

## 강04. 함수와 배열

## 9장. C 언어의 핵심! 함수

- main 함수 다시 보기 : 함수의 기본 형태



- 함수를 정의하는 이유
  - 모듈화에 의한 프로그램의 질 향상이 가능
  - 유지 보수 및 확장의 용이성
  - 문제 해결의 용이성 : "Divide and Conquer!"
- 4가지 형태의 함수
  - 전달 인자 有, 반환 값 有
  - 전달 인자 有, 반환 값 無
  - 전달 인자 無, 반환 값 有
  - 전달 인자 無, 반환 값 無

- 전달 인자와 반환 값, 둘 다 있는 함수

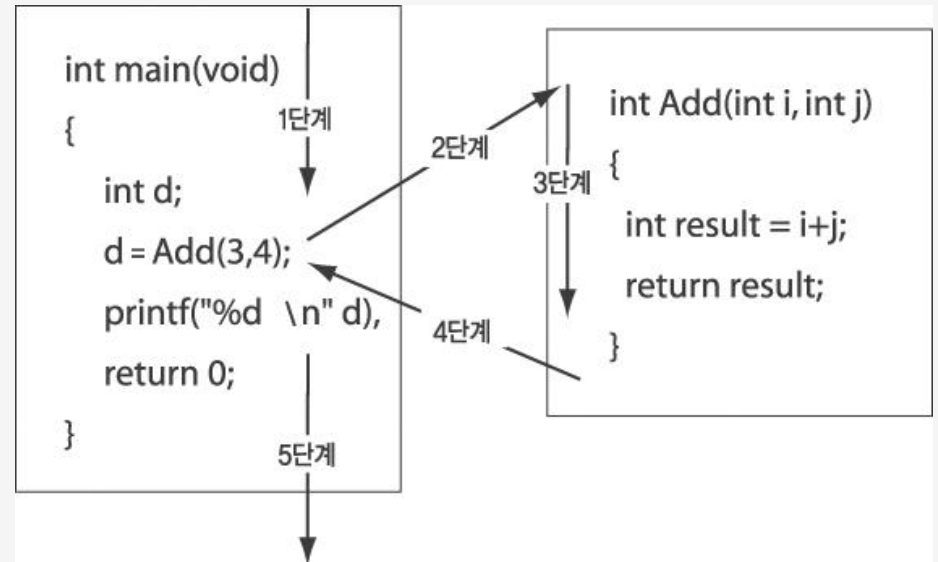
```
int Add(int i, int j)
{
    int result = i+j;
    return result;
}
```

```
  a      b      c
int  Add  (int i,int j)
{
    int result = i+j;
  d  return result;
}
```

- a 반환 형
- b 함수 이름
- c 매개 변수
- d 값의 반환

- 함수 호출 과정의 이해

```
#include <stdio.h>
int Add(int i, int j)
{
    int result = i + j;
    return result;
}
int main(void)
{
    int d;
    d = Add(3, 4);
    printf("%d \n", d);
    return 0;
}
```



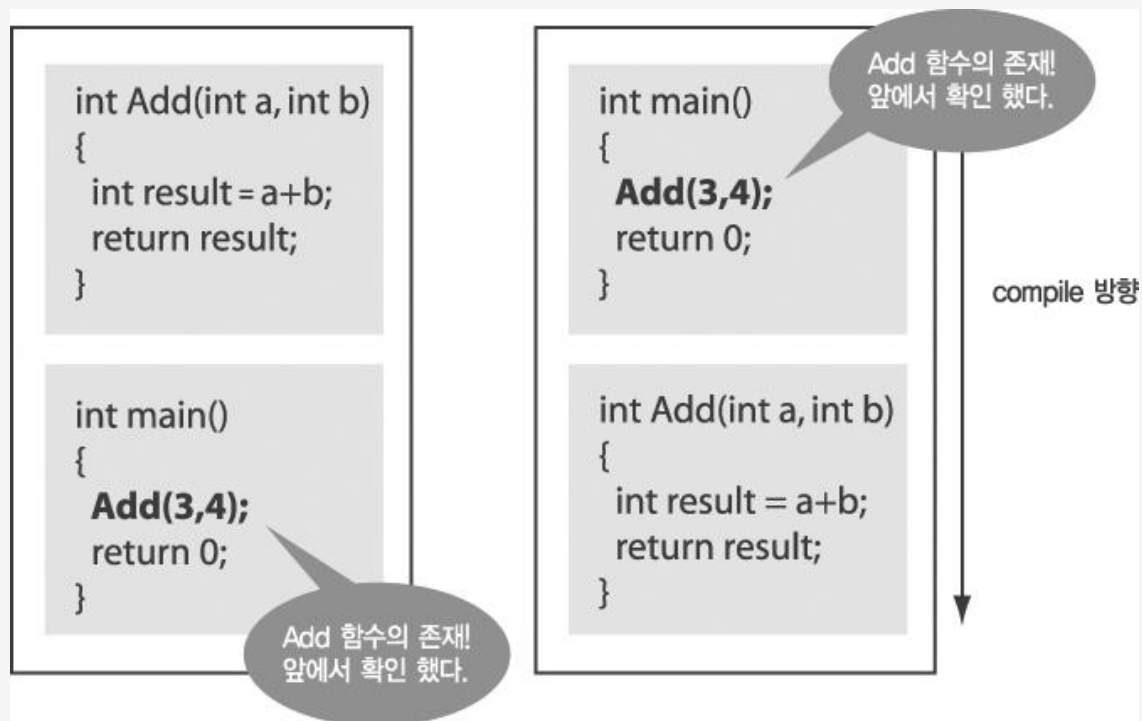
- 다양한 형태의 함수

```
void Result_Print(int val)
{
    printf("덧셈에 대한 결과 : %d \n", val);
    printf("***** END ***** \n");
}
```

```
int Input(void)
{
    int input;
    scanf("%d", &input);
    return input;
}
```

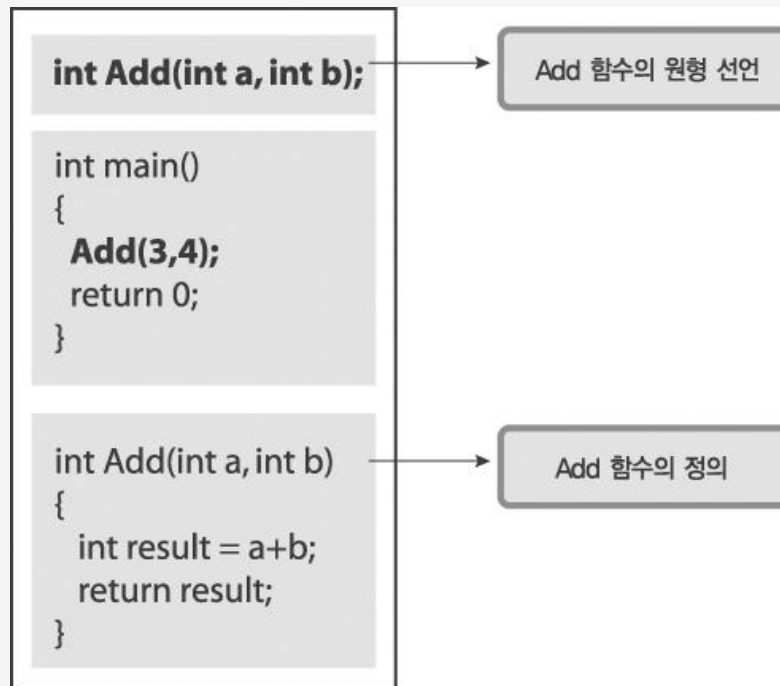
```
void Intro(void)
{
    printf("***** START ***** \n");
    printf("두개의 정수 입력 : ");
}
```

- 함수 선언의 필요성
  - 컴파일러의 특성상, 함수는 호출되기 전에 정의되어야 한다.





