

Computer Programming

Lecture 8

이윤진
서울대학교
2007.1.14.

Slide Credits

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

순서

- Scoping
 - Name & Scope
 - Rules
 - Examples
- Pointers
 - 기본
 - Examples
- Q&A

Name and Scope

- Declaration
 - Where the Nature of the Object Is Stated w/o Storage Allocated
 - e.g., `int subroutine();`
- Definition (Serving as a Declaration)
 - Where the Object Is Created or Assigned Storage
 - e.g., `int i = 1;`
- Scope of a Name
 - Part of Program within Which the Name Can Be Used

Initialization goes only w/ definition

Scoping Rules

- Basic Terminology
 - Local/Automatic (wrt a Function)
 - Coming and going with the function invocation
 - External
 - Accessible by name by any function
- Scopes
 - External Variables
 - Declarations to the end of the file
 - Local Variables
 - Functions in which the names are declared

Scoping Rules (계속)

- Use of **extern**

- External Variables or Functions

- To be used if defined in different files

- e.g., `extern int val[];`

Array size is optional

- Single Definition Rule

- External Variables or Functions

- Single definitions among all files

- Local Variables

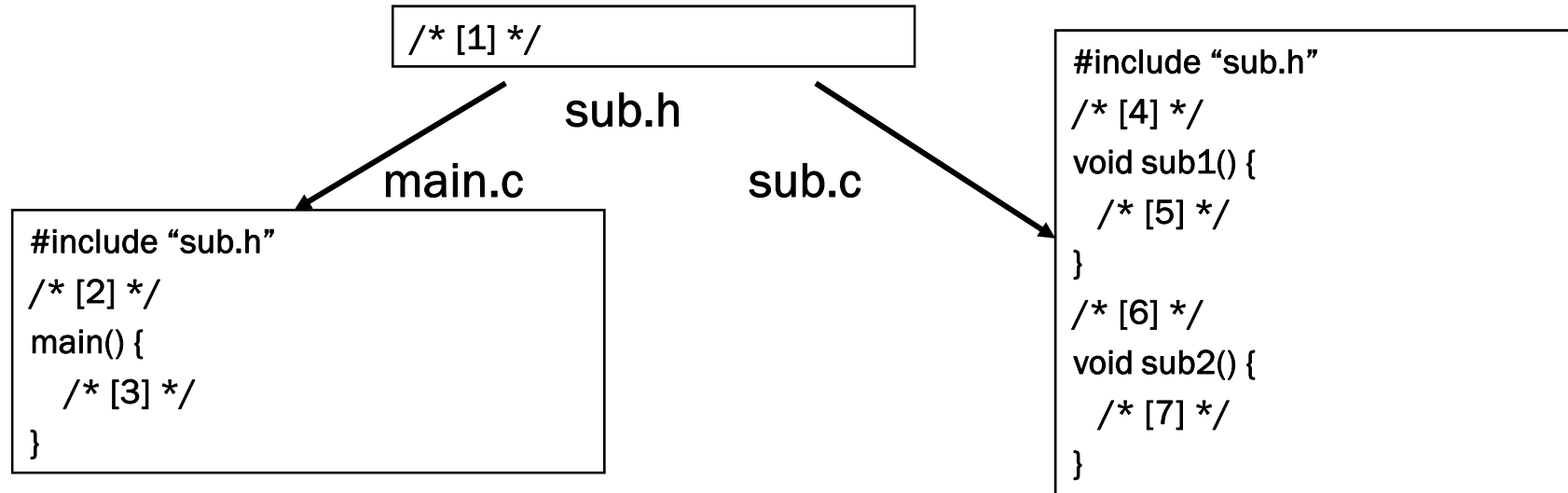
- Single definitions within the files

- Use of **static**

- External Variables or Functions

- To be used for limiting the scopes to the rest of the files

Scoping Example



Type	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Dec/Def	int i		O		O		O
Dec/Def		int i	O		X		X
Dec/Def			int i O		X		X
Dec/Def			X	int i	O		O
Dec/Def					int i O		X
Dec/Def			X		X	int i	O
Dec/Def			X		X		int i O

Scoping 보충

- Single Definition Rule
 - External Variables or Functions
 - Single definitions among all files

main.c

```
int i; // dec/def
int k = 3; // def; Error
main() {
    printf("i = %d\n", i); // OK
    printf("j = %d\n", j); // Error
    sub1();
    sub2();
}
```

i = 0

sub.c

```
int i; // dec/def
int j = 1; // def
void sub1() {
    printf("i = %d\n", i); // OK
}
int k = 2; // def; Error
void sub2() {
    printf("j = %d\n", j);
}
```

