C Programming II Mentoring

단원 별 핵심 개념 정리

제작 : 임재영 멘토

Part I. 함수(Function)

1. 함수의 유형 4가지

함수 유형	반환 값의 유무	
전달 인자의 유무	전달 인자 O 반환 값 O	전달 인자 O 반환 값 X
	전달 인자 X 반환 값 O	전달 인자 X 반환 값 X

Comment

전달인자로 들어온 값 잘 활용하기

반환 값의 타입 잘 체크해서 값 올바르게 반환하기

당연하지만, 전달 인자도 있고, 반환 값도 있는 유형의 함수가 시험에 가장 많이 나온다!

2. static 변수

```c
#include <stdio.h>

void increaseNumber()
{
    static int num1 = 0;

    printf("%d\n", num1);

    num1++;
}

int main()
{
    increaseNumber();
    increaseNumber();
    increaseNumber();
    increaseNumber();
    return 0;
}
```

Comment

static 변수는 **지역변수지만 전역변수의 성격**을 가지고 있기 때문에,

위 코드처럼 **함수를 벗어나도 값이 유지되는 성질**을 이용하는 문제가 많이 나온다.

Part II. 배열(Array)

1. 배열의 가장 기본적인 개념

- **배열 선언 방법**: 자료형 배열이름[길이] = {원소는 쓰던지 말던지};
- 배열의 **인덱스 값은 0부터 시작**한다! (arr[0], arr[1], arr[2] …)
- 배열의 **길이와 크기**는 다르다!
    - **길이**: 원소 수
    - **크기**: 원소 수 x 자료형의 크기
- **배열의 이름**은 포인터와 같으며, **배열의 가장 첫 번째 원소**를 가리킨다!

2. 문자열 입력 받기

Practice 2 #문자열 #null문자 #인덱스

프로그램 사용자로부터 하나의 영단어를 입력 받아서 입력 받은 영단어의 길이를 계산하여 출력하는 프로그램을 작성해보자.

```
출력 결과
영단어를 입력하세요: programming
영단어의 길이: 10
```

5

- **프로그램에서 중괄호의 열고 닫음은 중요하다. 중괄호가 옳게 열리고 닫혔는지를 테스트하는 함수를 작성하여 테스트하라. 중괄호를 10개의 배열로 만들어라.**

(hint)

중괄호가 열렸을 때 count를 +하고, 닫혔을 때 -를 하고, 합이 0이면 된다

그렇다면, {}}}}}{{{{이것은 정상적으로 열리고 닫혔는가?

어떻게 하면 좋을까?

<출력예시>

```
중괄호: 00000
중괄호가 정상적으로 열리고 닫혔습니다.
```

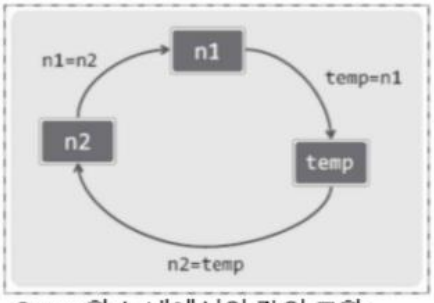
위 문제들처럼 문자열을 입력 받아야 할 상황이 왔을 때,
여러분이 많이 사용하는 scanf_s를 사용해서 문자열을 입력 받을 수 있다!!

scanf_s 함수의 원형은 **scanf_s("입력 받을 형태", &변수 이름, 입력 받을 크기);** 로 정의할 수 있는데,
여러분들은 세 번째에 위치한 '입력 받을 크기'를 자주 쓰지 않았을 것이다...

왜냐면 **일반 변수** 같은 경우에는 **하나의 변수에 할당된 크기가 고정**되어 있기 때문에 쓰지 않아도 되지만,
배열 같은 경우에는 **원소 몇 개를 입력 받느냐에 따라서 크기가 달라지기 때문**에 꼭 크기를 명시해야 한다.

따라서, 문자열을 입력 받고 싶다면...
scanf("%s", arr, sizeof(arr)); 와 같은 방식으로 문자열을 입력 받길 바란다.
하지만 scanf에서 %s 형식으로 입력 받을 때는, **공백 포함해서 받을 수 없다!!**
왜 arr 앞에 &를 안 붙였는지 모르겠다면 배열의 개념을 다시 보고 오도록!

