C언어 기초 part. 2

Week 2 – 배열, 포인터

QnA 메일 : edujongkook@gmail.com

Pdf 파일 : github.com/edujongkook
/pdf_sbs_c_weekend

목차 A table of contents

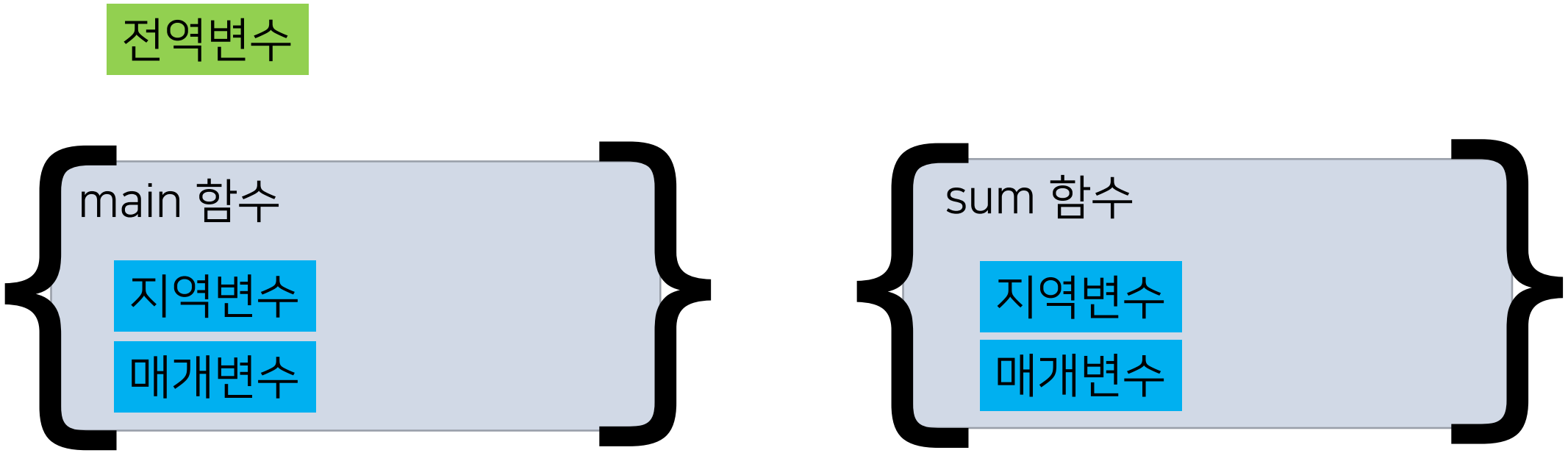
1.

소스파일 영역

복습 변수의 영역

지역변수 와 전역변수

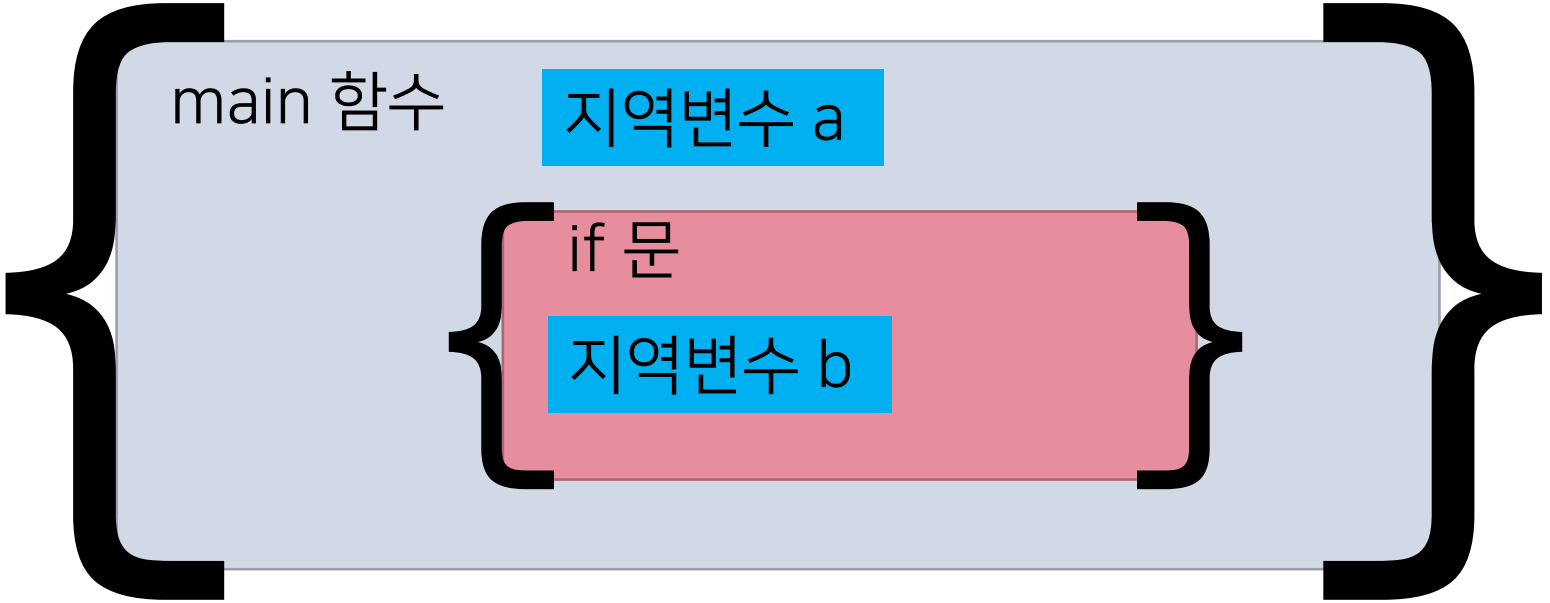
지역변수는 블록안에서만 전역변수는 모든 블록에서 사용가능



복습 변수의 영역

main함수의 변수 a는 if문에서도 사용가능

if문 안에서 만든 변수 b는 if문 밖에서는 사용 못함



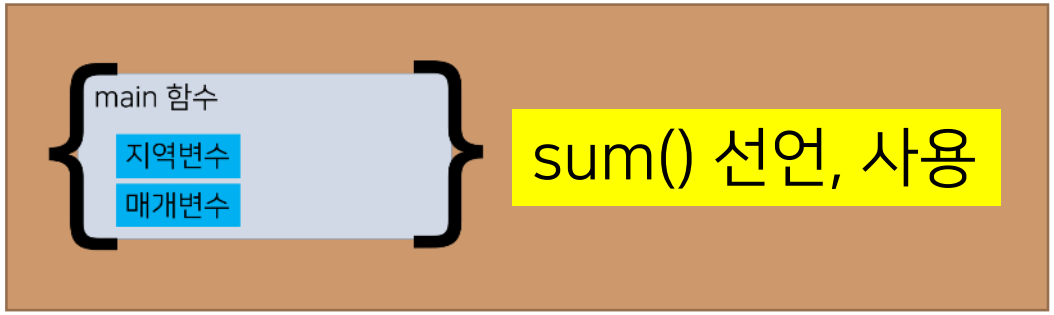
1 함수의 영역

함수의 경우 기본적으로 모든 소스파일에서 사용가능 (외부 함수)