- 함수 선언의 의미
  - 이후에 정의될 함수에 대한 정보 제공



- 실습 문제

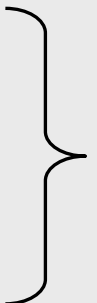
- 입력받은 KEY 값을 함수의 매개변수로 전달하고  
함수내에서 대문자/소문자/숫자/특수키 종류를  
구분하여 출력하는 프로그램을 작성

```
>a : 소문자  
>A : 대문자  
>@ : 특수문자  
>4 : 숫자
```

- 변수의 특성에 따른 분류
  - 지역 변수(Local Variable)
    - 중 괄호 내에 선언되는 변수
  - 전역 변수(Global Variable)
    - 함수 내에 선언되지 않는 변수
  - 정적 변수(Static Variable)
    - 함수 내부, 외부 모두 선언 가능
  - 레지스터 변수(Register Variable)
    - 선언에 제한이 많이 따름


- 지역 변수의 접근 범위
  - 지역 변수는 선언 된 함수 내에서만 접근 가능

```
int fct_one(void)
{
    int one=0;
    ...
    return 0;
}
```



범위 1

```
int fct_two(void)
{
    int two=0;
    int one=0;
    ...
    return 0;
}
```

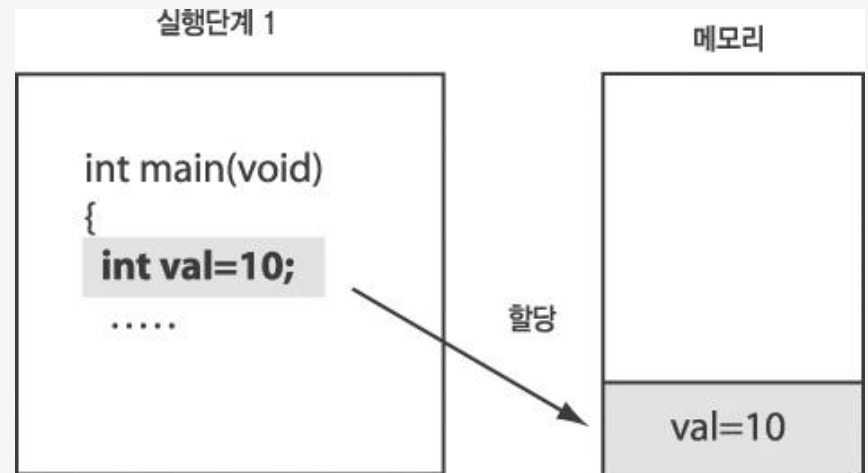


범위 2

- 지역 변수의 메모리상 존재 기간

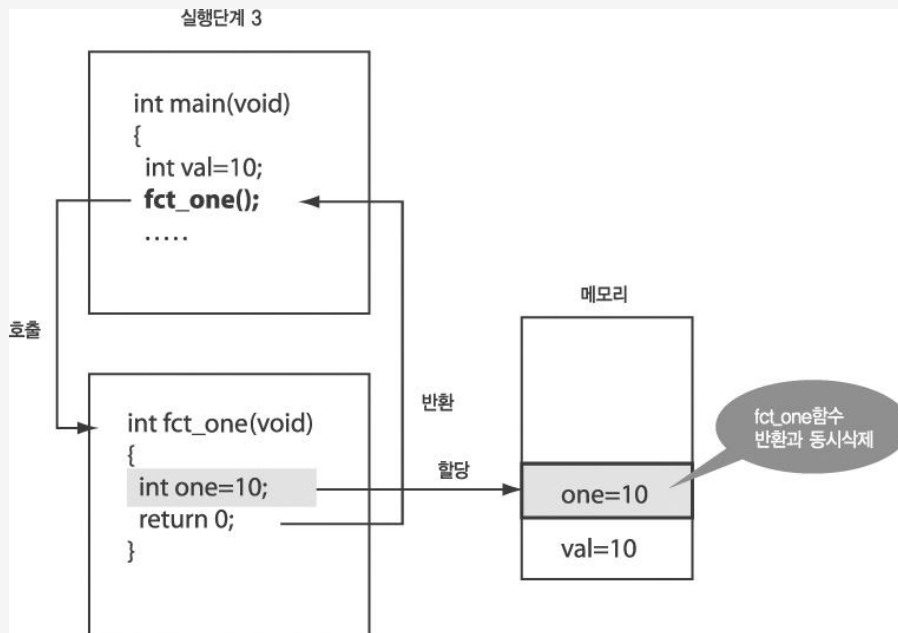
```
int fct_one(void)
{
    int one=10;
    return 0;
}
int fct_two(void)
{
    int one=20;
    int two=30;
    return 0;
}
int main(void)
{
    int val=10;
    fct_one();
    fct_two();
    return 0;
}
```

## 1 단계

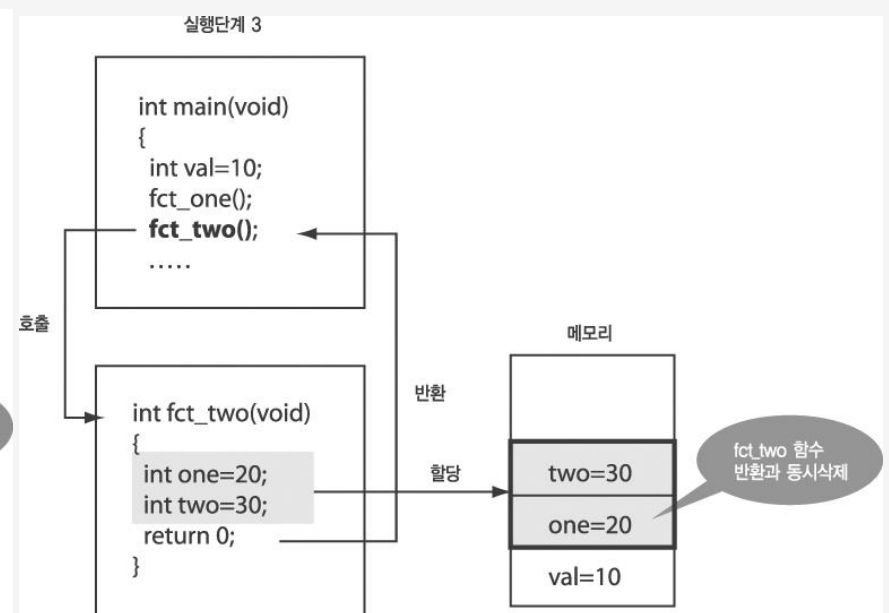


- 지역 변수의 메모리상 존재 기간

## 2단계

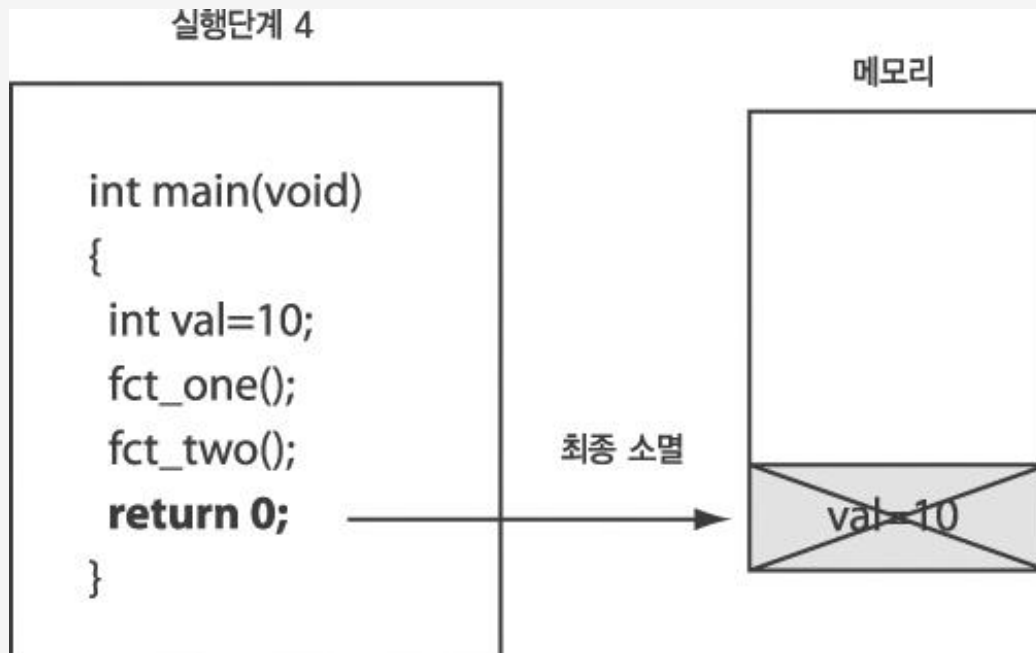


## 3단계



- 지역 변수의 메모리상 존재 기간

## 4단계

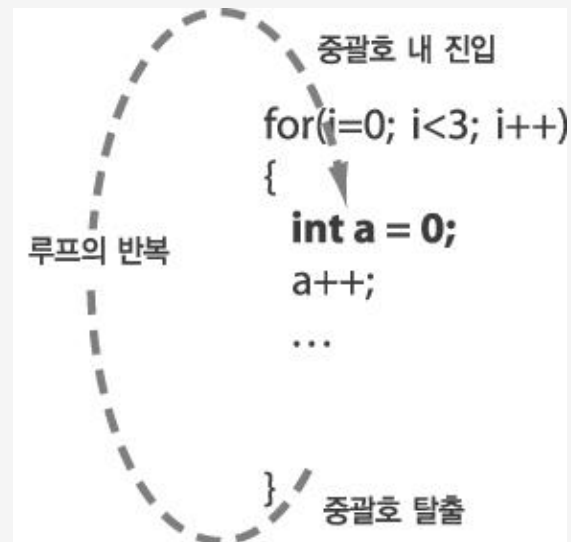


- 또 다른 형태의 지역 변수
  - while, for, if와 같은 문장에 의해 형성 되는 지역 내에서 선언되는 변수

```
int main(void)
{
    int i;
    for(...)
    {
        .....
    }

    if(...)
    {
        .....
    }
    return 0;
}
```

