Introduction to Software Design

C06. Parameter Passing, Const Pointer & String, Struct

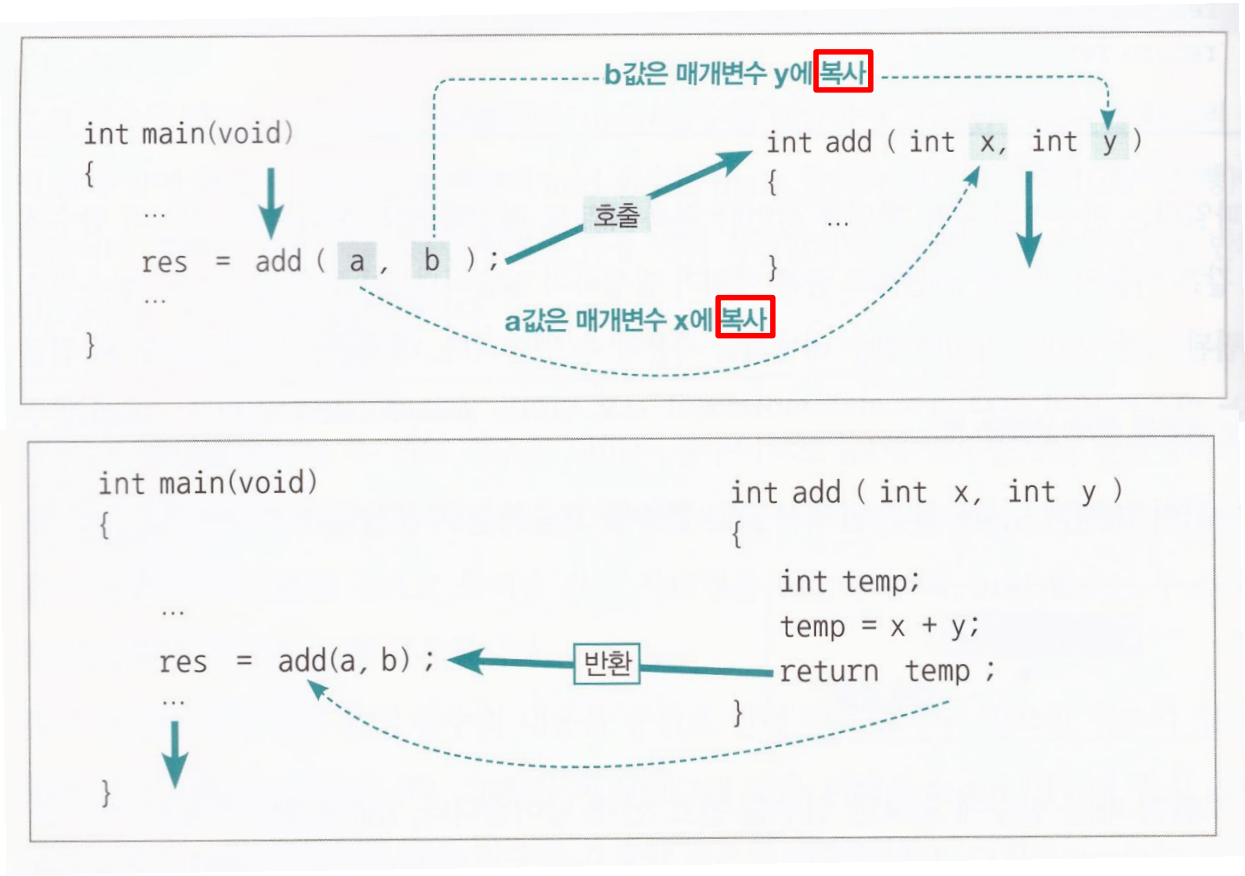
Tae Hyun Kim

Topics

- Parameter Passing: Call-by-value vs. Call-by-reference
- Pointer to Constant & Constant Pointer
- Two ways of declaring C Strings
- C Struct & Typedef
- C Struct with Array, Pointer, Functions

함수 호출 과정

```
int add(int x, int y)
{
    int temp;
    temp = x + y;
    return temp;
}

int main()
{
    int a = 2, b = 5;
    int res = add(a, b);
    printf("%d₩n", res);
    return 0;
}
```



C에서 함수 인자 전달의 기본 방식은 값의 복사이다!

