C Programming - Day 4

2017.09.07.

JunGu Kang
Dept. of Cyber Security

아주대학교 Whois

다중포인터

다중포인터

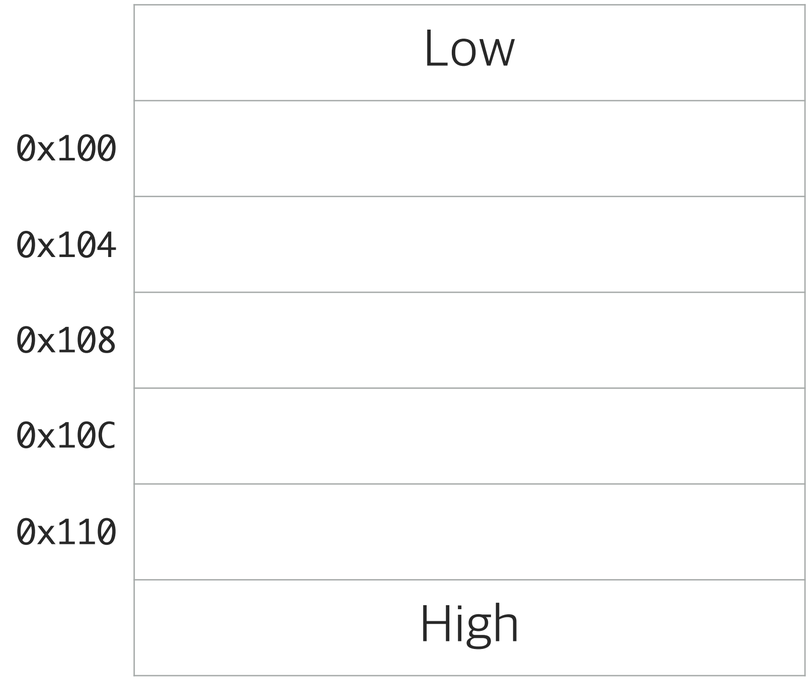
포인터 변수를 가리키는 포인터 변수가 존재할 수 있다.

다중포인터

```
int a = 37;
int * ptr1 = &a;
int ** ptr2 = &ptr1;
int *** ptr3 = &ptr2;
int **** ptr4 = &ptr3;
```

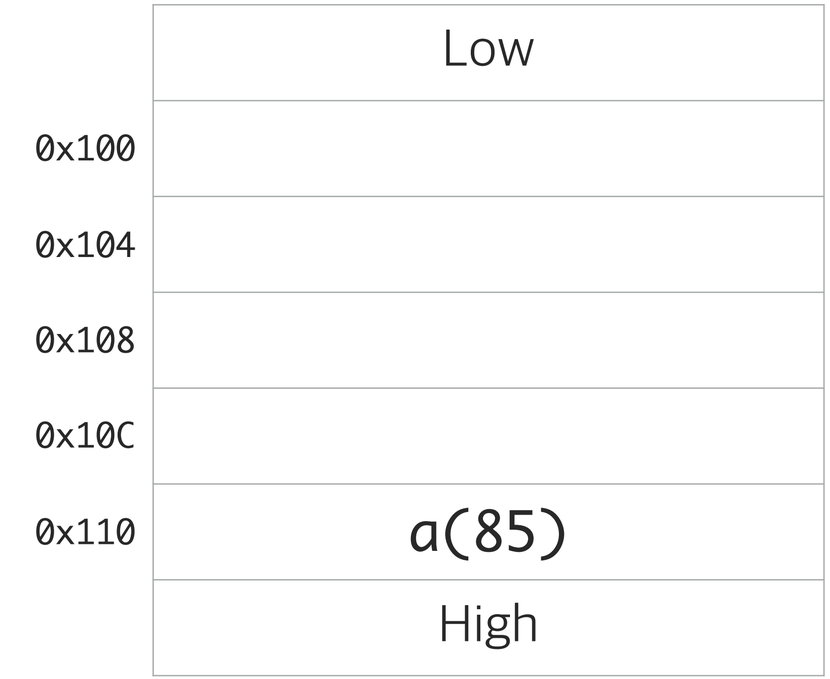
다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf("%p, %d", ptr1, *ptr1);
    printf("%p, %d", ptr2, **ptr2);
    printf("%p, %d", ptr3, ***ptr3);
    printf("%p, %d", ptr4, ****ptr4);
}
```



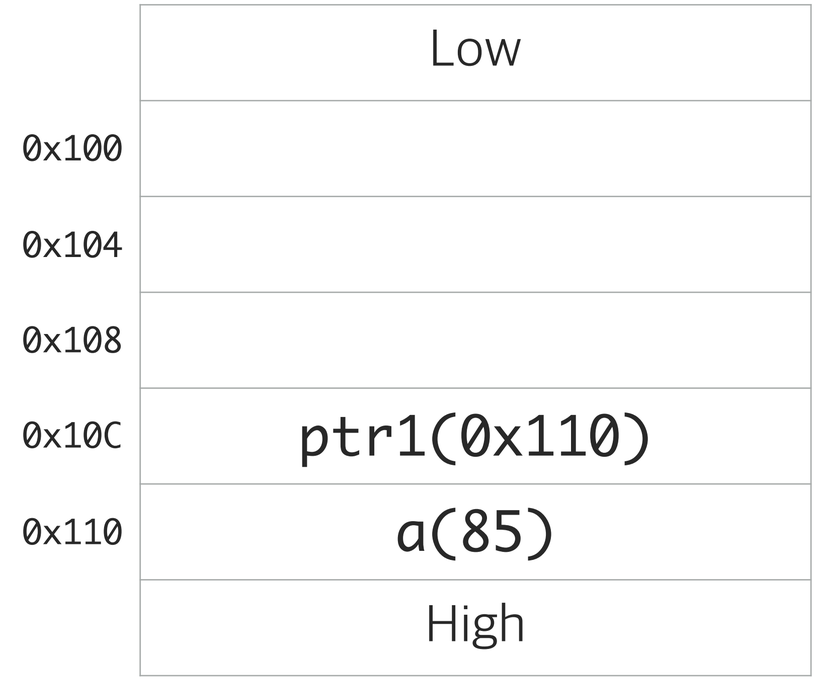
다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf("%p, %d", ptr1, *ptr1);
    printf("%p, %d", ptr2, **ptr2);
    printf("%p, %d", ptr3, ***ptr3);
    printf("%p, %d", ptr4, ****ptr4);
}
```



