

포인터 특강

20200921

dereference.c

```
#include <stdio.h>

int main()
{
    int *numPtr;    // 포인터 변수 선언
    int num1 = 10;  // 정수형 변수를 선언하고 10 저장

    numPtr = &num1; // num1의 메모리 주소를 포인터 변수에 저장

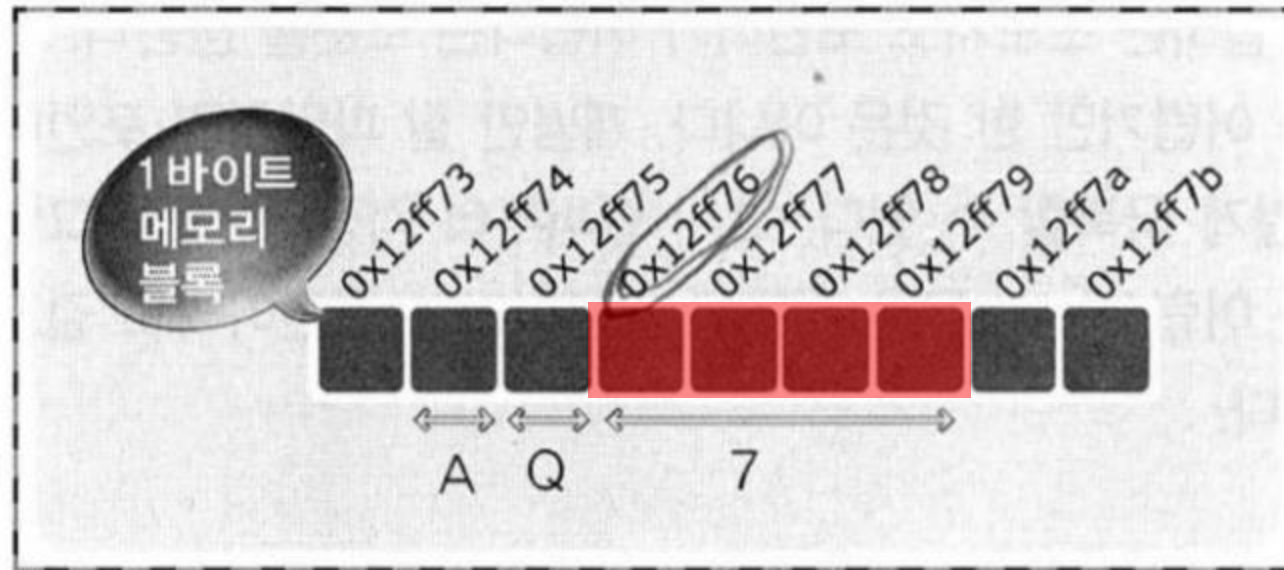
    printf("%d\n", *numPtr); // 10: 역참조 연산자로 num1의 메모리 주소에 접근하여 값을 가져옴

    return 0;
}
```

