

# 컴퓨터 프로그래밍 및 실습

9주차. 포인터

# 실습 안내

## ■ 실습 제출 안내

- 솔루션 이름은 “Practice week 9”
- 프로젝트 이름과 소스코드 이름은 Problem1, Problem2, ...
  - 실습1의 프로젝트 이름은 Problem1, 소스코드 이름은 problem1.c
  - 실습 2의 프로젝트 이름은 Problem2, 소스코드 이름은 problem2.c ...
- 솔루션 폴더를 압축하여 Practice\_week9\_학번\_이름.zip 으로 제출
- 제출기한: 당일 19시 까지

# 1. 포인터 변수

## ■ 포인터 변수

- int 변수는 정수를 담는 변수
- double 변수는 배정밀도 부동소수점 실수를 담는 변수
- 포인터 변수는 “메모리의 주소 값을 담는 변수”

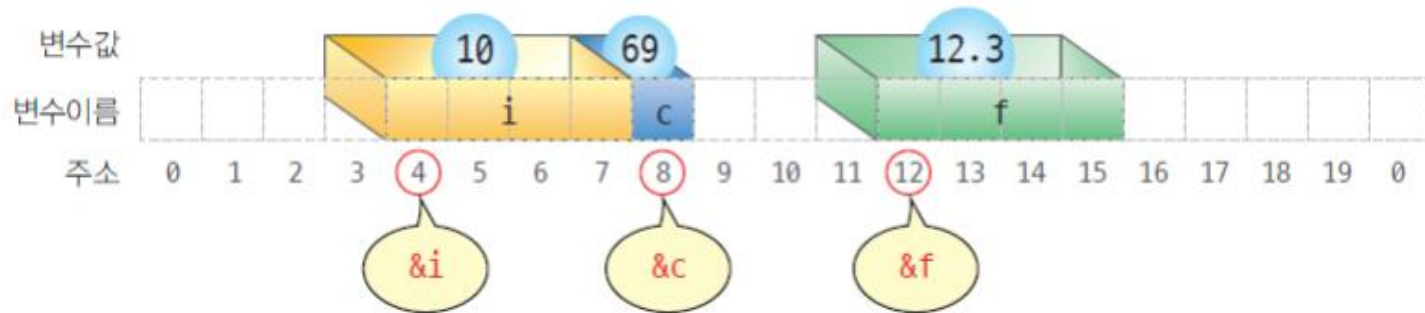
Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011

Address	Hex dump	ASCII	Address	Value	ASCII
0028FEA4	00 00 00 00 C8 00 00 00	È	0028FE90	00001000	+
0028FEAC	64 00 00 00 30 15 48 00	d 0+H	0028FE94	2CEB57B6	qwe,
0028FEB4	19 00 00 00 84 FF 28 00	† „ÿ(	0028FE98	0028FEB8	,b(
0028FEB8	EE 13 40 00 01 00 00 00	ì  @	0028FE9C	00401F6E	n @
0028FEC4	70 15 48 00 18 1F 48 00	p+H † H	0028FEA0	00401F10	† @
0028FEC8	00 00 00 00 00 00 00 00		0028FEA4	00000000	
0028FED4	00 00 00 00 00 00 00 00		0028FEA8	000000C8	È
0028FEDC	00 00 00 00 CC CC CC CC	iiii	0028FEAC	00000064	d
0028FEE4	CC CC CC CC CC CC CC CC	iiiiiiii	0028FEB0	00481530	0+H
0028FEE8	CC CC CC CC CC CC CC CC	iiiiiiii	0028FEB4	00000019	†
0028FEF4	CC CC CC CC CC CC CC CC	iiiiiiii	0028FEB8	0028FF84	„ÿ(
0028FEFC	CC CC CC CC 00 00 00 00	iiii	0028FEB8	004013EE	ì  @
0028FF04	00 00 00 00 00 00 00 00		0028FEC0	00000001	
0028FF08	00 00 00 00 08 00 00 00	0	0028FEC4	00481570	p+H
0028FF14	01 00 00 00 70 15 48 00	p+H	0028FEC8	00481F18	† H
0028FF1C	6F 45 D1 01 C8 AF D1 8A	oEÑ È ÑŠ	0028FECC	00000000	
0028FF24	6F 45 D1 01 00 00 00 00	oEÑ	0028FED0	00000000	
0028FF2C	00 00 00 00 00 00 00 00		0028FED4	00000000	
0028FF34	00 00 00 00 00 00 00 00		0028FED8	00000000	