다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf(“%p, %d”, ptr1, *ptr1);
    printf(“%p, %d”, ptr2, **ptr2);
    printf(“%p, %d”, ptr3, ***ptr3);
    printf(“%p, %d”, ptr4, ****ptr4);
}
```



다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf(“%p, %d”, ptr1, *ptr1);
    printf(“%p, %d”, ptr2, **ptr2);
    printf(“%p, %d”, ptr3, ***ptr3);
    printf(“%p, %d”, ptr4, ****ptr4);
}
```

	Low
0x100	
0x104	
0x108	ptr2(0x10C)
0x10C	ptr1(0x110)
0x110	a(85)
	High

다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf(“%p, %d”, ptr1, *ptr1);
    printf(“%p, %d”, ptr2, **ptr2);
    printf(“%p, %d”, ptr3, ***ptr3);
    printf(“%p, %d”, ptr4, ****ptr4);
}
```

	Low
0x100	
0x104	ptr3(0x108)
0x108	ptr2(0x10C)
0x10C	ptr1(0x110)
0x110	a(85)
	High

다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf(“%p, %d”, ptr1, *ptr1);
    printf(“%p, %d”, ptr2, **ptr2);
    printf(“%p, %d”, ptr3, ***ptr3);
    printf(“%p, %d”, ptr4, ****ptr4);
}
```

	Low
0x100	ptr4(0x104)
0x104	ptr3(0x108)
0x108	ptr2(0x10C)
0x10C	ptr1(0x110)
0x110	a(85)
	High

다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf("%p, %d", ptr1, *ptr1);  // 0x110, 85
    printf("%p, %d", ptr2, **ptr2);
    printf("%p, %d", ptr3, ***ptr3);
    printf("%p, %d", ptr4, ****ptr4);
}
```

	Low
0x100	ptr4(0x104)
0x104	ptr3(0x108)
0x108	ptr2(0x10C)
0x10C	ptr1(0x110)
0x110	a(85)
	High

다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf("%p, %d", ptr1, *ptr1);
    printf("%p, %d", ptr2, **ptr2);  // 0x10C, 85
    printf("%p, %d", ptr3, ***ptr3);
    printf("%p, %d", ptr4, ****ptr4);
}
```

	Low
0x100	ptr4(0x104)
0x104	ptr3(0x108)
0x108	ptr2(0x10C)
0x10C	ptr1(0x110)
0x110	a(85)
	High

