



# 20190926 실습 3주차

박건호

KITRI BoB 8<sup>th</sup>

5422 RTDCS

[devgunho.github.io](https://devgunho.github.io)



# Short-Circuit Evaluation (2)

## -연산 생략

```
#include <iostream>

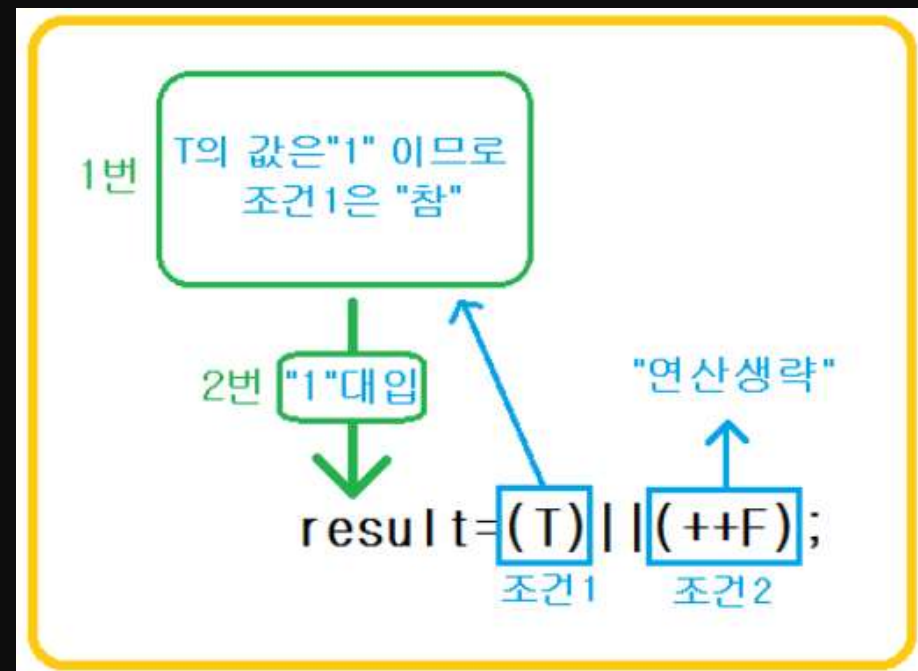
using namespace std;

int main()
{
    int T = 1;
    int F = 0;
    int result=0;

    result = (T) || (++F);

    cout << result << endl;
    cout << F << endl;

    return 0;
}
```



참고 : <https://devgunho.tistory.com/11>



# strlen() vs string::length

- **strlen (C언어에서의 <string.h>)**

strlen(문자열포인터);

strlen(문자배열);

size\_t strlen(const \*\_Str); // 문자열의 길이를 반환

- **string::length (C++에서의 string class <string>)**

string base = "hello world!";

base.length(); // base.size();

size()와 length()는 이름만 다를 뿐 같은 일을 하는 멤버 함수다.



# Function

- Call by value

함수에 인자로 값을 복사해서 전달해주는 방식.

- Call by reference

함수에 인자로 변수의 주소를 전달해주는 방식.



# Function

- Call by value

함수에 인자로 값을 복사해서 전달해주는 방식.

- Call by reference

함수에 인자로 **변수의 주소**를 전달해주는 방식.

# Pointer 실습 (C++)

메모리의 주소를 저장하기 위한 변수를 포인터라 한다.

- 선언

```
type* name;
```

```
type *name;
```

```
type * name;
```

- 위의 3가지 방법 모두 포인터 변수를 선언하는 동일한 방법이다.

- 포인터 변수에는 주소를 넣어줘야 한다.
- 변수의 주소를 포인터 변수에 대입시켜 주면 된다.
- & 연산자를 사용하여 변수의 주소를 대입시키면 된다.