# 1. 포인터 변수

## ❖ 변수의 주소

- 변수의 주소를 계산하는 연산자: &
- 변수 i의 주소: &i



```
int main(void)
{
    int i = 10;
    char c = 69;
    float f = 12.3;

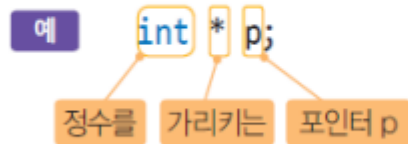
    printf("i의 주소: %u\n", &i); // 변수 i의 주소 출력
    printf("c의 주소: %u\n", &c); // 변수 c의 주소 출력
    printf("f의 주소: %u\n", &f); // 변수 f의 주소 출력
    return 0;
}
```

i의 주소: 1245024  
c의 주소: 1245015  
f의 주소: 1245000

# 1. 포인터 변수

## ■ 포인터 변수의 선언

Syntax: 포인터 선언



- 포인팅 할 주소 값이 담고 있는 데이터의 타입과 함께 선언되어야 한다.
- 예시)
  - int 변수를 포인팅하는 포인터의 타입: int\*
  - double 변수를 포인팅하는 포인터의 타입: double\*
  - char 변수를 포인팅하는 포인터의 타입: char\*

# 1. 포인터 변수

## ■ 포인터 변수의 선언

```
#include <stdio.h>

int main(void)
{
    int a = 5;
    char b = 'k';

    int* address_of_a = &a; // 변수 a의 주소 값이 담긴 상태
    char* address_of_b = &b; // 변수 b의 주소 값이 담긴 상태

    double* address_of_some_variable = NULL; // NULL은 아무 주소 값도 포인팅 하고 있지 않는 상태

    printf("%p", address_of_a); // %p는 포인터(주소 값)를 출력하는 포맷

    return 0;
}
```

# 1. 포인터 변수

- 지난 시간에 배운 내용을 확인해보자
  - 배열은 메모리 공간에 연속적으로 할당?

	address	arr[5]	
arr	→ 1024	1	arr[0]
	1028	2	arr[1]
	1032	3	arr[2]
	1036	4	arr[3]
	1040	5	arr[4]

```
#include <stdio.h>

int main(void) {
    int arr[10] = { 0 };

    printf("%p %p\n", &arr[0], &arr[1]);

    printf("%d\n", (int)&arr[1] - (int)&arr[0]);
    return 0;
}
```

- 결과값: 4
- int형 변수는 4byte의 크기를 가지고 있다.

Microsoft Visual Studio 디버그 콘솔

```
00000001F693F778 00000001F693F77C
4
```

# 1. 포인터 변수

## ■ 실습 1

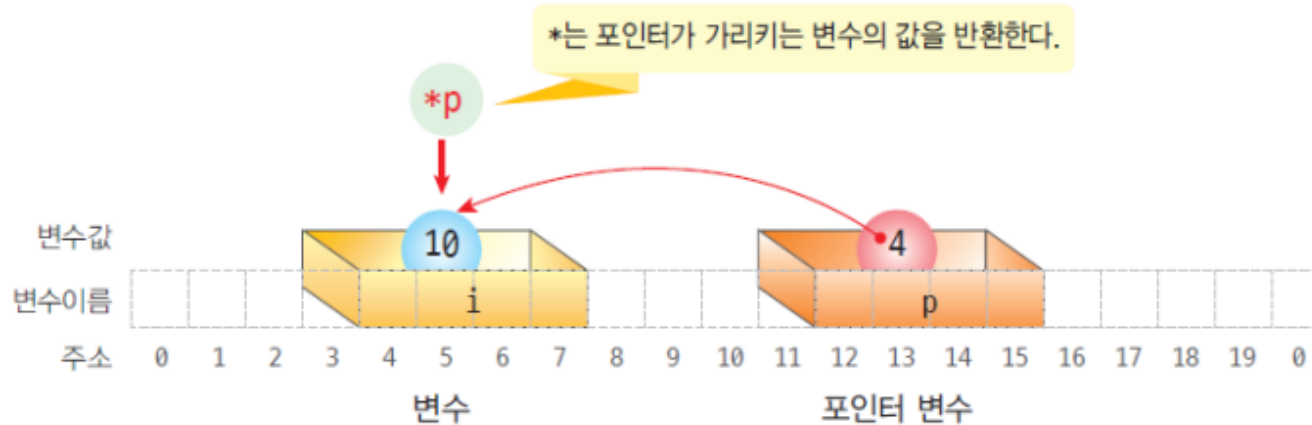
- 포인터 변수의 크기는 몇 byte일까?
- `int*`과 `double*`, `char*` 포인터 변수는 각각 크기가 다를까?
- 위 내용을 증명할 수 있는 프로그램을 개발해보자



