



C프로그래밍

Lecture 14. 포인터와 함수에 대한 이해

동덕여자대학교
데이터사이언스 전공
권 범

목차

- ❖ 01. 함수의 인자로 배열 전달하기
- ❖ 02. Call-by-value vs. Call-by-reference
- ❖ 03. 포인터 대상의 const 선언
- ❖ 04. 연습 문제

01. 함수의 인자로 배열 전달하기

02. Call-by-value vs. Call-by-reference

03. 포인터 대상의 const 선언

04. 연습 문제

01. 함수의 인자로 배열 전달하기

❖ 인자전달의 기본방식은 값의 복사다!

- 배열을 함수의 매개변수에 전달하는 이유는 함수 내에서 배열에 저장된 값을 참조하기 위함입니다.
- 그런데 배열을 통째로 전달하지 않아도 이러한 일이 가능합니다.

```
...
int SimpleFunc(int num)
{
    ...
}
int main(void)
{
    int age = 17;
    SimpleFunc(age);
    ...
}
```

age에 저장된 값이
매개변수 num에 복사됩니다.

실제 전달되는 것은 age가 아니라
age에 저장된 값입니다.

코드에서 보이는 바와 같이, 배열을 함수의 인자로 전달하려면
배열을 통째로 복사할 수 있도록 배열이 매개변수로 선언되어야 합니다.

- ✓ 그러나 C언어는 매개변수로 배열의 선언을 허용하지 않습니다.
- ✓ 결론! 배열을 통째로 복사하는 방법은 C언어에 존재하지 않습니다.
- ✓ 배열을 통째로 복사해서 전달하는 방식 대신에, 배열의 주소 값을 전달하는 방식을 취합니다.

01. 함수의 인자로 배열 전달하기

❖ 배열을 함수의 인자로 전달하는 방식

```
int main(void)
{
    int arr[3] = {1, 2, 3};
    int* ptr = arr;
    ...
}
```

배열의 이름은 int형 포인터!

- ✓ 배열의 이름은 int형 포인터입니다!
- ✓ 따라서 int형 포인터 변수에 배열의 이름이 지니는 주소 값을 저장할 수 있습니다.

위의 예제를 통해서 아래와 같은 코드의 구성이 가능함을 유추할 수 있습니다.

```
void SimpleFunc(int* param)
{
    printf("%d %d\n", param[0], param[1]);
}

int main(void)
{
    int arr[3] = {1, 2, 3};
    SimpleFunc(arr);
    ...
}
```

배열 이름 arr은 int형 포인터이므로
매개변수는 int형 포인터 변수!


포인터 변수를 이용해서도 배열의 형태로 접근 가능!

배열 이름 arr이 지니는 주소 값의 전달

01. 함수의 인자로 배열 전달하기

❖ 배열을 함수의 인자로 전달하는 예제 (1/2)

```
1  #include <stdio.h>
2  void ShowArrayElement(int* param, int len)
3  {
4      int j;
5      for (j = 0; j < len; j++)
6          printf("%d ", param[j]);
7      printf("\n");
8  }
9
10 int main(void)
11 {
12     int arr1[3] = {1, 2, 3};
13     int arr2[5] = {4, 5, 6, 7, 8};
14
15     ShowArrayElement(arr1, sizeof(arr1) / sizeof(int));
16     ShowArrayElement(arr2, sizeof(arr2) / sizeof(int));
17     return 0;
18 }
```