다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf("%p, %d", ptr1, *ptr1);
    printf("%p, %d", ptr2, **ptr2);
    printf("%p, %d", ptr3, ***ptr3);  // 0x108, 85
    printf("%p, %d", ptr4, ****ptr4);
}
```

	Low
0x100	ptr4(0x104)
0x104	ptr3(0x108)
0x108	ptr2(0x10C)
0x10C	ptr1(0x110)
0x110	a(85)
	High

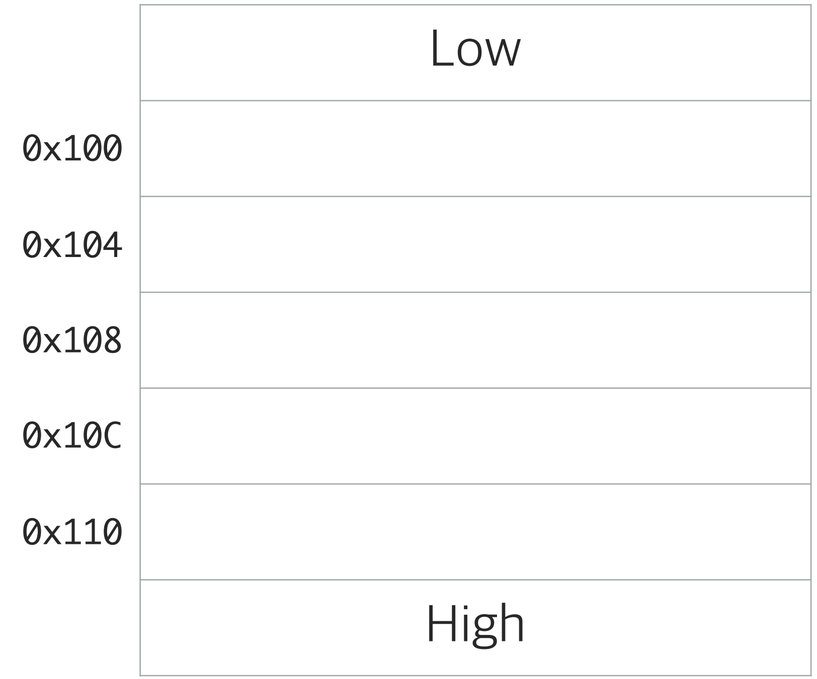
다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf("%p, %d", ptr1, *ptr1);
    printf("%p, %d", ptr2, **ptr2);
    printf("%p, %d", ptr3, ***ptr3);
    printf("%p, %d", ptr4, ****ptr4);  // 0x104, 85
}
```

	Low
0x100	ptr4(0x104)
0x104	ptr3(0x108)
0x108	ptr2(0x10C)
0x10C	ptr1(0x110)
0x110	a(85)
	High

다중포인터

```
main() {
    int a = 85;
    int * ptr1 = &a;
    int ** ptr2 = &ptr1;
    int *** ptr3 = &ptr2;
    int **** ptr4 = &ptr3;
    printf("%p, %d", ptr1, *ptr1);
    printf("%p, %d", ptr2, **ptr2);
    printf("%p, %d", ptr3, ***ptr3);
    printf("%p, %d", ptr4, ****ptr4);
}
```



포인터 배열

포인터 배열

포인터 변수들을 이어붙여 만든 배열

포인터 배열

```
type * name[length]
```

포인터 배열

```
int * int_arr[];
double * double_arr[];
```

문자열들의 배열

문자열 상수는 **포인터**로 가리킬 수 있다.

문자열들의 배열

여러 문자열 상수는
문자열 포인터의 배열로 가리킬 수 있다.

포인터 배열

```
main() {
    char * arr[] = {“String 1”, “String 2”, “String 3”};

    printf(“%s\n”, arr[0]);
    printf(“%s\n”, arr[1]);
    printf(“%s\n”, arr[2]);
}
```

함수 포인터

함수 포인터

함수도 메모리에 저장된 후 호출된다.

함수 포인터

그렇다면 포인터로 함수를 가리킬 수도 있다.