# 1. 포인터 변수

❖ 간접 참조 연산자 \*: 포인터가 가리키는 값을 가져오는 연산자

```
int i = 10;  
  
int* p;  
p = &i;  
  
printf("%d \n", *p);
```



# 1. 포인터 변수

## ■ 간접 참조 연산자 – 단순 참조

```
#include <stdio.h>

int main(void) {
    int a = 5;

    int* pa = &a;

    int k = *pa;    // 간접 참조 연산자 *을 사용하여 포인터가 가르키고 있는 값을 가져옴

    printf("k == pa is %s\n", k == a ? "True" : "False");

    return 0;
}
```

# 1. 포인터 변수

- 간접 참조 연산자 – 값 대입

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

```
int main(void) {  
    int a = 5;  
    int* pa = &a;  
  
    *pa = 41;  
    printf("%d\n", *pa);  
    printf("%d\n", a);  
  
    return 0;  
}
```

```
41  
41
```

# 1. 포인터 변수

## ■ 증감 연산자를 이용한 포인터 접근

### ❖ 포인터 연산

- 덧셈과 뺄셈
- 증가 연산의 경우 증가되는 값은 포인터가 가리키는 객체의 크기

포인터 타입	++연산후 증가되는값
char	1
short	2
int	4
float	4
double	8

# 1. 포인터 변수

## ■ 증감 연산자를 이용한 포인터 접근

### ❖ 포인터 연산

```
#include <stdio.h>
int main(void)
{
    char *pc;
    int *pi;
    double *pd;

    pc = (char *)10000;
    pi = (int *)10000;
    pd = (double *)10000;
    printf("증가 전 pc = %d, pi = %d, pd = %d\n", pc, pi, pd);

    pc++;
    pi++;
    pd++;
    printf("증가 후 pc = %d, pi = %d, pd = %d\n", pc, pi, pd);
    printf("pc+2 = %d, pi+2 = %d, pd+2 = %d\n", pc+2, pi+2, pd+2);
    return 0;
}
```

증가 전 pc = 10000, pi = 10000, pd = 10000  
증가 후 pc = 10001, pi = 10004, pd = 10008  
pc+2 = 10003, pi+2 = 10012, pd+2 = 10024

# 1. 포인터 변수

- 증감 연산자를 이용한 포인터 접근

```
#include <stdio.h>

int main(void)
{
    int arr[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    for (int i = 0; i < 10; i++) {
        printf("%d\n", arr[i]);
        printf("%d\n", *(arr + i));
    }

    return 0;
}
```

# 1. 포인터 변수

## ■ 실습 2

- 배열은 메모리 공간에 연속적으로 할당되어 있는 것을 배웠다.
- 만약 `int arr[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };` 이 있을 때  
    `&arr[0]`이 `0x0004`이라면, `&arr[1]`은 `0x0008`일 것 이다.
- 위 사실을 이용해서 증감 연산자와 반복문 만을 사용하여 배열 `arr`의 요소를 모두 출력해보자. `arr[i]` 형태로 접근 불가

# 1. 포인터 변수

- 포인터 타입을 잘못 선언하는 벌어지는 일

- 변수 pc가 char\*이 아닌 int\*로 선언 되었다.
- \*pc연산으로 'A'가 담긴 공간 보다 더 큰 공간에 접근하여 메모리 공간을 엿볼 수 있다.
- \*pc = 13... 연산으로 다른 메모리 공간에 값을 덮어씌울 수 있다.