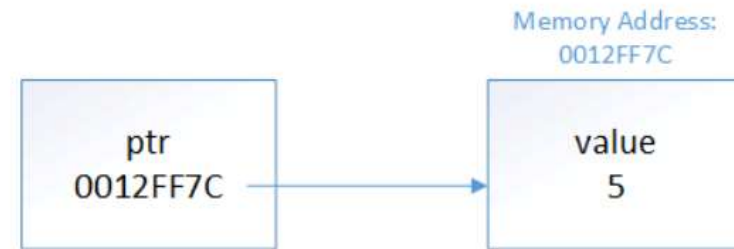
```
int number = 123;
```

```
int* pointerNumber;
```

```
pointerNumber = &number;
```

- &number 에서 &는 변수 number의 메모리 주소를 가리키는 연산자이다.

개념적으로, 위 코드를 다음과 같이 나타낼 수 있다.



ptr 은 값으로 value 변수 값의 주소를 가지고 있다. 그러므로 ptr 을 value 변수를 '가리키는' 값이라고 할 수 있다.

코드를 사용해서 쉽게 확인할 수 있다.

```
#include <iostream>

int main()
{
    int value = 5;
    int *ptr = &value; // 변수 값의 주소로 ptr 초기화

    std::cout << &value << '\n'; // value 변수의 주소 출력
    std::cout << ptr << '\n'; // ptr 변수 값 출력

    return 0;
}
```





# Pointer 사용 목적

- **배열은 포인터를 사용하여 구현된다.** 포인터는 배열을 반복할 때 사용할 수 있다.  
(배열 인덱스 대신 사용 가능)
- C/C++에서 **동적으로 메모리를 할당**할 수 있는 유일한 방법이다.
  - (가장 흔한 사용 사례)
- 데이터를 복사하지 않고도 많은 양의 데이터를 함수에 전달할 수 있다.
  - = **큰 오버헤드 없이 데이터 구조를 포인터로 전달**
  - (네트워크 프로그래밍)
- 함수를 매개 변수로 다른 함수에 **전달**하는 데 사용할 수 있다. (**데이터 보호**)
- 상속을 다룰 때 다형성을 달성하기 위해 사용한다.
- 하나의 구조체/클래스 포인터를 다른 구조체/클래스에 두어 **체인을 형성**하는 데 사용할 수 있다. 이는 연결리스트 및 트리와 같은 고급 자료구조에서 유용하다.



# 메모리? 주소? 값? (직접 눈으로 확인하기)

- [https://github.com/devgunho/HUFS\\_19-2\\_ComputerProgramming/tree/master/CP\\_20190926](https://github.com/devgunho/HUFS_19-2_ComputerProgramming/tree/master/CP_20190926) 실습3주차
- Debug > Step Over
  - F10
- Debug > Windows > Memory
  - Ctrl+Alt+'M' / 1 2 3 4





# 메모리? 주소? 값? (직접 눈으로 확인하기)

## 간접 참조 연산자와 증감 연산자

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i = 10;
```

```
    int *pi = &i;
```

```
    printf("i = %d, pi = %p\n", i, pi);
```

```
    (*pi)++;
```

```
    printf("i = %d, pi = %p\n", i, pi);
```

```
    printf("i = %d, pi = %p\n", i, pi);
```

```
    *pi++;
```

```
    printf("i = %d, pi = %p\n", i, pi);
```

```
    return 0;
```

```
}
```

pi가 가리키는 위치의 값을 증가한다.

pi가 가리키는 위치에서 값을 가져온 후에 pi를 증가한다.

```
i = 10, pi = 0012FF60
```

```
i = 11, pi = 0012FF60
```

```
i = 11, pi = 0012FF60
```

```
i = 11, pi = 0012FF64
```



# 메모리? 주소? 값? (직접 눈으로 확인하기)

```
int a[5] = { 10, 20, 30, 40, 50 };
printf(" size of a = %d, size of int = %d \n\n", sizeof(a), sizeof(int));

int n = sizeof(a) / sizeof(int);
int i;
int* p = a;

printf(" a = %d, p = %d \n\n", a, p);
// 배열은 결국 포인터로 구현된다는 것을 알 수 있다.