1 2 3
4 5 6 7 8

01. 함수의 인자로 배열 전달하기

❖ 배열을 함수의 인자로 전달하는 예제 (2/2)

```
1  #include <stdio.h>
2  void ShowArrayElement(int* param, int len)
3  {
4      int j;
5      for (j = 0; j < len; j++)
6          printf("%d ", param[j]);
7      printf("\n");
8  }
9
10 void AddArrayElement(int* param, int len, int add)
11 {
12     int j;
13     for (j = 0; j < len; j++)
14         param[j] += add;
15 }
16
```

```
17 int main(void)
18 {
19     int arr[3] = {1, 2, 3};
20     AddArrayElement(arr, sizeof(arr) / sizeof(int), 1);
21     ShowArrayElement(arr, sizeof(arr) / sizeof(int));
22     AddArrayElement(arr, sizeof(arr) / sizeof(int), 2);
23     ShowArrayElement(arr, sizeof(arr) / sizeof(int));
24     AddArrayElement(arr, sizeof(arr) / sizeof(int), 3);
25     ShowArrayElement(arr, sizeof(arr) / sizeof(int));
26     return 0;
27 }
```



2	3	4
4	5	6
7	8	9

01. 함수의 인자로 배열 전달하기

❖ 배열을 함수의 인자로 전달받는 함수의 또 다른 선언

```
void ShowArrayElement(int* param, int len){ ... }  
void AddArrayElement(int* param, int len, int add){ ... }
```

동일한 선언!

```
void ShowArrayElement(int param[], int len){ ... }  
void AddArrayElement(int param[], int len, int add){ ... }
```

- ✓ 매개변수의 선언에서는 int* param과 int param[]은 동일한 선언입니다.
- ✓ 따라서 배열을 인자로 전달받는 경우에는 int param[]이 더 의미 있어 보이므로 주로 사용됩니다.

```
int main(void)  
{  
    int arr[3] = {1, 2, 3};  
    int* ptr = arr;           // int ptr[] = arr;로 대체 불가능!  
}
```

하지만 그 이외의 영역에서는 int* ptr의 선언을 int ptr[]으로 대체할 수 없습니다.

02. Call-by-value vs. Call-by-reference

- 01. 함수의 인자로 배열 전달하기
- 03. 포인터 대상의 const 선언
- 04. 연습 문제

02. Call-by-value vs. Call-by-reference

❖ 값을 전달하는 형태의 함수 호출: Call-by-value

- 함수를 호출할 때 단순히 값을 전달하는 형태의 함수 호출을 가리켜 Call-by-value라 하고,
- 메모리의 접근에 사용되는 주소 값을 전달하는 형태의 함수 호출을 가리켜 Call-by-reference라 합니다.
- 즉, Call-by-value와 Call-by-reference를 구분하는 기준은 함수의 인자로 전달되는 대상에 있습니다.

```
void NoReturnType(int num)
{
    if(num < 0)
        return;
    ...
}
```

Call-by-value

```
void ShowArrayElement(int* param, int len)
{
    int j;
    for (j = 0; j < len; j++)
        printf("%d ", param[j]);
    printf("\n");
}
```

Call-by-reference

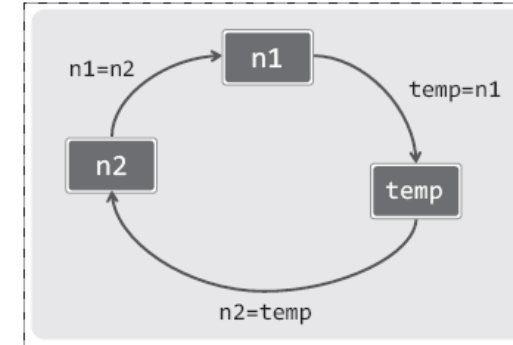
- ✓ Call-by-value와 Call-by-reference라는 용어를 기준으로 구분하는 것이 중요한 게 아닙니다.
- ✓ 중요한 것은 각 함수의 특징을 이해하고 적절한 형태의 함수를 정의하는 것입니다.
- ✓ Call-by-value 형태의 함수에서는 함수 외부에 선언된 변수에 접근이 불가능합니다.
- ✓ 그러나 Call-by-reference 형태의 함수에서는 외부에 선언된 변수에 접근이 가능합니다.

