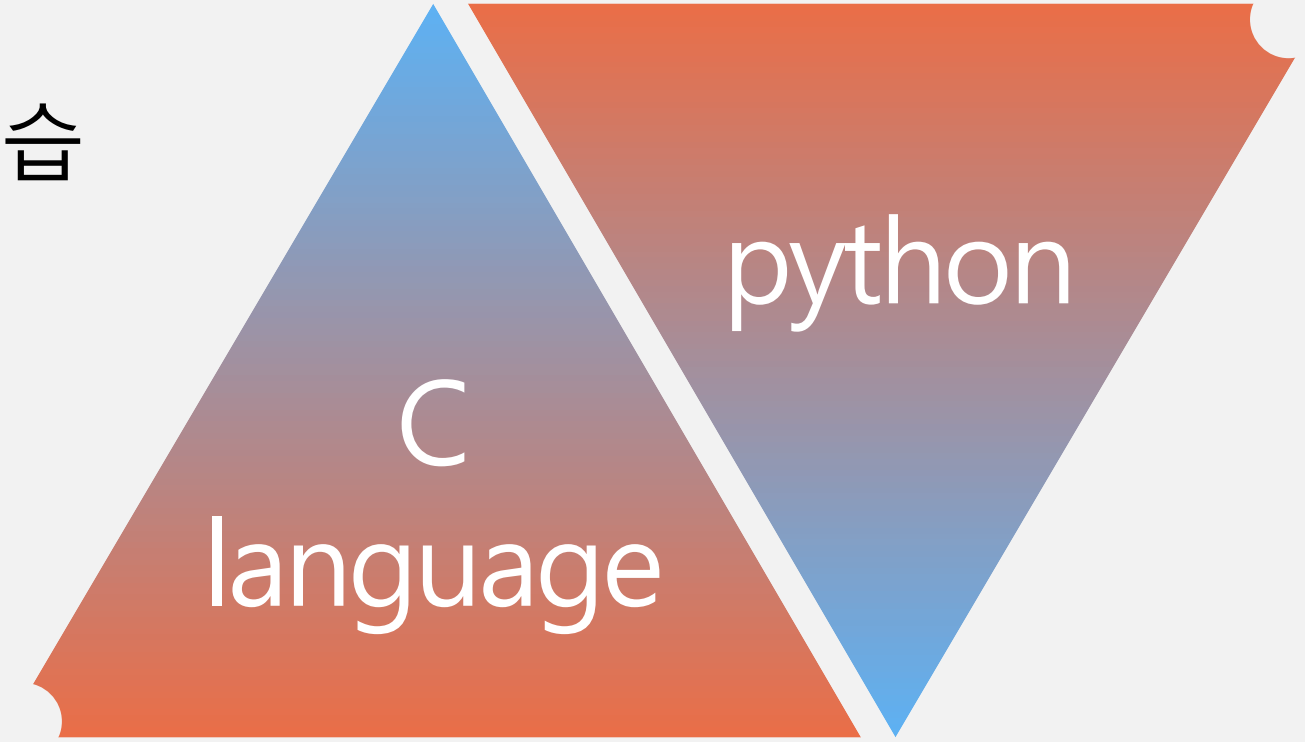
소입설
멘토링 수업
(10주차)

-이현민 멘토

CONTENTS

1. 지난 주 수업 복습



10L. 선언

```
int number;
float rate;
char choice;

int func();

[Specifier] [Declarator] = [Initializer] ;
double pi, rate = 3.5;
int ;

[hm]
int* num1, num2;
```

C01_Hello_World

VS vs gcc

C01_Hello_World

UNIX 명령어

- ls : 현재 디렉토리의 파일 검색 (**lis**t)
- mkdir : 디렉토리 만들기 (**mak**e **dir**ectory)
- cd : 특정 디렉토리로 이동 (**c**hange **d**irectory)
- rm : 파일 삭제 (**r**e**m**ove)
- cat : 파일 내용 출력 (con**cat**enate)
- find : 파일 검색
- grep : 특정 문자 검색
- vim : 에디터(VS와 비슷), 주로 vi로 사용
- gedit : 에디터(메모장과 비슷)

C02. Guess the Number

```
#include<stdlib.h>
        → rand()
        → srand(unsigned int seed)
#include<time.h>
        → time(time_t *timeptr)
```

C03. Dragon Realm

```
#include <string.h>
        → strcmp(char[], char[])

python:          a and b          a or b
C:               a && b           a || b

#include <windows.h>
        → Sleep(1000)            :Windows
#include <unistd.h>
        → usleep(1000000)        :UNIX
```

C05. Operators

```
#define CUBE(x) x*x*x
void main(){
        printf(“%d”,CUBE(2+2));
}
```

C05. Operators

- **Data Objects**
  a region of data storage that can be used to hold values
- **L values**
  a name or expression that identifies a particular Data Object
- **R values**
  quantities that can be assigned to modifiable L values
- **Operands**
  Operands are what operator operate on
- **Unary and binary operators**
  ex) 36-12
  ex) -16
  ex) –(12-20)

C05. Operators

```
// Operator Precedence
    ① ()                →
    ② + - (unary)       ←
    ③ * /               →
    ④ + - (binary)      →
    ⑤ =                 ←
```