3. 두 원소 바꾸기



Swap 함수 내에서의 값의 교환

n1과 n2의 값을 바꾸고 싶다면, 아래 순서를 천천히 따라와 보도록...

0) n1 = 10, n2 = 20 이 들어 있다고 가정
1) **temp라는 임시 저장 공간**을 하나 만든다.
2) **temp**에 **n1**을 넣는다. (temp = 없음 -> 10)
3) **n1**에 **n2**를 넣는다. (n1 = 10 -> 20)
4) **n2**에 **temp**를 넣는다. (n2 = 20 -> 10)
5) 수행 결과 : n1 = 20, n2 = 10 으로 두 값이 바뀐 것을 알 수 있다.

코드

```
void Swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
}
```

이 알고리즘을 여기서 다룬 이유?

- 홍교수님 과제 중에 배열 원소 값을 서로 바꾸는 문제가 있더라...
- 사실 이 알고리즘은 함수와 포인터 부분에서 중요하게 다루니 꼭 기억할 것!

Part III. 포인터(Pointer)

1. 포인터의 가장 기본적인 개념

- **&은 주소 연산자** / ***는 역참조 연산자** (참고: &은 Ampersand / *은 Asterisk라고 읽는다)
- **포인터 변수의 크기**는 64비트 환경 -> **8바이트** / 32비트 환경 -> 4바이트
- 주소에 +1을 하면 진짜 +1이 아니라, **그 자료형만큼 뛰는거다!**

Ex) int형 배열 첫번째 원소의 주소가 1000이라면, 두번째 원소의 주소는 1004

첫번째 원소의 포인터를 **p(=&arr[0])**라고 하면,

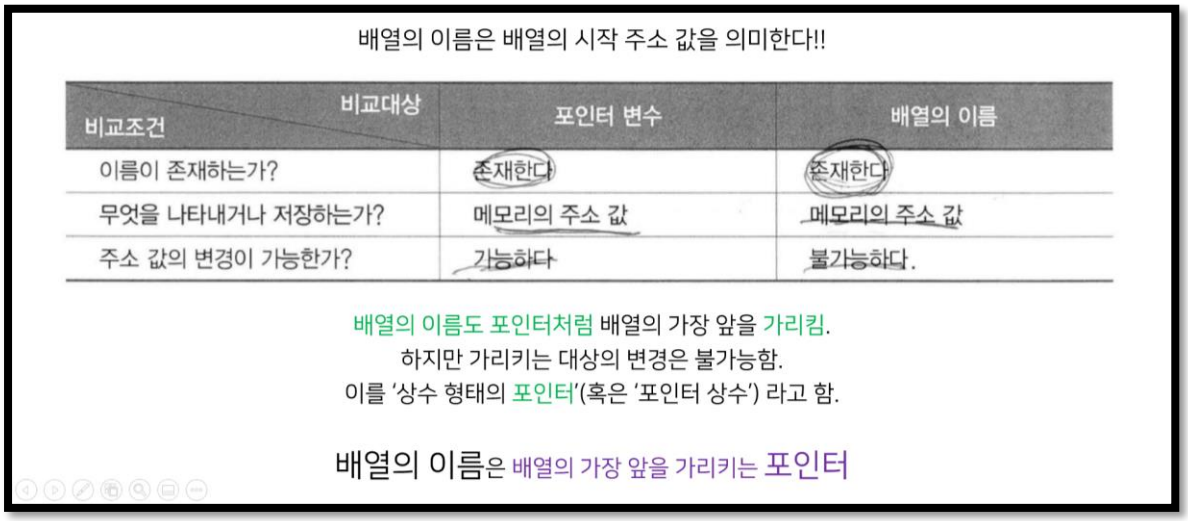
두번째 원소의 포인터는 **p+1**이라고 쓰고 제대로 된 주소는 **(&arr[0] + 4)**로, **int형 자료형만큼 뛴거임**

- 그리고 항상 중요한 것... 너네들만의 포인터 그림을 머리 속에 꼭 가지고 있어야 함!

2. 포인터와 배열

- 가장 중요한 사실은, 배열도 포인터다!

  왜냐하면, **배열의 이름**은 **배열의 가장 첫번째 원소**를 가리키거든!