for (i = 0; i < n; i++)
{
    //printf(" a[%d] ==> %d 주소 ==> %d \n", i, a[i], &a[i]);
    //printf(" p[%d] ==> %d 주소 ==> %d \n", i, p[i], &p[i]);
    //printf(" *(a+%d) ==> %d 주소 ==> %d \n", i, *(a+i), a+i);
    printf(" *(p+%d) ==> %d 주소 ==> %X \n", i, *(p + i), p + i);
}
```

배열식		포인터식		배열식		포인터식
a[i]	==	*(a+i)	==	p[i]	==	*(p+i)
&a[i]	==	a+i	==	&p[i]	==	p+i

이렇게 배열과 포인터는 완벽하게 호환됩니다.



# 메모리? 주소? 값? (직접 눈으로 확인하기)

- 엔디언(Endianness)은 컴퓨터의 메모리와 같은 1차원의 공간에 여러 개의 연속된 대상을 배열하는 방법을 뜻하며, 바이트를 배열하는 방법을 특히 바이트 순서(Byte order)라 한다.
- 엔디언은 보통 큰 단위가 앞에 나오는 빅 엔디언(Big-endian)과 작은 단위가 앞에 나오는 리틀 엔디언(Little-endian)으로 나눌 수 있으며, 두 경우에 속하지 않거나 둘을 모두 지원하는 것을 미들 엔디언(Middle-endian)이라 부르기도 한다.



# 메모리? 주소? 값? (직접 눈으로 확인하기)

- Big-endian : 최상위 바이트(MSB, Most Significant Byte)부터 차례로 저장하는 방식
- Little-endian : 최하위 바이트(LSB, Least Significant Byte)부터 차례로 저장하는 방식

메모리에 0x12345678을 대입						
Big-Endian						
				→ 메모리 주소 증가		
메모리 주소	....	0x100	0x101	0x102	0x103	...
변수 값		0x12	0x34	0x56	0x78	
Little-Endian						
				→ 메모리 주소 증가		
메모리 주소	....	0x100	0x101	0x102	0x103	...
변수 값		0x78	0x56	0x34	0x12	

배열식	포인터식	배열식	포인터식
-----	------	-----	------

<code>a[i]</code>	<code>==</code>	<code>*(a+i)</code>	<code>==</code>	<code>p[i]</code>	<code>==</code>	<code>*(p+i)</code>
-------------------	-----------------	---------------------	-----------------	-------------------	-----------------	---------------------

<code>&amp;a[i]</code>	<code>==</code>	<code>a+i</code>	<code>==</code>	<code>&amp;p[i]</code>	<code>==</code>	<code>p+i</code>
------------------------	-----------------	------------------	-----------------	------------------------	-----------------	------------------

이렇게 배열과 포인터는 완벽하게 호환됩니다.

- 배열과 포인터의 관계
- 코드를 다시한번 생각해보기...

scanf\_string\_array.c

```
#define _CRT_SECURE_NO_WARNINGS // scanf 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>

int main()
{
    char s1[10]; // 크기가 10인 char형 배열을 선언

    printf("문자열을 입력하세요: ");
    scanf("%s", s1); // 표준 입력을 받아서 배열 형태의 문자열에 저장

    printf("%s\n", s1); // 문자열의 내용을 출력

    return 0;
}
```





# Pointer 동적할당 (malloc) (2)

```
#include <stdio.h>
#include <stdlib.h>    // malloc, free 함수가 선언된 헤더 파일

int main()
{
    int num1 = 20;      // int형 변수 선언
    int* numPtr1;       // int형 포인터 선언

    numPtr1 = &num1;    // num1의 메모리 주소를 구하여 numPtr에 할당

    int* numPtr2;       // int형 포인터 선언

    numPtr2 = (int*)malloc(sizeof(int));    // int의 크기 4바이트만큼 동적 메모리 할당

    printf("%p\n", numPtr1);    // 006BFA60: 변수 num1의 메모리 주소 출력
                                // 컴퓨터마다, 실행할 때마다 달라짐