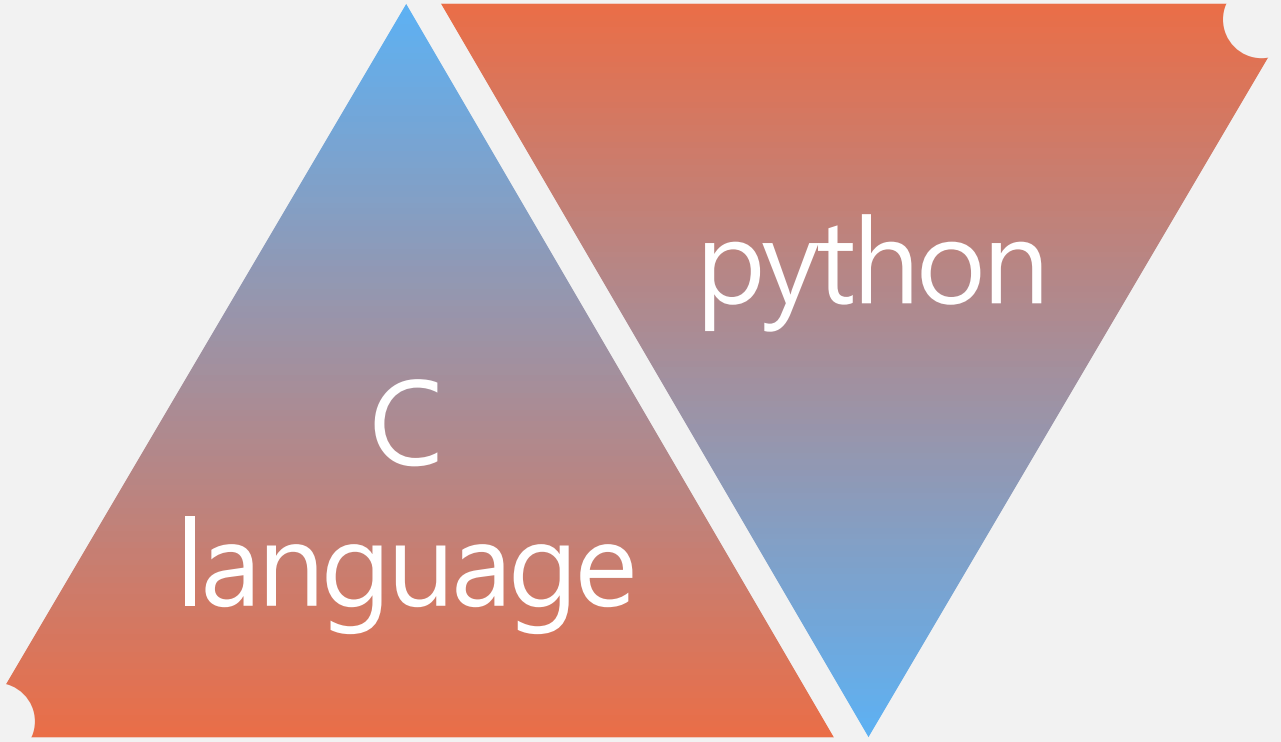
C05. Operators

```
// Other operators
    • sizeof: 메모리칸에 할당되는
              bytes 수
    • %: 나머지
    • ++, --: x=x+1, x=x-1
              postfix, prefix!
```

C05. Operators

```
// Compound Statements (Block)

<fragment 1>
index=0;
while (index++<10)
sam=10*index+2;
printf(“sam=%d\n”,sam);

<fragment 2>
index=0;
while (index++<10){
    sam=10*index+2;
    printf(“sam=%d\n”,sam);
}
```