02. Call-by-value vs. Call-by-reference

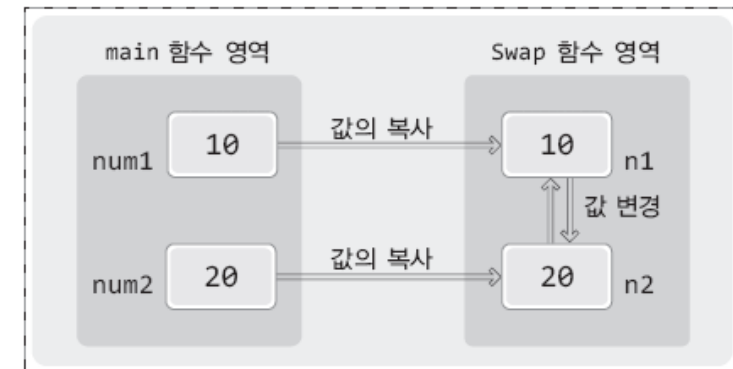
❖ 잘못 적용된 Call-by-value

```
1  #include <stdio.h>
2  void Swap(int n1, int n2)
3  {
4      int temp = n1;
5      n1 = n2;
6      n2 = temp;
7      printf("n1 n2: %d %d\n", n1, n2);
8  }
9  int main(void)
10 {
11     int num1 = 10, num2 = 20;
12     printf("num1 num2: %d %d\n", num1, num2);
13
14     Swap(num1, num2); // num1과 num2에 저장된 값이 서로 바뀌길 기대!
15     printf("num1 num2: %d %d\n", num1, num2);
16     return 0;
17 }
```

Swap 함수 내에서의 값의 교환



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



**Swap 함수 내에서의 값의 교환은
외부에 영향을 주지 않습니다!**

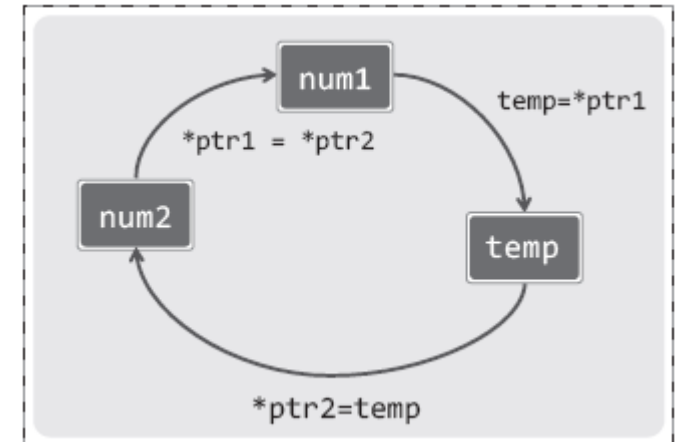
02. Call-by-value vs. Call-by-reference

❖ 주소 값을 전달하는 형태의 함수 호출: Call-by-reference

```
1  #include <stdio.h>
2  void Swap(int* ptr1, int* ptr2)
3  {
4      int temp = *ptr1;
5      *ptr1 = *ptr2;
6      *ptr2 = temp;
7  }
8
9  int main(void)
10 {
11     int num1 = 10, num2 = 20;
12     printf("num1 num2: %d %d\n", num1, num2);
13
14     Swap(&num1, &num2);
15     printf("num1 num2: %d %d\n", num1, num2);
16     return 0;
17 }
```

- ✓ Swap 함수 내에서의 *ptr1은 main 함수의 num1을 의미하게 됩니다.
- ✓ 그리고 *ptr2는 main 함수의 num2를 의미하게 됩니다.

num1 num2: 10 20
num1 num2: 20 10



Swap 함수 내에서
함수 외부에 있는 변수간 값의 교환

02. Call-by-value vs. Call-by-reference

❖ scanf 함수 호출 시 & 연산자를 붙이는 이유

```
#include <stdio.h>

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

```
#include <stdio.h>

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

변수 num 앞에 & 연산자를 붙이는 이유는?