    printf("%p\n", numPtr2);    // 009659F0: 새로 할당된 메모리의 주소 출력
                                // 컴퓨터마다, 실행할 때마다 달라짐

    free(numPtr2);        // 동적으로 할당한 메모리 해제

    return 0;
}
```



# Pointer 동적할당 (malloc) (2)

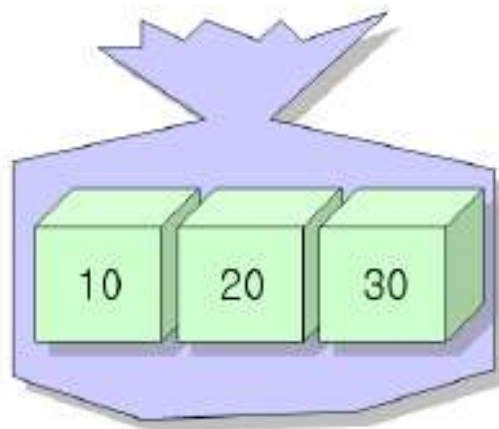
- 동적할당이라는 것은 프로그램 실행 중에 동적으로 메모리를 할당하는 것을 말한다.
- 동적으로 메모리를 할당할 때 Heap(힙)영역에 할당한다.
- 함수의 원형은 `void* malloc(size_t size)`이다.
- 해당 함수를 사용하기 위해서는 `<stdlib.h>` 헤더파일을 include 해야 한다.
- 함수 동작은 매개변수에 해당하는 `size_t`만큼의 크기를 메모리에 할당하고,
  - 성공하면 : 할당한 메모리의 첫번째 주소를 리턴해준다.
  - 실패하면 : NULL을 리턴한다.
- 가장 중요한 점은 **할당한 메모리는 반드시 해제** 해주어야 한다.
  - 그렇지 않으면 메모리 릭, 메모리 누수가 발생한다.



# Struct

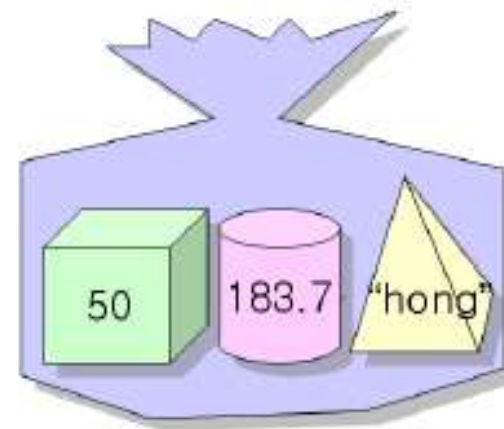


## ■ 구조체 vs 배열



배열

같은 타입의 집합



구조체

다른 타입의 집합



# Struct



```
// x값과 y값으로 이루어지는 화면의 좌표
struct point {
    int x;           // x 좌표
    int y;           // y 좌표
};
```

```
// 복소수
struct complex {
    double real;     // 실수부
    double imag;     // 허수부
};
```

```
// 날짜
struct date {
    int month;
    int day;
    int year;
};
```

```
// 사각형
struct rect {
    int x;
    int y;
    int width;
    int grade;
};
```

```
// 직원
struct employee {
    char name[20];   // 이름
    int age;         // 나이
    int gender;      // 성별
    int salary;      // 월급
};
```

# Struct



- 구조체 정의와 구조체 변수 선언은 다르다.

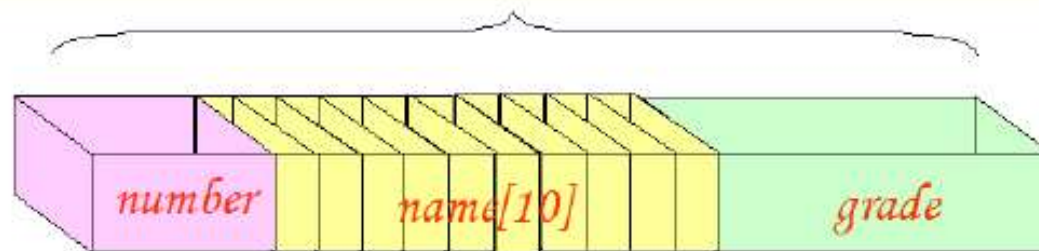
```
struct student {  
    int number;  
    char name[10];  
    double grade;  
};
```

구조체 정의