2. Pointer and Array



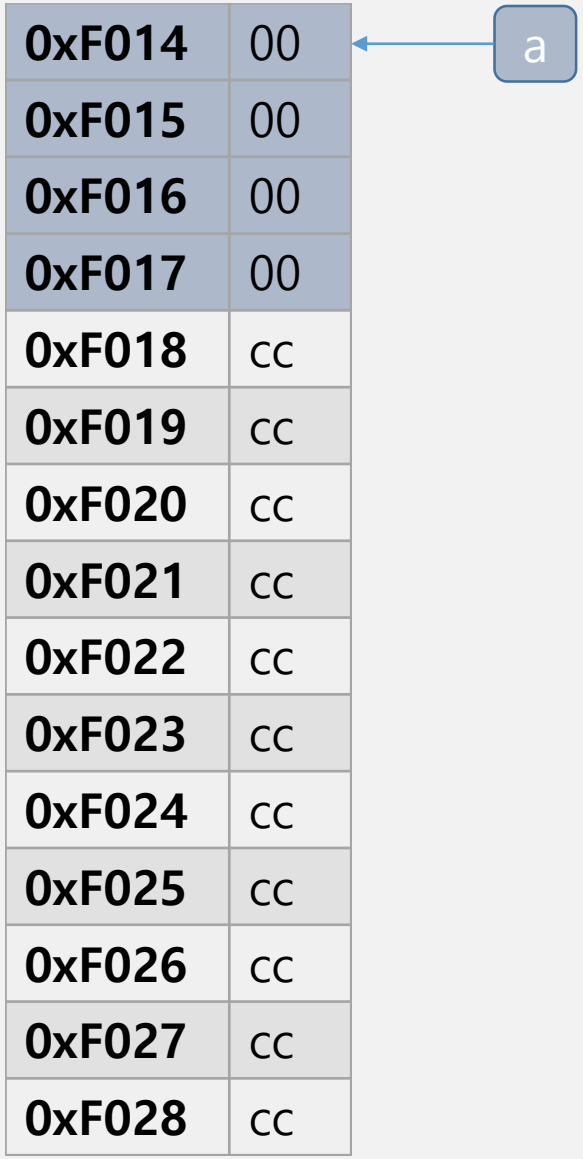
Pointer

1 Byte = 8 bits
00000000
~
11111111

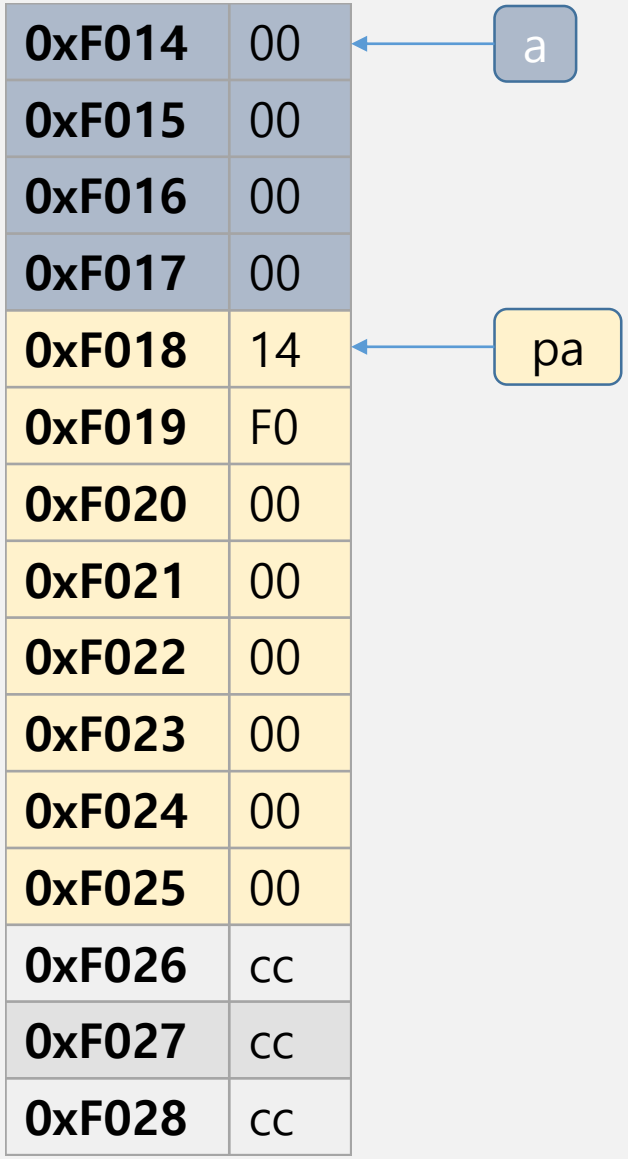
0xF014	cc
0xF015	cc
0xF016	cc
0xF017	cc
0xF018	cc
0xF019	cc
0xF020	cc
0xF021	cc
0xF022	cc
0xF023	cc
0xF024	cc
0xF025	cc
0xF026	cc
0xF027	cc
0xF028	cc

int a;

int: 4Bytes(32bits)



```
int a;

int* pa=&a;
```


0xF014 00
0xF015 00
0xF016 00
0xF017 00
0xF018 14
0xF019 F0
0xF020 00
0xF021 00
0xF022 00
0xF023 00
0xF024 00
0xF025 00
0xF026 cc
0xF027 cc
0xF028 cc
a
pa

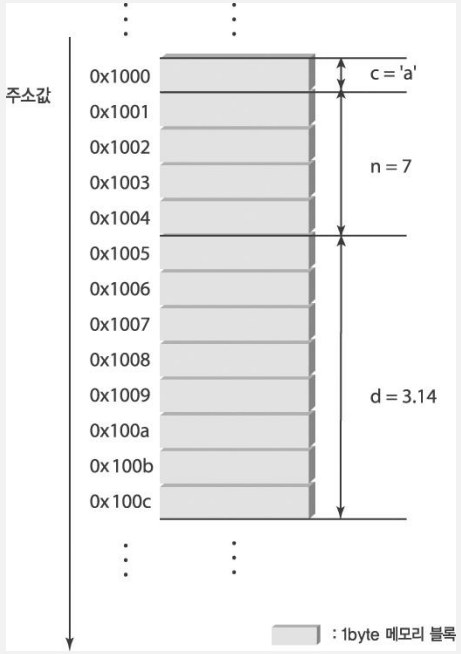
포인터와 포인터 변수

메모리의 주소 값을 저장하기 위한 변수

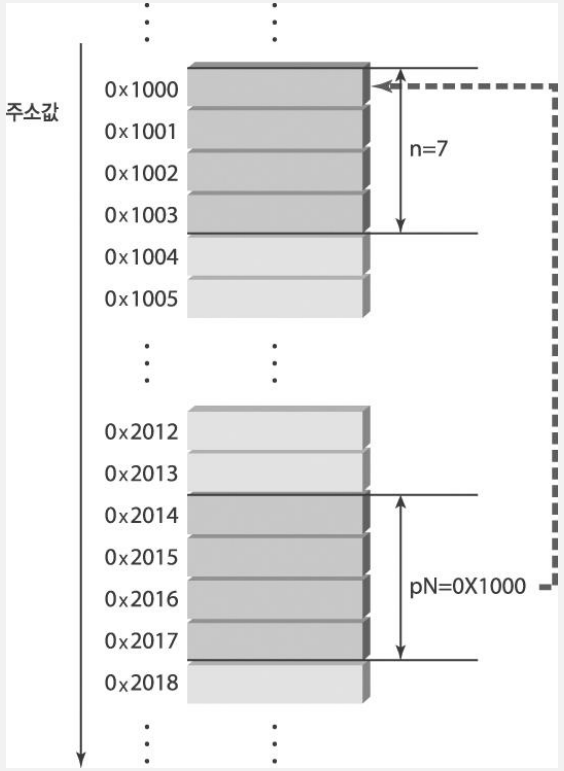
"포인터"를 흔히 "포인터 변수"라 한다.

주소 값과 포인터는 다른 것이다.

```
int main(void)
{
    char c='a';
    int n=7;
    double d=3.14;
    . . . . .
```