- ✓ scanf 함수 내에서 외부에 선언된 변수 num에 접근 하기 위해서는 num의 주소 값을 알아야 합니다.
- ✓ 그래서 scanf 함수는 변수의 주소 값을 요구하는 것입니다.

배열 이름 str 앞에 & 연산자를 붙이지 않는 이유는?

- ✓ str은 배열의 이름이고 그 자체가 주소 값이기 때문에 & 연산자를 붙이지 않습니다.
- ✓ str을 전달하는 것은 scanf 함수 내부로 배열 str의 주소 값을 전달하는 것과 같습니다.

03. 포인터 대상의 const 선언

- 01. 함수의 인자로 배열 전달하기
- 02. Call-by-value vs. Call-by-reference
- 04. 연습 문제

03. 포인터 대상의 const 선언

❖ 포인터 변수의 참조 대상에 대한 const 선언

```
int main(void)
{
    int num = 20;
    const int* ptr = &num;
    *ptr = 30;      // 컴파일 에러!
    num = 40;      // 컴파일 성공!
    ...
}
```

왼편의 const 선언이 갖는 의미

- ✓ 포인터 변수 ptr을 이용해서 ptr이 가리키는 변수에 저장된 값을 변경하는 것을 허용하지 않겠다는 의미입니다!

변수 num에 저장된 값 자체의 변경이 불가능한 것은 아닙니다.
단지 ptr을 통한 변경을 허용하지 않을 뿐입니다.

03. 포인터 대상의 const 선언

❖ 포인터 변수의 상수화

```
int main(void)
{
    int num1 = 20;
    int num2 = 30;
    int* const ptr = &num1;

    ptr = &num2;    // 컴파일 에러!
    *ptr = 40;      // 컴파일 성공!
    ...
}
```

왼편의 const 선언이 갖는 의미

- ✓ 포인터 변수 ptr에 저장된 값을 상수화 하겠다는 의미입니다.
- ✓ 즉, ptr에 저장된 값은 변경이 불가능합니다.
- ✓ ptr이 가리키는 대상의 변경을 허용하지 않습니다.

```
const int* ptr = &num;
int* const ptr = &num;
```

```
const int* const ptr = &num;
```

두 가지 const 선언을 동시에 할 수 있습니다.

03. 포인터 대상의 const 선언

❖ const 선언이 갖는 의미

```
1  #include <stdio.h>
2  int main(void)
3  {
4      double PI = 3.1415;
5      double rad;
6      PI = 3.07;      // 실수로 잘못 삽입된 문장, 컴파일 시 발견 안됨.
7      scanf("%lf", &rad);
8      printf("circle area %f\n", PI * rad * rad);
9      return 0;
10 }
```

```
1  #include <stdio.h>
2  int main(void)
3  {
4      const double PI = 3.1415;
5      double rad;
6      PI = 3.07;      // 컴파일 시 발견되는 오류 상황.
7      scanf("%lf", &rad);
8      printf("circle area %f\n", PI * rad * rad);
9      return 0;
10 }
```



const 선언을 통해
안정성이 높아집니다

- ✓ const 선언은 추가적인 기능을 제공하기 위한 것이 아니라, 코드의 안전성을 높이기 위한 것입니다.
- ✓ 따라서 이러한 const의 선언을 소홀히하기 쉬운데, const의 선언과 같이 코드의 안전성을 높이는 선언은 가치가 매우 높은 선언입니다.

04. 연습 문제

- 01. 함수의 인자로 배열 전달하기
- 02. Call-by-value vs. Call-by-reference
- 03. 포인터 대상의 const 선언

04. 연습 문제

❖ 연습 문제 1.