```
int main(void){  
    struct student s1;  
    ...  
}
```

구조체 변수 선언

*s1*



# Struct 실습 (2)

Microsoft Visual Studio Debug Console

홍길동

Enter name: 가나다


Enter age: 11

Displaying information

Name: 가나다

Age: 11

D:\Workspace\19\_ComputerProgramming\C



```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

struct student // 구조체 선언
{
    char name[50];
    int age;
};

// 함수 프로토타입 선언
void display(struct student s);

int main()
{
    struct student s1;
    strcpy(s1.name, "홍길동"); // c의 string.h
    printf("%s\n\n", s1.name);

    printf("Enter name: ");
    scanf("%s", s1.name);
    printf("Enter age: ");
    scanf("%d", &s1.age); // 구조체 멤버의 주소 전달

    display(s1); // 보여주는 함수 호출

    return 0;
}

void display(struct student s)
{
    // 
```



# 두 점 사이 거리 구하기 (10)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <math.h>

struct point {
    int xpos;
    int ypos;
};

int main() {
    struct point pos1, pos2;
    double distance;

    //printf("point1 pos: ");
    scanf("%d %d", &pos1.xpos, &pos1.ypos);

    //printf("point2 pos: ");
    scanf("%d %d", &pos2.xpos, &pos2.ypos);

    distance = ?

    //printf("두 점 사이의 거리 : %0.2f\n", distance);
    printf("%0.2f\n", distance);

    return 0;
}
```

Microsoft Visual Studio Debug Console

point1 pos: 10 10  
point2 pos: 20 20  
두 점 사이의 거리 : 14.14

D:\\_Workspace\19\_ComputerProgramming\C  
Press any key to close this window . .



# Struct 비교 연산

- 같은 구조체 변수끼리 대입은 가능하지만 비교는 불가능하다.

```
struct point {  
    int x;  
    int y;  
};  
  
int main(void)  
{  
    struct point p1 = {10, 20};  
    struct point p2 = {30, 40};  
  
    p2 = p1; // 대입 가능  
  
    if( p1 == p2 ) // 비교 -> 컴파일 오류!!  
        printf("p1와 p2이 같습니다.");  
  
    if( (p1.x == p2.x) && (p1.y == p2.y) ) // 올바른 비교  
        printf("p1와 p2이 같습니다.");  
}
```

# References as shortcuts

## References as shortcuts

참조형의 또 다른 장점은 중첩된 데이터에 쉽게 접근할 수 있게 한다는 것이다.

```
struct Something
{
    int value1;
    float value2;
};

struct Other
{
    Something something;
    int otherValue;
};

Other other;
```

`other.something.value1`에 접근할 경우 타이핑 양이 많아지고 여러 개면 코드가 엉망이 될 수 있다.

참조를 통해 좀 더 쉽게 접근할 수 있다.

```
int& ref = other.something.value1;
// ref can now be used in place of other.something.value1
```

따라서 다음 두 명령문은 같다.

```
other.something.value1 = 5;
ref = 5;
```

이렇게 하면 코드가 더 명확하고 읽기 쉽게 유지할 수 있다.