main.c

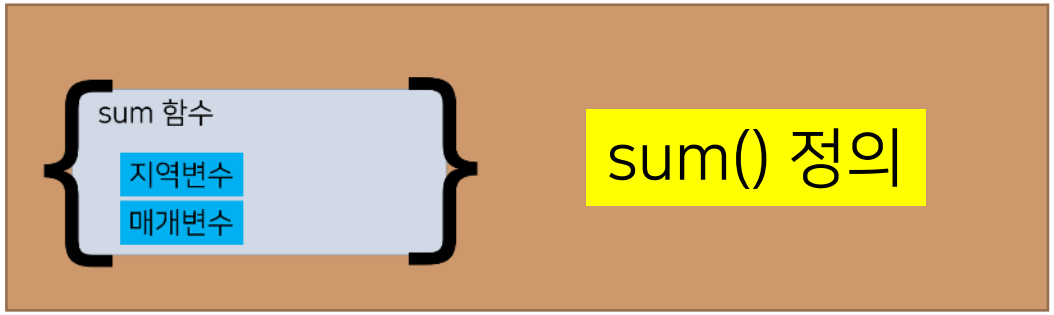
```c
int sum(int a, int b);

int main(void) {
    printf("%d", sum(4, 10));
}
```



sum.c

```c
int sum(int a, int b) {
    return a + b;
}
```



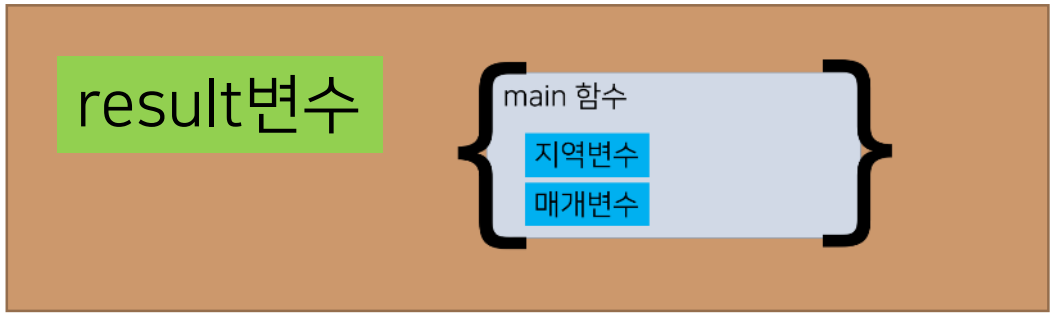
1 변수의 영역

전역변수라 해도 기본적으로 소스파일 내부에서만 사용가능 (내부 변수)

main.c

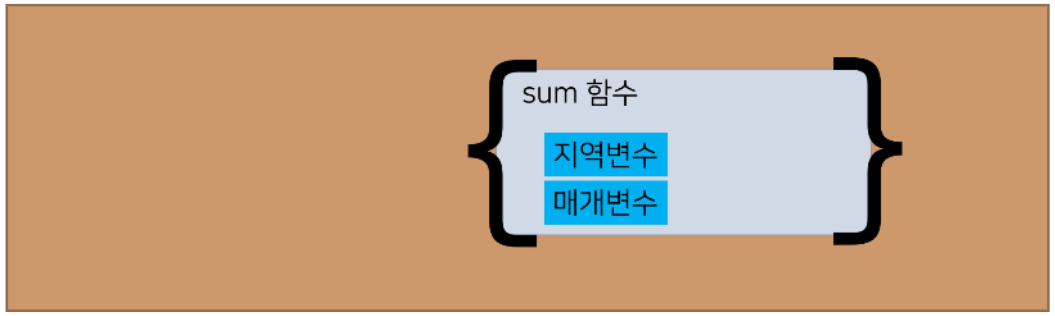
```c
int sum(int a, int b);
int result;

int main(void) {
    printf("%d", result);
}
```



sum.c

```c
int sum(int a, int b) {
    result = a + b;
}
```



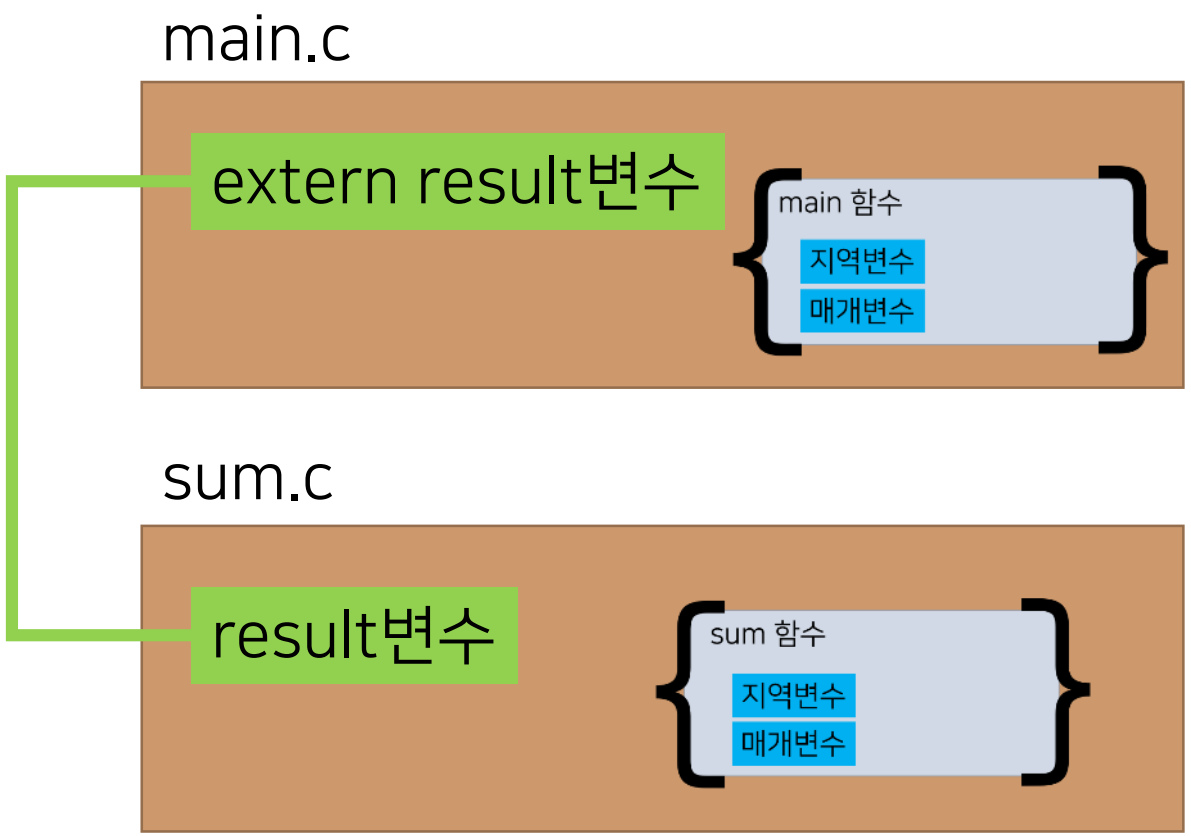
1 extern 외부 변수

전체로 공유하기 위해서는 extern 키워드를 사용 (외부변수로 선언)

main.c

```c
#include <stdio.h>
int sum(int a, int b);
extern int result;

int main(void) {
    result = sum(3, 5);
    printf("%d", result);
}
```