- 사용자로부터 정수를 하나 입력받아서 변수 a에 저장합니다. 입력받은 값의 제곱을 계산해서 다시 a에 대입해 주는 프로그램을 구현해 보세요. 단, 입력받은 값의 제곱을 계산하는 함수를 독립적으로 구현해서 main 함수가 이를 호출하는 형식으로 구현하세요.
함수의 이름은 Square라고 하세요. 아래의 두 가지 방법으로 함수를 구현할 수 있습니다.

Call-by-value에 의한 방법

- ✓ Square 함수 호출 시 변수 a를 전달합니다.
- ✓ 그리고 함수는 제곱 값을 계산해서 반환해 줍니다.
- ✓ 반환 값을 변수 a에 다시 저장합니다.

함수 호출 형태: `a = Square(a);`

Call-by-reference에 의한 방법

- ✓ Square 함수 호출 시 변수 a의 주소 값을 전달합니다.
- ✓ 그리고 함수는 주소 값을 참조해서 변수 a의 값을 알아낸 다음 제곱 값을 계산합니다.
- ✓ 그리고 변수 a의 값을 변경합니다(주소 값을 알고 있으므로 변수 a로의 접근이 가능합니다).

함수 호출 형태: `Square(&a);`

04. 연습 문제

❖ 연습 문제 1. 정답 및 해설 (1/2)

```
1  /* example1.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4  int Sqaure(int n)
5  {
6      return n * n;
7  }
8
9  int main(void)
10 {
11     int a;
12     printf("정수 입력: ");
13     scanf("%d", &a);
14
15     a = Sqaure(a);
16     printf("연산 결과: %d\n", a);
17
18     return 0;
19 }
```

Call-by-value에 의한 방법

정수 입력: 5
연산 결과: 25

04. 연습 문제

❖ 연습 문제 1. 정답 및 해설 (2/2)

```
1  /* example1.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4  void Sqaure(int* pA)
5  {
6      *pA = (*pA) * (*pA);
7  }
8
9  int main(void)
10 {
11     int a;
12     printf("정수 입력: ");
13     scanf("%d", &a);
14
15     Sqaure(&a);
16     printf("연산 결과: %d\n", a);
17
18     return 0;
19 }
```

Call-by-reference에 의한 방법

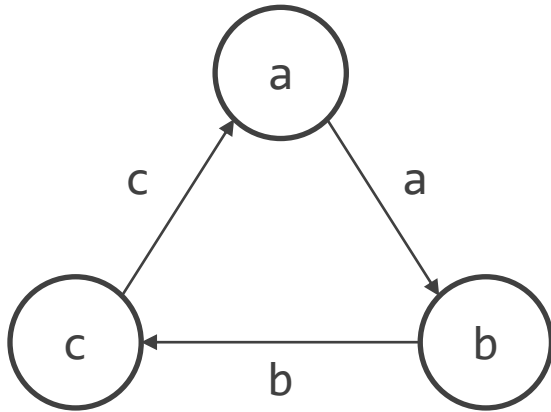
정수 입력: 5
연산 결과: 25

04. 연습 문제

❖ 연습 문제 2.

- 두 개의 값을 서로 바꾸는 Call-by-reference 형식의 Swap 함수를 공부하였습니다.
이번에는 변수 a, b, c가 지니는 값을 다음과 같이 변경시키는 Swap 함수를
Call-by-reference 형식으로 구현해 보세요.
즉, a=10, b=20, c=30이었다면 Swap 함수 호출 후에는 a=30, b=10, c=20이 되어야 합니다.