# Struct Pointer



## ■ 구조체를 가리키는 포인터

```
struct student *p;
```

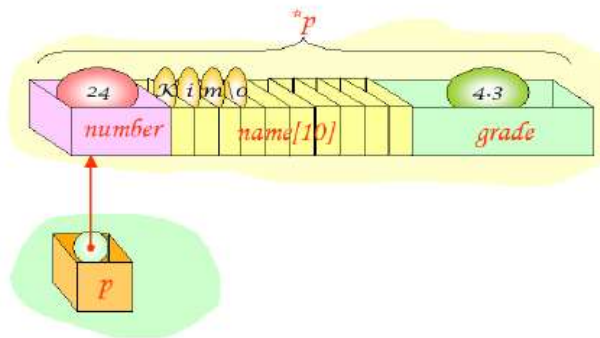
```
struct student s = { 20070001, "홍길동", 4.3 };
```

```
struct student *p;
```

```
p = &s;
```

```
printf("학번=%d 이름=%s 학점=%f \n", s.number, s.name, s.grade);
```

```
printf("학번=%d 이름=%s 학점=%f \n", (*p).number, (*p).name, (*p).grade);
```







# Struct Pointer

여기서, today는 구조체 변수이고,

date\_ptr은 구조체 포인터 변수로 선언되어 있다.

즉, date\_ptr은 Date형의 구조체 변수인 today를 가리키는 포인터 변수로 선언된 것이다.

따라서, date\_ptr은 Date와 같은 구조체를 가진다.

만약  $*(date\_ptr + 1)$  하면  $*date\_ptr$ 은 첫번째 구조체가 되며  $*(date\_ptr + 1)$ 은 두 번째 구조체가 된다.

구조체 포인터 변수를 사용할 때의 구조체 멤버를 참조하는 방법에는 직접 참조와 간접 참조 두 가지 방법이 있다.

즉, 구조체 멤버를 참조하기 위해 도트 연산자를 이용하여 **멤버를 직접 참조하는 방법**과 포인터 변수를 이용하여 **간접적으로 구조체 멤버를 참조하는 방법**이 있다.

구조체의 포인터 연산자 ( $\rightarrow$  : structure pointer operation)를 사용하여 멤버를 참조하는 경우의 일반 형식은 다음과 같다. (다음장의 예제확인)

```
struct Date
```

```
{
```

```
    int year;
```

```
    int month;
```

```
    int day;
```

```
};
```

```
struct Date today; // 구조체 변수 today 선언
```

```
struct Date *date_ptr; // 구조체 포인터 변수 date_ptr 선언
```

```
date_ptr = &today;
```