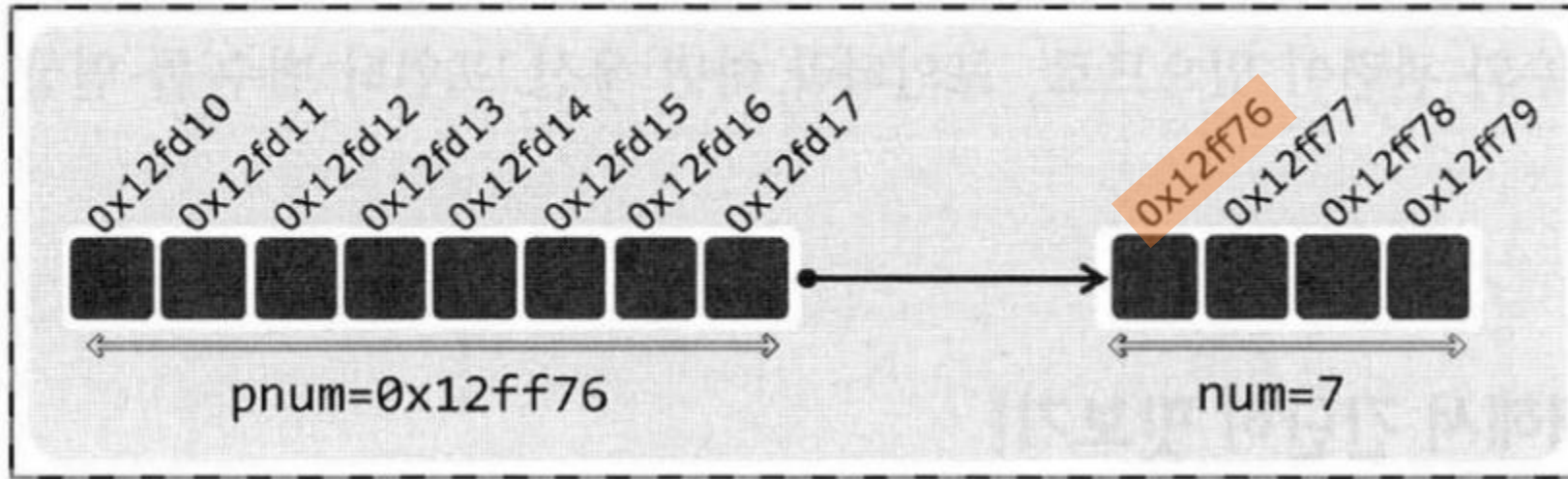
주소 값의 저장을 목적으로 선언되는 포인터 변수

포인터 변수는
메모리의 주소 값을
저장하기 위한 변수

```
int main(void) {  
  
    char ch1 = 'A', ch2 = 'Q';  
    int num = 7;  
  
}
```