함수 포인터의 선언

```
return_type (* name) (argument type, …)
```

함수 포인터의 선언

```
int (* ptr_1) (int);  // int 하나를 받고, int를 return
double (* ptr_2) (double, double);  // double 두개를 받고, double을 return
```

함수 포인터의 선언

함수의 이름은 함수를 가리키는 포인터

함수 포인터의 선언

```
int do_something(int);

main() {
    int (* ptr) (int) = do_something;  // 함수의 이름은 포인터
    ptr(3);  // do_something(3)과 같음
}
```

void 포인터

void 포인터

type이 **`void`**인 포인터

void 포인터

type이 없는 포인터

void 포인터

주소를 저장만 할 수 있고, 어떠한 연산도 할 수 없다.

사용자 정의 자료형

사용자 정의 자료형

필요로 하는 자료형을 만들어 쓸 수 있다.

typedef

자료형에 **새로운 이름**을 붙인다.

typedef

```
typedef type_name new_name;
```

typedef

```
typedef int * INT_PTR;

main() {
    int a = 10;
    INT_PTR a_ptr = &a;
    printf(“%d\n”, *a_ptr);
}
```

구조체

하나 이상의 변수를 묶어 정의한 자료형

구조체의 정의

```
struct coordinate {
    double x;
    double y;
};  // 뒤에 세미콜론을 붙인다.
```

구조체 변수의 선언

```
struct structure_name variable_name;
```

구조체 변수의 선언

```
struct coordinate {
    double x;
    double y;
};

main() {
    struct coordinate coord;
}
```

구조체의 정의

```
typedef struct {
    double x;
    double y;
} Coordinate;
```

구조체 변수의 선언

```
type_name variable_name;
```

구조체 변수의 선언

```
typedef struct {
    double x;
    double y;
} Coordinate;

main() {
    Coordinate coord;
}
```

구조체의 멤버

```
typedef struct {
    int number;
    char string[10];  // 배열도 구조체의 멤버가 될 수 있다.
    float * ptr;  // 포인터도 구조체의 멤버가 될 수 있다.
} Member;
```

구조체의 초기화

```
typedef struct {
    int number;
    char string[10];
    float * ptr;
} Member;

main() {
    float a = 10.3;
    Member members = {123, “string”, &a};
}
```

구조체 멤버에 접근

```
typedef struct {
    int number;
    char string[10];
    float * ptr;
} Member;

main() {
    float a = 10.3;
    Member members = {123, “string”, &a};
    printf(“%d\n”, members.number);  // 123
    printf(“%s\n”, members.string);  // string
    printf(“%p\n”, members.ptr);  // &a
    printf(“%d\n”, *members.ptr);  // 10.3
}
```

중첩 구조체

구조체를 멤버로 가지는 구조체

중첩 구조체의 정의

```
typedef struct {
    char first_name[20];
    char last_name[20];
} Name;

typedef struct {
    int student_no;
    Name student_name;
} Student;
```

중첩 구조체의 정의

```
typedef struct {
    char first_name[20];
    char last_name[20];
} Name;

typedef struct {
    int student_no;
    Name student_name;
} Student;

main() {
    Student = {123, {“GilDong”, “Hong”}}
    // Student = {123, “GilDong”, “Hong”} 로 써도 된다.
}
```

구조체 포인터

구조체도 메모리에 저장되므로 포인터로 가리킬 수 있다.

구조체 포인터

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num = {123, 234}
    Numbers * num_ptr = &num;
}
```

구조체 포인터

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num = {123, 234}
    Numbers * num_ptr = &num;
    printf("%d\n", num_ptr->number1);
    printf("%d\n", num_ptr->number2);
}
```

중첩 구조체

```
(*ptr).member == ptr->member
```

구조체 배열

구조체도 type이므로 **배열로 선언할 수 있다.**

구조체 배열

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num_arr[3] = { {1, 2}, {3, 4}, {5, 6} };
    for(int i = 0; i < 3; i++)
        printf("%d, %d\n", num_arr[i].number1, num_arr[i].number2);
}
```

구조체 연산

구조체를 대상으로 하는 연산은 제한적이다.

구조체 연산

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num1 = {123, 234};
    Numbers num2 = num1;  // num1을 num2로 복사
    printf("%d, %d\n", num2.number1, num2.number2);
}
```

구조체 연산

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num = {123, 234};
    printf(“%ld\n”, sizeof num);
}
```

구조체 연산

```
typedef struct {
    int number1;
    int number2;
} Numbers;

main() {
    Numbers num1 = {123, 234};
    Numbers num2 = {345, 456};
    num1 + num2;  // 이런 연산을 할 수 없다.
}
```

구조체 연산

구조체 변수를 대상으로 연산을 하기 위해서는
따로 함수를 정의해야 한다.

공용체

같은 메모리 공간을 다양한 방법으로 접근한다.

공용체의 정의

