

# C Programming

## Ch.7-3 포인터와 함수에 대한 이해

# Contents

---

1. 함수의 인자로 배열 전달하기
2. Call-by-value vs. Call-by-reference
3. 포인터와 const

# 기본적인 인자의 전달 방식

## ▶ 값에 의한 전달 (Call-by-value)

```
int main(void)
{
    int num1 = 3, num2 = 4;

    int result = Sum(num1, num2);

    print("%d \n", result);
}
```

num1과 n1은 다른 변수  
num2와 n2도 다른 변수  
값이 전달된다!  
→ Call-by-value

```
int Sum(int n1, int n2)
{
    return n1 + n2;
}
```

### C 언어의 인자 전달 방식

- 원칙적으로 Call-by-value(값 전달)만 존재
- 흔히 얘기하는 C 언어의 Call-by-reference 또한 Call-by-value임(뒤에서 확인)

# 배열을 함수의 인자로 전달하는 방식

- ▶ 배열 전체를 값의 복사에 의해 전달하는 방법은 없음!
- ▶ 배열과 포인터의 관계 (Review)

```
int main(void)
{
    int arr[3] = { 1, 2, 3 };
    int* p = arr;                // 첫 번째 원소의 주소

    for (int i = 0; i < 3; i++)
        print("%d \n", p[i]);    // 포인터는 배열처럼 사용 가능!
}
```

- ▶ 배열 이름(=배열 첫 요소 주소=포인터) 전달
  - 포인터로 받음

# 배열을 함수의 인자로 전달 예

```
int SumAndChange(int* p, int count)
{
    int sum = 0;
    for (int i = 0; i < count; i++)
    {
        sum += p[i];           // sum += *(p + i);
        p[i]++;
    }
    return sum;
}
```

- 배열 첫 요소 주소 전달 → 포인터
  - 요소의 개수도 함께 전달
- 포인터는 배열과 동일하게 사용
- 값 변경 시 원본 배열의 값이 변경됨

```
int main(void)
{
    int arr[5] = { 1, 2, 3, 4, 5 };
    int sum = SumAndChange(arr, 5);

    printf("합계 : %d \n", sum);
    for (int i = 0; i < 5; i++)
        printf("[%d] %d \n", i, arr[i]);
}
```

합계 : 15

[0] 2

[1] 3

[2] 4

[3] 5

[4] 6

# 배열 이름과 포인터에 대한 sizeof 연산 결과는?

- ▶ sizeof(배열이름) : 배열 전체의 바이트 수 반환
- ▶ sizeof(포인터) : 포인터 크기 반환 → 타입 무관 항상 4

64비트 시스템 : 8

```
int main(void)
{
    int arr[5] = { 1, 2, 3, 4, 5 };
    int* p = arr;

    printf("sizeof(배열명) : %d \n", sizeof(arr));
    printf("sizeof(포인터) : %d \n", sizeof(p));
    printf("원소 개수 : %d \n", sizeof(arr) / sizeof(int));
}
```

sizeof(배열명) : 20  
sizeof(포인터) : 4  
원소 개수 : 5

# 배열을 인자로 전달받는 함수의 또 다른 선언

- ▶ `int Func(int p[], int count);`
  - `int* p` 와 `int p[]` 는 같다!
    - `int p[5]`도 가능하나 대괄호 내의 숫자는 무의미
  - 일반적인 변수 선언 시에는 `int p[] = &num;` 불가능
- ▶ 예 : 배열 요소 중 최대값 반환

```
int MaxVal(int p[], int n);

int main(void)
{
    int arr[10] = { 4, 8, 3, 7, 2 };
    int max;

    max = MaxVal(arr, sizeof(arr) / sizeof(int));
    printf("최대 값 : %d \n", max);
}
```

```
int MaxVal(int p[], int n)
{
    int max, i;

    max = p[0];
    for (i = 1; i < n; i++)
        if (max < p[i])
            max = p[i];

    return max;
}
```