위 두가지 경우 모두 보안 취약점으로 활용될 수 있음  
따라서 요즘에는 위 접근이 발생하면 프로그램이 정지됨

- 반대로 int a = 135468;
- char\* pa = &a;
- 위 코드는 a보다 작은 공간을 분할하여 접근

```
#include <stdio.h>

int main(void) {
    char c = 'A';

    int* pc = &c;

    printf("%d", *pc);

    *pc = 13845321;

    printf("%d", *pc);

    return 0;
}
```



# 1. 포인터 변수

- 포인터 주소를 임의로 지정하면 벌어지는 일
  - 하드웨어 등을 제어하는 경우를 제외

```
● ● ●  
  
#include <stdio.h>  
  
int main(void) {  
    int* opponent_position = 0x5ABF1C;  
  
    attack(*opponent_position);  
  
    int* pin_number = 0x32E1FB5;  
    save_database(*pin_number);  
  
    return 0;  
}
```

## 2. 포인터와 함수

### ❖ 함수 호출 시에 인수 전달 방법

- 값에 의한 호출(call by value)
  - ✓ 함수로 복사본이 전달됨
  - ✓ C언어에서의 기본적인 방법
- 참조에 의한 호출(call by reference)
  - ✓ 함수로 원본이 전달됨
  - ✓ C에서는 포인터를 이용하여 흉내 낼 수 있음

## 2. 포인터와 함수

### ■ 포인터와 함수

- 매개변수는 일반적인 경우와 동일
- 포인터를 통해 값을 변경
  - Call by reference

```
#include <stdio.h>

void call_function(int* a, double* b);

int main(void) {
    int a = 135;
    double b = 513.147;

    call_function(&a, &b);

    printf("%d\n", a);

    return 0;
}

void call_function(int* a, double* b) {
    printf("%d    %lf\n", *a, *b);
    *a = 1;
}
```

## 2. 포인터와 함수

### ❖ swap() 함수 (값에 의한 호출)

```
#include <stdio.h>
void swap(int x, int y);
int main(void)
{
    int a = 100, b = 200;
    printf("a=%d b=%d\n", a, b);

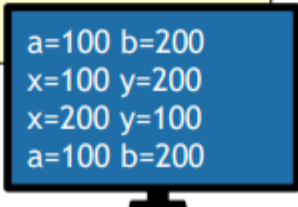
    swap(a, b);

    printf("a=%d b=%d\n", a, b);
    return 0;
}
```

```
void swap(int x, int y)
{
    int tmp;
    printf("x=%d y=%d\n", x, y);

    tmp = x;
    x = y;
    y = tmp;

    printf("x=%d y=%d\n", x, y);
}
```



```
a=100 b=200
x=100 y=200
x=200 y=100
a=100 b=200
```