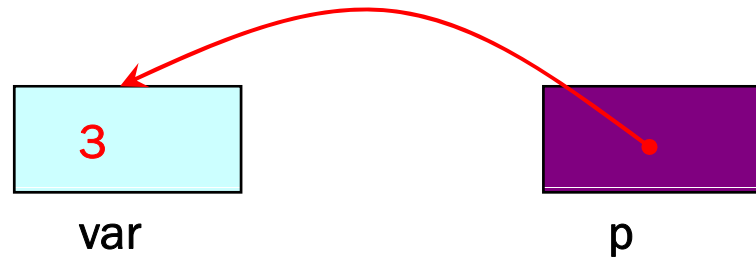
j = 1

Pointer 기본

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

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

- Result



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);
```

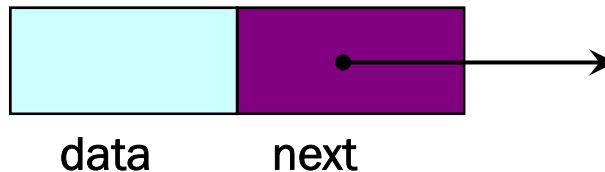
- Result

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

Self-Referential Structure

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

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



- Example

```
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);
```

Stack Module Example

Interface

Implementation

```
void make_empty();  
int is_empty();  
void push(int i);  
int pop();
```

stack.h

calc.c

```
#include "stack.h"  
main() {  
    make_empty();  
    ...  
}
```

stack.c

```
#include "stack.h"  
static int contents[100];  
struct node {  
    int data;  
    struct node *next;  
};  
static struct node *top = 0;  
void make_empty() {  
    ...  
}  
int is_empty() {  
    ...  
}  
static int is_full() {  
    ...  
}  
void push(int i) {  
    ...  
}  
int pop() {  
    ...  
}
```

순서

- Pointers
 - Pointer & Array
 - Valid Pointer Operations
 - Use of the * and ++/-- Operators
 - Stack Module Example
 - Pointer to void
 - Character Pointer & Pointer Array
 - Pointer & Multi-Dimensional Array
 - Pointer & Function Argument
- Q&A

Pointer & Array

- Use of Arrays
 - Indexing
 - Pointer Arithmetic

```
int i, *p;
int a[5] = {1, 2, 3, 4, 5}, sum1 = 0, sum2 = 0, sum3 = 0;
for (i = 0; i < 5; ++i)
    sum1 += a[i];
for (p = a; p < &a[5]; ++p)
    sum2 += *p;
for (i = 0; i < 5; ++i)
    sum3 += *(a + i);
printf("%d %d %d\n", sum1, sum2, sum3);
```

- Result

```
15 15 15
```

Valid Pointer Operations

- Assignment of Pointers of the Same Type

```
int a[5], *p, *q;  
p = a; /* OK */
```

not a variable

- Adding/Subtracting a Pointer and an Integer

```
q = p + 3; /* OK */
```

Relations: ==, !=, <, >=, etc.

- Subtracting/Comparing Two Pointers to Members of the Same Array

```
if (q > p) printf("%d\n", q-p);
```

3

- Assigning/Comparing to Zero

```
if (q == 0) printf("%d\n", q);
```

Use of the * and ++/-- Operators

- Right to Left Association Rule

```
int a[5] = {1, 3, 5, 7, 9}, *p, *q;
```

```
p = a;
```

```
printf("%d %d\n",*++p, *p);    /* = *(++p), *p */
```

```
printf("%d %d\n",++*p, a[1]); /* = ++(*p), a[1] */
```

```
printf("%d %d\n",*p++, a[1]); /* = *(p++), a[1] */
```

```
printf("%d %d\n",(*p)++, a[1]); /* != *p++, a[1] */
```

```
printf("%d %d\n",*p, a[2]);
```

right to left for these arguments

- Result

```
3 1
```

```
4 3
```

```
4 4
```

```
5 4
```

```
6 6
```

Stack Module Example

```
#include "stack.h"
static int contents[100];
static int *top_ptr = contents;
...
void push(int i) {
    if (is_full())
        ...
    else
        *top_ptr++ = i;
}
int pop() {
    if (is_empty())
        ...
    else
        return *--top_ptr;
}
```

stack.c

Array implementation w/ pointer arithmetic

Pointer to void

- void
 - Nonexistent
- Pointers to void (`void *`)
 - Generic Pointers
 - Any pointer may be converted to type `void *`
 - w/o loss of information
 - w/o a cast