- 그림을 통한 포인터의 이해
  - 컴퓨터의 주소 체계에 따라 크기가 결정
  - 32비트 시스템 기반 : 4 바이트

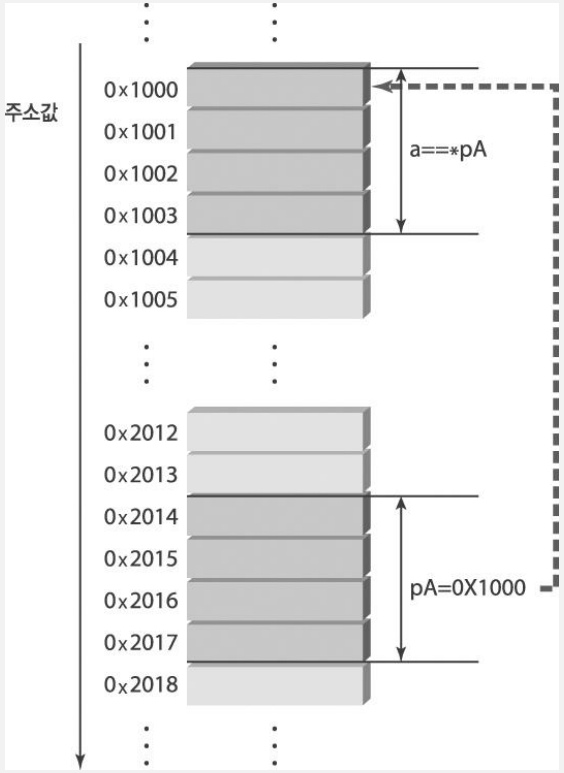


- 포인터의 타입과 선언
  - 포인터 선언 시 사용되는 연산자 : *
  - A형 포인터(A*) : A형 변수의 주소 값을 저장

```
int main(void)
{
   int *a;              // a라는 이름의 int형 포인터
   char *b;             // b라는 이름의 char형 포인터
   double *c;// c라는 이름의 double형 포인터
    . . . . .
```

- 주소 관련 연산자
  - & 연산자 : 변수의 주소 값 반환
  - * 연산자 : 포인터가 가리키는 메모리 참조

```
int main(void)
{
   int a=2005;
   int *pA=&a;
   printf("%d", a);   //직접 접근
   printf("%d", *pA);   // 간접 접근
   . . . . .
```



```
#include <stdio.h>

int main(void)
{
    int a=2005;
    int* pA=&a;

    printf("pA : %d ₩n", pA);
    printf("&a : %d ₩n", &a);

    (*pA)++;            //a++와 같은 의미를 지닌다.

    printf("a    : %d ₩n", a);
    printf("*pA : %d ₩n", *pA);

    return 0;
}
```

- 포인터에 다양한 타입이 존재하는 이유
  - 포인터 타입은 참조할 메모리의 크기 정보를 제공

```
#include <stdio.h>
 int main(void)
 {
     int a=10;
     int *pA = &a;
     double e=3.14;
     double *pE=&e;

     printf("%d %f", *pA, *pE);
     return 0;
 }
```

- 사례 1

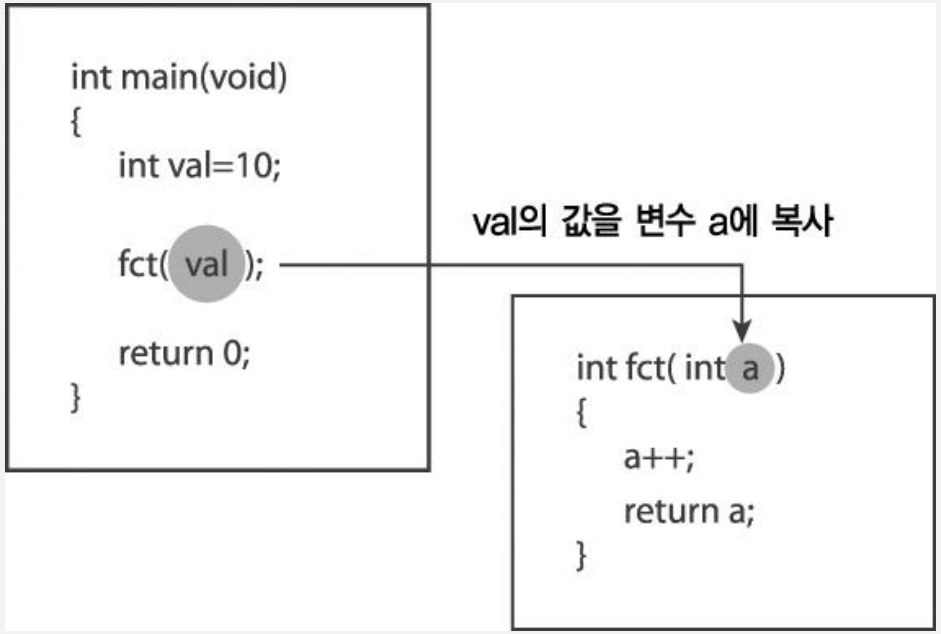
```
int main(void)
{
   int *pA;       // pA는 쓰레기 값으로 초기화 됨
   *pA=10;
   return 0;
}
```

- 사례 2

```
int main(void)
{
   int* pA=100;   // 100이 어딘 줄 알고???
   *pA=10;
   return 0;
 }
```

기본적인 인자의 전달 방식

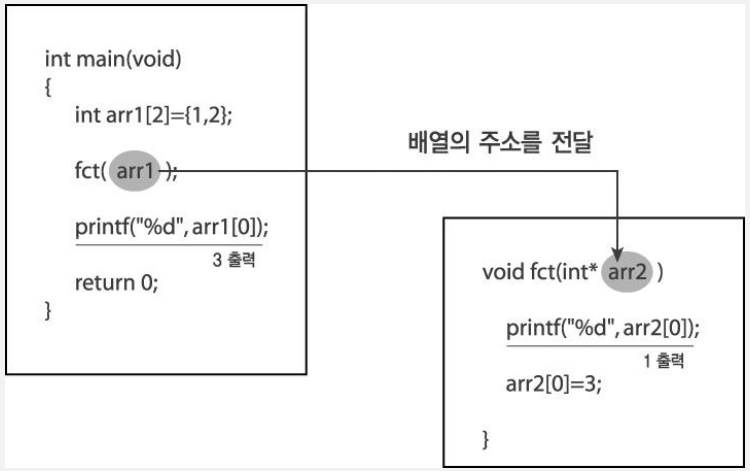
값의 복사에 의한 전달

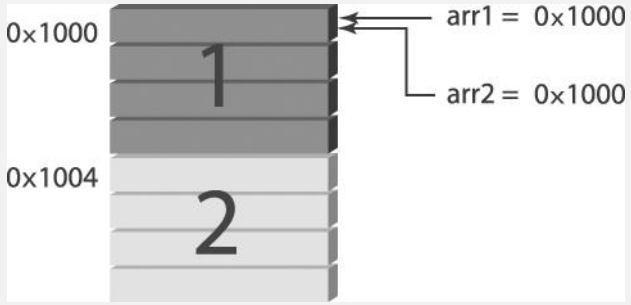


배열의 함수 인자 전달 방식

배열 이름(배열 주소, 포인터)에 의한 전달

```
#include <stdio.h>
void fct(int *arr2);

int main(void)
{
    int arr1[2]={1, 2};

    fct(arr1);
    printf("%d ₩n", arr1[0]);
    return 0;
}
void fct(int *arr2)
{
    printf("%d ₩n", arr2[0]);
    arr2[0]=3;
}
```