함수 인자 전달 시 값의 복사 – 변수의 값 전달

```
void swap1(int n1, int n2)
{
    int temp = n1;
    n1 = n2;
    n2 = temp;
}

int main()
{
    int num1=10, num2=20;
    swap1(num1, num2);
    printf("%d %d",num1,num2);
    return 0;
}
```

num1과 n1은 별도의 메모리 공간에 저장되는 서로 다른 변수.

num1의 값 10이 n1에 **복사**되는 것

n1, n2는 지역변수이므로 함수 호출이 끝나면 사라짐.

출력결과: 10 20

함수 인자 전달 시 값의 복사 – 변수의 주소 전달

```
void swap2(int* p1, int* p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int main()
{
    int num1=10, num2=20;
    swap2(&num1, &num2);
    printf("%d %d",num1,num2);
    return 0;
}
```

num1과 p1은 별도의 메모리 공간에 저장되는 서로 다른 변수.

num1의 **주소값**이 **p1**에 복사되는 것

p1, p2는 지역변수이므로 함수 호출이 끝나면 사라짐.
하지만 p1, p2가 가리키는 변수의 값은 변경된 채로 남아있음.

→ 출력결과: 20 10

값을 전달하는 형태의 함수 호출: Call-by-value

```
void swap1(int n1, int n2)
{
    int temp = n1;
    n1 = n2;
    n2 = temp;
}

int main()
{
    int num1=10, num2=20;
    swap1(num1, num2);
    // num==10, num2==20
    return 0;
}
```

- 인자의 **값**을 복사하여 함수 호출
- 호출된 함수에서는 **호출한 함수에서 선언한 변수**의 값을 **변경 불가능.**

주소값을 전달하는 형태의 함수 호출: Call-by-reference

```
void swap2(int* p1, int* p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int main()
{
    int num1=10, num2=20;
    swap2(&num1, &num2);
    // num==20, num2==10
    return 0;
}
```

- 인자의 **주소값**을 복사하여 함수 호출
- 호출된 함수에서 **호출한 함수에서 선언한 변수**의 값을 **변경 가능.**

C Examples

```c
#include <stdio.h>

void swap1(int n1, int n2)
{
    int temp = n1;
    n1 = n2;
    n2 = temp;
}

int main()
{
    int num1=10, num2=20;
    swap1(num1, num2);
    printf("%d %d\n", num1, num2);
    return 0;
}
```

```c
#include <stdio.h>

void swap2(int* p1, int* p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int main()
{
    int num1=10, num2=20;
    swap2(&num1, &num2);
    printf("%d %d\n", num1, num2);
    return 0;
}
```

C & Python Examples

- C

```c
#include <stdio.h>

void addOne(int* p1)
{
    *p1 += 1;
}

int main()
{
    int num1=10;
    addOne(&num1);
    printf("%d\n", num1);
    return 0;
}
# call by reference
```

- Python

```python
def addOne(v):
    v[0] += 1

num1 = [10]
addOne(num1)
print(num1) # [11]
```

```python
def addOne(v):
    v += 1

num1 = 10
addOne(num1)
print(num1) # 10
```

- Python의 인자 전달 방식은 **call by object reference**라 불린다.
- Mutable object(예: list)는 전달받은 reference를 통해 직접 object의 내용을 변경한다.
- Immutable object(예: tuple, str)는 내용을 변경할 수 없으므로 새로운 object를 만들고 reference가 이것을 가리키게 한다.
  - 참조: https://www.geeksforgeeks.org/is-python-call-by-reference-or-call-by-value/

Call-by-reference의 예: scanf()

- 우리가 그 동안 사용했던 scanf()함수도 call-by-reference 방식으로 호출
- scanf(“%d”, **&**num);
- scanf() 함수 안에서 함수 밖에 선언된 변수 num의 내용을 변경하기 위해서는, call-by-reference 방식을 사용할 수 밖에 없음.
- call-by-reference로 전달하는 parameter (즉, pointer parameter)를 **out parameter**라고 부르기도 함.