## 2. 포인터와 함수

---

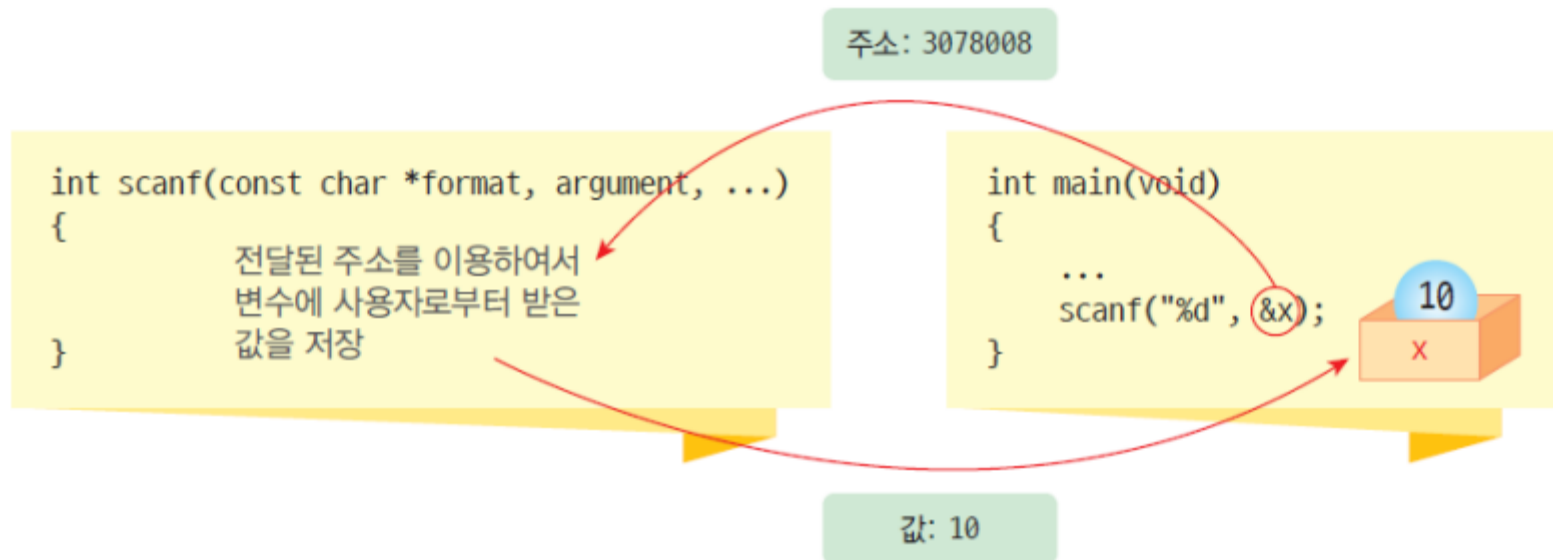
### ■ 실습 3

- 실습 노트 19페이지의 swap() 함수는 제대로 역할을 수행하지 못한다.
- Swap함수를 호출하여 main 함수에서도 값이 변경되도록 개발해보자

## 2. 포인터와 함수

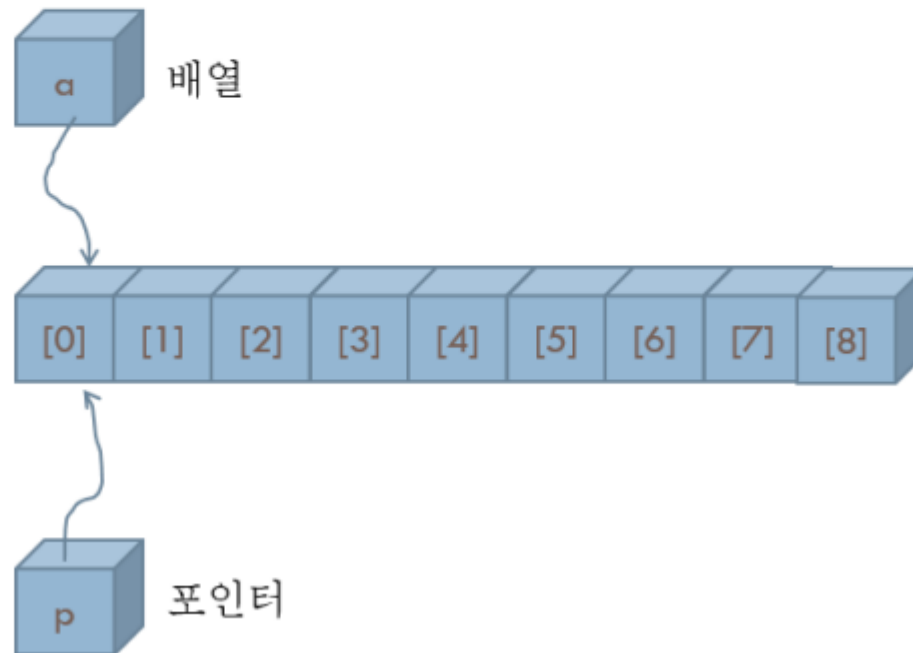
### ❖ scanf() 함수

- 변수에 값을 저장하기 위하여 변수의 주소를 받음



### 3. 포인터와 배열

- ❖ 배열과 포인터는 아주 밀접한 관계를 가지고 있음
- ❖ 배열 이름이 바로 포인터임
- ❖ 포인터는 배열처럼 사용 가능



### 3. 포인터와 배열

#### ❖ 포인터와 배열의 관계

// 포인터와 배열의 관계

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

```
int main(void)
```

```
{
```

```
    int a[] = { 10, 20, 30, 40, 50 };
```

```
    printf("&a[0] = %u\n", &a[0]);
```