"int * pArr" vs. "int pArr[ ]"

둘 다 같은 의미를 지닌다.

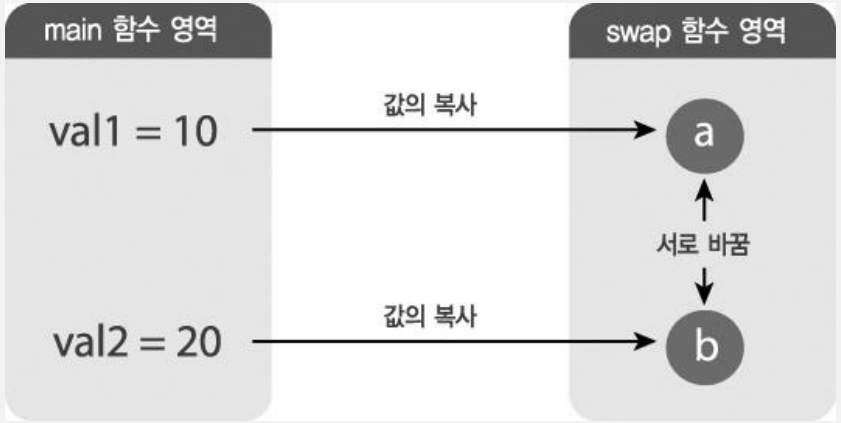
선언 "int pArr[ ]"은 함수의 매개 변수 선언 시에만 사용 가능

좀 있다 다시 살펴봄

```
int function(int pArr[])
{
   int a=10;
   pArr=&a;          // pArr이 다른 값을 지니게 되는 순간
   return *pArr;
}
```

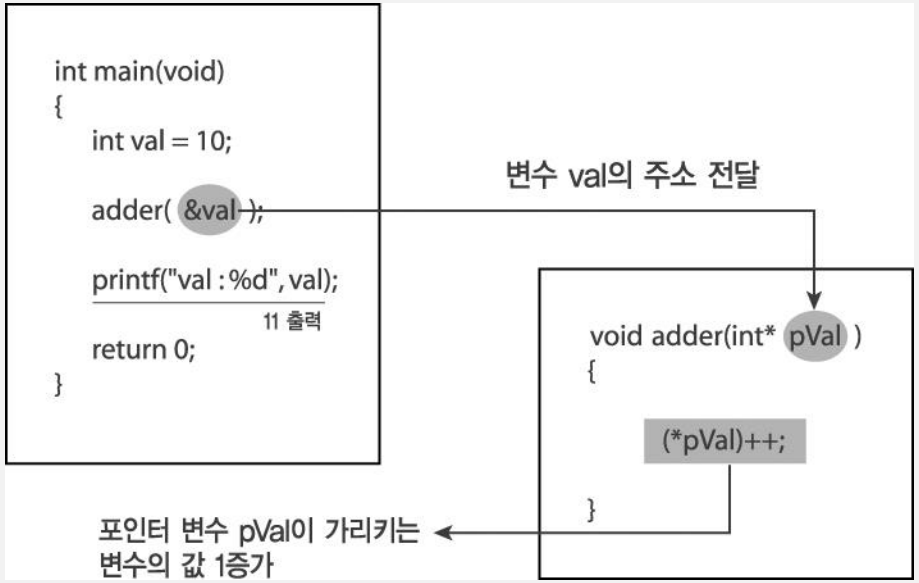
int parameter에 의한 swap

```
int main(void)
{
    int val1=10;
    int val2=20;
    swap(val1, val2);

    printf("val1 : %d ₩n", val1);
    printf("val2 : %d ₩n", val2);
    return 0;
}
void swap(int a, int b)
{
    int temp=a;
    a=b;
    b=temp;

    printf("a : %d ₩n", a);
    printf("b : %d ₩n", b);
}
```



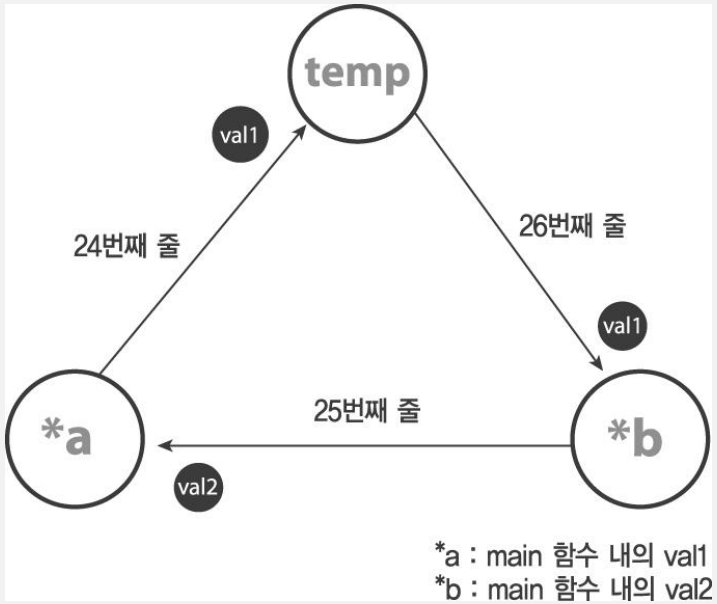
Pointer parameter, address argument

주소값을 인자로 전달하는 형태의 함수 호출



pointer parameter에 의한 swap

```
int main(void)
{
    int val1=10;
    int val2=20;
    printf("Before val1 : %d ₩n", val1);
    printf("Before val2 : %d ₩n", val2);
    swap(&val1, &val2);      //val1, val2 주소 전달

    printf("After val1 : %d ₩n", val1);
    printf("After val2 : %d ₩n", val2);
    return 0;
}
void swap(int* a, int* b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
```