변수 i에 접근이 가능한  
main 영역의 범위

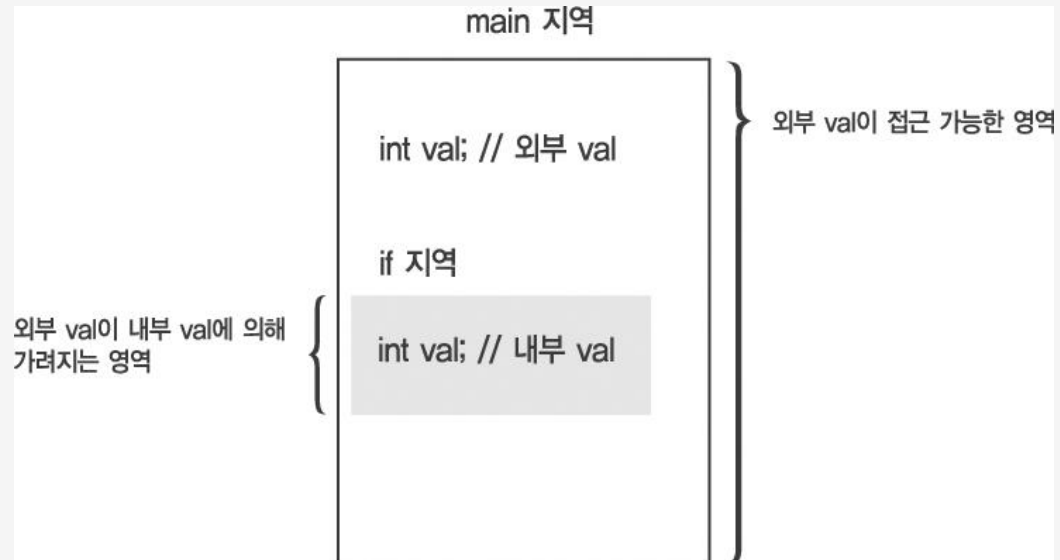




- 지역 변수의 또 다른 특성
  - 경우에 따라서 지역 변수는 다른 지역 변수를 가리기도 한다.

```
/* local_val2.c */
#include <stdio.h>

int main(void)
{
    int val=0;
    if(1) //무조건 true
    {
        int val=0;
        val+=10;
        printf("%d", val);
    }
    printf("%d", val);
    return 0;
}
```



- 지역 변수와 매개 변수
  - 매개 변수도 지역 변수의 일종이다.

```
int fction (int a, int b)
{
    a+=10;
    b+=20
    return a+b;
}
```

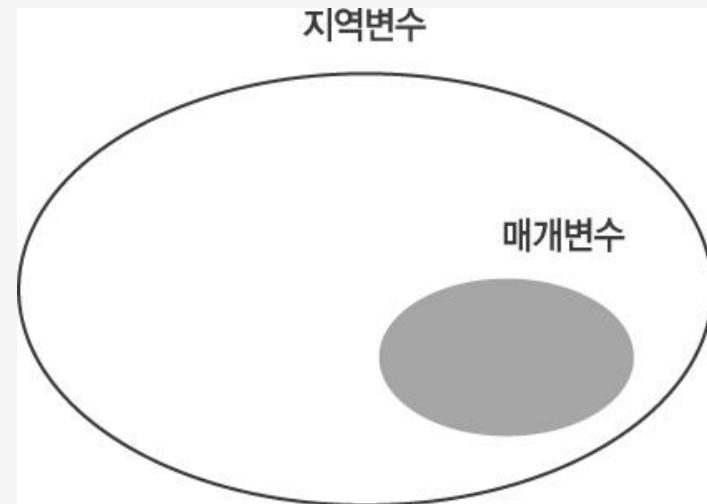


그림 9-15

- 전역 변수
  - 프로그램 어디에서나 접근이 가능한 변수
  - 특정 영역(함수, 반복문...)에 속하지 않는다.
  - 프로그램이 종료될 때까지 존재
- 전역 변수의 또 다른 특징
  - 같은 이름의 지역 변수에 의해서 가려지기도 한다.

- static 변수
  - 함수 내부 및 외부에 선언 가능하다.
  - 한번만 초기화된다 : 전역 변수의 특징
  - 함수 내부에서 선언될 경우 함수 내에서만 접근이 가능하다 : 지역 변수의 특징
- 보다 빠르게! register 변수

```
int main(void)
{
    int a ;
    register int b;      // 레지스터 변수 b 선언
    . . . . .
```

# C/C++

---

```
/* static_val.c */
#include <stdio.h>

void fct(void);

int main(void)
{
    int i;
    for(i=0; i<5; i++)
        fct();

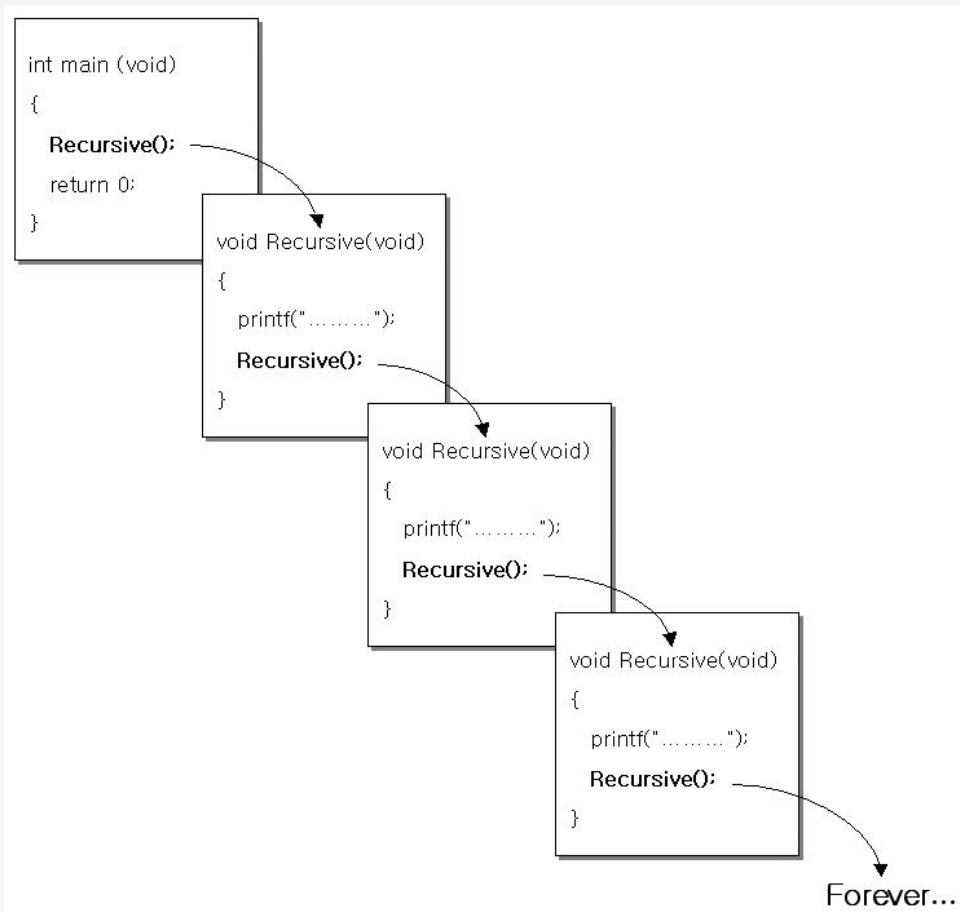
    return 0;
}

void fct(void)
{
    int val=0;      // static int val=0;
    val++;
    printf("%d ",val);
}
```