포인터를 const로 선언하는 방법 1 (Pointer to Constant)

```
int num = 20;
const int* ptr = &num;
```

- **포인터를 통해 포인터가 가리키는 변수의 값**을 변경할 수 없게 한다.

```
*ptr = 30;  // 빌드 에러!
```

- 하지만 num 변수 자체를 상수로 만드는 것은 아님.

```
num = 30;   // 문제 없이 빌드 및 실행됨.
```

포인터를 const로 선언하는 방법 2 (Constant Pointer)

```
int num1 = 20;
int num2 = 30;
int* const ptr = &num1;
```

- **포인터 변수 ptr**을 상수로 만든다.
- → **ptr의 값**을 변경할 수 없다.
- → **ptr이 다른 변수를 가리키도록 변경할 수 없다.**

```
ptr = &num2;  // 빌드 에러!
```

- 하지만 포인터가 가리키는 변수의 값을 변경하는 것은 가능.

```
*ptr = 30;    // 문제 없이 빌드 및 실행됨.
```

두 방법의 비교

- `const int* ptr = &num;`
- 포인터를 상수로 만드는 것이 **아님.**
- **포인터를 통해 포인터가 가리키는 변수의 값**을 변경할 수 없게 한다.

- `int* const ptr = &num;`
- **포인터를 상수**로 만듬.
- **포인터가 다른 변수를 가리키도록 변경할 수 없게** 한다.

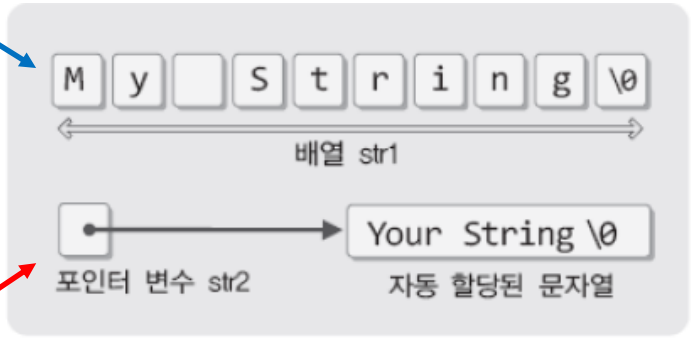
- **헷갈리기 쉽지만 꼭 알아둘 필요가 있다!**

C Example

```c
#include <stdio.h>
int main()
{
    int num1 = 20;
    int num2 = 30;

    const int* ptr1 = &num1;
    int* const ptr2 = &num1;
    const int* const ptr3 = &num1;//이렇게 선언하는 것도 가능!

    *ptr1 = 30;  // 빌드 에러!
    num1 = 30;   // 문제 없이 빌드 및 실행됨.

    ptr2 = &num2;   // 빌드 에러!
    *ptr2 = 30;     // 문제 없이 빌드 및 실행됨.

    // const int* const로 선언한 ptr3는 ptr3 자체의 값 변경, ptr3를 통
한 변수의 값 변경 모두 허용되지 않는다.
    ptr3 = &num2;   // 빌드 에러!
    *ptr3 = 40;     // 빌드 에러!

    return 0;
}
```