```
union test {
     char char_member;
     int int_member;
     long long_member;
};
```

공용체의 선언

```
union test {
     char char_member;
     int int_member;
     long long_member;
};

main {
     union test test_union;
     test_union = 0x12345678;
}
```

공용체의 정의

```
typedef union {
    char char_member;
    int int_member;
    long long_member;
} Test;
```

공용체의 선언

```
typedef union {
    char char_member;
    int int_member;
    long long_member;
} Test;

main {
    Test test_union;
    test_union = 0x12345678;
}
```

공용체

```
typedef union test {
    char char_member;
    int int_member;
    long long_member;
} Test;

main {
    Test test_union;
    test_union = 0x12345678;
    printf(“%ld\n”, sizeof test_union);  // 8
}
```

공용체

```
typedef union test {
    //생략
} Test;

main {
    Test test_union;
    test_union = 0x12345678;
    printf(“%p\n”, &test_union.char_member);
    printf(“%p\n”, &test_union.int_member);
    printf(“%p\n”, &test_union.long_member);
}
```

공용체

```
typedef union test {
    //생략
} Test;

main {
    Test test_union;
    test_union = 0x12345678;
    printf(“%ld\n”, test_union.char_member);
    printf(“%ld\n”, test_union.int_member);
    printf(“%ld\n”, test_union.long_member);
}
```

열거형

상수에 이름을 붙이자

열거형 정의

```
enum color {
    RED = 1, BLUE = 2, GREEN = 3
};
```

열거형 정의

```
enum color {
    ZERO, ONE, TWO, THREE  // 0, 1, 2, 3
};
```

열거형 정의

```
enum color {
    ZERO, ONE, FOUR = 4, FIVE  // 0, 1, 4, 5
};
```

열거형 정의

```
typedef enum color {
    RED = 1, BLUE = 2, GREEN = 3
} Color;
```

열거형 변수의 선언

```
typedef enum color {
    RED = 1, BLUE = 2, GREEN = 3
} Color;

main() {
    Color favorite_color;
    favorite_color = BLUE;
    printf("%d\n", favorite_color);  // 2
}
```

열거형 변수의 선언

```
typedef enum color {
    RED = 1, BLUE = 2, GREEN = 3
} Color;

main() {
    Color favorite_color;
    favorite_color = BLUE;

    switch(favorite_color) {
        case RED:
            printf(“Favorite Color is Red!\n”);
        // 생략
    }
}
```

함수의 인자

Argument

- 함수를 호출할 때 전달하는 값
- 지역변수이다.

Call by Value vs Call by Reference

Call by Value는 **값**을 전달하고,
Call by Reference는 **포인터**를 전달한다.

예제

두 변수를 입력받아 변수에 들어있는 값을
서로 바꿔주는 함수를 만들고 싶다면?

Call by Value

```
void swap(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

main() {
    int a = 6, b = 3;
    swap(a, b);
    printf("%d, %d", a, b);
}
```

Call by Value

변수가 아니라 **변수에 들어있는 값이 전달**되기 때문에,
절대 변수에 들어있는 값이 바뀌지 않는다.

Call by Reference

```
void swap(int * a, int * b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

main() {
    int a = 6, b = 3;
    swap(a, b);
    printf("%d, %d", a, b);
}
```

Call by Reference

포인터를 전달해 변수에 직접 접근하도록 해야 한다.

예제

두 포인터 변수를 입력받아 포인터가 가리키는 값을
서로 바꿔주는 함수를 만들고 싶다면?

Call by Value

```
void swap(int * a, int * b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

main() {
    int a = 6, b = 3;
    int * a_ptr = &a, * b_ptr = &b;
    swap(a, b);
    printf("%d, %d", a_ptr, b_ptr);
}
```

Call by Reference