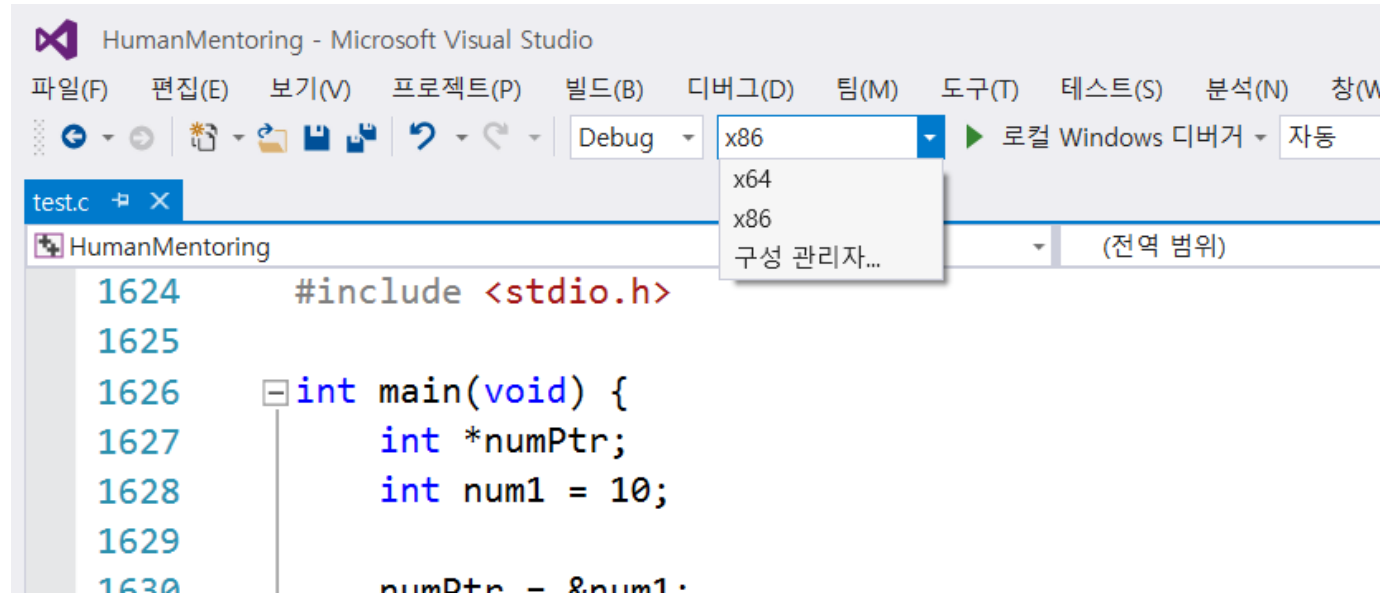


포인터와 변수의 참조관계



포인터 변수 `pnum`에는
변수 `num`의 시작주소가 저장됨

포인터 변수의 크기



포인터 변수의 크기는 몇 비트 시스템이냐에 따라 달라짐

32비트 시스템(x86) => 4바이트

64비트 시스템(x64) => 8바이트

포인터 변수의 크기

<code>int *</code>	int형 포인터
<code>int * pnum1;</code>	int형 포인터 변수 pnum1
<code>double *</code>	double형 포인터
<code>double * pnum2;</code>	double형 포인터 변수 pnum2

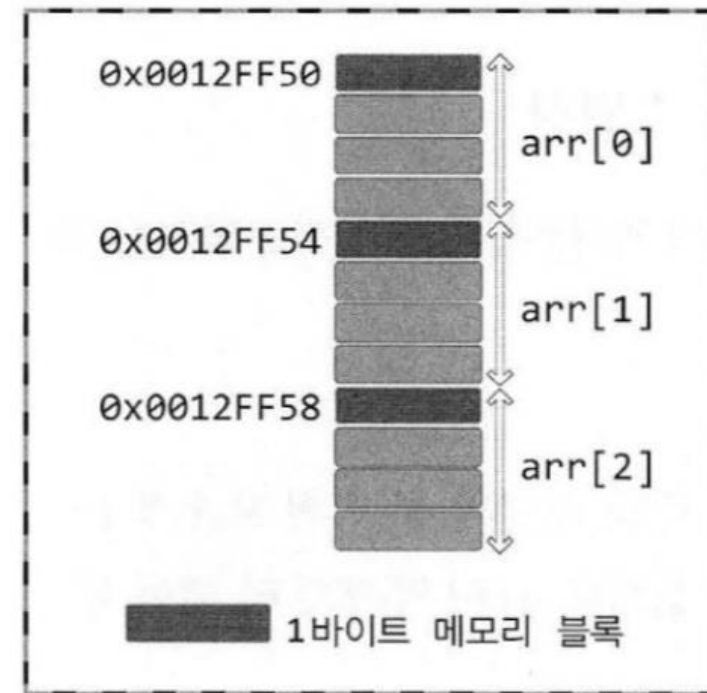
Q : 자료형에 따라서 포인터 변수의 크기가 달라지지 않을까?

A : LL 포인터 변수는 메모리의 주소 값을 저장하기 때문

포인트와 배열

배열의 이름

```
int main(void) {  
  
    int arr[3] = {0, 1, 2};  
  
    printf("배열의 이름: %p \n", arr);  
    printf("첫 번째 요소(0번째 인덱스): %p \n", &arr[0]);  
    printf("두 번째 요소(1번째 인덱스): %p \n", &arr[1]);  
    printf("세 번째 요소(2번째 인덱스): %p \n", &arr[2]);  
  
    return 0;  
}
```



참고) 16진수에서 4의 배수 : 0 4 8 12(C) 16(F)

배열의 이름

배열의 이름은 배열의 시작 주소 값을 의미한다!!

비교조건 \ 비교대상	포인터 변수	배열의 이름
이름이 존재하는가?	존재한다	존재한다
무엇을 나타내거나 저장하는가?	메모리의 주소 값	메모리의 주소 값
주소 값의 변경이 가능한가?	가능하다	불가능하다.

배열의 이름도 포인터처럼 배열의 가장 앞을 가리킴.

하지만 가리키는 대상의 변경은 불가능함.

이를 '상수 형태의 포인터'(혹은 '포인터 상수') 라고 함.