문자열을 선언하는 두 가지 방법

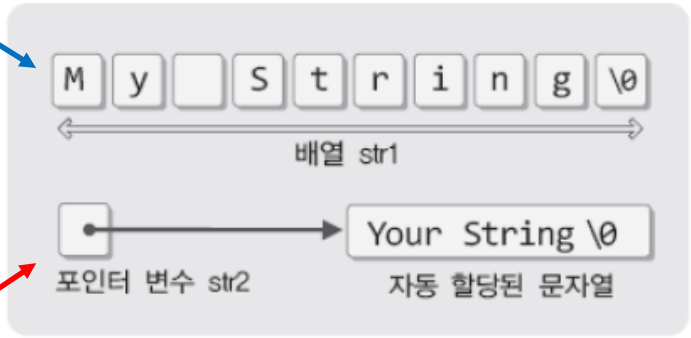
- char str1[] = “My String”;
- char형 배열로 문자열을 선언
- str1: 문자열 전체를 저장하는 **배열**



- const char* str2 = “Your String”;
- const char형 포인터로 문자열을 선언
- str2: 메모리 어딘가에 저장되어 있는 문자열(string literal)의 시작 위치를 가리키고 있는 **포인터**

문자열을 선언하는 두 가지 방법

- char str1[] = “My String”;
- “**변수** 형태의 문자열”
- 배열의 각 요소에 접근해서 **문자열 내용을 수정 가능**



- const char* str2 = “Your String”;
- “**상수** 형태의 문자열“
- 메모리에 저장되어 있는 string literal을 가리키는 포인터로 **문자열 내용 수정 불가능**

상수 형태의 문자열

- const char* str2 = “Your String”;
- -> 메모리에 저장된 리터럴 “Your String”의 시작주소가 str2에 저장된다.
  - 예) 만일 “Your String”이 10266번지에 저장이 되어있다면, 아래와 같은 식으로 동작한다.
  - const char* str2 = 10266;
- str2는 Pointer to Constant이기 때문에 나중에 다른 문자열의 시작주소로 바꾸는 것이 가능하다.
- str2 = “string2”;
  - 앞 슬라이드의 str1은 위와 같이 하는 것이 불가능 (배열의 이름에 다른 값을 대입할 수 없기 때문)

C Example

- Python의 string (str 객체)은 변경불가능(immutable)한 string literal을 가리킨다는 측면에서 C의 상수 형태 문자열 (const char*)와 유사하다고 볼 수 있다.

```c
#include <stdio.h>
int main()
{
    char str1[] = "string1";
    const char* str2 = "string2";

    // 동일한 방식으로 출력이 가능
    printf("%s %s\n", str1, str2);

    // str1 = "string11"; // 빌드 에러! - 배열의 이름에 다른 값을 대입할 수 없다!
    str2 = "string22";
    printf("%s %s\n", str1, str2);

    str1[0] = 'X';
    //str2[0] = 'X';    // 빌드 에러! - 상수형태문자열 (const char*)의 내용을 변경하려고 시도했기 때문.
    printf("%s %s\n", str1, str2);

    return 0;
}
```

문자열 리터럴을 함수로 전달할 때

- printf(“Age: %d\n”, num);
- 메모리에 저장된 리터럴 “Age: %d\n”의 시작주소가 printf의 인자로 전달된다.
  - 예) 만일 “Age: %d\n”이 10633번지에 저장이 되어있다면, 아래와 같은 식으로 동작한다.
  - printf(10633, num);
- 그래서 const char형 포인터로 받는다
- int printf (const char * format, ...);

포인터의 const 선언이 갖는 의미

- int printf (**const** char * format, ...)
- -> printf 함수 내부에서 format으로 전달받은 **문자열의 내용을 바꾸지 않을 것이다.**

- void swap (**int*** p1, **int*** p2)
- -> p1이 가리키는 변수의 값, p2가 가리키는 변수의 값이 **함수 내부에서 변경될 것이다.**

- 라는 것을 **암묵적으로** 의미.