# Call-by-value와 Call-by-reference

## ▶ Call-by-value : 값 전달

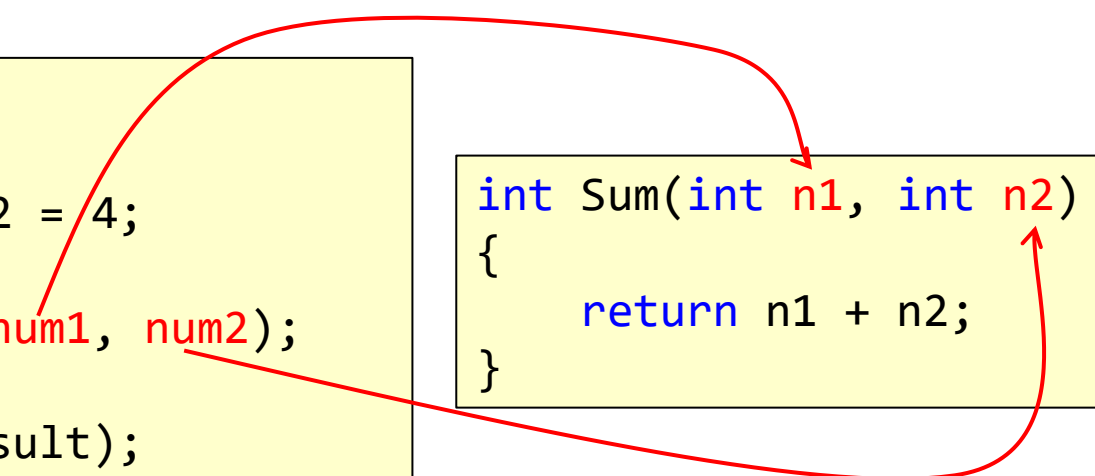
### ◦ 예 (Review)

```
int main(void)
{
    int num1 = 3, num2 = 4;

    int result = Sum(num1, num2);

    print("%d \n", result);
}
```

```
int Sum(int n1, int n2)
{
    return n1 + n2;
}
```



Call-by-reference도 결국 값 전달의 일종. 용어 자체도 공식 용어가 아님 → 용어보다 동작 원리 이해 중요

## ▶ Call-by-reference : 주소값 전달

### ◦ 예 : 배열 전달 시 첫 번째 요소의 주소값 전달



# 잘못 적용된 Call-by-value

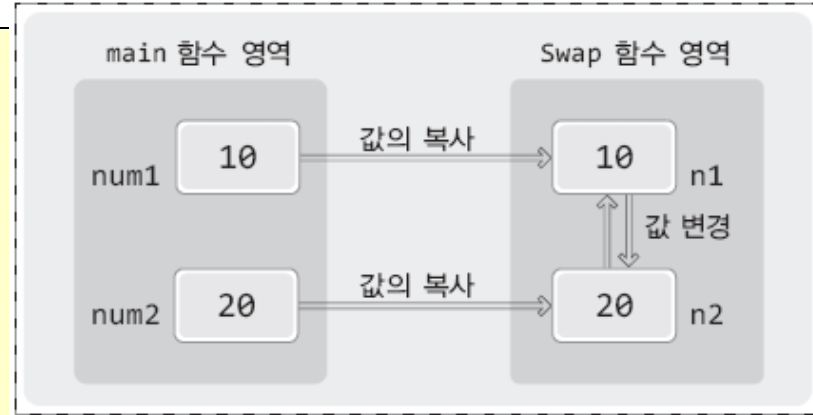
- ▶ 두 변수의 값을 교환하는 Swap 함수 작성

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

    Swap(num1, num2);    // num1과 num2에 저장된 값이 서로 바뀌길 기대!
    printf("num1 num2: %d %d \n", num1, num2);
}
```