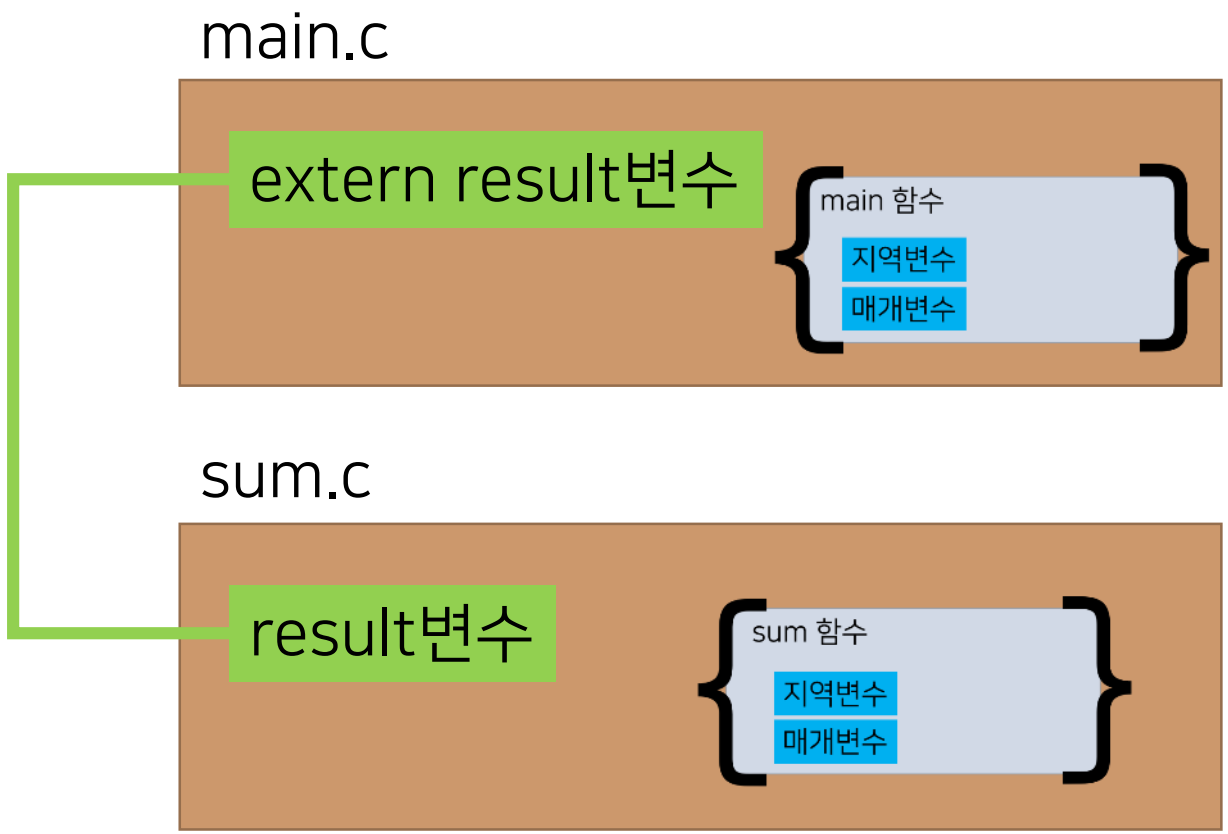
1 extern 외부 변수

외부변수로 선언된 변수를 다른 파일에서 전역변수로 선언하고 값을 할당

sum.c

```c
int result;

int sum(int a, int b) {
    result = a + b;
}
```

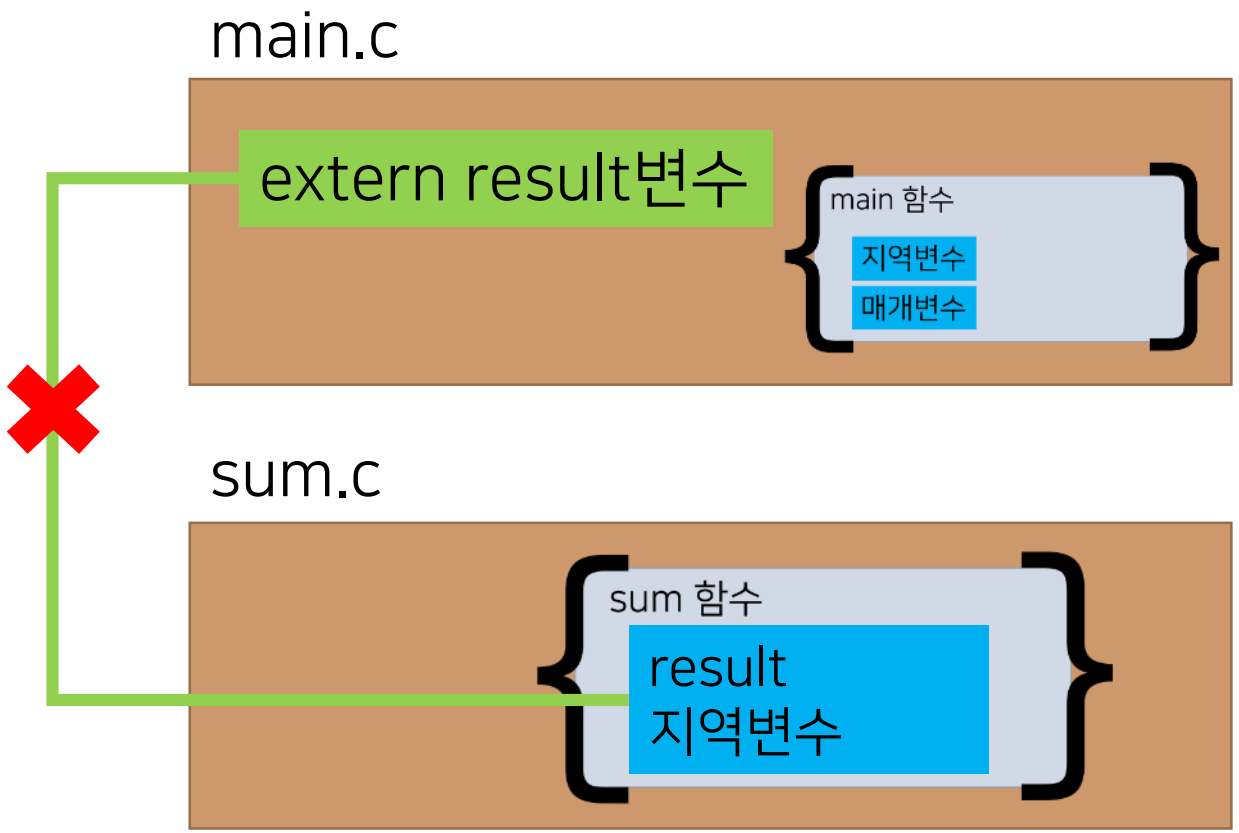


1 extern 외부 변수

외부변수는 지역변수와는 공유되지 않음

sum.c

```
int sum(int a, int b) {
    int result;
    result = a + b;
}
```



1 static 내부 함수

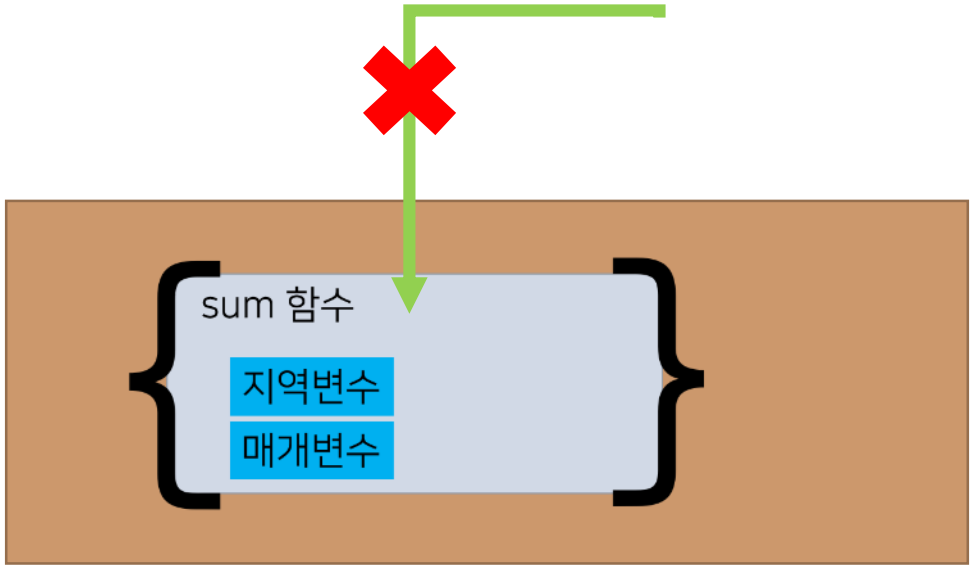
함수는 기본적으로 외부 함수 -> 내부 함수로 만들어주는 static 키워드

