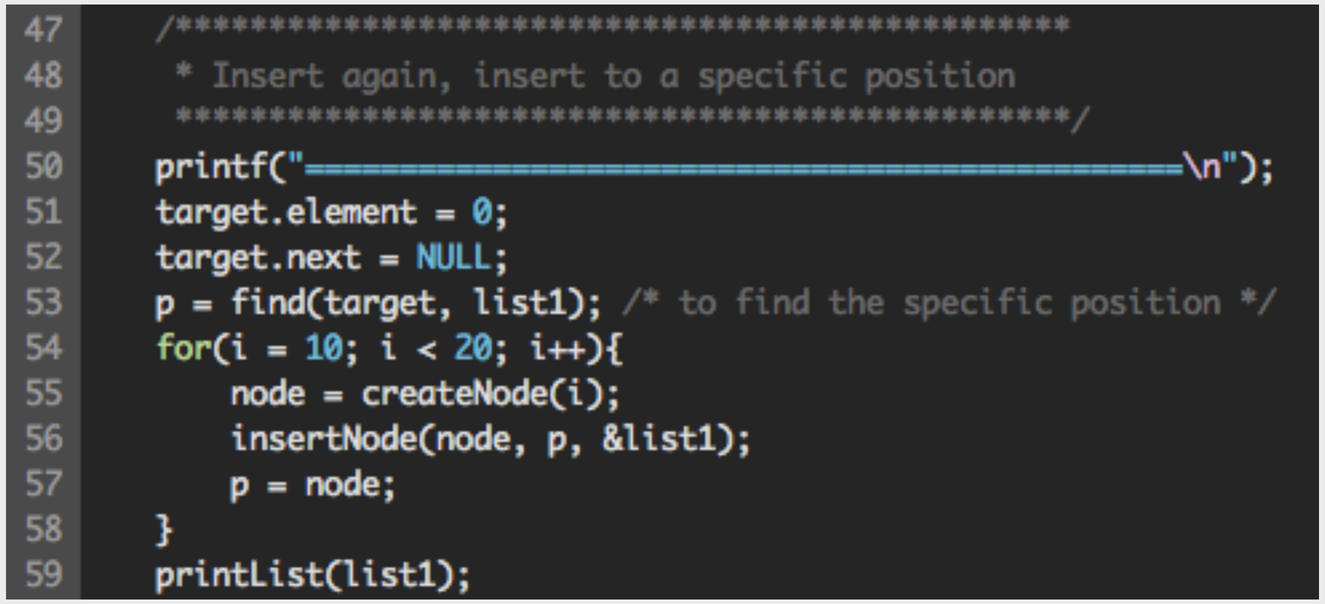
```c
/***************************************
 * Test find()
 ***************************************/
printf("===============================================\n");
target.element = 5;
target.next = NULL;
p = find(target, list1);
printf("%d is at %p\n", target.element, (void *) p);

target.element = 10;
target.next = NULL;
p = find(target, list1);
printf("%d is at %p\n", target.element, (void *) p);
```



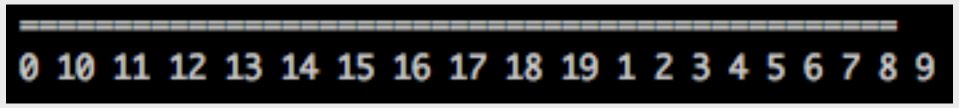
```
===============================================
5 is at 0x1003000d0
10 is at 0x0
```