```
num1 num2: 10 20
n1 n2: 20 10
num1 num2: 10 20
```



Swap 함수 내에서의 값 교환  
은 외부에 영향을 주지 않음

# 주소값을 전달하는 Call-by-reference

## ▶ Call-by-reference

- 변수의 주소값을 전달함으로써 포인터를 사용하여 간접적으로 원본 변수의 값 사용 가능

```
void Adder(int* n)
{
    (*n)++;
}

int main(void)
{
    int num = 1;
    printf("num : %d \n", num);

    Adder(&num);
    printf("num : %d \n", num);
}
```

num 변수의 주소값이 포인터 변수 n으로 전달됨  
→ n을 통해 num 값 사용 가능

num : 1  
num : 2

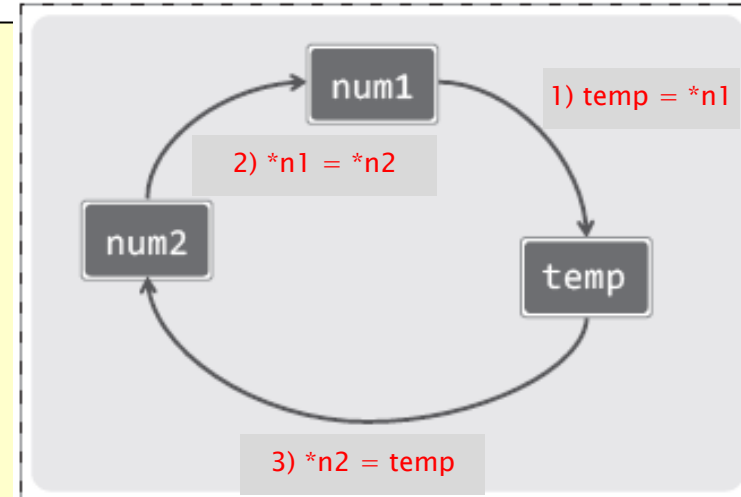
# Call-by-reference에 의한 Swap 함수

```
void Swap(int* n1, int* n2)
{
    // n1과 n2는 num1과 num2를 가리킴
    int temp = *n1;
    *n1 = *n2;
    *n2 = temp;
    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);
```

```
    Swap(&num1, &num2); // num1과 num2에 저장된 값이 서로 바뀌길 기대!
    printf("num1 num2: %d %d \n", num1, num2);
}
```

```
num1 num2: 10 20
*n1 *n2: 20 10
num1 num2: 20 10
```



포인터 변수 n1과 n2를 통해  
num1과 num2 값 교환

# scanf 함수 호출 시 &를 붙이는 이유

- ▶ scanf 함수를 통해 주소값을 전달해야 값 변경 가능!

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

변수 num의 주소를 전달하여 사용자 입력값을 num에 넣는다.

```
int main(void)
{
    char str[30];
    scanf("%s", str);
    . . . .
}
```

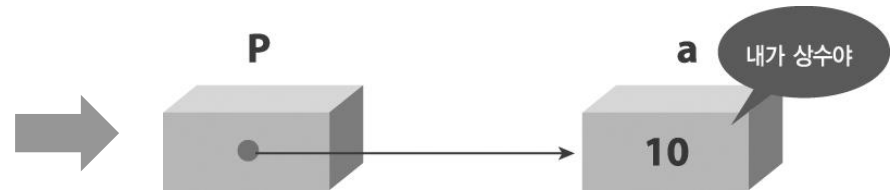
저장소의 주소를 알려 줄테니 값을 저장해줘.

배열 이름 자체가 첫 번째 요소의 주소값  
이므로 주소 연산자(&)를 붙이지 않는다.  
→ 역시 주소 전달!

# 포인터 선언 시 const를 붙이는 2가지 방법

## ▶ 1. 포인터가 가리키는 변수의 상수화

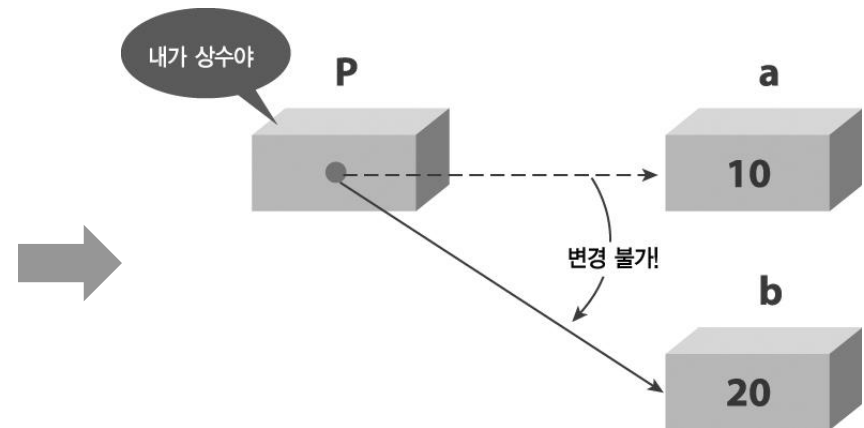
```
int a = 10;
const int* p = &a;
*p = 30;           // Error!
a = 30;            // OK!
```



- p를 통해 a의 값 변경 불가
- p가 다른 변수를 가리킬 수 있음

## ▶ 2. 포인터 자체의 상수화