scanf 함수 호출 시 &를 붙이는 이유

case 1

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

case 2

```
int main(void)
{
    char str[100];
    printf("문자열 입력 : ");
    scanf("%s", str);
    . . . . .
```

일반 변수의 상수화

```
const int a = 10;
// int const a = 10; (same)

a=30            // Error!
```

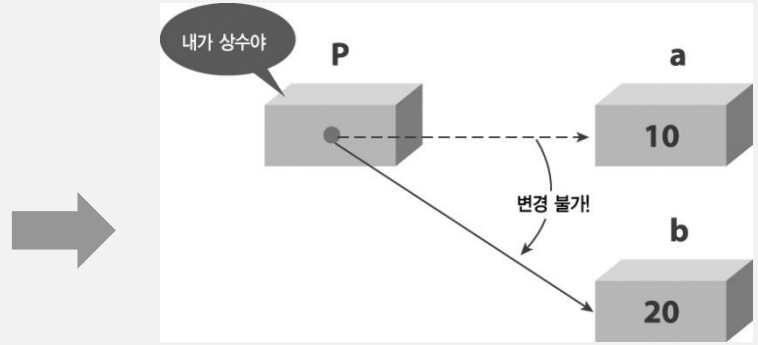
const 키워드 사용

포인터가 가리키는 변수의 상수화

```
int a = 10;
const int * p = &a;
*p=30            // Error!
a=30             // OK!
```

포인터 상수화

```
int a=10;
int b=20;
int * const p = &a;
p=&b             // Error!
*p=30            // OK!
```



const 키워드를 사용하는 이유

컴파일 시 잘못된 연산에 대한 에러 메시지

프로그램을 안정적으로 구성

```
#include <stdio.h>
float PI=3.14;

int main(void)
{
    float rad;
    PI=3.07;      // 분명히 실수!!

    scanf("%f", &rad);
    printf("원의 넓이는 %f ₩n", rad*rad*PI);
    return 0;
}
```

```
#include <stdio.h>
const float PI=3.14;

int main(void)
{
    float rad;
     PI=3.07;     // Compile Error 발생!

    scanf("%f", &rad);
    printf("원의 넓이는 %f ₩n", rad*rad*PI);
    return 0;
}
```

Array

둘 이상의 변수를 동시에 선언하는 효과를 지닌다.

많은 양의 데이터를 일괄적으로 처리해야 하는 경우에 유용하다.

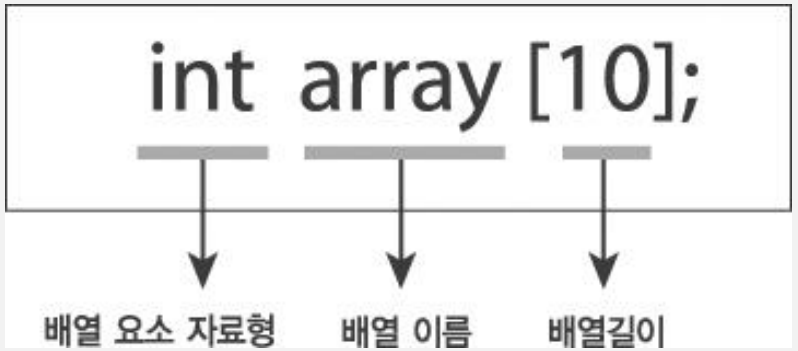
지역적 특성을 지닐 수도 있고, 전역적 특성을 지닐 수도 있다.

배열 선언에 있어서 필요한 것 세 가지

배열 길이 : 배열을 구성하는 변수의 개수
(반드시 상수를 사용)