핵심 코드

```
int main(void) {

    int arr[3] = { 11, 22, 33 };

    int* ptr = arr;  // 1. 포인터로 배열을 가리키고

    *(ptr + 2) = 44; // 2. 포인터를 증가/감소 시켜서
                     // 3. 그 안에 있는 원소에 접근

    printf("%d %d %d", arr[0], arr[1], arr[2]);

    return 0;
}
```

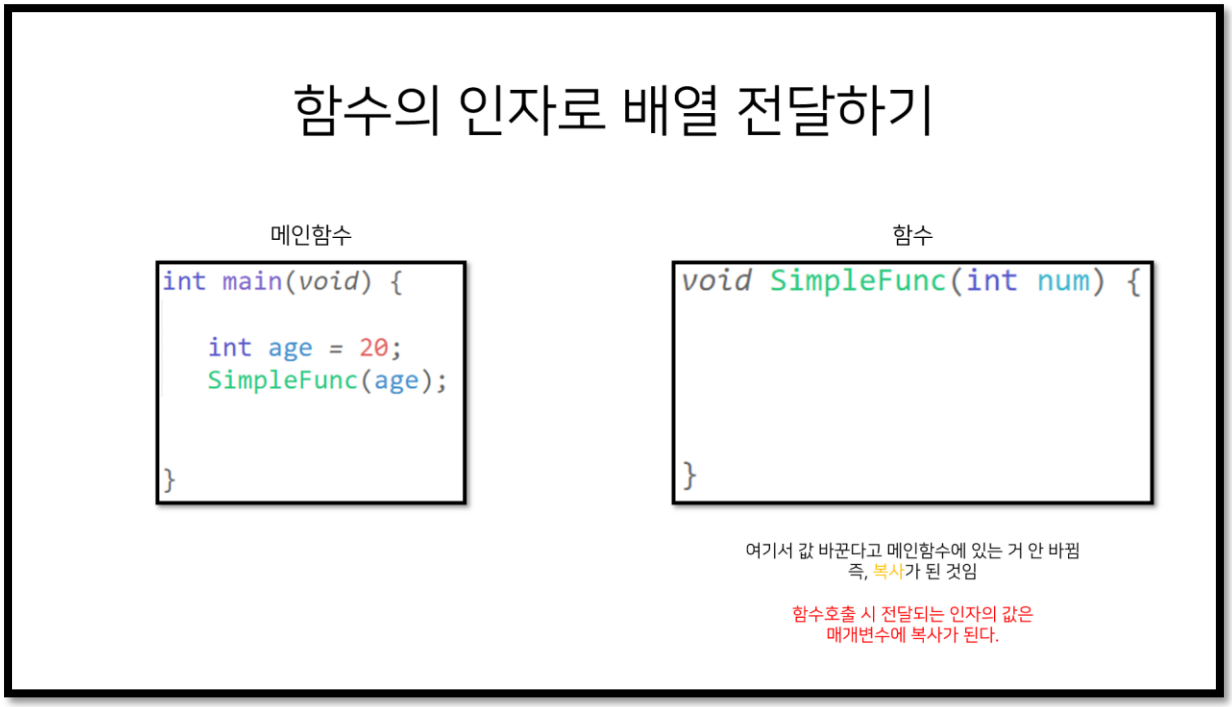
사실 핵심은,

1) 포인터로 **배열을 가리킴 (& 연산자 사용)**
2) 포인터를 **증가 혹은 감소**시켜서 (그래서 주소에 +1 연산한 것들 주의하라고 한 거야)
3) 그 안에 있는 **원소에 접근 (* 연산자 사용)**

이게 전부 다입니다!

따라서 저 위에 있는 예제 코드를 이해하고 출력이 어떻게 나오는지 이해했으면 한다!