바람직한 함수의 인자 전달 방법

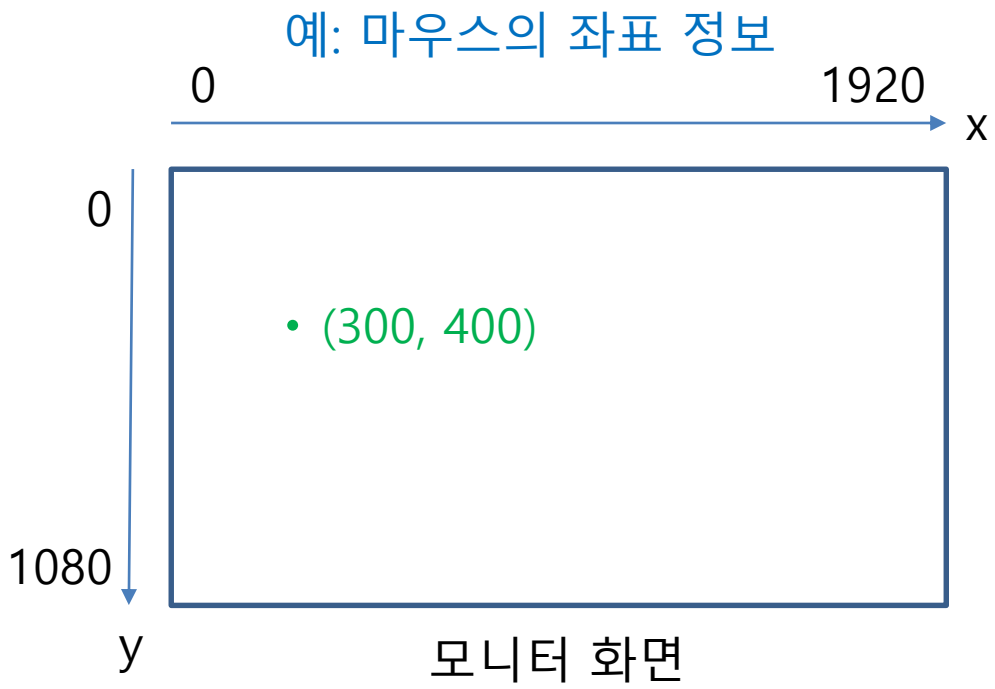
- 함수를 작성할 때, **함수 안에서 전달되는 변수의 값을 변경하지 않는 경우 (read-only인 경우),**

- 인자가 int나 double같은 기본 자료형인 경우 그냥 변수를 전달하면 됨.
- void printNumber(int n)

- 전달되는 변수가 **문자열**이거나 **크기가 큰 구조체 변수**인 경우, **const char* 나 const Point* 처럼 const 포인터를 전달**하는 것이 바람직함.
- void printString(const char* str)
- void printPoint(const Point* p)

구조체 (struct)

- 구조체 : 여러 자료형의 변수를 묶어서 **나만의 새로운 자료형**을 만들 수 있다.



구조체 이름

```
struct point
{
    int xpos;
    int ypos;
};
```

구조체 멤버

구조체 (struct) | 또 다른 예

- 이름, 전화번호, 나이 등 각 개인의 정보를 묶어서 저장하는 경우

```
struct person
{
    char name[20];      // 이름 저장
    char phoneNum[20];    // 전화번호 저장
    int age;    // 나이 저장
};
```

구조체 변수

- 앞에서 만든 point 타입의 변수를 선언하려면?

```
struct point pos1;
```

- pos1의 멤버에 접근하려면?

C & Python Examples

- C

```c
#include <stdio.h>
struct point
{
    int xpos;
    int ypos;
};
int main()
{
    struct point p1, p2;

    // p1의 위치는 코드에서 입력
    p1.xpos = 10;
    p1.ypos = 20;

    // p2의 위치는 사용자로부터 입력 받음
    scanf("%d %d", &p2.xpos, &p2.ypos);

    // 두 점의 위치 출력
    printf("p1: %d, %d\n", p1.xpos, p1.ypos);
    printf("p2: %d, %d\n", p2.xpos, p2.ypos);

    return 0;
}
```