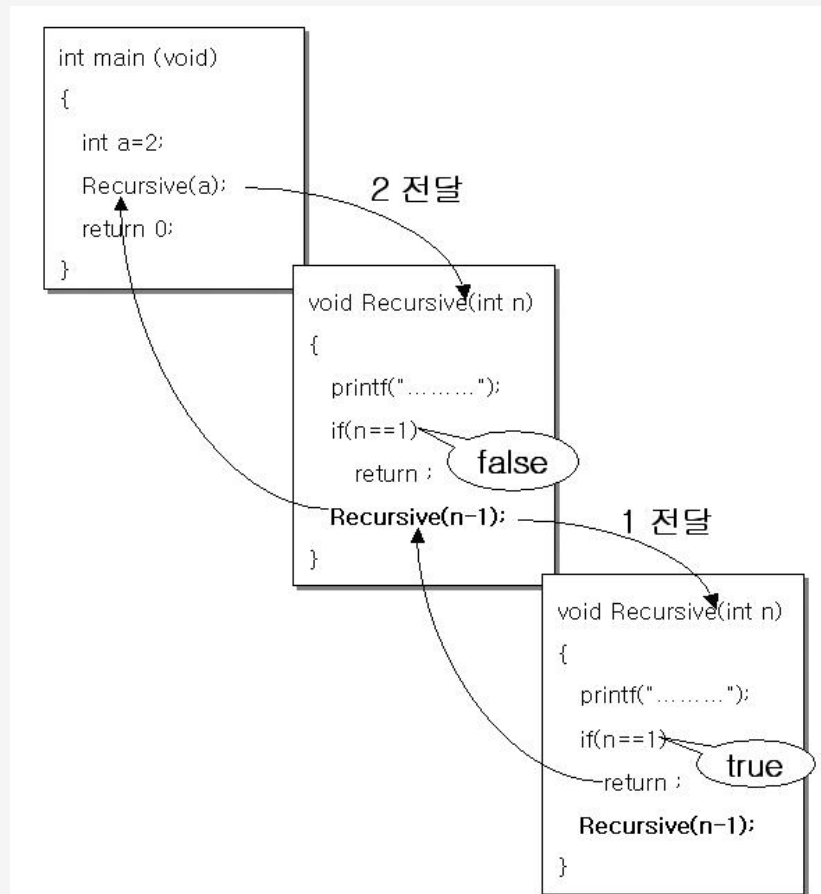
- 재귀 함수의 기본적 이해
  - 자기 자신을 다시 호출하는 형태의 함수

```
/* recursive_basic.c */  
#include <stdio.h>  
  
void Recursive(void)  
{  
    printf("Recursive Call! \n");  
    // Recursive();  
}  
  
int main(void)  
{  
    Recursive();  
    return 0;  
}
```

- 탈출 조건의 필요성
  - 무한 재귀 호출을 피하기 위해서



- 탈출 조건의 이해





# C/C++

---

```
/* recursive_basic2.c */
#include <stdio.h>

void Recursive(int n)
{
    printf("Recursive Call! %d\n", n);
    if(n==1)
        return;
    Recursive(n-1);
}

int main(void)
{
    int a=2;
    Recursive(a);
    return 0;
}
```

- 재귀 함수 Design 사례
  - 팩토리얼(factorial) 계산을 위한 알고리즘

$$n! = n \times (n-1) \times (n-2) \times (n-3) \dots \times 2 \times 1$$

↓

$$n! = n \times (n-1)!$$

$$f(n) = \begin{cases} n \times f(n-1) & n \text{이 } 1 \text{ 이상인 경우} \\ 1 & n \text{이 } 0 \text{인 경우} \end{cases}$$

- 재귀 함수 Design 예시
  - 알고리즘을 코드로 옮기기 위한 pseudo code와 C 코드

// 시작 조건 :  $n$ 은 0 이상이다.

시작(START) :  $f(n)$  호출

1. 만약에  $n$ 이 0이면 1을 반환
2. 그렇지 않다면  $n \times f(n-1)$ 을 반환.

끝(END)

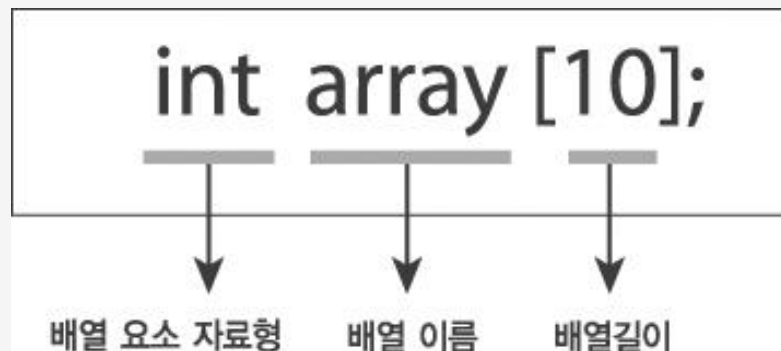
```
int f(int n)
{
    if (n==0)
        return 1;
    else
        return n*f(n-1);
}
```

## 10장. 1차원 배열

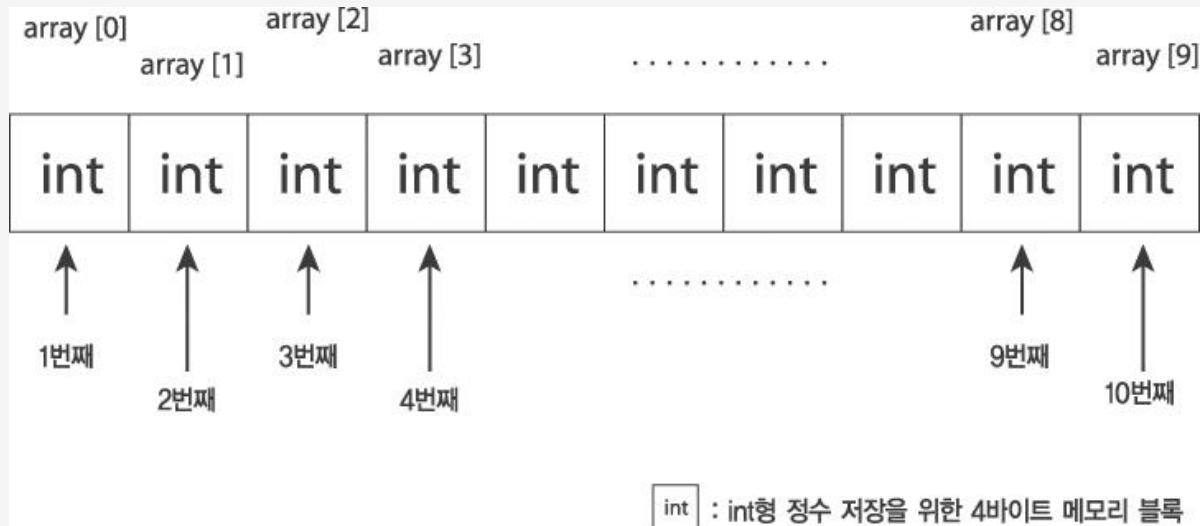
## [ 배열이란? ]

- 둘 이상의 변수를 동시에 선언하는 효과를 지닌다.
- 많은 양의 데이터를 일괄적으로 처리해야 하는 경우에 유용하다.
- 지역적 특성을 지닐 수도 있고, 전역적 특성을 지닐 수도 있다.

- 배열 선언에 있어서 필요한 것 세 가지
  - 배열 길이 : 배열을 구성하는 변수의 개수  
(반드시 상수를 사용)
  - 배열 요소 자료형 : 배열을 구성하는 변수의 자료형
  - 배열 이름 : 배열에 접근할 때 사용되는 이름



- 1차원 배열의 접근
  - 배열 요소의 위치를 표현 : 인덱스(index)
  - 인덱스는 0에서부터 시작



- 배열 선언과 접근의 예

```
int main(void)
{
    int array[10];    // 배열 선언
    array[0]=10;      // 첫 번째 요소 접근
    array[1]=20;      // 두 번째 요소 접근
    array[2]=30;      // 세 번째 요소 접근
    . . . . .
    return 0;
}
```

array[s] = 10;



S+1번째 요소에 10을 대입하라.