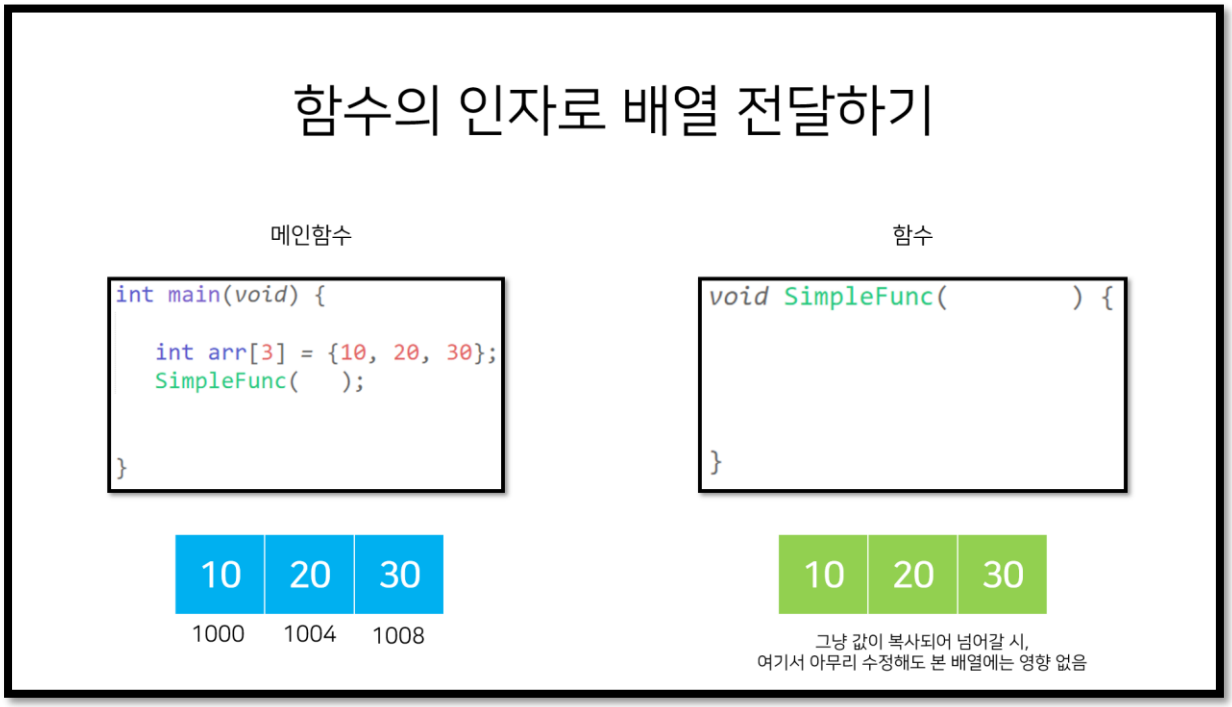
3. 포인터와 함수 ★★★ (좀 중요)



여기서 알고 가야할 사실!

메인함수에서 다른 함수로 전달 값을 보냈을 때, 값이 실제로 전달되는 것이 아니라 **'복사'**가 되는 것임

쉽게 말해서, 메인함수 age 변수에 담긴 **20이라는 값**이, 다른 함수 **num 변수에** 20이 **복사된 것**.



배열도 똑같다!

다른 함수 입장에서는 **그냥 값이 복사되어 넘어간 것**일뿐...

저 함수 안에서 배열 값을 아무리 수정해도 원본 배열에는 영향을 미치지 않음.

함수의 인자로 배열 전달하기

함수에서 본 배열을 수정할 수 있는 방법은 없을까?
함수에서도 '원래 배열에 접근할 수 있게 해주는 무언가'가 필요

메인함수

```
int main(void) {

   int arr[3] = {10, 20, 30};
   SimpleFunc(    );


}
```

함수

```
void SimpleFunc(          ) {



}
```

10	20	30
1000	1004	1008

그렇다면, **다른 함수에서도 메인함수 내의 원본 배열을 수정할 수 있는 방법**은 없을까?
우리에게 필요한 것은 **다른 함수 내에서도 '원본 배열에 접근할 수 있게 해주는 무언가'**가 필요해.

메인함수

```
int main(void) {

   int arr[3] = {10, 20, 30};
   SimpleFunc(    );


}
```

함수

```
void SimpleFunc(          ) {



}
```

Step 1. 배열의 주소 값을 넘겨주어야 함 (main)
"배열의 이름이 뭐라고 했지?"

Step 2. 배열의 주소를 받아야 함(SimpleFunc)
"포인터 변수는 뭐를 저장하는 변수라고 했지?"
"주소를 저장하는 것이 무엇이라고 했지?"

Step 3. 포인터를 통해서 원래 배열에 접근 가능(SimpleFunc)
"접근 어떤 방식으로 할 수 있다고 했지?"