- Python

```python
# Python에서 여러 자료형의 변수를 묶으려면
class를 사용하면 된다.
# Python의 class는 본 강좌의 범위를
넘어서므로, 아래 코드는 참고만 하기 바람.
class Point:
    def __init__(self):
        self.xpos = 0
        self.ypos = 0

p1 = Point()
p2 = Point()

p1.xpos = 10
p1.ypos = 20

p2.xpos = int(input())
p2.ypos = int(input())

print('p1: %d, %d'%(p1.xpos, p1.ypos))
print('p2: %d, %d'%(p2.xpos, p2.ypos))
```

typedef

- typedef : 이미 존재하는 자료형에 새로운 이름을 붙인다.

```
typedef unsigned int UINT;
```

// unsigned int 자료형에 UINT라는 이름을 추가로 붙여줌

```
UINT count;
```
// unsigned int count; 와 동일한 선언

typedef로 정의되는 자료형의 이름은 대문자로 시작되는 것이 관례이다 (코드에서 기본 자료형의 이름과 쉽게 구분할 수 있도록).

typedef와 구조체

```
struct point
{
    int xpos;       // 이런 구조체가 있을 때
    int ypos;
};

struct point pos1; // 그 구조체 타입의 변수 선언
```

```
typedef struct point Point;
```

// typedef로 struct point에 Point라는 새로운 이름을 붙여주면

```
Point pos1;
```

// 더 편하게 구조체 변수 선언이 가능하다

typedef와 구조체 사용하는 여러 방법들

```
struct point
{
    int xpos;
    int ypos;
};

typedef struct point Point;
```

이렇게 하는 대신...

```
typedef struct point
{
    int xpos;
    int ypos;
} Point;
```

이렇게 할 수도 있고

```
typedef struct
{
    int xpos;
    int ypos;
} Point;
```

이렇게 할 수도 있다 (구조체 이름은 생략가능)

구조체 변수의 초기화

```
typedef struct
{
    int xpos;
    int ypos;
} Point;
```

xpos=10, ypos=20의 값을 갖는 Point 타입의 변수 p1을 만들려면,

```
Point p1;
p1.xpos = 10;
p1.ypos = 20;
```

or

초기화 리스트

```
Point p1 = {10, 20};
```

비교) 배열의 초기화 방식과 동일

```
int arr1[5]={1, 2, 3, 4, 5};
```

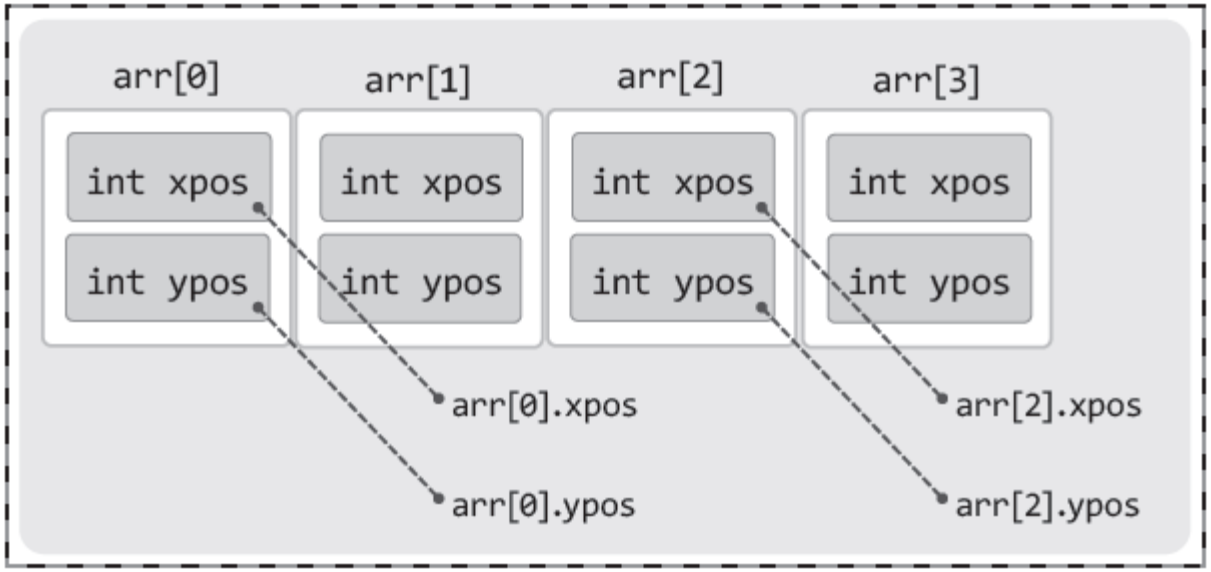
C Example

```c
#include <stdio.h>

struct point
{
    int xpos;
    int ypos;
};
typedef struct point Point;

// 아래 두 가지 방식도 해볼 것
//typedef struct point
//{
//    int xpos;
//    int ypos;
//} Point;

//typedef struct
//{
//    int xpos;
//    int ypos;
//} Point;

int main()
{
    Point pos1;
    pos1.xpos = 10;
    pos1.ypos = 20;

    // 아래의 방식으로도 해보자
    // Point pos1 = {10, 20};

    printf("pos1: %d, %d\n", pos1.xpos, pos1.ypos);

    return 0;
}
```