# C/C++

---

```
/* array1.c */
#include <stdio.h>

int main(void)
{
    double total;
    double val[5];

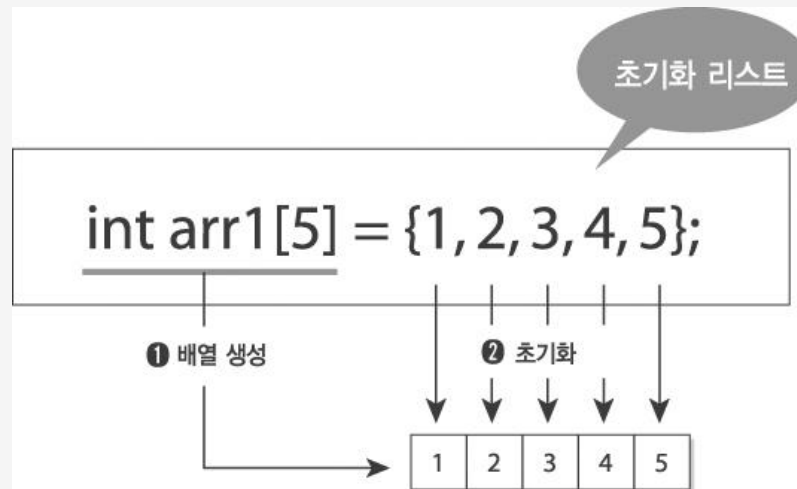
    val[0]=1.01;
    val[1]=2.02;
    val[2]=3.03;
    val[3]=4.04;
    val[4]=5.05;

    total=val[0]+val[1]+val[2]+val[3]+val[4];
    printf("평균 : %lf \n", total/5);

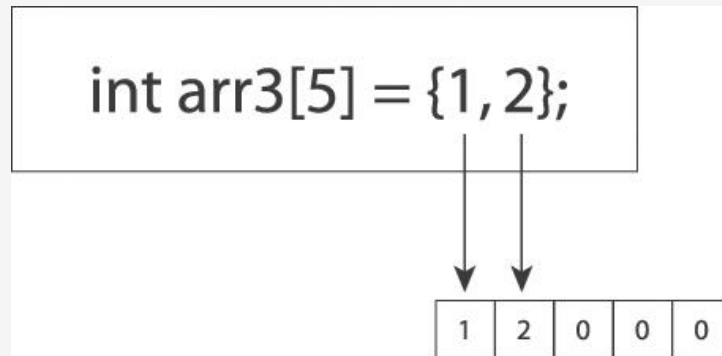
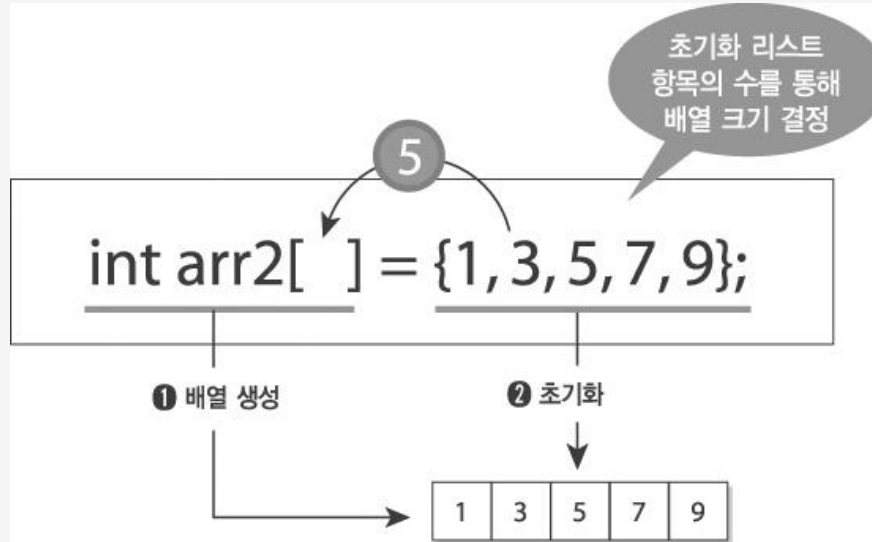
    return 0;
}
```

- 선언과 동시에 초기화

```
int main(void)
{
    int arr1[5]={1, 2, 3, 4, 5};
    int arr2[ ]={1, 3, 5, 7, 9};
    int arr3[5]={1, 2}
}
```



# C/C++



- 문자열 상수

- 문자열이면서 상수의 특징을 지닌다.

```
printf("Hello World! \n");
```

- 문자열 변수

- 문자열이면서 변수의 특징을 지닌다.

```
char str1[5]="Good";  
char str2[]="morning";
```

```
/* ar_str.c */
#include <stdio.h>

int main(void)
{
    char str1[5]="Good";
    char str2[]="morning";

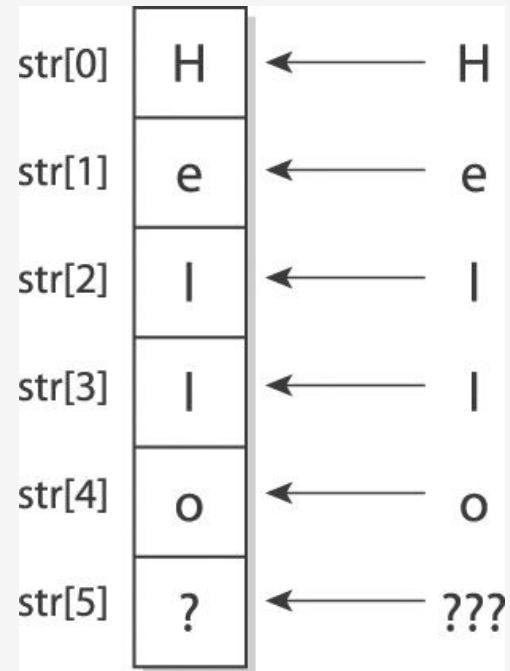
    printf("%s \n", str1);
    printf("%s %s \n ", str1, str2);

    return 0;
}
```

- 문자열의 특징

- 문자열은 널(null)문자를 끝에 지닌다.
- 널(null) 문자 : '\0'(아스키 코드 값으로 0)

```
int main(void)
{
    char str[6]="Hello";
    printf("Hello");
    . . . . .
```



- 널(null) 문자를 지녀야 하는 이유
  - 문자열의 끝을 표현하기 위해서
  - 쓰레기 값과 실제 문자열의 경계를 나타내기 위해
  - printf 함수는 널 문자를 통해서 출력의 범위를 결정 짓는다.

```
int main(void)
{
    char str[100]="Hello World!";
    printf("%s\n", str);
    . . . . .
```

- 문자열과 char형 배열의 차이점

```
char arr1[ ] = "abc";  
char arr2[ ] = {'a', 'b', 'c'};  
char arr3[ ] = {'a', 'b', 'c', '\0'};
```



# C/C++

```
/* va_str.c */
#include <stdio.h>

int main(void)
{
    int i;
    char ch;
    char str[6]="Hello";

    printf("--변경 전 문자열--\n");
    printf("%s\n", str);

    for(i=0; i<6; i++)
        printf("%c | ", str[i]);

    /* 문자열 변경 */
    for(i=0; i<3; i++)
    {
        ch=str[4-i];
        str[4-i]=str[i];
        str[i]=ch;
    }

    printf("\n\n--변경 후 문자열--\n");
    printf("%s\n", str);
    return 0;
}
```

# C/C++

- scanf 함수를 이용하여 문자열을 입력받아 한 문자씩 띄어서 출력하시오.  
( char 배열을 이용할 것 ) “abcde” ==> “a b c d e?????????”
- 대소문자가 혼합된 입력받은 문자열을 모두 대문자로 일괄 변환하기.

```
char str[100];
scanf("%s",str);
for(i=0;i<100;i++)
{
    char a;
    a = str[i];
    if(str[i] == 0) break;
    if(a>96 && a<123) a -= 32; // a가 소문자이면
    printf("%c",a);
}
```

## 11. 다차원 배열

- 다차원 배열이란 무엇인가?
  - 2차원 이상의 배열을 의미함
- 다차원 배열의 선언

배열 선언 예	몇 차원 배열인가?
int arr[100]	1차원 배열
int arr[10][10]	10×10, 2차원배열
int arr[5][5][5]	5×5×5, 3차원 배열