```
void swap(int ** a, int ** b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

main() {
    int a = 6, b = 3;
    int * a_ptr = &a, * b_ptr = &b;
    swap(a, b);
    printf("%d, %d", &a_ptr, &b_ptr);
}
```

main 함수의 인자

main 함수로도 인자를 전달할 수 있다.

main 함수의 인자

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

argv는 더블 포인터

argv

Strings

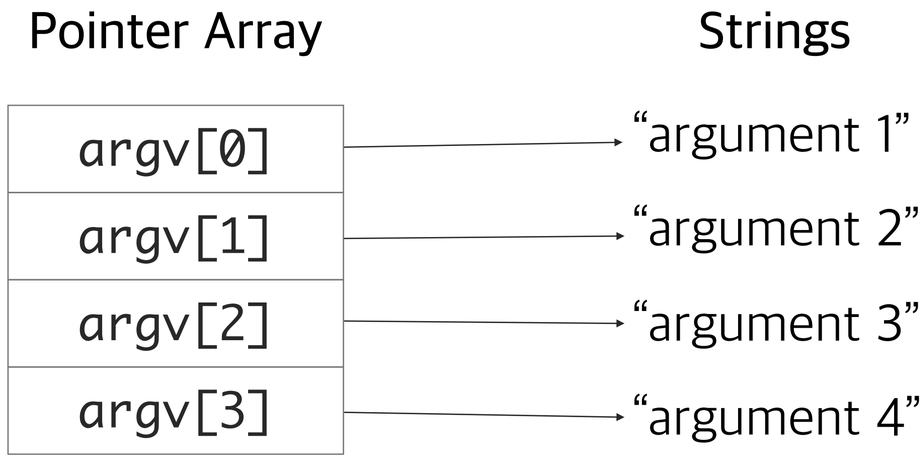
“argument 1”

“argument 2”

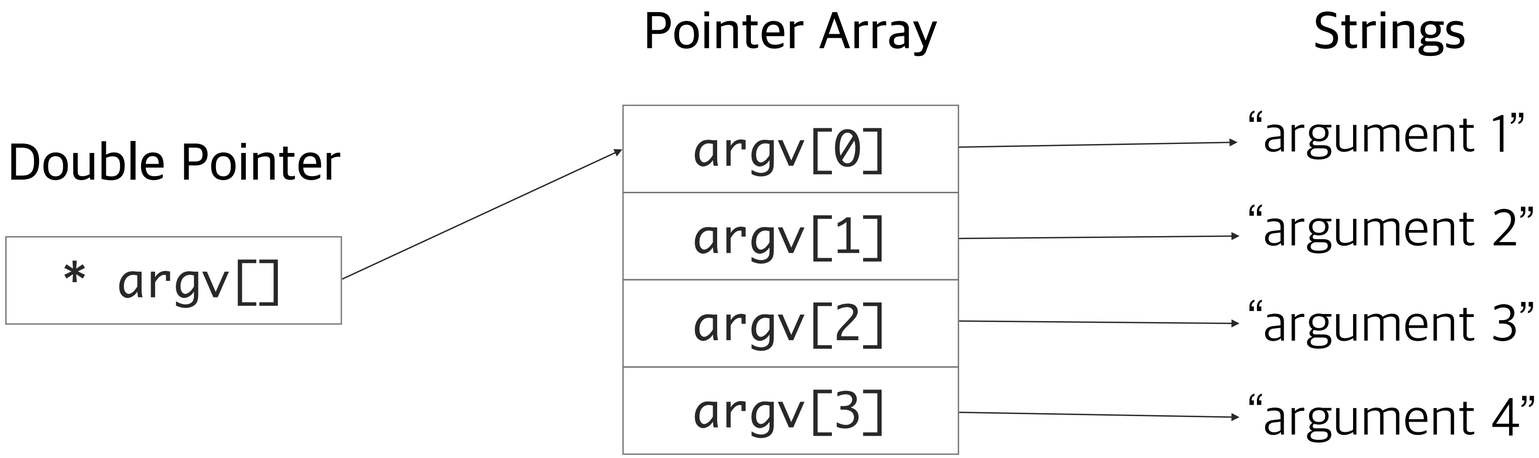
“argument 3”

“argument 4”

argv



argv



구조체

구조체는 자료형이기 때문에
그대로 인자로 전달받고, 반환할 수 있다.

Call by Reference

```
typedef struct {
    // 생략
} Argument;

Argument do_something(Argument); // Argument를 받고, Argument를 반환

main() {
    Argument arg;
    do_something(arg);
}
```

구조체

물론 Call by Reference를 원한다면,
구조체를 가리키는 포인터를 전달해야 한다.

다음 수업 준비

복습 및 과제

- 오늘 수업 내용 복습
- 과제 반드시 제출
  - 질문은 얼마든지 가능하니 반드시 제출
- 학습시 리눅스 이용

다음 수업 예습

- 입출력
- 메모리와 동적 할당

C Programming - Day 4

2017.09.07.

JunGu Kang
Dept. of Cyber Security

아주대학교 Whois