내부 함수가 되면 다른 소스파일에서는 사용불가

sum.c

```
int result;

static int sum(int a, int b) {
    result = a + b;
}
```



1 static 지역변수

지역변수는 기본적으로 블록이 끝나면 소멸

main.c

```c
int main(void) {
    sum(3, 5);
    sum(4, 2);
}

int sum(int a, int b) {
    int count = 0;
    count += 1;
    printf("count: %d₩n", count);
    return a + b;
}
```

1 static 지역변수

static 지역변수를 만들면 소멸하지 않고 프로그램 종료시까지 유지

main.c

```c
int main(void) {
    sum(3, 5);
    sum(4, 2);
    sum(2, 1);
}

int sum(int a, int b) {
    static int count = 0;
    count += 1;
    printf("count: %d₩n", count);
    return a + b;
}
```

초기화가 최초에 한번만 실행

2.

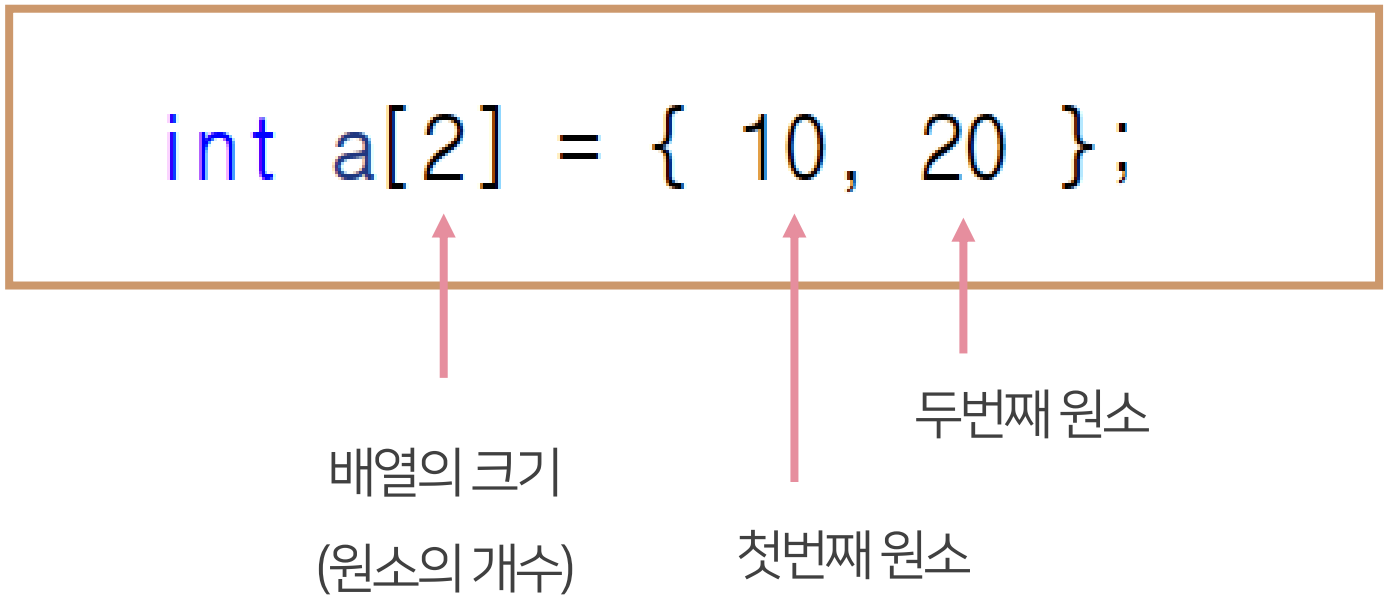
배열과 문자열


IVE THE LIFE YOU'VE
LWAYS DREAMED OF.
BE FEARLESS IN THE
FACE OF ADVERSITY.
EVER STOP LEARNING.
SE YOUR IMAGINATION
WHENEVER POSSIBLE.
ECOGNIZE THE BE AUTY
THAT SURROUN
REMEMBER WHI
AME FROM, BU
LOSE SIGHT OF
YOU ARE GO

복습 배열의 구조

순서가 있는 값들을 배열(array)이라고 합니다.

int 타입의 2개의 원소를 가지고있는 배열 a를 만드는 법은 아래와 같습니다.



복습 배열의 인덱스

배열의 원소를 사용하기 위해서 인덱스를 사용합니다.

인덱스는 0부터 시작하여 1씩 증가합니다.

```
int main(void) {
    int a[2] = { 10, 20 };
    printf("%d₩n", a[0]); //첫번째 원소 index는 0
    a[0] = 100;
}
```

인덱스(index)

복습 배열의 크기

배열은 변수와 마찬가지로 자신의 크기를 가지고 있습니다.

```
int main(void) {
    int a[2] = { 10, 20 };
    printf("%d₩n", a[0]); //첫번째 원소 index는 0
    a[0] = 100;
    printf("%d₩n", sizeof(a));
}
```

2 문자열 구조

문자열은 문자형 값들로 이루어진 배열입니다.

```
int main(void) {
    char c = 'a';
    char str[4] = { 'a','b','c','₩0' };
    printf("%s₩n", str);
}
```

2 문자열 선언과 초기화

다양한 방식으로 선언하고 초기화 할 수 있습니다.

모두 같은 문자열입니다.

```
int main(void) {
    char str[4] = { 'a','b','c','₩0' };
    char str2[] = { 'a','b','c','₩0' };
    char str3[4] = "abc";
    char str4[] = "abc";
    printf("%s₩n", str4);
}
```

2 문자열 선언과 초기화

str4 의 방식으로 값을 넣을 때도 실제로는 마지막에 널문자가 자동포함.

```
int main(void) {
    char str[4] = { 'a','b','c','₩0' };
    char str2[] = { 'a','b','c','₩0' };
    char str3[4] = "abc";
    char str4[] = "abc";
    printf("%d₩n", sizeof(str4));
}
```

2 문자열 선언과 초기화

선언만 하는 경우는 반드시 크기를 지정해 주어야합니다.

```
int main(void) {
    char str[4] = { 'a','b','c','₩0' };
    char str2[] = { 'a','b','c','₩0' };
    char str3[4] = "abc";
    char str4[] = "abc";
    char str5[100]; // 선언만
    printf("%d₩n", sizeof(str5));
}
```

2 문자열 접근

마찬가지로 인덱스를 이용하여 각각의 문자에 접근가능

printf의 경우 %s 로 전체 문자열 출력

```
int main(void) {
    char str[] = "abc";
    printf("%c %c %c₩n", str[0], str[1], str[2]);
    printf("%s", str);
}
```

2 Null 문자

문자열 마지막을 의미하는 널문자 는 숫자로는 0

문자 '0' 과는 다릅니다.

```
int main(void) {
    char str[] = "abc";
    printf("%c₩n", str[3]);
    printf("%d₩n", str[3]); // NULL문자
    printf("%d₩n", '0'); // 문자 0
}
```

2 배열의 사용

어떤 성적 데이터를 배열로 만드는 예시입니다.