- 2차원 배열의 선언

- 2차원적 메모리 구조를 구성

```
int main(void)
```

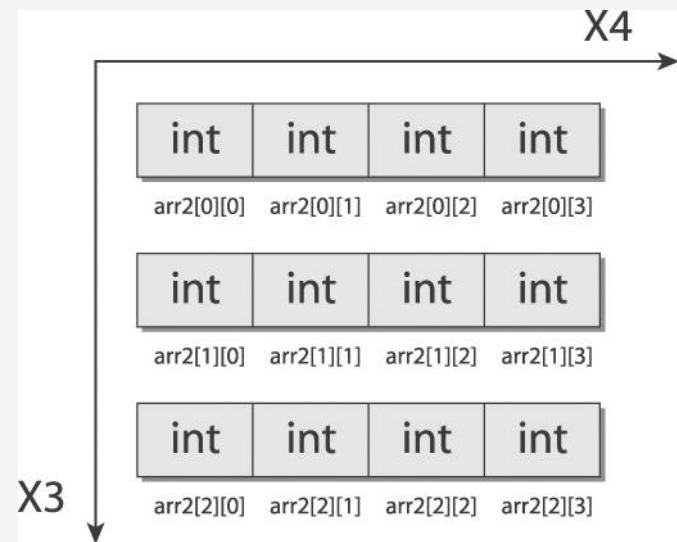
```
{
```

```
    int arr1[4];
```

```
    int
```

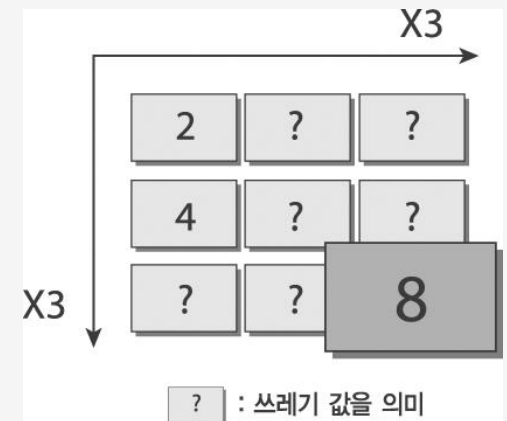
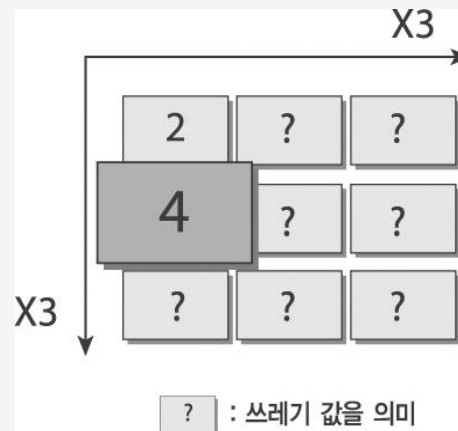
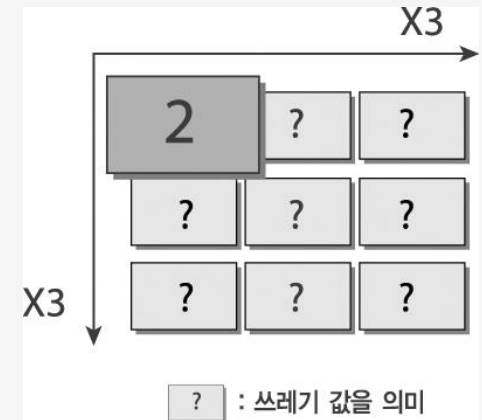
```
    arr2[3][4];
```

```
    . . . . .
```



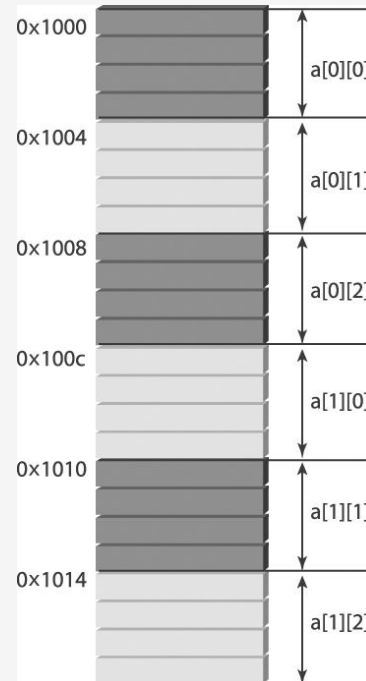
## • 2차원 배열 요소의 접근 방법

```
int main(void)
{
    int arr[3][3];
    arr[0][0]=2;
    arr[1][0]=4;
    arr[2][2]=8;
    . . . . .
}
```

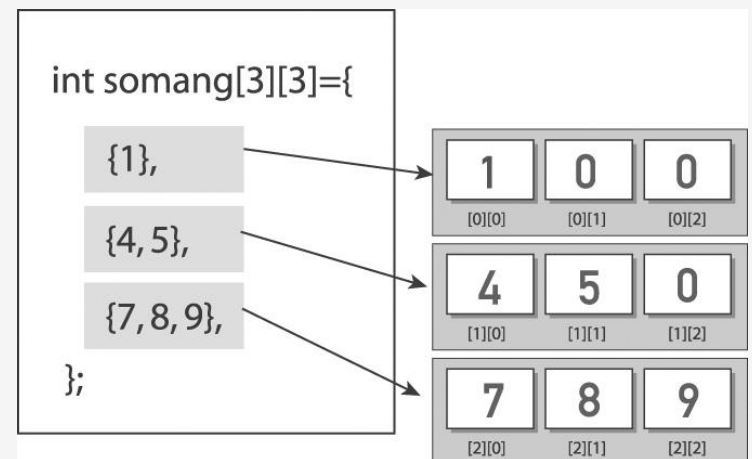
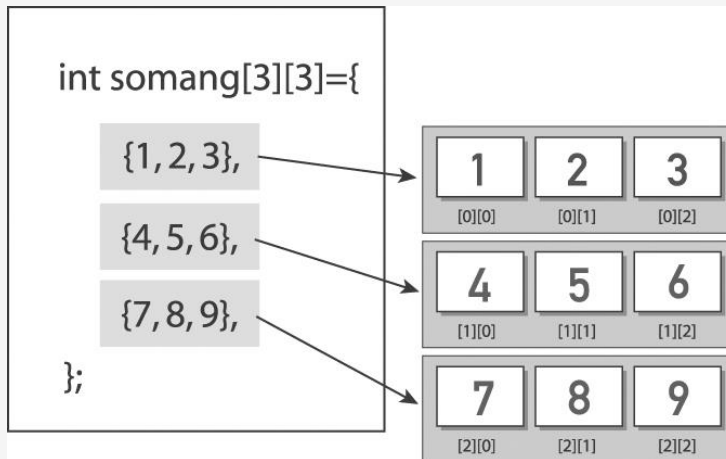


- 다차원 배열의 실제 메모리 구성
  - 1차원 배열과 동일하다.  
다만 접근 방법을 2차원적으로 해석할 뿐이다.
  - 2차원적으로 이해하는 것이 좋은 습관!

```
int a[2][3]
```



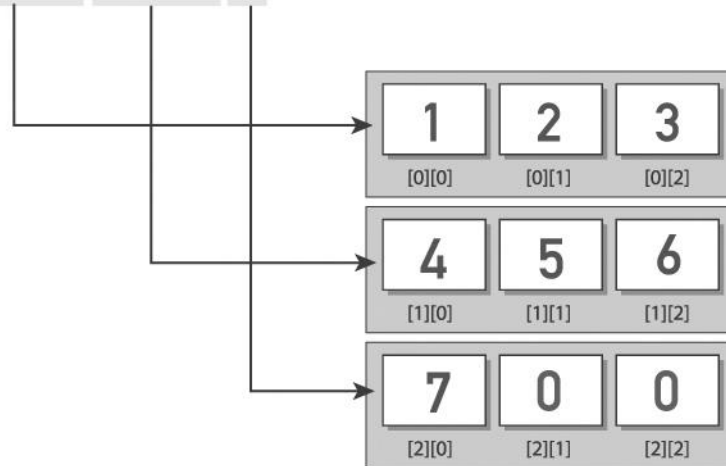
- 2차원 배열! 선언과 동시에 초기화
  - case 1 : 행 단위로 모든 요소들을 초기화
  - case 2 : 행 단위로 일부 요소들만 초기화