함수 호출 형태: `Swap(&a, &b, &c);`



```
Microsoft Visual Studio 디버그 콘솔
a = 10, b = 20, c = 30
a = 30, b = 10, c = 20

C:\Users\Bkwon\source\repos\Project1\x64\Debug\Project1.exe(프로세스 19960개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요....
```

04. 연습 문제

❖ 연습 문제 2. 정답 및 해설

```
1  /* example2.c */
2  #include <stdio.h>
3
4  void Swap(int* pA, int* pB, int*pC)
5  {
6      int temp = *pA;
7      *pA = *pC;
8      *pC = *pB;
9      *pB = temp;
10 }
11
12 int main(void)
13 {
14     int a = 10, b = 20, c = 30;
15     printf("a = %d, b = %d, c = %d\n", a, b, c);
16     Swap(&a, &b, &c);
17     printf("a = %d, b = %d, c = %d\n", a, b, c);
18     return 0;
19 }
```

a = 10, b = 20, c = 30

a = 30, b = 10, c = 20

04. 연습 문제

❖ 연습 문제 3.

- 아래 print 함수의 정의를 살펴 보세요. 인자로 전달되는 정보를 참조해서 int형 배열의 전체 요소를 출력하는 함수입니다. 이 함수에서 매개변수 arr 선언 시 const 키워드를 사용하는 이유는 무엇일까요? print 함수를 정의한 사람의 의도를 답해 보세요.

```
void print(const int* arr, int size)
{
    int j;
    for (j = 0; j < size; j++)
    {
        printf("%d ", arr[j]);
    }
}
```


04. 연습 문제

❖ 연습 문제 3. **정답 및 해설**

- `print` 함수의 기능은 인자로 전달된 배열의 전체 요소를 출력하는 것입니다.
- 따라서 프로그래머가 실수로라도 배열 요소의 값을 바꾸는 일이 없어야 합니다.
- 그래서 매개변수 `arr` 선언 시 `const` 키워드를 붙여준 것입니다.
- 이제 프로그래머가 실수로 배열 요소의 값을 바꾸는 코드를 작성할 경우 컴파일 에러가 발생할 것입니다.
- 그리고 프로그래머는 자신의 실수를 인식하고 적절히 코드를 고칠 기회를 얻게 됩니다.

04. 연습 문제

❖ 연습 문제 4.

- 다음 예제는 문제를 지니고 있습니다. 문제점을 찾고, 그것이 왜 문제가 되는지 답해 보세요.
특히 print 함수를 유심히 살펴보세요.

```
1  #include <stdio.h>
2  void print(const int* ptr);
3  int main(void)
4  {
5      int a = 10;
6      int* p = &a;
7      print(p);
8      return 0;
9  }
10 void print(const int* ptr);
11 {
12     int* p = ptr;
13     printf("%d \n", *ptr);
14     *p = 20;
15 }
```

04. 연습 문제

❖ 연습 문제 4. 정답 및 해설

- print 함수는 전달되는 인자를 포인터 ptr로 받고 있습니다. ptr은 const 키워드에 의해서 상수화되어 있는데, 이는 print 함수 내에서 ptr이 가리키는 변수의 조작을 허용하지 않겠다는 뜻입니다. 그런데 12번째 줄에서는 const int* 타입의 포인터를 int* 타입의 포인터 p에 대입하고 있습니다. print 함수 내에서 포인터 ptr이 가리키는 대상의 데이터 조작을 허용하지 않으려 했는데, 12번째 줄과 같은 방법을 통해서 데이터 조작을 허용하려고 할 경우 컴파일러가 오류로 인식합니다. 즉, const int* 타입의 포인터 값은 int*로의 대입이 허용되지 않습니다. 이러한 코드는 컴파일러에 의해 오류로 인식됩니다.

```
10 void print(const int* ptr);  
11 {  
12     int* p = ptr;  
13     printf("%d \n", *ptr);  
14     *p = 20;  
15 }
```

- ❖ 01. 함수의 인자로 배열 전달하기
- ❖ 02. Call-by-value vs. Call-by-reference
- ❖ 03. 포인터 대상의 const 선언
- ❖ 04. 연습 문제

THANK YOU!

Q & A

- Name: 권범
- Office: 동덕여자대학교 인문관 B821호
- Phone: 02-940-4752
- E-mail: bkwon@dongduk.ac.kr