국어	영어	수학
80	95	100

```
int main(void) {
    int score[3] = { 80, 95, 100 };
}
```

2 배열의 사용

각각의 점수를 순서대로 index로 접근하여 사용

국어	영어	수학
80 - score[0]	95 - score[1]	100 - score[2]

```
int main(void) {
    int score[3] = { 80, 95, 100 };
    int size = sizeof(score);
    printf("%d", size);
}
```

2 다차원 배열

아래와 같은 성적 데이터를 배열로 표현하려고 합니다.

번호	국어	영어	수학
1	80	95	100
2	60	82	75
...	...	...	...
20	99	100	95

2 다차원 배열

2번까지 배열로 표현하면 다음처럼 2차원 배열로 만들 수 있습니다.

번호	국어	영어	수학
1	80 - score[0][0]	95 - score[0][1]	100 - score[0][2]
2	60 - score[1][0]	82 - score[1][1]	75 - score[1][2]

```
int main(void) {
    int score[2][3] = { {80, 95, 100}, {60, 82, 75} };
}
```

2 다차원 배열

3번까지 배열로 표현하면

번호	국어	영어	수학
1	80	95	100
2	60	82	75
3	88	92	64

```
int main(void) {
    int score[3][3] = {{80, 95, 100},{60, 82, 75},{88, 92, 64}};
}
```

2 다차원 배열

반이 2개면 3차원으로 표현될 수도 있습니다.

B반

번호	국어	영어	수학
1	80	95	100
2	60	82	75
3	88	92	64

A반

번호	국어	영어	수학
1	80	95	100
2	60	82	75
3	88	92	64

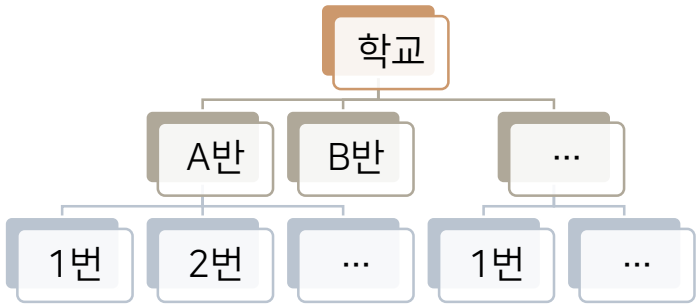
```
int main(void) {
    int score[2][3][3] = { {{80, 95, 100},{60, 82, 75},{88, 92, 64}},
                           {{52, 23, 64},{34, 77, 97},{35, 75, 84}} };
}
```

2

다차원 배열

배열만으론 복잡도가 점점 증가하기 때문에 다른 방법들을 사용하게 됩니다.

1 계층화된 자료구조



2 파일 시스템

3 데이터베이스

2 배열의 반복문 사용

구구단의 2단의 결과값을 1차원 배열에 저장하고자 하는 경우

```
int main(void) {
    int gugu[9]; // 2단의 곱셈값
    gugu[0] = 2 * 1;
    gugu[1] = 2 * 2;

}
```

2 배열의 반복문 사용

구구단의 2단의 결과값을 1차원 배열에 저장하고자 하는 경우

```
int main(void) {
    int gugu[9]; // 2단의 곱셈값

    for (int m = 0; m < 9; m++) {
        gugu[m] = 2 * (m + 1);
        printf("%d₩n", gugu[m]);
    }

}
```

2 배열의 반복문 사용

2단 ~ 9단의 결과값을 2차원 배열에 저장하고자 하는 경우

```
int main(void) {
    int gugu[8][9]; // 2~9단의 곱셈값

    for (int d = 0; d < 8; d++) {
        for (int m = 0; m < 9; m++) {
            gugu[d][m] = (d + 2) * (m + 1);
            printf("%d ", gugu[d][m]);
        }
        printf("₩n");
    }
}
```

2 배열을 함수의 매개변수로 사용

변수처럼 배열을 함수로 전달할 수 있습니다.

배열을 매개변수로 받는 함수 sum_array 의 선언과 호출부분

```
int sum_array(int nums[]);

int main(void) {
    int nums[] = { 10, 20, 30 };
    int ret = sum_array(nums);
    printf("sum_array 리턴 값 : %d₩n", ret);
}
```

2 배열을 함수의 매개변수로 사용

배열을 매개변수로 받는 함수 sum_array 의 정의 부분

```
int sum_array(int nums[]) {
    int sum = 0;
    for (int i = 0; i < 3; i++) {
        sum += nums[i];
    }
    return sum;
}
```

2 배열을 함수의 매개변수로 사용

배열의 크기를 3으로 함수내에서 정해주고 있습니다.

배열의 크기를 모르는 경우에는 함수를 어떻게 바꿔야 할까요.

```
int sum_array(int nums[]) {
    int sum = 0;
    for (int i = 0; i < 3; i++) {
        sum += nums[i];
    }
    return sum;
}
```

2 배열을 함수의 매개변수로 사용

매개변수 nums의 크기를 구해보면 항상 같은 값인 것을 확인할 수 있습니다.

사실 매개변수 int nums[ ] 는 배열이 아닙니다. 배열의 주소값만 들어옵니다.

```
int sum_array(int nums[]) {
    printf("%d", sizeof(nums));
    int sum = 0;
    for (int i = 0; i < 3; i++) {
        sum += nums[i];
    }
    return sum;
}
```

2 배열을 함수의 매개변수로 사용

따라서 별도로 size 크기를 매개변수로 받아야 합니다.

(전역 변수, 상수 등을 크기값으로 사용할 수도 있습니다.)

```
int sum_array(int nums[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += nums[i];
    }
    return sum;
}
```

2 배열을 함수의 매개변수로 사용

크기는 sizeof 키워드를 이용해 구할 수 있습니다.

원소의 개수 = 배열의 크기 / 자료형의 크기

```
int main(void) {
    int nums[] = { 10, 4, 48, 1 };
    int size = sizeof(nums) / sizeof(int);
    printf("max: %d", max(nums, size));
}
```

2 배열을 함수의 매개변수로 사용

배열안의 숫자중에 가장 큰값을 리턴하는 max함수를 만들어보세요

```
int max(int nums[], int size);
int main(void) {
    int nums[] = { 10, 4, 48, 1 };
    int size = sizeof(nums) / sizeof(int);
    printf("max: %d", max(nums, size));
}
```

2 배열을 함수의 매개변수로 사용

배열안의 숫자중에 가장 큰값을 리턴하는 max함수를 만들어보세요

```
int max(int nums[], int size) {
    int max = 0;
    for (int i = 0; i < size; i++) {
        if (nums[i] > max) {
            max = nums[i];
        }
    }
    return max;
}
```

3.