배열의 이름은 배열의 가장 앞을 가리키는 포인터

천천히 치환하면서 가자

1차원 배열 이름의 포인터 타입 결정하는 방법

- 배열의 이름이 가리키는 변수의 자료형을 근거로 판단
- int 형 변수를 가리키면 int * 형
 - int arr1[5]; 에서 **arr1** 은 **int *** 형
- double 형 변수를 가리키면 double * 형
 - double arr2[7]; 에서 **arr2** 는 **double *** 형

```
int main(void) {  
    int arr1[3] = { 1, 2, 3 };  
    double arr2[3] = { 1.1, 2.2, 3.3 };  
  
    printf("%d %g \n", *arr1, *arr2);  
    *arr1 += 100;    배열 이름을 대상으로  
    *arr2 += 120.5; 포인터 연산을 하고 있음에 주목!  
    printf("%d %g \n", arr1[0], arr2[0]);  
    return 0;  
}
```

실행 결과

천천히 치환하면서 가자

```
int main(void)
{
    int arr[3]={1, 2, 3};
    arr[0] += 5;
    arr[1] += 7;
    arr[2] += 9;
    . . . . .
}
```

- arr 은 int 형 포인터이니 int 형 포인터를 대상으로 배열접근을 위한 [idx] 연산을 진행한 셈
- 실제로 포인터 변수 ptr 을 대상으로 ptr[0], ptr[1], ptr[2] 와 같은 방식으로 메모리 공간에 접근이 가능함

```
int main(void) {
    int arr[3] = { 15, 25, 35 };
    int *ptr = &arr[0]; // int * ptr = arr; 과 동일한 문장

    printf("%d %d \n", ptr[0], arr[0]);
    printf("%d %d \n", ptr[1], arr[1]);
    printf("%d %d \n", ptr[2], arr[2]);
    printf("%d %d \n", *ptr, *arr);
    return 0;
}
```

포인터 변수를 이용해서 배열의 형태로
메모리 공간에 접근하고 있음에 주목!

실행 결과

15
25
35
15

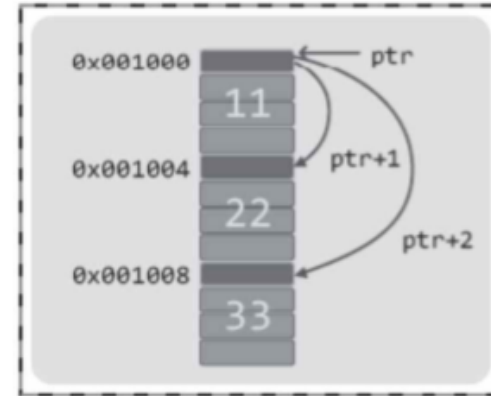
다음 코드를 분석하시오.

```
int main(void) {  
    int arr[3] = { 11, 22, 33 };  
    int * ptr = arr; // int * ptr = &arr[0]; 과 같은 문장  
    printf("%d %d %d \n", *ptr, *(ptr + 1), *(ptr + 2));  
  
    printf("%d ", *ptr); ptr++; // printf 함수호출 후, ptr++ 실행  
    printf("%d ", *ptr); ptr++;  
    printf("%d ", *ptr); ptr--; // printf 함수호출 후, ptr-- 실행  
    printf("%d ", *ptr); ptr--;  
    printf("%d ", *ptr); printf("\n");  
    return 0;  
}
```

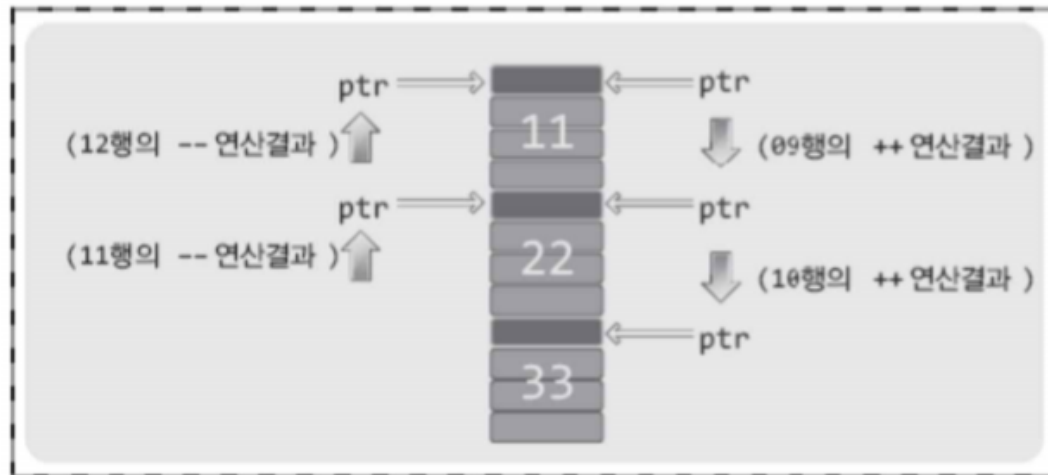
분석 결과

```
int main(void) {
    int arr[3] = { 11, 22, 33 };
    int * ptr = arr; // int * ptr = &arr[0]; 과 같은 문장
    printf("%d %d %d \n", *ptr, *(ptr + 1), *(ptr + 2));

    printf("%d ", *ptr); ptr++; // printf 함수호출 후, ptr++ 실행
    printf("%d ", *ptr); ptr++;
    printf("%d ", *ptr); ptr--; // printf 함수호출 후, ptr-- 실행
    printf("%d ", *ptr); ptr--;
    printf("%d ", *ptr); printf("\n");
    return 0;
}
```