배열 요소 자료형 : 배열을 구성하는 변수의 자료형

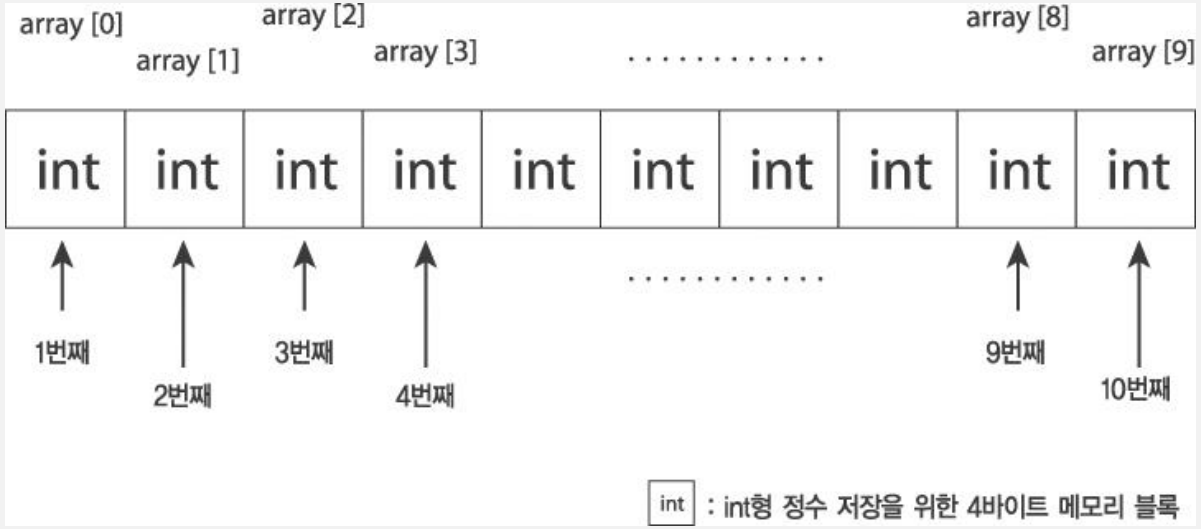
배열 이름 : 배열에 접근할 때 사용되는 이름



1차원 배열의 접근

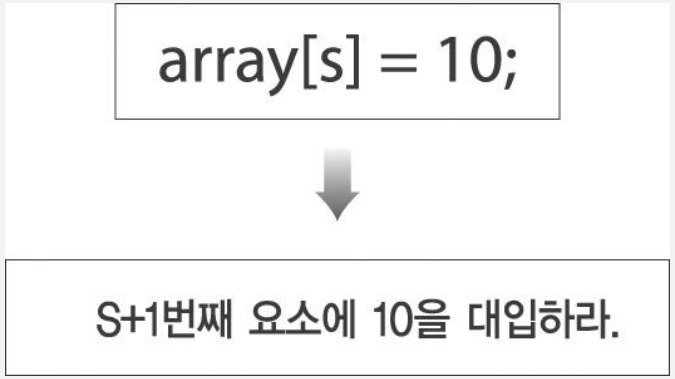
배열 요소의 위치를 표현 : 인덱스(index)

인덱스는 0에서부터 시작



- 배열 선언과 접근의 예

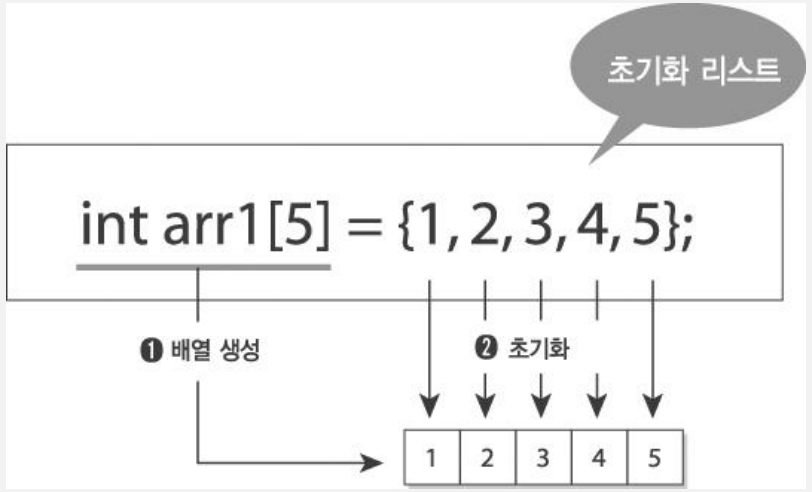
```
int main(void)
{
    int array[10];      // 배열 선언
    array[0]=10;        // 첫 번째 요소 접근
    array[1]=20;        // 두 번째 요소 접근
    array[2]=30;        // 세 번째 요소 접근
    . . . . .
    return 0;
}
```

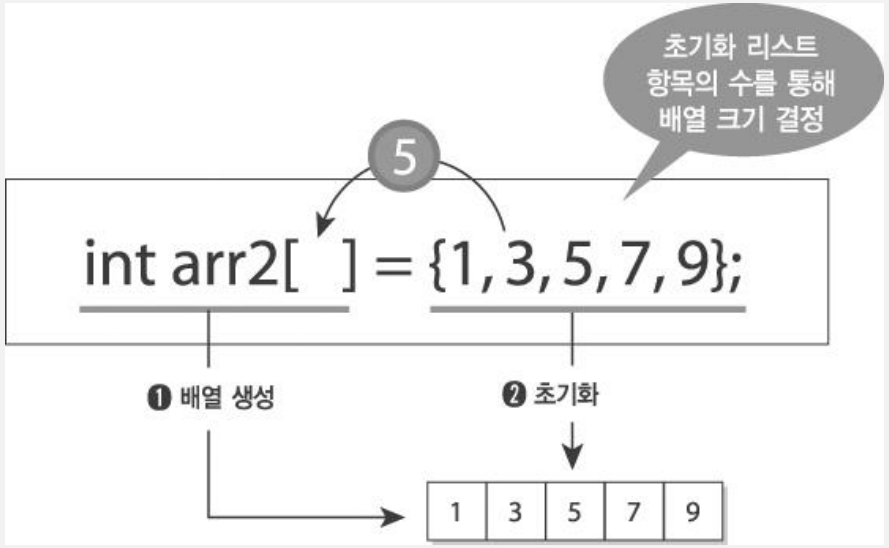


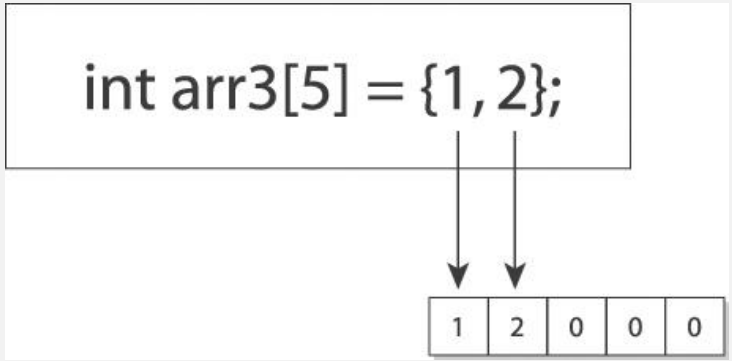
```
#include <stdio.h>

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}
}
```