10	20	30
1000	1004	1008

그것이 바로 **포인터**라는 것이여.
어떻게 사용하냐면,

1) 배열의 **주소 값을 전달 인자로 넘겨주어야** 해! Ex) SimpleFunc(arr);
2) 다른 함수에서 **배열의 주소 값을 받을 변수를 선언**해주어야 해! Ex) void SimpleFunc(int* ptrarr)
3) 이제 **그 함수 안에서 원래 배열에 접근** 가능해짐 Ex)ptrarr[0] = 15; -> 메인함수 arr[0]도 바뀜

함수의 인자로 배열 전달하기

2. 포인터 변수로 받는다

```
void ShowArrElem(int* param, int len) {
   for (int i = 0; i < len; i++)
      printf("%d ", param[i]);
   printf("\n");          3. 접근 가능
}

int main(void) {
   int arr1[3] = { 1,2,3 };
   int arr2[5] = { 4,5,6,7,8 };   1. 주소 값을 보내고
   ShowArrElem(arr1, sizeof(arr1) / sizeof(int));
   ShowArrElem(arr2, sizeof(arr2) / sizeof(int));
   return 0;
}
```

위와 같은 방식으로 사용할 수 있다.

위 코드는 사용법을 익히는데 중요하니, 꼭 알아 둘 것!

함수의 인자로 배열 전달하기

★ 국 룰 ★

외부 함수에서 메인 함수에 있는 값을 건드리고 싶다면

함수의 전달인자로 주소 값을 보내주어라.

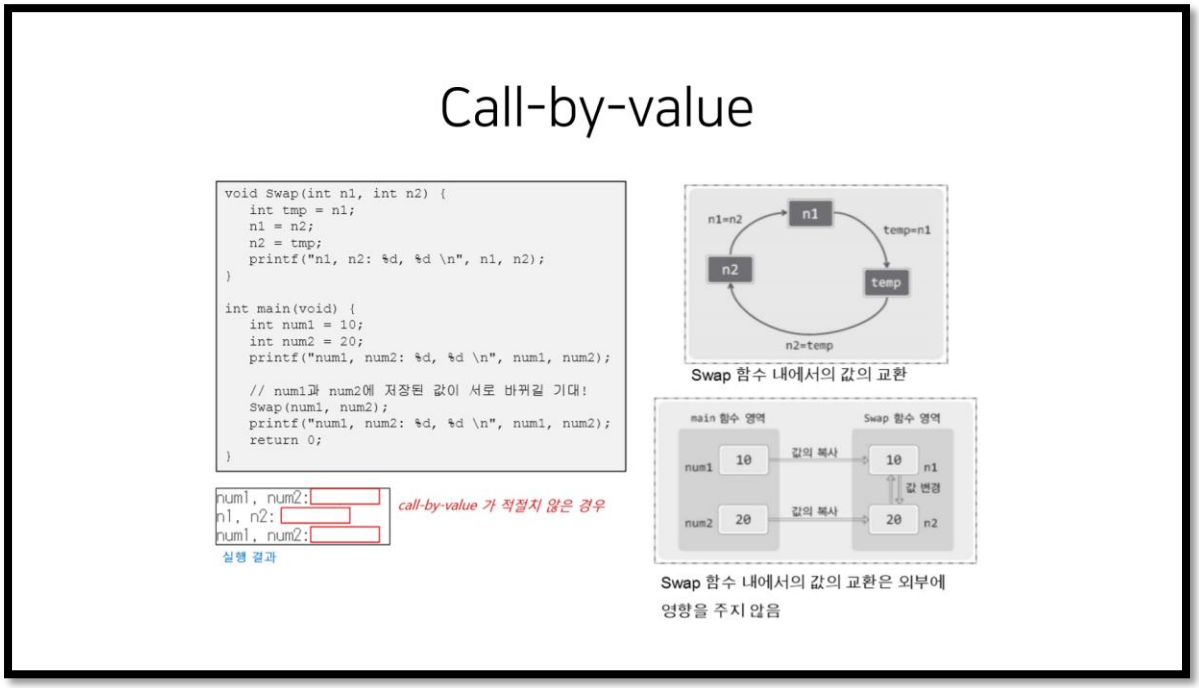
그리고 외부 함수에서는 매개변수로

포인터 변수를 선언하여 그 주소 값을 받아라.