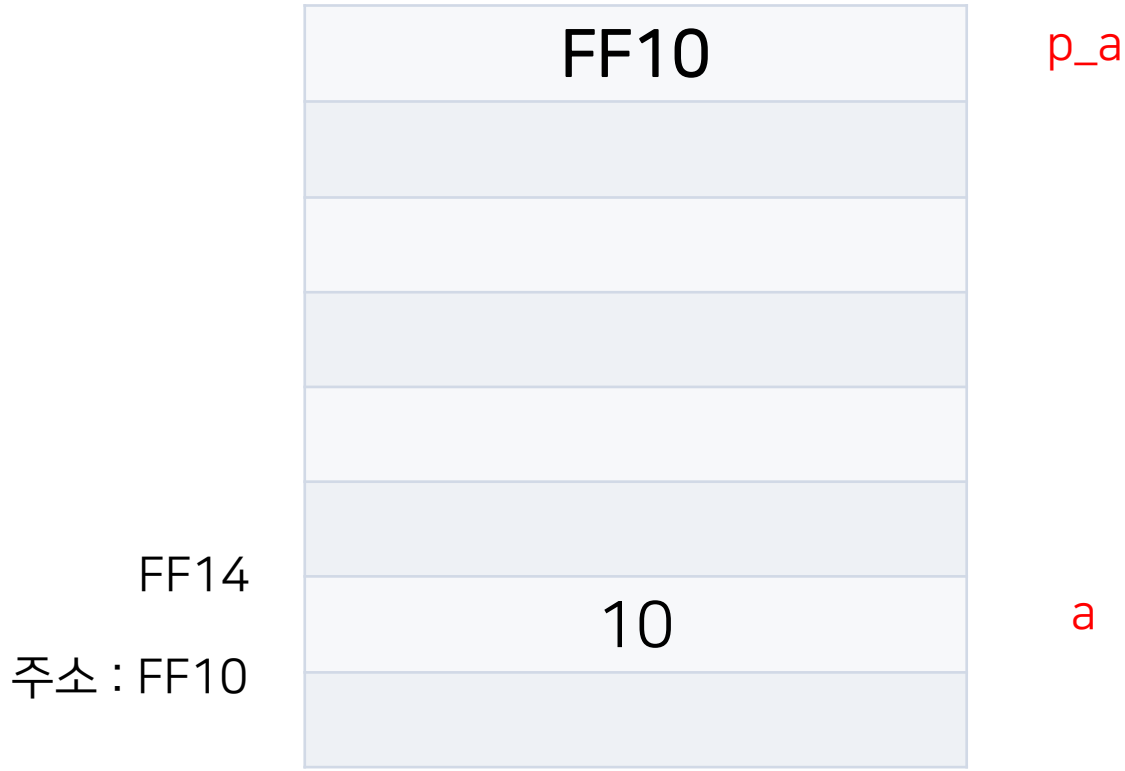
포인터

3 포인터 변수

포인터 (pointer) 란 주소값을 저장하는 변수입니다.

```c
#include <stdio.h>

int main(void) {
    int a = 10;
    int* p_a = &a;
    printf("%p₩n", &a);
    printf("%p", p_a);
}
```

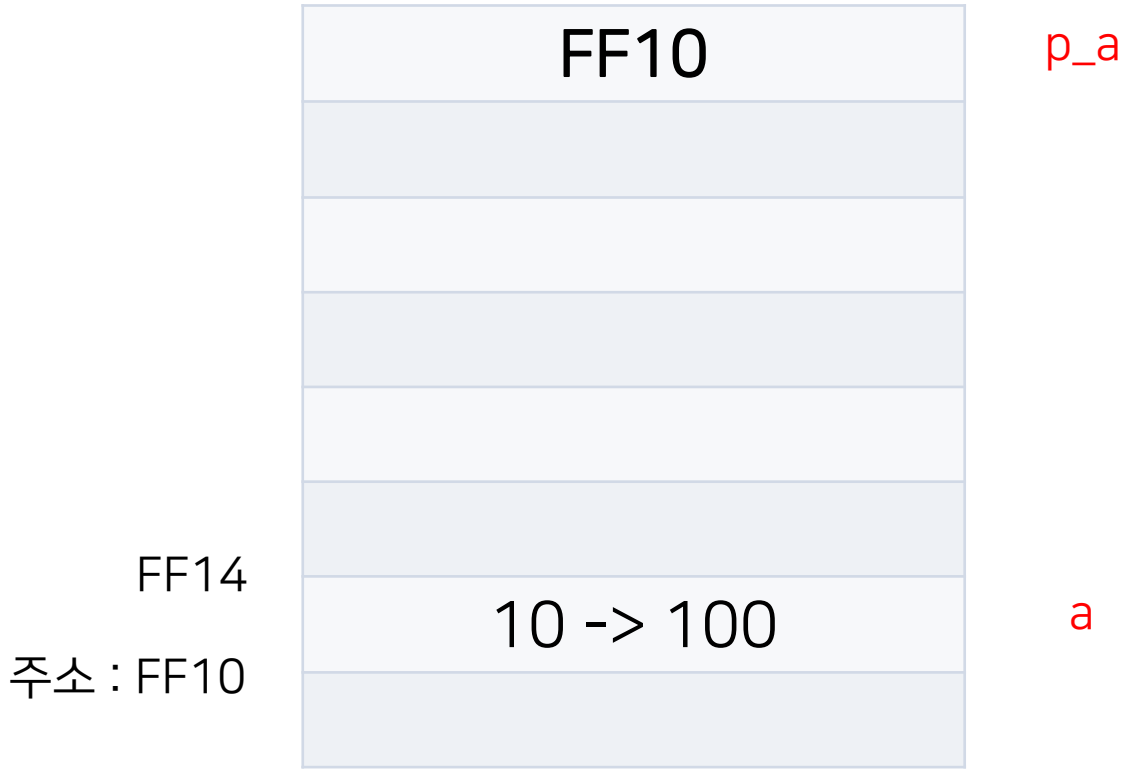


3 * 연산자

아래코드의 *p_a = 100의 *기호는 간접참조연산자 라고 불리는 연산자로

포인터 p_a 에 저장된 주소를 찾아가서 값을 처리할 수 있습니다.

```
int main(void) {
    int a = 10;
    int* p_a = &a;
    *p_a = 100;
    printf("%d", a);
}
```



3 포인터의 크기

포인터는 프로그램이 동작할 cpu 의 사양에 따라 크기가 달라집니다.

64bit / 32bit 등 (visual studio 기본설정은 4byte - 32bit)

```c
int main(void) {
    int a = 10;
    int* p_a = &a;
    printf("%d₩n", sizeof(p_a));
    printf("%p", p_a);
}
```

포인터의 크기는 주소의 크기:

012FFADC

3 포인터의 자료형

포인터와 변수의 자료형이 일치하지 않으면 정확한 값을 사용할 수 없습니다.

```c
int main(void) {
    int a = 10000;
    char* p_a = &a;
    printf("%d", *p_a);
}
```

십진수 10000을 비트로 표현하면

… 0010 0111 0001 0000

char* 로 선언하면 해당위치에서 1byte만큼만 값으로 인식합니다 (char의 크기 1byte).

3 포인터의 형변환

강제 형변환(casting) 하면 원래 int 값 전체를 읽어올 수 있습니다.

```
int main(void) {
    int a = 10000;
    char* p_a = &a;
    printf("%d", *(int*)p_a);
}
```