초기화 리스트
항목의 수를 통해
배열 크기 결정
5
int arr2[ ] = {1, 3, 5, 7, 9};
❶ 배열 생성
❷ 초기화
1
3
5
7
9


int arr3[5] = {1, 2};
1
2
0
0
0

- 문자열 상수
  - 문자열이면서 상수의 특징을 지닌다.

```
printf("Hello World! ₩n");
```

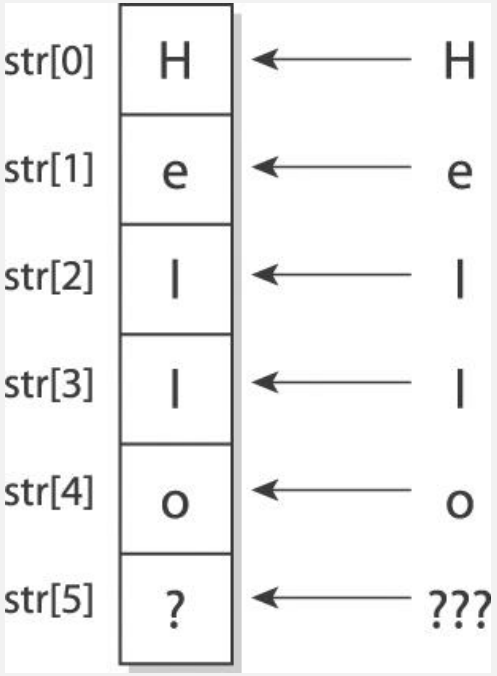
- 문자열 변수
  - 문자열이면서 변수의 특징을 지닌다.

```
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);
    . . . . .
```

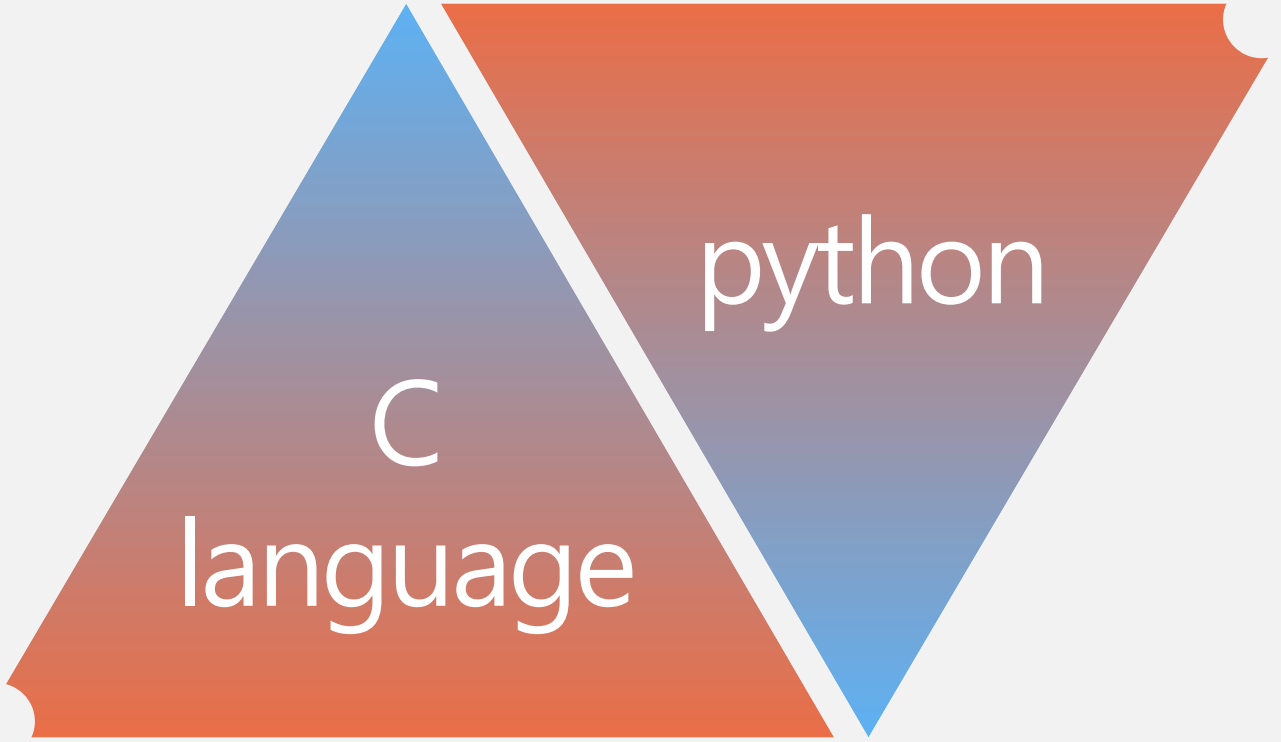
- 문자열 변수를 활용한 다양한 예제
  - va_str.c : 문자열 변수의 데이터 조작
  - scanf_str.c : 문자열 변수를 통한 문자열의 입력

- 문자열과 char형 배열의 차이점

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

```
#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;
}
```

3. C codes



```
/* code01: swap by value */

int swap( int operand1, int operand2 ) ;

int main(){
        int num1=10, num2=-3;
        printf("num1=%d\nnum2=%d\n"
                ,num1, num2);
        swap(/*TODO: 인자?*/);
        printf("num1=%d\nnum2=%d\n"
                ,num1, num2);
        return 0;
}

int swap( int operand1, int operand2 ){
        /* TODO: swap함수 정의 */
}
```

```
/* code02: swap by address */

int swap( int* operand1, int* operand2 ) ;

int main(){
        int num1=10, num2=-3;
        printf("num1=%d\nnum2=%d\n"
                ,num1, num2);
        swap(/*TODO: 인자?*/);
        printf("num1=%d\nnum2=%d\n"
                ,num1, num2);
        return 0;
}

int swap( int* operand1, int* operand2 ){
        /* TODO: swap함수 정의 */
}
```

```c
/* code03: loop */
int main(){
        int num[50];
        /* for loop 나
         * while loop 를 이용해서
         * num Array에 50개의
         * 랜덤값(1~500)을 집어넣은 후
         * num Array의 원소들을 출력하고
         * 주소값을 출력해보기 */

        return 0;
}
```

복습