ModList

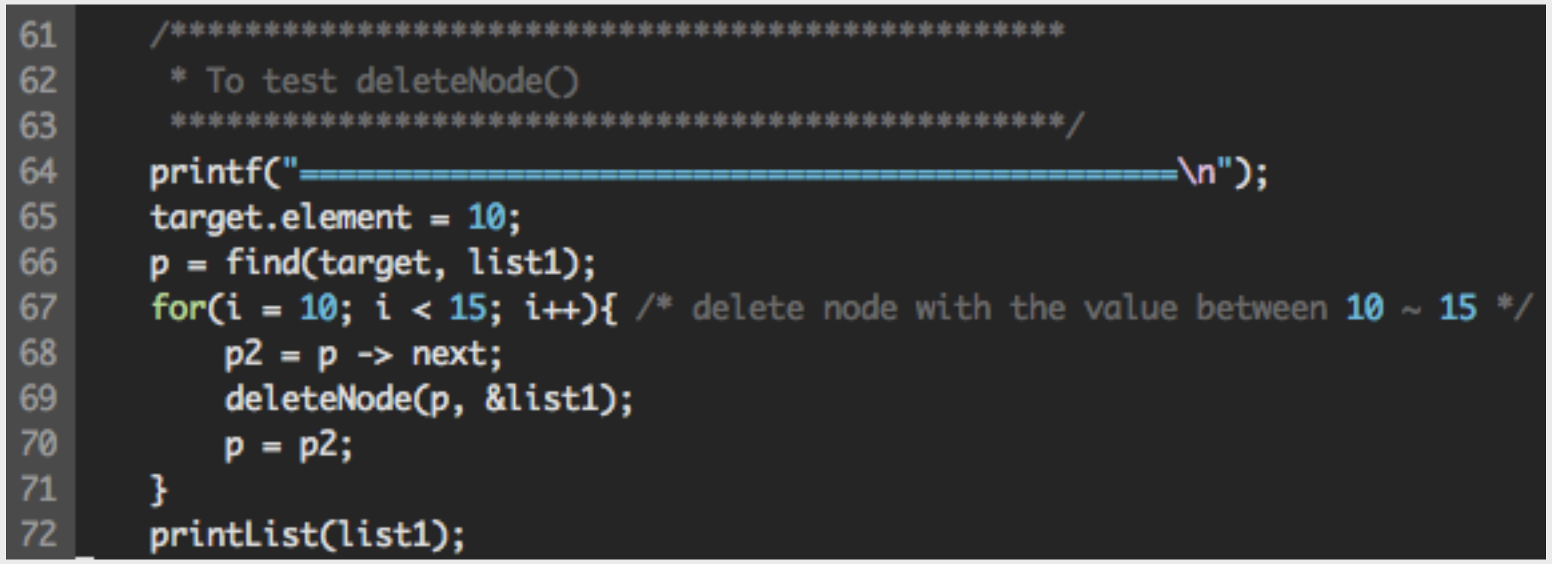
- Example: modList/Makefile, list.h, list.c, test.c



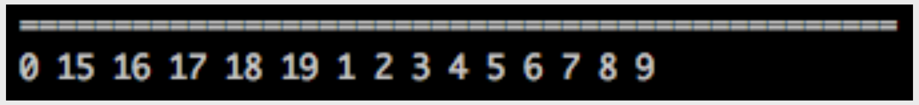
```c
/**********************************************
 * Insert again, insert to a specific position
 **********************************************/
printf("=============================================\n");
target.element = 0;
target.next = NULL;
p = find(target, list1); /* to find the specific position */
for(i = 10; i < 20; i++){
    node = createNode(i);
    insertNode(node, p, &list1);
    p = node;
}
printList(list1);
```



```
=============================================
0 10 11 12 13 14 15 16 17 18 19 1 2 3 4 5 6 7 8 9
```