# Struct Pointer 실습1 (2)

```
ne_code (Global Scope)

#include <stdio.h>

struct student {
    int number;
    char name[20];
    double grade;
};

int main()
{
    struct student s = { 201901123, "홍길동", 4.5 };
    struct student* p;

    p = &s;

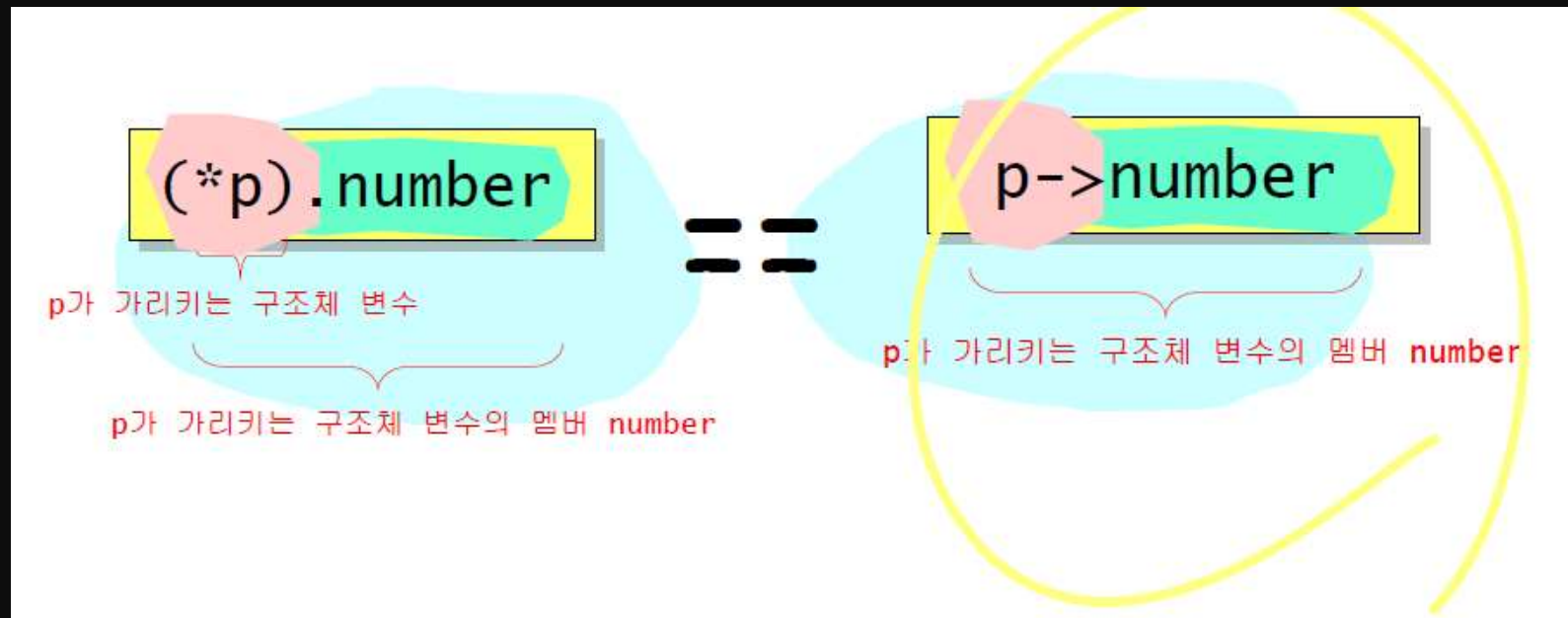
    printf("학번=%d 이름=%s 성적=%.2f\n", s.number, s.name, s.grade);
    printf("학번=%d 이름=%s 성적=%.2f\n", (*p).number, (*p).name, (*p).grade);
    printf("학번=%d 이름=%s 성적=%.2f\n", p->number, p->name, p->grade);

    return 0;
}
```

MICROSOFT VISUAL STUDIO DEBUG CONSOLE

```
학번=201901123 이름=홍길동 성적=4.50
학번=201901123 이름=홍길동 성적=4.50
학번=201901123 이름=홍길동 성적=4.50
```

# Struct Pointer





# Struct Pointer 실습2 (2)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

struct dog
{
    char name[10];
    char breed[10];
    int age;
    char color[10];
};
```

```
int main()
{
    struct dog my_dog = { "hi", "Bulldog", 5, "white" };
    struct dog* ptr_dog;

    printf("%d\n", sizeof(struct dog*));    // 뭐든 포인터 변수의 크기는 4byte

    ptr_dog = &my_dog;

    printf("Dog's name: %s\n", ptr_dog->name);
    printf("Dog's breed: %s\n", ptr_dog->breed);
    printf("Dog's age: %d\n", ptr_dog->age);
    printf("Dog's color: %s\n", ptr_dog->color);

    // 개의 이름을 jack으로 바꾸려면?
    strcpy(ptr_dog->name, "jack");
    // strcpy() 함수를 사용해서 새로운 이름으로 바꿔야 한다.
    // because we can't assign a string value directly to ptr_dog->name using assignment operator!

    // increasing age of dog by 1 year
    ptr_dog->age++;

    printf("Dog's new name is: ");
    printf("Dog's age is: %d\n");

    return 0;
}
```

Microsoft Visual Studio Debu

```
4
Dog's name: hi
Dog's breed: Bulldog
Dog's age: 5
Dog's color: white
Dog's new name is: hi
Dog's age is: 6
```



# Struct Pointer 실습3 (2)