구조체 배열

```
typedef struct
{
    int xpos;
    int ypos;
} Point;
```

- 점(Point형 변수)을 4개 만들고 싶다면?
- → Point arr[4];



C Example

```c
#include <stdio.h>
typedef struct
{
    int xpos;
    int ypos;
} Point;

int main()
{
    int i;
    Point points[5];

    // 사용자가 좌표를 직접 입력
    for(i=0; i<5; ++i)
        scanf("%d %d", &points[i].xpos, &points[i].ypos);

    //// for 루프로 자동으로 좌표 입력
    //for(i=0; i<5; ++i)
    //{
    //    points[i].xpos = i;
    //    points[i].ypos = i*2;
    //}

    // 모든 점의 좌표를 출력
    for(i=0; i<5; ++i)
        printf("[%d] %d %d\n", i, points[i].xpos,
points[i].ypos);
    return 0;
}
```

구조체 배열의 초기화

- 구조체의 초기화

```
Point p1 = {10, 20};
```

- 배열의 초기화

```
int arr1[5]={1, 2, 3, 4, 5};
```

- 구조체 배열의 초기화

```
Point points[4] = {
    {1,1},
    {2,2},
    {3,3},
    {4,4},
};
```

구조체 포인터 변수와 -> 연산자

```
struct point pos={11, 12};
struct point * pptr=&pos;
```

구조체 point의 포인터 변수 선언

```
(*pptr).xpos=10;
```

pptr이 가리키는 구조체 변수의 멤버 xpos에 접근

```
(*pptr).ypos=20;
```

pptr이 가리키는 구조체 변수의 멤버 ypos에 접근

```
(*pptr).xpos=10;  ⟷  pptr->xpos=10;

(*pptr).ypos=20;  ⟷  pptr->ypos=20;
```

"->" 연산자는
매우 많이 쓰임!

C Example

```c
#include <stdio.h>

typedef struct
{
    int xpos;
    int ypos;
} Point;

int main()
{
    Point p1 = {10, 20};
    Point* pp1 = &p1;

    printf("%d %d\n", p1.xpos, p1.ypos);

    // 아래 세 문장은 모두 동일한 결과를 만든다
    //p1.xpos = 100;
    //(*pp1).xpos = 100;
    pp1->xpos = 100;

    printf("%d %d\n", p1.xpos, p1.ypos);

    return 0;
}
```

구조체와 함수

- 구조체 변수 역시 함수의 인자로 전달될 수 있고, 함수로부터 반환될 수도 있다.
- 예)
- void printPoint(Point p)
  - // 점 p의 좌표를 화면에 출력하는 함수

- Point getScale2xPoint(Point p)
  - // 점 p의 x, y 좌표에 2를 곱한 좌표값을 가지는 점을 리턴하는 함수

- 참고) 구조체 변수를 함수에 전달할 때
  - 대부분의 경우 const 구조체* 타입으로 전달한다고 보면 된다.
  - Point getScale2xPoint(const Point* p)
  - 단, 함수 내부에서 전달받은 인자의 값을 변경하려는 경우(out-parameter)에는 구조체* 타입으로 전달
  - 실제로는 위의 printPoint() 예제처럼 구조체 타입으로 전달하는 경우는 없다.