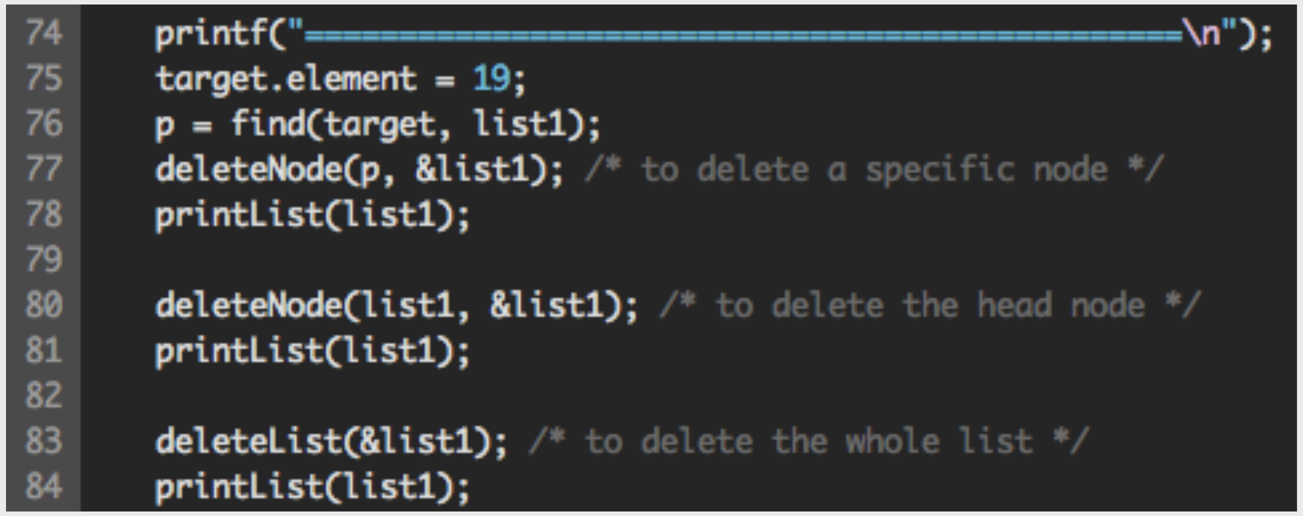
```c
/**********************************************
 * To test deleteNode()
 **********************************************/
printf("=============================================\n");
target.element = 10;
p = find(target, list1);
for(i = 10; i < 15; i++){ /* delete node with the value between 10 ~ 15 */
    p2 = p -> next;
    deleteNode(p, &list1);
    p = p2;
}
printList(list1);
```



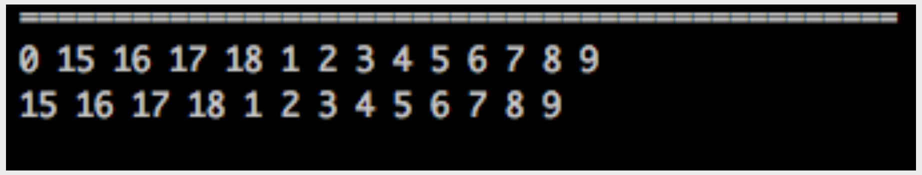
```
=============================================
0 15 16 17 18 19 1 2 3 4 5 6 7 8 9
```

ModList

- Example: modList/Makefile, list.h, list.o, test.c



```c
    printf("==============================================\n");
    target.element = 19;
    p = find(target, list1);
    deleteNode(p, &list1); /* to delete a specific node */
    printList(list1);

    deleteNode(list1, &list1); /* to delete the head node */
    printList(list1);

    deleteList(&list1); /* to delete the whole list */
    printList(list1);
```



```
==============================================
0 15 16 17 18 1 2 3 4 5 6 7 8 9
15 16 17 18 1 2 3 4 5 6 7 8 9
```

The Makefile for Multiple Files

- Makefile contains the following sections
  - Comments
  - Macros
  - Explicit rules
  - Default rules
- For more details, please refer to the previous slides

The Makefile for Multiple Files

- **`hist.o: hist.c ia.h`**

  tell **make** to create **`hist.o`** from **`ia.h`** and **`hist.c`**. Because no command is specified, the default is used

Dividing A Task into Modules

- Modules should be designed to minimize the amount of information that has to pass between them
- A module should make public only the minimum number of functions and data needed to do the job

Guidelines of Designing Modules

- Some general guidelines
    - The number of public functions in a module should be small
    - The information passed between modules should be limited
    - All the functions in a module should perform related jobs