- 2차원 배열! 선언과 동시에 초기화
  - case 3 : 1차원 배열 형태의 초기화

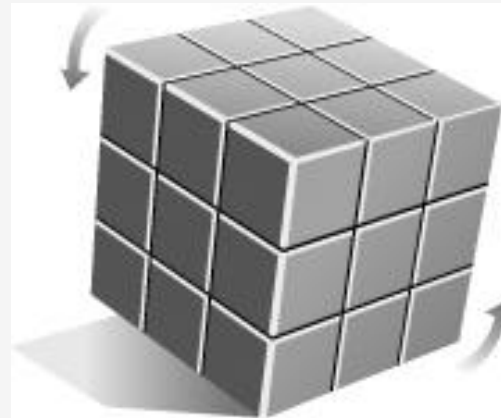
```
int somang[3][3]={ 1,2,3, 4,5,6, 7};
```



- 초기화 리스트에 의한 배열 크기의 결정
  - 1차원 배열의 예
    - `int arr[]={1, 2, 3, 4, 5};`
  - 2차원 배열의 예
    - `int arr[][]={1, 2, 3, 4, 5, 6, 7, 8};`    `//Error!`
    - `int arr[][4]={1, 2, 3, 4, 5, 6, 7, 8};`    `//OK!`
    - `int arr[][2]={1, 2, 3, 4, 5, 6, 7, 8};`    `//OK!`

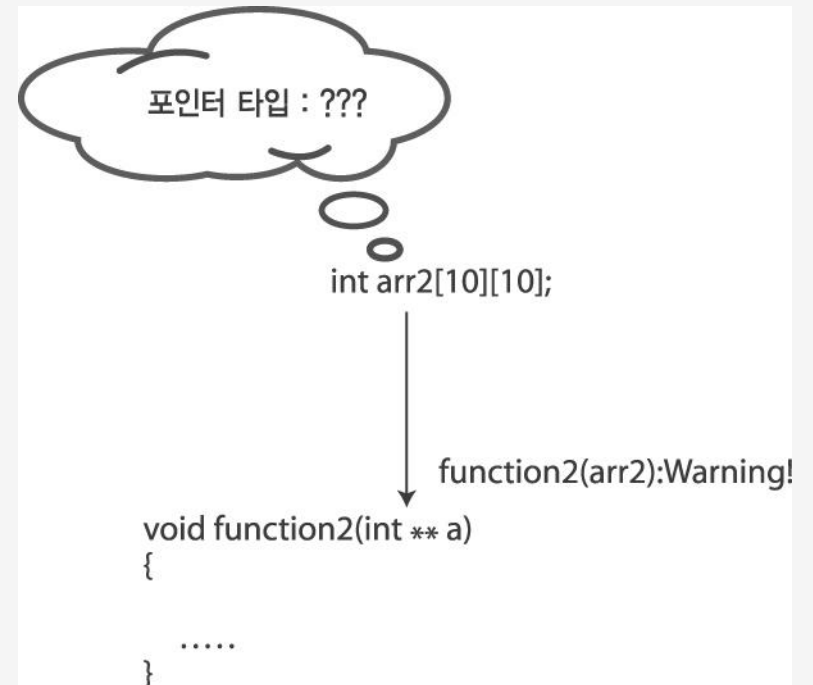
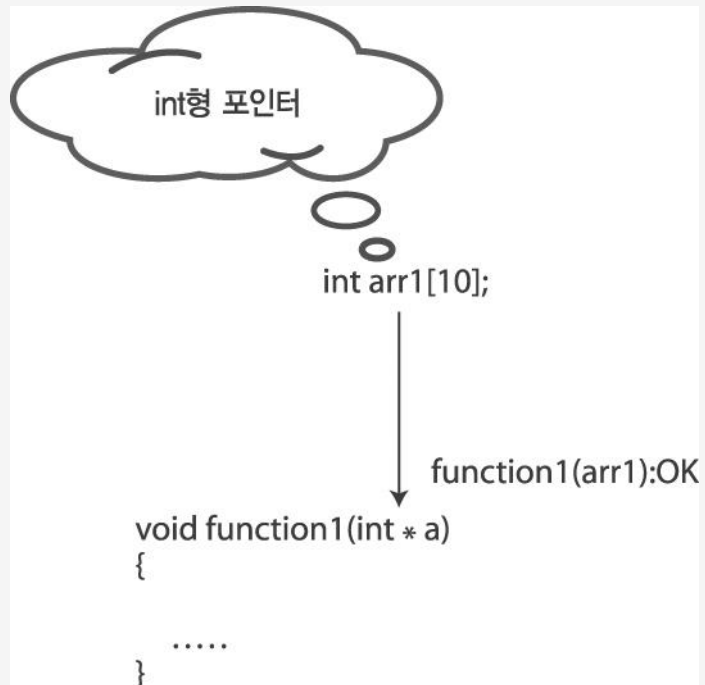
- 3차원 배열의 선언과 의미
  - 3차원적 메모리 구조를 의미함
  - 개념만 이해하면 충분, 일반적으로 필요 없다.
  - 4차원 이상의 배열은 4차원의 형태가 되므로 구조적인 이해 불가!!

```
int  
a[3][3][3]
```



## 12장. 다차원 배열 그리고 포인터

# C/C++



- 1차원 배열 이름의 포인터 타입 결정 포인트!
  - 포인터가 가리키는 요소의 자료형
  - 포인터 연산 시 증가하는 바이트의 크기
- 1차원 배열 이름
  - 배열 이름이 가리키는 요소의 자료형이 일치 한다면, 포인터 연산 시 증가하는 값의 크기도 일치.
  - 따라서 1차원 배열 이름의 경우 가리키는 요소만 참조.

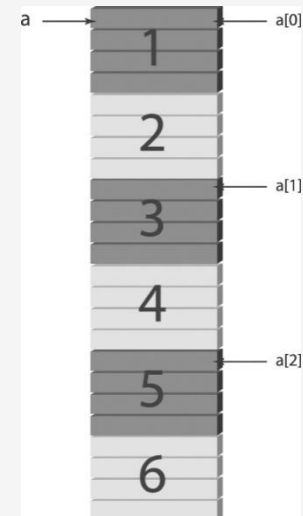
- 다차원 배열의 포인터 타입 결정 포인트!
  - 포인터가 가리키는 요소의 자료형
  - 포인터 연산 시 증가하는 바이트의 크기
- 2차원 배열 이름
  - 포인터가 가리키는 요소의 자료형이 같아 해도 포인터 연산 시 증가하는 값의 크기 불일치!
  - 포인터 연산 결과도 생각해 봐야 함.

- 2차원 배열 이름의 특성 이해(1단계)

```
/* two_array1.c */
#include <stdio.h>
int main(void)
{
    int a[3][2]={1, 2, 3, 4, 5, 6};

    printf("a[0] : %d \n", a[0]);
    printf("a[1] : %d \n", a[1]);
    printf("a[2] : %d \n", a[2]);
    printf("a   : %d \n", a);

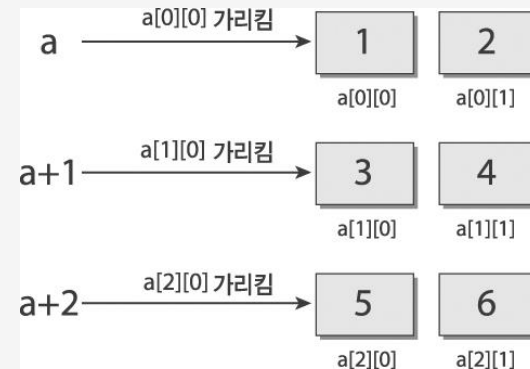
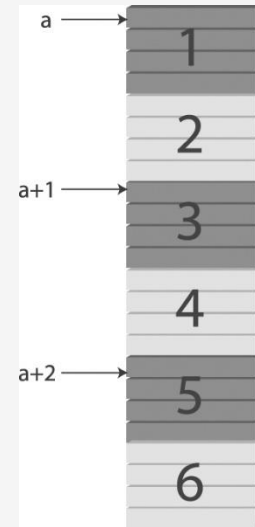
    return 0;
}
```