```
typedef void * SDT /* Stack Data Type */  
void push(SDT i) {  
...}  
SDT pop() {  
...}
```

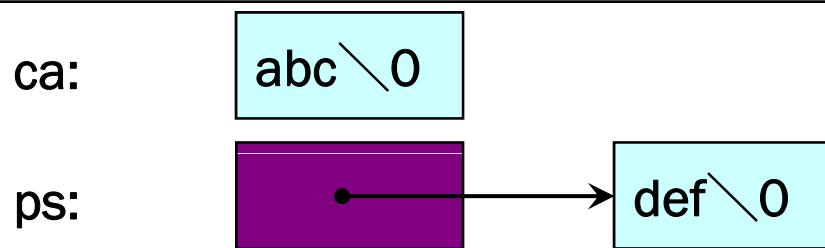
```
main() {  
    int i = 3;  
    push(&i); /* w/o a cast */  
    printf("%d\n", *((int *)pop()));  
}
```

Cast required

Character Pointer & Pointer Array

- Character Array vs Pointer to a String Constant

```
char ca[] = "abc";  
char *ps = "def";
```



- Pointer Array

assumed to be defined already

```
char *line_ptr[MAXLINES]; /* String-constant-pointer array */  
for (int i = 0; i < MAXLINES; i++) {  
    printf("%s \n", line_ptr[i]); /* Character pointer */  
    printf("%c \n", *line_ptr[i]); /* First character */  
}
```

Initialization of Pointer Arrays

```
/* month_name: return the name of the n-th month */
char *month_name(int n) {
    static char *mnpa[] = {
        "Invalid Month Number",
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December"
    };
    return (n < 1 || n > 12) ? mnpa[0] : mnpa[n];
}
```

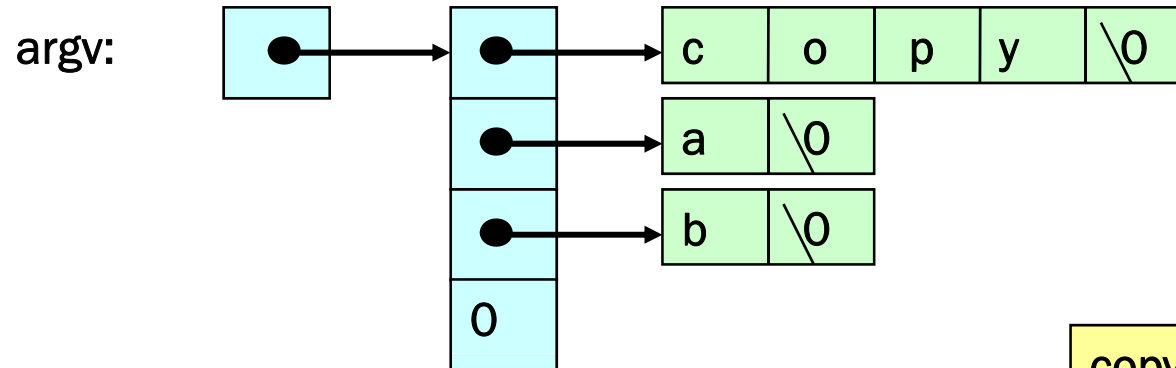
Conditional expression

Command-Line-Argument Example

```
main(int argc, char *argv[]) {
```

```
...
```

```
martini:~>copy a b
```



```
for (int i = 0; i < argc; i++)  
    printf("%s %c \n", *(argv + i), *argv[i]);  
printf("%c \n", *++argv[0]);  
printf("%c \n", *++(argv[0]));  
printf("%c \n", (*++argv)[0]);  
printf("%c \n", *++argv[0]);
```

copy c

a a

b b

o

p

a

Pointer & Multi-Dimensional Array

- Multi-Dimensional Array
 - One-Dimensional Array of Arrays

```
int twoda[2][3] = { /* [row][col] */  
    {1, 2, 3},  
    {4, 5, 6}  
};
```

Array of two rows, where each row is a three-integer array

```
void f(int twoda[][3]) {  
    ... }
```

```
void f(int (*twoda)[3]) {  
    ... }
```

- Different definition

```
void f(int *twoda[3]) { /* Error */  
    ... }
```

Pointer & Function Argument

- Call by Value

w/ a lower possibility of memory leak

- Lead to More Compact Programs

- Call by Reference (by Passing Pointers)

- Permit Altering a Caller Variable in the Callee
- Permit Returning Multiple Data Items
- Permit Passing Large Structured Data Efficiently

- Example

wrong!

```
void swap(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
}
```

```
swap(a, b);
```

```
void swap(int *px, int *py) {  
    int temp;  
    temp = *px;  
    *px = *py;  
    *py = temp;  
}
```

```
swap(&a, &b);
```