실행 결과



int 형 포인터 변수의 **값은 4씩 증가 및 감소**를 하니, int 형 포인터 변수가 **int 형 배열을 가리키면**, int 형 포인터 변수의 값을 증가 및 감소시켜서 배열 요소에 순차적으로 접근이 가능함

분석 결과

중요한 결론! $\text{arr}[i] == *(\text{arr} + i)$

```
int main(void)
{
    int arr[3]={11, 22, 33};
    int * ptr = arr;
    printf("%d %d %d \n", *ptr, *(ptr+1), *(ptr+2));
    . . . .
}
```

배열이름도 포인터이니, 포인터 변수를 이용한 배열의 접근 방식을 배열의 이름에도 사용할 수 있음. 배열의 이름을 이용한 접근방식도 포인터 변수를 대상으로 사용할 수 있음. 결론은 **arr 이 포인터 변수의 이름이건 배열의 이름이건**

$\text{arr}[i] == *(\text{arr} + i)$

```
printf("%d %d %d \n", *ptr, *(ptr+1), *(ptr+2));    // *(ptr+0)는 *ptr 과 같음
printf("%d %d %d \n", ptr[0], ptr[1], ptr[2]);
printf("%d %d %d \n", *(arr+0), *(arr+1), *(arr+2)); // *(arr+0)는 *arr 과 같음
printf("%d %d %d \n", arr[0], arr[1], arr[2]);
```

Summary

1. 배열의 이름은 배열의 가장 첫 번째 인덱스를 가리키는 포인터
2. 배열의 첫 번째 요소에 포인터를 가리키고 포인터를 하나 증가시키면, 배열의 그 다음 요소를 가리킨다.

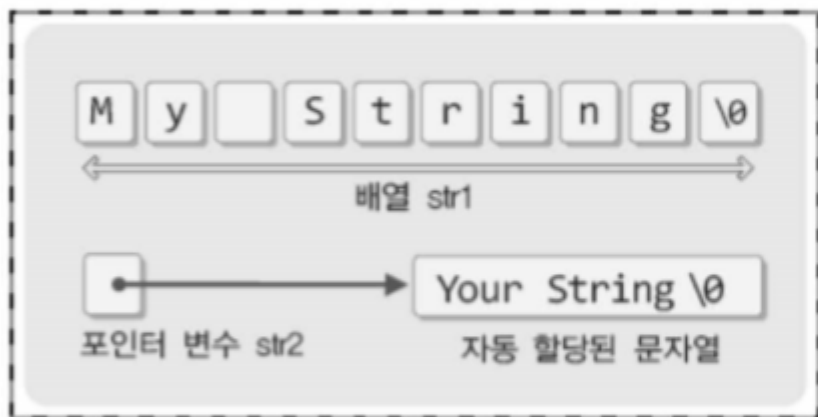
(포인터 하나 올리면 배열도 하나 같이 올라감)

두 가지 형태의 문자열 표현

```
char str1[ ] = "My String";  
char * str2 = "Your String";
```



문자열의 저장방식



```
int main(void){  
    char * str = "Your team";  
    str = "Our team"; // 의미 있는 문장  
    . . . . .  
}
```