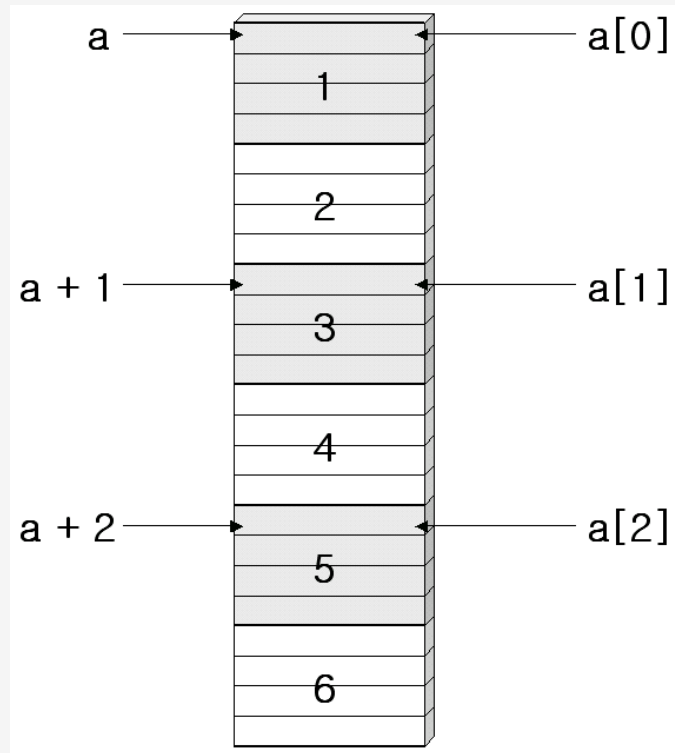


- 2차원 배열 이름의 특성 이해(2단계)

```
/* arr2_name.c */  
#include <stdio.h>  
int main(void)  
{  
    int a[3][2]={1, 2, 3, 4, 5, 6};  
    printf("a   : %d \n", a );  
    printf("a+1 : %d \n", a+1);  
    printf("a+2 : %d \n", a+2);  
    return 0;  
}
```



- 2차원 배열 이름의 특성 이해(결론1)



- 2차원 배열 이름의 특성 이해(결론2)

- 2차원 배열 이상의 포인터 타입 구성
  - 가리키는 대상의 자료형
  - 포인터 연산 시 증가하는 바이트의 크기

```
/* arr2_ptr.c */
#include <stdio.h>
int main(void)
{
    int arr1[3][2];

    printf("arr1   : %d \n", arr1);
    printf("arr1+1 : %d \n", arr1+1);
    printf("arr1+2 : %d \n", arr1+2);
    return 0;
}
```

```
/* arr2_ptr.c */
#include <stdio.h>
int main(void)
{
    int arr2[2][3];

    printf("arr2   : %d \n", arr2);
    printf("arr2+1 : %d \n", arr2+1);
    printf("arr2+2 : %d \n", arr2+2);
    return 0;
}
```

- 2차원 배열 이름에 일치하는 포인터 선언

```
int arr[2][4];
```



```
int (*pArr)[4];
```

```
int (*pArr) [4];
```

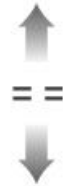
4칸씩 건너 뛴다

pArr은 포인터!

int형 변수를 가리킨다

- 매개 변수로 선언되는 포인터의 또 다른 표현

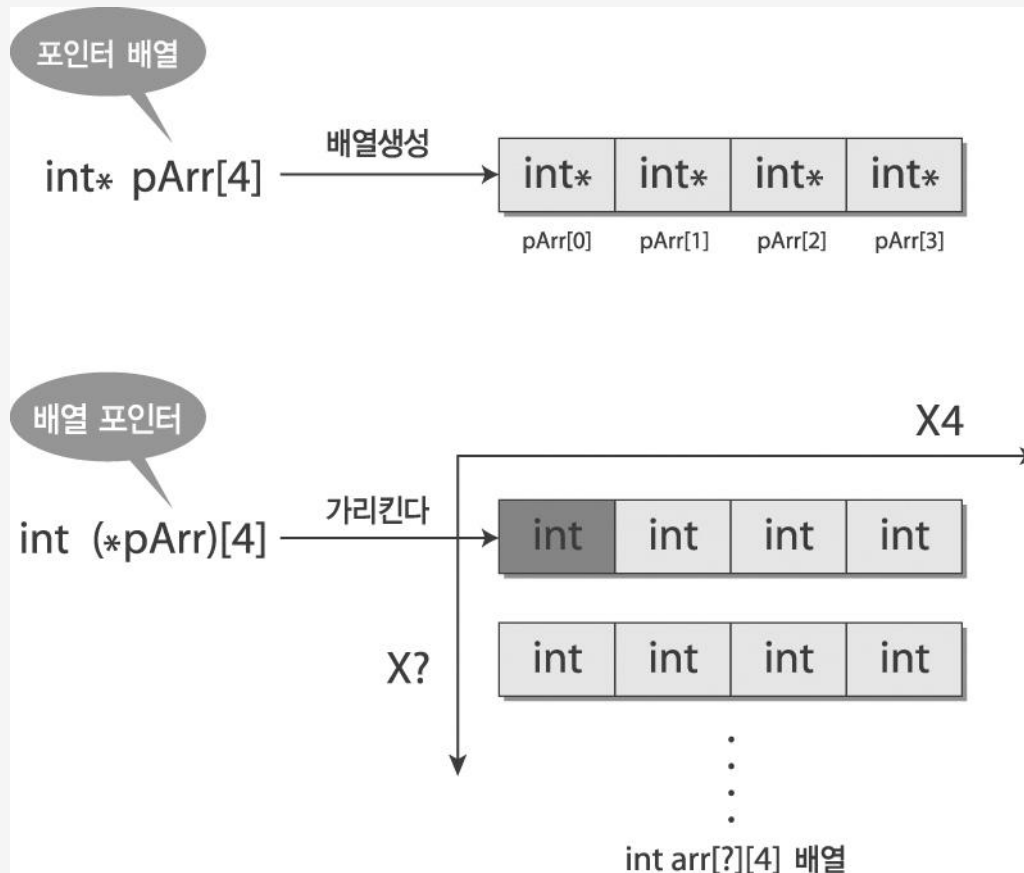
```
void show_data(int (*ptr)[4], int a);
```



```
void show_data(int ptr[ ][4], int a);
```

```
int main (void)
{
    int arr1[2][4]={1, 2, 3, 4, 5, 6, 7, 8};
    int (*ptr1)[4]=arr1;           // OK
    int ptr2[][4]=arr1;            // ERROR
    . . . . .
```

- “int (\*pArr)[4]” 과 “int\* pArr[4]”의 차이점



- 다양한 형태의 배열 요소 접근 방법

```
/* two_array2.c */
#include <stdio.h>
int main(void)
{
    int a[3][2]={ {1, 2}, {3, 4}, {5, 6} };

    printf("a[0]   : %d \n", a[0]);
    printf("*a+0 : %d \n", *(a+0));

    printf("a[1]   : %d \n", a[1]);
    printf("*a+1 : %d \n", *(a+1));

    printf("a[2]   : %d \n", a[2]);
    printf("*a+2 : %d \n", *(a+2));

    printf("%d, %d \n", a[1][0], (*(a+1))[0]);
    printf("%d, %d \n", a[1][2], *(a[1]+2));
    printf("%d, %d \n", a[2][1], *(*a+2)+1);
    return 0;
}
```

- 문1) scanf 함수를 이용하여 문자열을 입력후 해당 문자열을 한 글자씩 공백을 삽입하여 출력하시오.
- 문2) scanf 함수를 이용하여 문자열을 입력후 getch() 함수를 이용하여 숫자 키를 누르면 해당 위치의 문자를 출력하시오



- 문3) 배열을 이용하여 아래 모양과 같이 출력하십시오.

[illegible]

## → 함수/최대값/배열의 크기

# C/C++

- 문) 두 과목의 성적이 다음과 같을 때 배열을 이용하여 초기화하고, 각각의 성적에 가중치를 곱한 후 개인별 합계를 구하여 합이 큰 순서대로 정렬하여 출력하시오.

자료(예)

이름	A	B	C	D	E	F	G	국어	82	93	71	69
78	84	75	---	가중치	0.3							
영어	76	91	67	73	86	63	83	---	가중치	0.7		

출력(예시1)

1 :	91.60
2 :	83.60
3 :	80.60
4 :	77.80
5 :	71.80
6 :	69.30
7 :	68.20

(예시2)

순위:	이름	국어	영어	합계
1:	B	93	91	91.60
2:	E	78	86	83.60
3:	G	75	83	80.60
4:	A	82	76	77.80
5:	D	69	73	71.80
6:	F	84	63	69.30
7:	C	71	67	68.20