십진수 10000을 비트로 표현하면

… 0010 0111 0001 0000

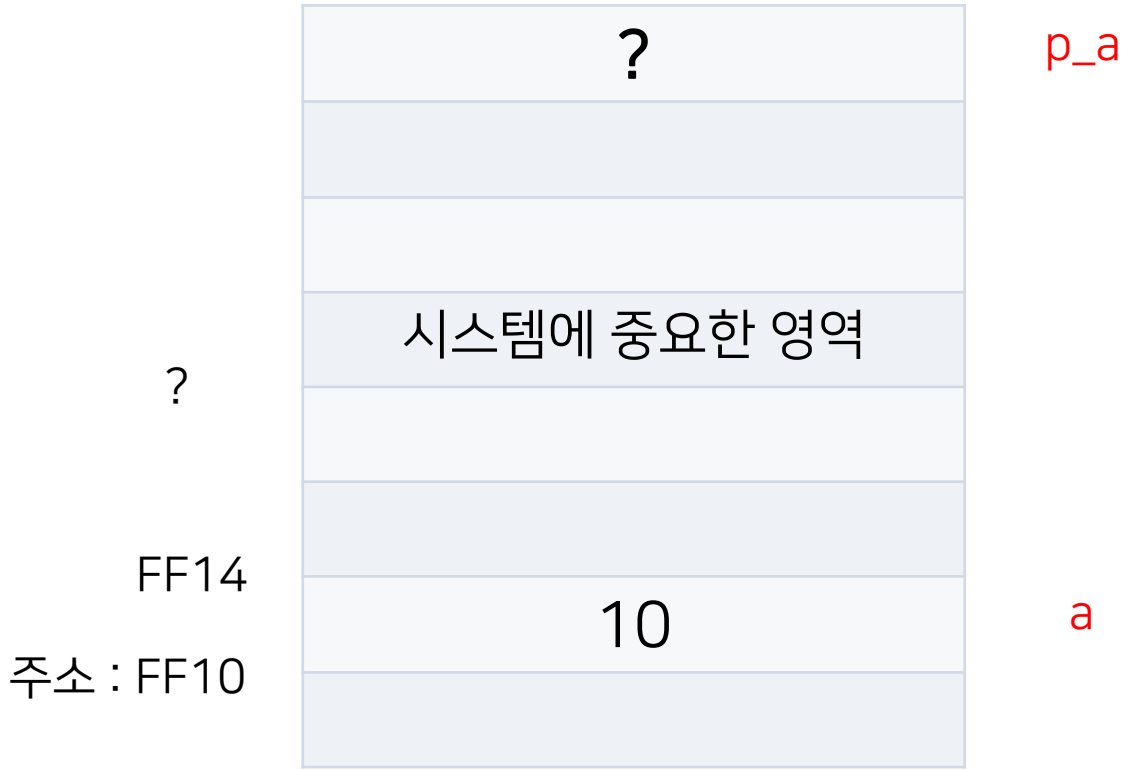
int형 포인터로 형변환을 하면 다시 4byte만큼 값으로 인식합니다 (int의 크기 4byte).

3 포인터의 쓰레기값

포인터를 선언만 하면 그 안에 어떤 값이 담길지 알 수 없습니다.

이러한 무작위주소의 값을 변경하면 시스템에 치명적일 수 있습니다.

```c
int main(void) {
    int a = 10;
    int* p_a = &a;
    *p_a = 100;
    printf("%d", a);
}
```

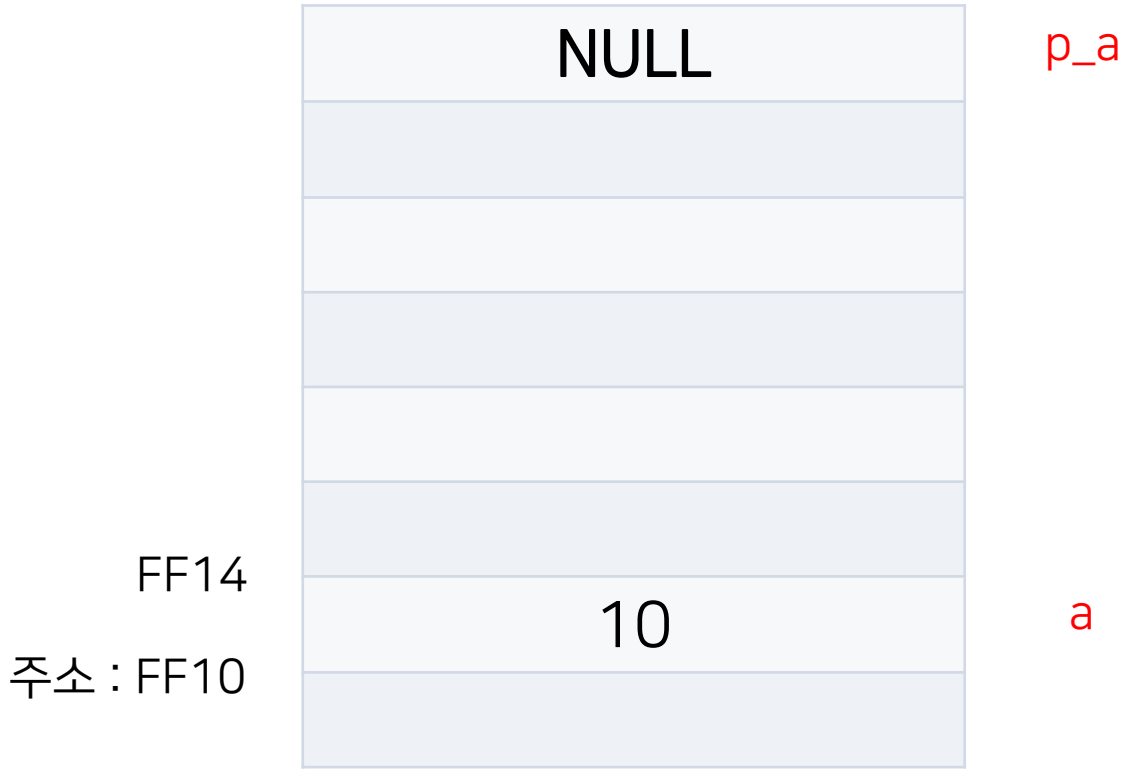


3 포인터의 초기화

물론 개발툴과 운영체제가 보호하지만 장담할 수가 없습니다.

선언만 하는 경우 NULL (아무런 주소도 없음) 을 사용하여 초기화 필요.

```
int main(void) {
    int a = 10;
    int* p_a = NULL;
    printf("%p", p_a);
}
```



3 포인터 주소로 하드웨어 제어

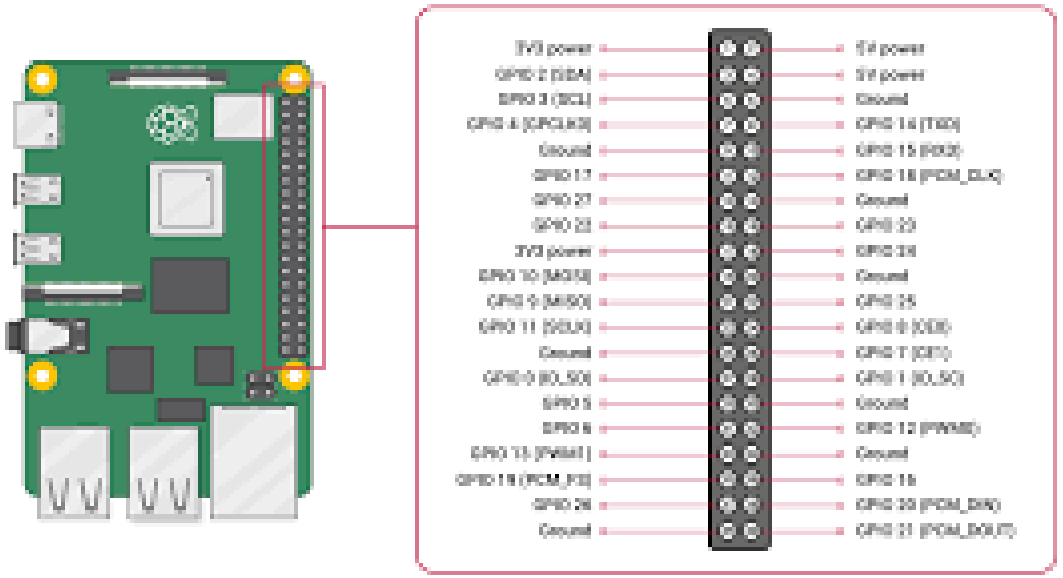
포인터의 주소에 임의의 주소를 직접 입력하여 하드웨어를 제어할 수 있습니다.