주소 값을 보내준 것을 비유하자면,

"내가 주소 보내줬으니까, 너가 직접 가서 변경해!" 라는 의미를 갖고 있음.

Call by value

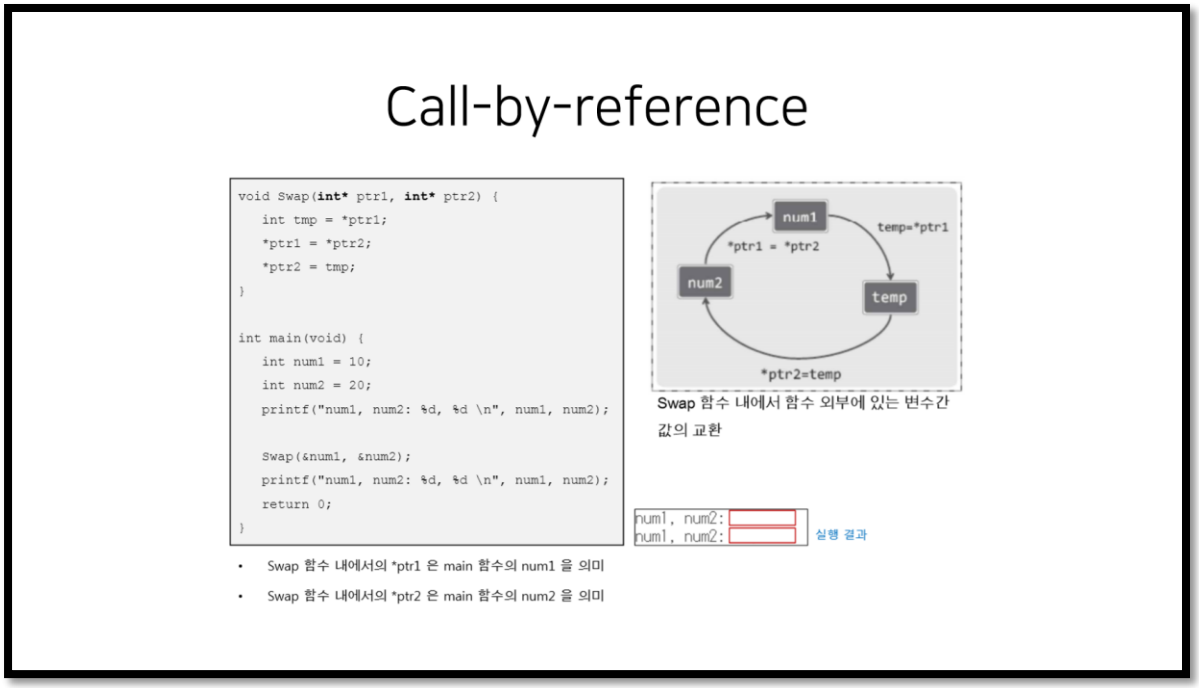


Call by value는 **함수의 전달 인자에 그냥 값을 넣어서 보내**주기 때문에,

다른 함수 입장에서는 **값을 복사 받기만 한 것**이다.

따라서 아무리 다른 함수에서 값을 수정하고 바꿔도, 원본 값에는 영향을 미치지 않는다.