함수 인자 전달의 기본 방식은 값의 복사: Call-by-value

```
Point getScale2xPoint(Point p)
{
    Point p_new;
    p_new.xpos = p.xpos * 2;
    p_new.ypos = p.ypos * 2;
    return p_new;
}

int main()
{
    Point p1 = {1,2};
    Point p2 = getScale2xPoint(p1);
    printf("%d %d₩n", p2.xpos, p2.ypos);
    return 0;
}
```

p1과 p는 별도의 메모리 공간에 저장되는 서로 다른 변수.
p1의 모든 멤버의 값이 p에 복사되는 것.

p_new와 p2는 별도의 메모리 공간에 저장되는 서로 다른 변수.
p_new의 모든 멤버의 값이 p2에 복사되는 것.

주소값을 복사하는 Call-by-reference

```
void scale2x(Point* pp)
{
    pp->xpos *= 2;
    pp->ypos *= 2;
}

int main()
{
    Point p1 = {1,2};
    scale2x(&p1);
    printf("%d %d₩n", p1.xpos, p1.ypos);
    return 0;
}
```

p1의 주소값이 pp에 복사됨.
주소를 통해 접근하기 때문에 p1의 멤버 값들을 수정 가능

C Example

```c
#include <stdio.h>
typedef struct
{
    int xpos;
    int ypos;
} Point;

Point getScale2xPoint(const Point* p)
{
    Point p_new;
    p_new.xpos = p->xpos * 2;
    p_new.ypos = p->ypos * 2;
    return p_new;
}

void scale2x(Point* pp)
{
    pp->xpos *= 2;
    pp->ypos *= 2;
}
```

```c
void printPoint(Point p)
{
    printf("%d %d\n", p.xpos,p.ypos);
}

int main()
{
    Point p1 = {1,2};
    Point p2 = getScale2xPoint(&p1);
    scale2x(&p2);
    printPoint(p1);
    printPoint(p2);
    return 0;
}
```

구조체 변수를 대상으로 가능한 연산

- C의 기본 자료형 (int, char 등)은 +, -, >, < 등 다양한 연산이 가능

- 구조체 변수에 대해서는 **=(대입연산), &(address-of 연산), sizeof** 정도만 사용 가능

- **=(대입연산)**의 경우, **모든 멤버값을 복사**

C & Python Examples

- C

```c
#include <stdio.h>

typedef struct
{
    int xpos;
    int ypos;
} Point;

int main()
{
    Point p1 = {1,2};
    Point p2;
    p2 = p1;    // C 구조체 변수의 대입연산은
모든 멤버들의 값을 복사한다.
    p2.xpos = 10;   // 따라서 p2 멤버값을
변경해도 p1 멤버값은 그대로이다.

    printf("p1: %d %d\n", p1.xpos, p1.ypos);
    printf("p2: %d %d\n", p2.xpos, p2.ypos);

    return 0;
}
```

- Python

```python
class Point:
    def __init__(self, x, y):
        self.xpos = x
        self.ypos = y

p1 = Point(1, 2)

p2 = p1      # Python class
object의 대입연산은 p2가 p1과 같은
object를 가리키게 한다.
p2.xpos = 10     # 따라서 p2
멤버값을 변경하면 p1의 값도 변경된다.

print('p1: %d, %d'%(p1.xpos,
p1.ypos))
print('p2: %d, %d'%(p2.xpos,
p2.ypos))
```