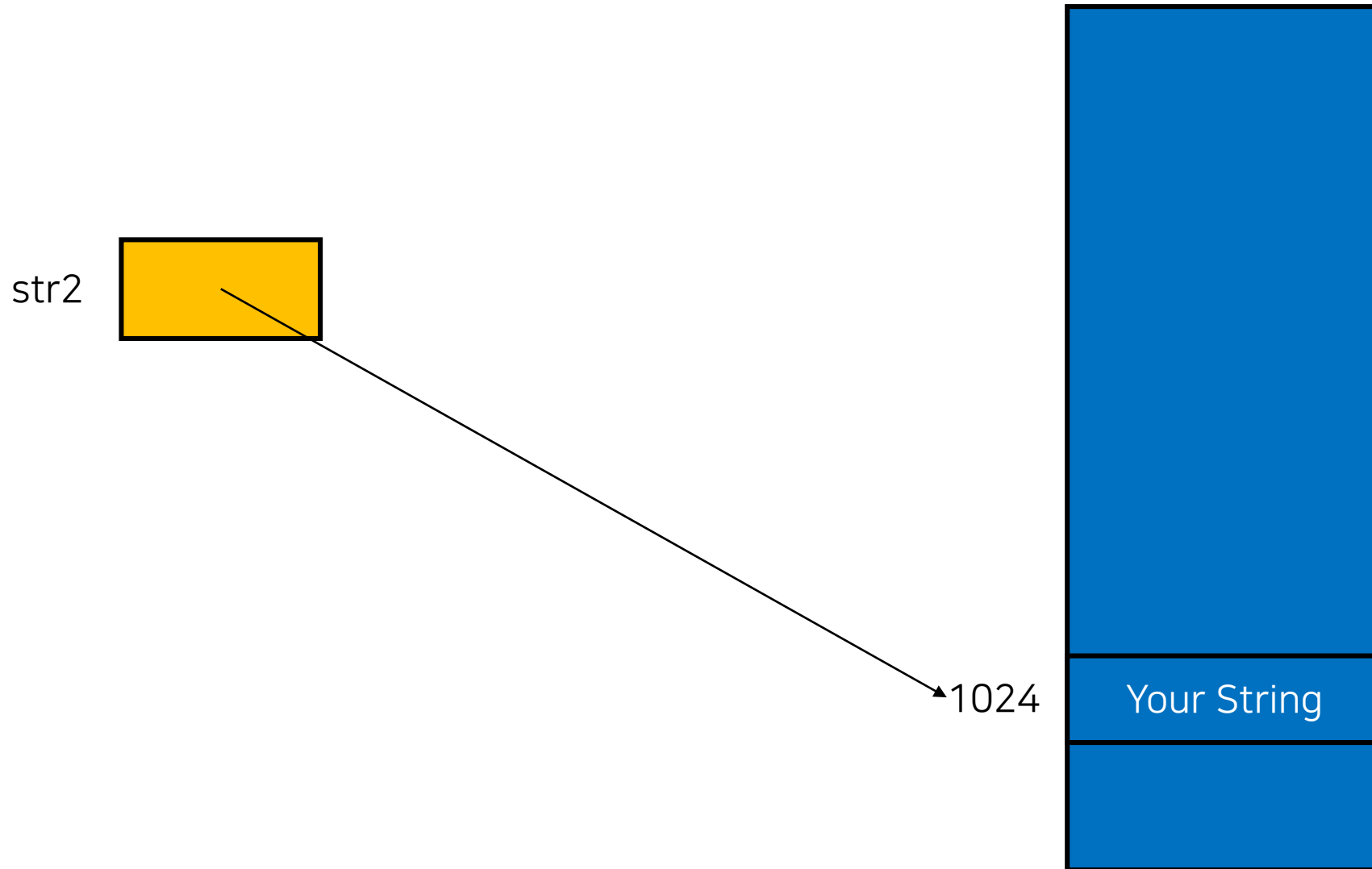
```
int main(void){  
    char str[] = "Your team";  
    str = "Our team"; // 의미 없는 문장  
    . . . . .  
}
```

- `str1` 은 문자열이 저장된 배열. 즉, 문자열 배열임. 따라서 변수 성향의 문자열
- `str2` 는 문자열의 주소 값을 저장함. 즉, 자동 할당된 문자열의 주소 값을 저장함.
따라서 상수 성향의 문자열