Call by reference



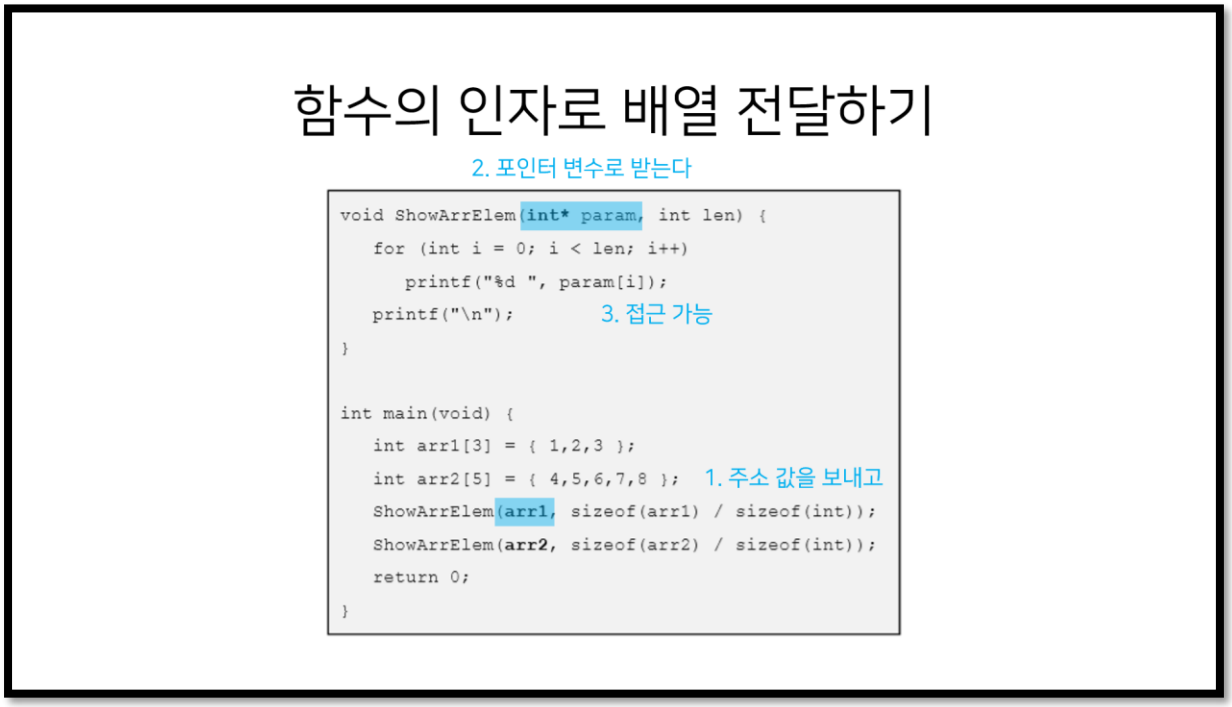
Call by reference는 **함수의 전달 인자에 주소를 넣어서 보내**주기 때문에,

다른 함수 입장에서는 **원본 값에 접근할 수 있는 주소를 받게 된 것**이다.

따라서 다른 함수에서도 메인함수 안에 있는 원본 값에 영향을 미칠 수 있게 된다.

이제 문제는 어떻게 나오냐?

아래 대표적인 코드를 보면 큰 틀을 알 수 있다.



아까 이 코드를 기억하라고 했는데, 저 코드에 모든 핵심이 다 들어가 있다.

1) 함수의 **전달 인자에 주소 값** 넣어서 보내기
2) 다른 함수에서는 **포인터 변수로 받기**
3) 그 함수에서 이제 **접근 가능**

하지만 궁금한 점...

질문) 2번 항목에서 **포인터 변수**로 받는다고 했는데, 그냥 **배열**로 받으면 안되나요?

답변) 됩니다. **배열도 포인터**라고 했기 때문에 둘 다 같은 의미입니다.

```c
#include <stdio.h>

void SimpleFunc(int* a) {
    a[1] = 25;
}

int main(void) {

    int arr[3] = { 10, 20, 30 };
    printf("%d %d %d\n", arr[0], arr[1], arr[2]);
    SimpleFunc(arr);
    printf("%d %d %d\n", arr[0], arr[1], arr[2]);
    return 0;
}
```

포인터 변수로 받았을 때

```c
#include <stdio.h>

void SimpleFunc(int a[]) {
    a[1] = 25;
}

int main(void) {

    int arr[3] = { 10, 20, 30 };
    printf("%d %d %d\n", arr[0], arr[1], arr[2]);
    SimpleFunc(arr);
    printf("%d %d %d\n", arr[0], arr[1], arr[2]);
    return 0;
}
```