```
int a = 10;
int b = 20;
int* const p = &a;
p = &b;           // Error!
*p = 30;          // OK!
```



- p를 통해 a의 값 변경 가능
- p가 다른 변수를 가리킬 수 없음 (반드시 p 선언과 함께 초기화)

# 포인터 선언 시 const를 붙이는 2가지 방법

- ▶ const 키워드가 둘 다 붙는 경우
  - p를 통해 a의 값 변경 불가
  - p가 다른 변수를 가리킬 수 없음 (선언과 동시에 초기화)

```
int a = 10;  
int b = 20;  
const int * const p = &a;  
p = &b;           // Error!  
*p = 30;          // Error!
```

# const 키워드를 사용하는 이유

- ▶ 잘못된 변수 값 변경에 대한 예러 처리
  - 코드의 안정성 증가

```
#include <stdio.h>
float PI = 3.14;
int main(void)
{
    float rad;
    PI = 3.07; // 분명히 실수!!

    scanf("%f", &rad);
    printf("원의 넓이는 %f \n",
           rad*rad*PI);
}
```



```
#include <stdio.h>
const float PI = 3.14;
int main(void)
{
    float rad;
    PI = 3.07; // Compile Error 발생!

    scanf("%f", &rad);
    printf("원의 넓이는 %f \n",
           rad*rad*PI);
}
```

# 생각해 보기

- ▶ 프로그래머 A가 Print라는 함수를 만들면서 다음과 같이 함수 프로토타입을 설계하였다. 이 함수에서 매개 변수 arr의 선언 시 const를 사용하였는데, 이와 관련된 프로그래머 A의 의도는 무엇일까?
  - `void Print(const int *arr, int size);`



# 이번 장에서 배운 것

- C 언어의 기본적인 매개변수 전달 방법은 값에 의한 전달(Call-by-value)이다.
- 매개변수 전달 시 변수의 주소값 전달이 가능한데, 이를 참조에 의한 전달(Call-by-reference)이라 한다. 엄밀히 말하자면 이 또한 값에 의한 전달의 일종이다. 진짜 참조에 의한 전달은 C++ 언어를 통해 배우게 된다.
- 배열 자체를 값에 의한 전달로 전달할 수는 없다. 배열을 함수를 전달하기 위해서는 배열의 첫 번째 요소의 주소값을 전달한다. 함수에서는 포인터로 받아서 배열처럼 사용할 수 있다. 이는 주소값 전달(참조에 의한 전달)의 한 예이다.
- 포인터 변수 선언 시 `const` 키워드를 두 가지 방식으로 추가할 수 있으며, 각각 대상의 상수화와 포인터 변수 자체의 상수화를 의미한다.