두 가지 형태의 문자열 표현

str1	M	y		S	t	r	i	n	g	₩0
	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009

두 가지 형태의 문자열 표현



포인터 배열의 선언

```
int * arr1[20];    // 길이가 20인 int 형 포인터 배열 arr1
double * arr2[30]; // 길이가 30인 double 형 포인터 배열 arr2
```

```
int main(void) {
    int num1 = 10, num2 = 20, num3 = 30;
    int* arr[3] = {&num1, &num2, &num3};

    printf("%d \n", *arr[0]);
    printf("%d \n", *arr[1]);
    printf("%d \n", *arr[2]);
    return 0;
}
```

주소를 담을 수 있는 배열 3칸이 만들어 진 것임
= 포인터 배열

실행 결과



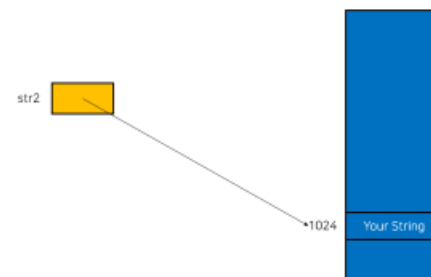
- 포인터 배열이라 해서 일반 배열의 선언과 차이가 나지는 않음.
- 변수의 자료형을 표시하는 위치에 **int** 나 **double** 을 대신해서 **int*** 나 **double*** 가 올 뿐.

포인터 배열의 선언

```
int main(void) {  
    char * strArr[3] = { "Simple", "String", "Array" };  
    printf("%s \n", strArr[0]);  
    printf("%s \n", strArr[1]);  
    printf("%s \n", strArr[2]);  
    return 0;  
}
```

실행 결과

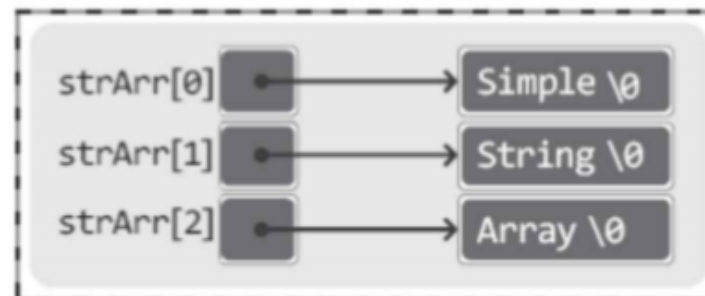
두 가지 형태의 문자열 표현



`char * strArr[3] = {"Simple", "String", "Array"};`



`Char * strArr[3] = {0x1004, 0x1048, 0x2012}; // 반환된 주소 값은 임의로 결정한 것`



포인터와 함수

함수의 인자로 배열 전달하기

메인함수

```
int main(void) {  
  
    int age = 20;  
    SimpleFunc(age);  
  
}
```

함수

```
void SimpleFunc(int num) {  
  
  
  
  
  
  
}
```

여기서 값 바꾼다고 메인함수에 있는 거 안 바뀔
즉, 복사가 된 것임

함수호출 시 전달되는 인자의 값은
매개변수에 복사가 된다.

함수의 인자로 배열 전달하기

메인함수

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

10	20	30
1000	1004	1008

함수

```
void SimpleFunc(      ) {  
  
  
  
  
  
  
  
  
}
```

10	20	30
----	----	----

그냥 값이 복사되어 넘어갈 시,
여기서 아무리 수정해도 본 배열에는 영향 없음

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

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?”

10	20	30
1000	1004	1008

Step 3. 포인터를 통해서 원래 배열에 접근 가능(SimpleFunc)

“접근 어떤 방식으로 할 수 있다고 했지?”

