

Computer Programming

Lecture 9

이윤진
서울대학교
2007.1.17.

Slide Credits

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

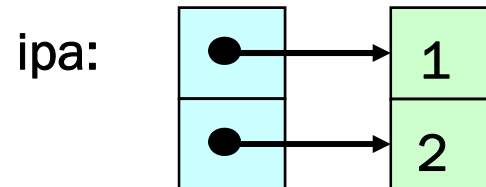
Pointer (2)

순서

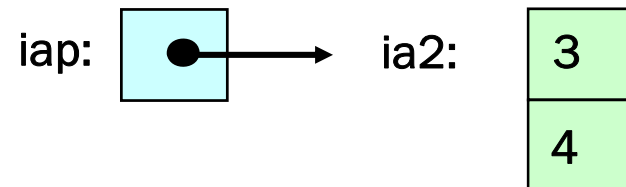
- Pointers
 - Pointer Array vs Array Pointer
 - const Pointer 보충
 - Function Pointer
 - Complicated Declarations
- Q&A

Pointer Array vs Array Pointer

```
int *ipa[2], i;  
int ia1[2] = {1, 2};  
for (i = 0; i < 2; i++) ipa[i] = ia1+i;
```



```
int (*iap)[2];  
int ia2[2] = {3, 4};  
iap = &ia2;
```



```
for (i = 0; i < 2; i++)  
    printf("%d %d\n", ***(ipa + i), **(*iap+i));
```

1 3
2 4

```
printf("%d %d\n", **++ipa, **++*iap);
```

wrong!

const Pointer 보충

- Prevent the Object from Being Changed

```
struct info {
    int a;
    int b;
}
void foo (const struct info *p) {
    p->a = 1; /* Error w/ gcc v3.4.3; Warning w/ gcc v3.0.4 */
    p->b = 2;
}
void bar (const int *d) {
    (*d) = 3; /* Error w/ gcc v3.4.3; Warning w/ gcc v3.0.4 */
}
main() {
    const char *ps = "abc";
    (*ps) = 'e'; /* Error w/ gcc v3.0.4 */
    const struct info c = {0 , 0};
    foo(&c);
    const int i = 5;
    bar(&i);
}
```

placed in the text segment

Function Pointer

- Permit Using Different Functions
 - By Passing a Function Pointer

```
/* charstrcmp, intstrcmp: return <0 if s<t, 0 if s==t, >0 s>t */
int charstrcmp(char *s, char *t) { /* s & t are pointers to character strings */
...
}
int intstrcmp(char *s, char *t) { /* s & t are pointers to integer strings */
...
}
void bsort(void *spa[], int num, int (*comp)(char *, char *)) {
....
if ((*comp)(spa[i], spa[j]) > 0)
...
}
```

Pointer to a function
e.g., charstrcmp & intstrcmp

Function Pointer (계속)

- By Using a Function Pointer Array

```
int handler1(int a) {  
    printf("handler1 %d\n", a);  
}  
int handler2(int a) {  
    printf("handler2 %d\n", a);  
}  
int (*pf[2])(int) = {handler1, handler2};  
void Handler(int n) {  
    (*pf[n])(n+11);  
}  
main() {  
    Handler(0);  
    Handler(1);  
}
```

optional

- Result

```
handler1 11  
handler2 12
```

Complicated Declarations

- Rules
 - Always read declarators from the inside out
 - When there's a choice, always favor [] and () over *
- Examples
 - `int *ap[10];` → array of pointers
 - `float *fp(float);` → function that returns a pointer
 - `void (*pf)(int);` → pointer to a function with an int argument
 - `int *(*x[10])(void);` → array of pointers to function with no arguments returning pointer to int

Complicated Declarations

<code>int *f()</code>	Function returning pointer to int
<code>int (*pf)()</code>	Pointer to function returning int
<code>char **argv</code>	Pointer to pointer to char
<code>int (*spa)[13]</code>	Pointer to array[13] of int
<code>int *spa[13]</code>	Array[13] of pointer to int
<code>void *comp()</code>	Function returning pointer to void
<code>void (*comp)()</code>	Pointer to function returning void
<code>char ((*x())[3])()</code>	Function returning pointer to array[] of pointer to function returning char
<code>char ((*x[3])())[3]</code>	Array[3] of pointer to function returning pointer to array[] of char

Memory Management

순서

- Memory Management
 - Dynamic Memory Allocation
 - Introduction
 - Possible Errors
- Q&A

Dynamic Memory Allocation

- malloc
 - Allocate a Specified Number of Bytes of Memory (w/ Indeterminate Initial Values)
- calloc
 - Allocate (Zero-Set) Space for a Specific Number of Objects of a Specified Size
- realloc
 - Change the Size of a Previous Allocated Area (w/ Indeterminate-Valued Added Area)
- free
 - Deallocate the Specified Space

Dynamic Memory Allocation (계속)

- Standard Library Function Prototype

Unsigned integer type

```
void *malloc(size_t size);  
void *calloc(size_t nobj, size_t size);  
void *realloc(void *ptr, size_t newsize);  
void free(void *ptr);
```

- Example

- Allocation & Deallocation Should Match

```
struct_node *new_node;  
int *pa, n = 10;  
new_node = malloc(sizeof(struct node));  
pa = calloc(n, sizeof(int));  
free(new_node);  
free(pa);
```

```
pa = malloc(n*sizeof(int));  
for (i=0; i<n; i++)  
    *(pa+i) = 0;
```

Stack: Linked List Implementation

```
struct node {
    int data;
    struct node *next;
};
static struct node *top = NULL;

bool Is_empty()
{
    if(top == NULL)
        return true;
    else
        return false;
}
```

Stack: Linked List Implementation

```
void push(int i)
{
    struct node *new_node
    = malloc(sizeof(struct node));
    new_node->data = i;
    new_node->next = top;
    top = new_node;
}
```

```
int pop()
{
    struct node *old_top = top;
    int val = old_top->data;
    if(top->next != NULL){
        top->next = top->next->next;
        top = top->next;
    }
    else {
        top = NULL;
    }
    free(old_top);
    return val;
}
```