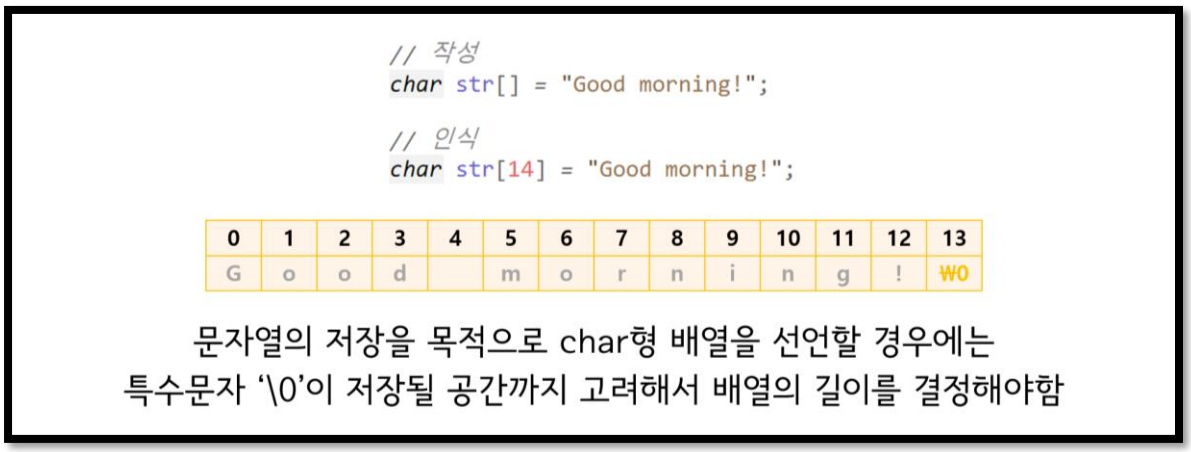
배열 변수로 받았을 때

두 코드 모두 결과는 똑같이 나옵니다.

Part IV. 문자와 문자열 (Character and String)

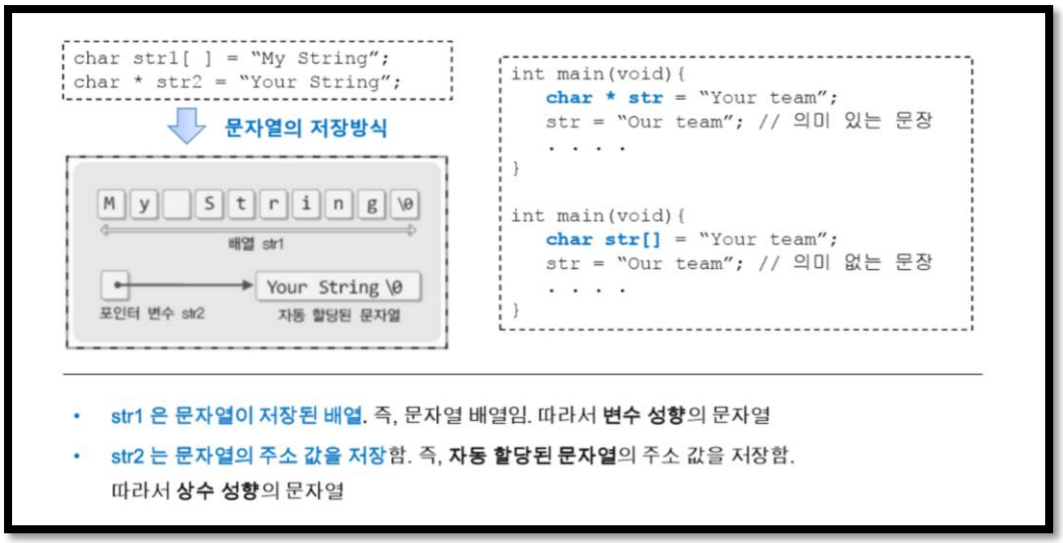
1. 문자열의 기본적인 개념

- 배열을 이용한 문자열 변수의 표현



문자열 가장 마지막에는 NULL 문자가 있다는 사실 잊지 말아야 함!

- 문자열 상수란 무엇인가?



문자열이 저장된 배열과, 문자열 상수는 다르다!

문자열이 저장된 배열은 각 배열의 원소에 문자들과, **맨 마지막 원소에는 NULL 문자**가 들어있는 배열

문자열 상수는 **포인터 변수**로 저장하기 때문에 **문자열의 주소 값**을 저장

따라서 이 부분에서는 **'문자열 맨 끝에는 NULL 문자가 존재한다'** 라는 사실을 아는 것이 핵심!

- **NULL 문자를 통해서 반복문을 제어**하는 문제가 가장 많이 쓰임
- for(int i = 0; **p[i] != '\0'**; i++) 이런 형식??

당연히 알겠지만, NULL 문자와 그냥 숫자 0의 아스키 코드는 서로 다르다! 헷갈리지 말 것.