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 };  
    ShowArrElem(arr1, sizeof(arr1) / sizeof(int));  
    ShowArrElem(arr2, sizeof(arr2) / sizeof(int));  
    return 0;  
}
```

1. 주소 값을 보내고

함수의 인자로 배열 전달하기

★ 국룰 ★

외부 함수에서 메인 함수에 있는 값을 건드리고 싶다면

함수의 전달인자로 주소 값을 보내주어라.

그리고 외부 함수에서는 매개변수로

포인터 변수를 선언하여 그 주소 값을 받아라.

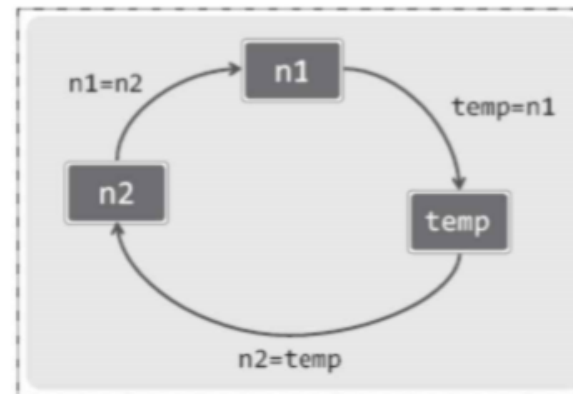
Call-by-value

```
void Swap(int n1, int n2) {  
    int tmp = n1;  
    n1 = n2;  
    n2 = tmp;  
    printf("n1, n2: %d, %d \n", n1, n2);  
}  
  
int main(void) {  
    int num1 = 10;  
    int num2 = 20;  
    printf("num1, num2: %d, %d \n", num1, num2);  
  
    // num1과 num2에 저장된 값이 서로 바뀌길 기대!  
    Swap(num1, num2);  
    printf("num1, num2: %d, %d \n", num1, num2);  
    return 0;  
}
```

num1, num2:
n1, n2:
num1, num2:

실행 결과

call-by-value 가 적절치 않은 경우



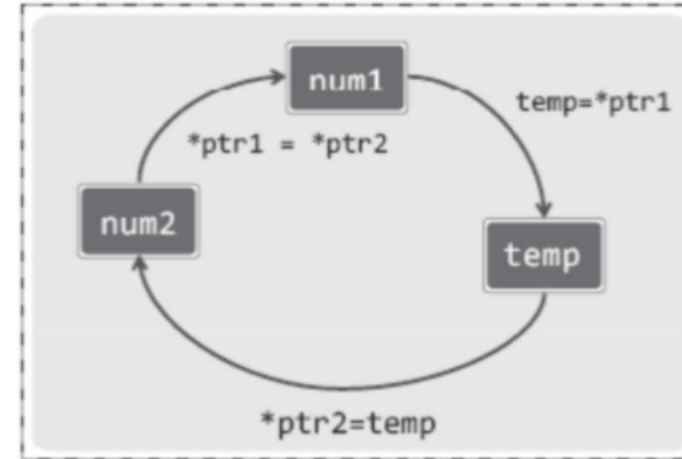
Swap 함수 내에서의 값의 교환



Swap 함수 내에서의 값의 교환은 외부에 영향을 주지 않음

Call-by-reference

```
void Swap(int* ptr1, int* ptr2) {  
    int tmp = *ptr1;  
    *ptr1 = *ptr2;  
    *ptr2 = tmp;  
}  
  
int main(void) {  
    int num1 = 10;  
    int num2 = 20;  
    printf("num1, num2: %d, %d \n", num1, num2);  
  
    Swap(&num1, &num2);  
    printf("num1, num2: %d, %d \n", num1, num2);  
    return 0;  
}
```



Swap 함수 내에서 함수 외부에 있는 변수간
값의 교환

num1, num2:	<input type="text"/>
num1, num2:	<input type="text"/>

 실행 결과

- Swap 함수 내에서의 *ptr1 은 main 함수의 num1 을 의미
- Swap 함수 내에서의 *ptr2 은 main 함수의 num2 을 의미

scanf 함수 호출 시 & 연산자를 붙이는 이유

```
int main(void) {  
    int num;  
    scanf("%d", &num);  
}
```

문자열 입력 받을 때 왜 & 연산자가 없었을까

배열을 이용한 문자열 변수의 표현

```
*****입력*****
// 직접 입력 (Hard Coding)
char str[14] = "Good morning!";

// scanf 통한 입력
char str[100];
scanf("%s", str);
*****
```

```
*****출력*****
// 한 번에 출력
printf("문자열 출력: %s \n", str);

// 각 인덱스 접근해서 출력
int index = 0;
while(str[index] != '\0') {
    printf("%c", str[index]);
    index++;
}
*****
```