현재 개발 환경에서는 사용할 수 없습니다. (메모리 엑세스 위반)

```
int main(void) {
    int a = 10;
    unsigned char* p_a = (unsigned char*)0x008FFBCC;
    *p_a = 0b00000001;
}
```

포인터 주소로 하드웨어 제어

GPIO(General Purpose Input Output)



이런 임베디드 환경에서는 메모리의
주소와 하드웨어가 직접 연결되어
포인터를 이용해서 제어할 수 있습니다.

```
#define GPIO_BASE       0x00006F80

#define GPACRL          ((volatile unsigned long*)(GPIO_BASE + 0x00000000))
#define GPAQSEL1        ((volatile unsigned long*)(GPIO_BASE + 0x00000002))
#define GPAQSEL2        ((volatile unsigned long*)(GPIO_BASE + 0x00000004))
#define GPAMUX1         ((volatile unsigned long*)(GPIO_BASE + 0x00000006))
#define GPAMUX2         ((volatile unsigned long*)(GPIO_BASE + 0x00000008))
#define GPADIR          ((volatile unsigned long*)(GPIO_BASE + 0x0000000A))
#define GPAPUD          ((volatile unsigned long*)(GPIO_BASE + 0x0000000C))
```

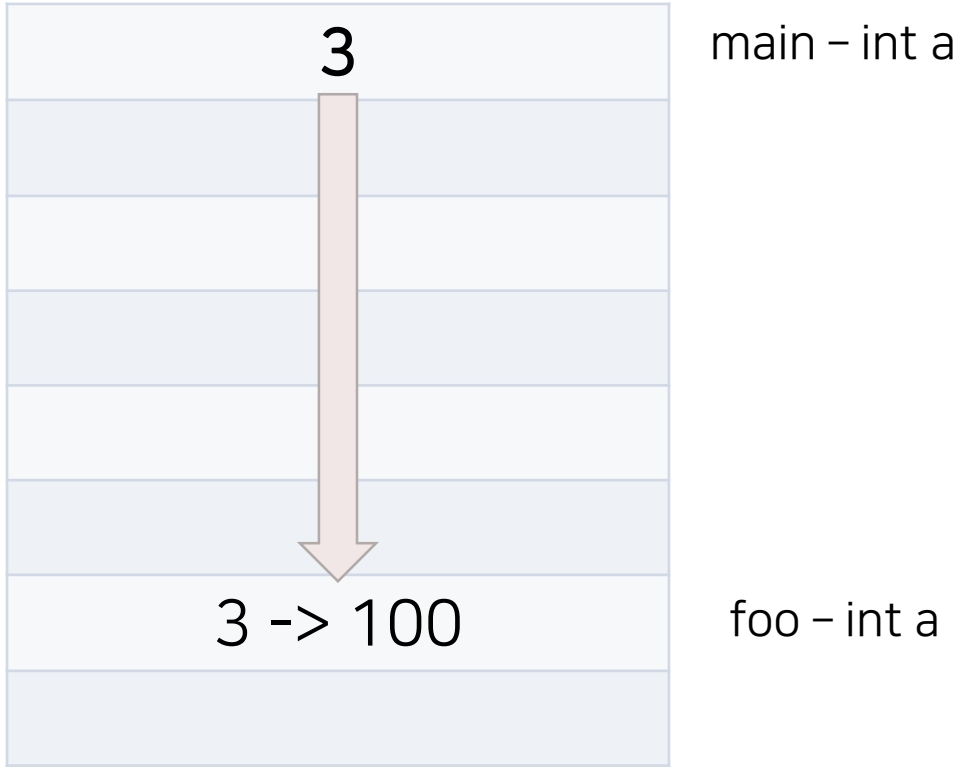
3 매개변수로의 포인터

C에서 함수에 매개변수를 전달할 때는 값의 복사가 일어납니다.

- call by value

```c
int main(void) {
    int a = 3;
    foo(a);
    printf("%d", a);
}

int foo(int a) {
    a = 100;
}
```



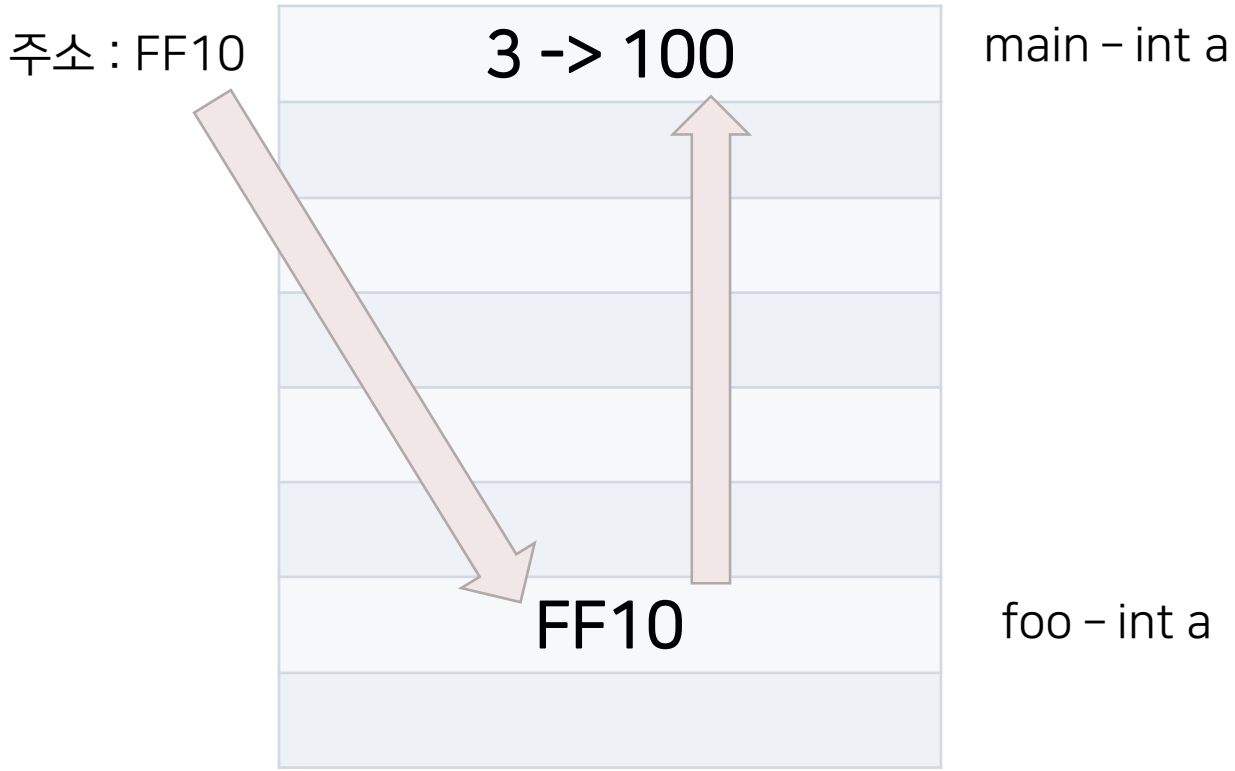
3 매개변수로의 포인터

다른 함수에게 포인터를 전달하면 원본을 전달한 것 같은 효과가 납니다.

- call by reference 의 효과

```c
int main(void) {
    int a = 3;
    foo(&a);
    printf("%d", a);
}

int foo(int* a) {
    *a = 100;
}
```



3 매개변수로의 포인터

따라서 함수를 이용해 변수의 원본을 조작하려는 경우

포인터를 사용해야만 합니다. 아래는 두 변수의 값을 서로 바꾸는 swap 함수

```c
int main(void) {
    int a = 3, b = 5;
    printf("swap 전 a: %d, b: %d₩n", a, b);
    swap(&a, &b);
    printf("swap 후 a: %d, b: %d₩n", a, b);
}

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