```
#include<stdio.h>

struct student
{
    char* name;
    int age;
    char* program;
    char* subjects[5];
};

int main()
{
    struct student stu = {
        "Lucy",
        25,
        "CS",
        {"CS-01", "CS-02", "CS-03", "CS-04", "CS-05"}
    };

    struct student* ptr_stu = &stu;
    int i;
```

```
printf("Accessing members using structure variable: \n\n");

printf("Name: %s\n", stu.name);
printf("Age: %d\n", stu.age);
printf("Program enrolled: %s\n", stu.program);

for (i = 0; i < 5; i++)
{
    printf("Subject : %s \n", stu.subjects[i]);
}

printf("\n\nAccessing members using pointer variable: \n\n");

printf("Name: %s\n", ptr_stu->name);
printf("Age: %d\n", ptr_stu->age);
printf("Program enrolled: %s\n", ptr_stu->program);

for (i = 0; i < 5; i++)
{
    printf("Subject : %s \n", ptr_stu->subjects[i]);
}

return 0;
}
```

**student  
구조체  
확인!**

## Microsoft Visual Studio Debug Console

Accessing members using structure variable:

Name: Lucy  
Age: 25  
Program enrolled: CS  
Subject : CS-01  
Subject : CS-02  
Subject : CS-03  
Subject : CS-04  
Subject : CS-05

Accessing members using pointer variable:

Name: Lucy  
Age: 25  
Program enrolled: CS  
Subject : CS-01  
Subject : CS-02  
Subject : CS-03  
Subject : CS-04  
Subject : CS-05

D:\Workspace\19\_ComputerProgramming\CP\_Lecture  
Press any key to close this window . . .



# Struct Pointer 실습4 (5)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

struct person {
    int age;
    float weight;
    char name[30];
};


int main()
{
    struct person* ptr;
    int i, n;
    printf("Enter the number of persons: ");
    scanf("%d", &n);
    // allocating memory for n numbers of struct person
    ptr = (struct person*) malloc(n * sizeof(struct person));
```

```
    for (i = 0; i < n; ++i)
    {
        printf("Enter first name and age respectively: ");
        // To access members of 1st struct person,
        // ptr->name and ptr->age is used
        // To access members of 2nd struct person,
        // (ptr+1)->name and (ptr+1)->age is used
        scanf("%s %d", (ptr + i)->name, &(ptr + i)->age);
    }
    printf("Displaying Information:\n");
    for (i = 0; i < n; ++i)
        printf("Name: %s\tAge: %d\n", (ptr + i)->name, (ptr + i)->age);
    return 0;
}
```

```
Enter the number of persons: 3
Enter first name and age respectively: aaa 22
Enter first name and age respectively: aaf 23
Enter first name and age respectively: asdf 23
Displaying Information:
Name: aaa      Age: 22
Name: aaf      Age: 23
Name: asdf     Age: 23
```



# Swap (10)

```
void swap(int* a, int* b) {  
    int temp;  
  
      
}
```

```
int main() {  
    int a, b;  
    int plus, min, multi;  
    float divi;  
  
    //printf("input the two numbers : ");  
    scanf("%d %d", &a, &b);  
  
    swap(&a, &b);  
  
    plus = a + b;  
    min = a - b;  
    multi = a * b;  
    divi = (double)a / (double)b;  
  
    printf("swap : %d %d\n", a, b);  
  
    printf("%d\n", plus);  
    printf("%d\n", min);  
    printf("%d\n", multi);  
    printf("%0.2f\n", divi);  
  
    return 0;  
}
```

```
input the two numbers : 3 5  
swap : 25 9  
34  
16  
225  
2.78
```



# Point Swap (10)



```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

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

void SwapPoint(Point* ptr1,         ) {
            
}
```

```
int main() {
    Point pos1, pos2;

    scanf("%d %d", &pos1.xpos, &pos1.ypos);
    scanf("%d %d", &pos2.xpos, &pos2.ypos);

    SwapPoint(        );

    printf("[%d, %d]\n", pos1.xpos, pos1.ypos);
    printf("[%d, %d]\n", pos2.xpos, pos2.ypos);

    return 0;
}
```

```
1 2
3 5
[3, 5]
[1, 2]
```