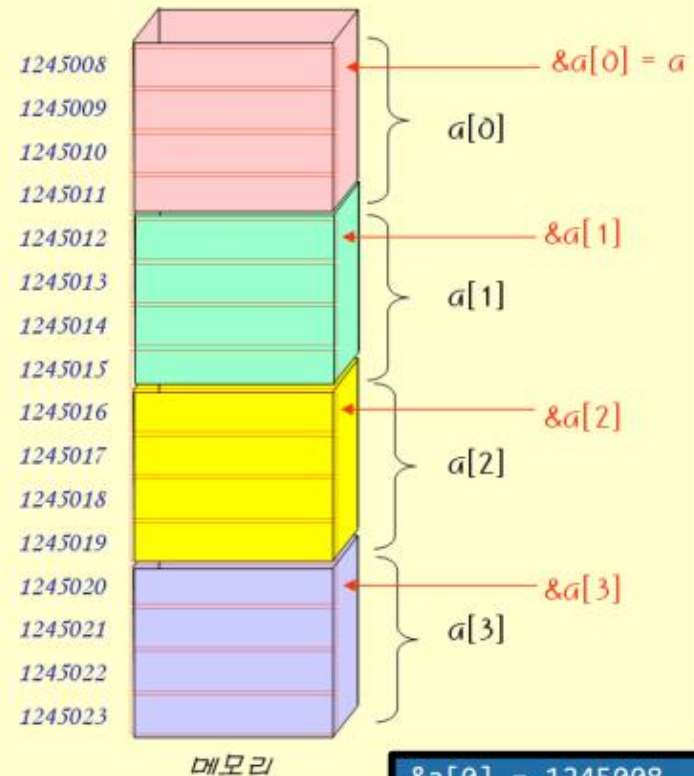
```
    printf("&a[1] = %u\n", &a[1]);
```

```
    printf("&a[2] = %u\n", &a[2]);
```

```
    printf("a = %u\n", a);
```

```
    return 0;
```

```
}
```



```
&a[0] = 1245008
&a[1] = 1245012
&a[2] = 1245016
a = 1245008
```



### 3. 포인터와 배열

```
#include <stdio.h>

int main(void) {
    int arr[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    printf("arr = &arr[0]: %c\n", arr == &arr[0] ? 'T' : 'F');

    return 0;
}
```

### 3. 포인터와 배열

#### ❖ 포인터를 배열처럼 사용

```
#include <stdio.h>
int main(void)
{
    int a[] = { 10, 20, 30, 40, 50 };
    int *p;

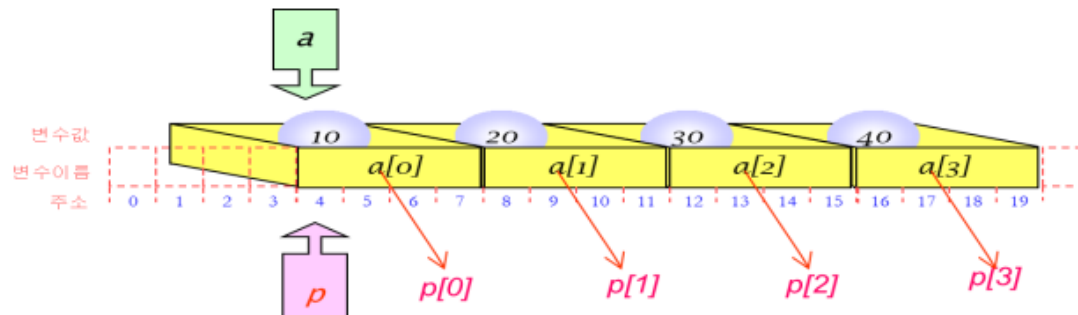
    p = a;
    printf("a[0]=%d a[1]=%d a[2]=%d \n", a[0], a[1], a[2]);
    printf("p[0]=%d p[1]=%d p[2]=%d \n\n", p[0], p[1], p[2]);

    p[0] = 60;
    p[1] = 70;
    p[2] = 80;

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

a[0]=10 a[1]=20 a[2]=30  
p[0]=10 p[1]=20 p[2]=30

a[0]=60 a[1]=70 a[2]=80  
p[0]=60 p[1]=70 p[2]=80



## 4. 마무리 문제

### ■ 실습 4

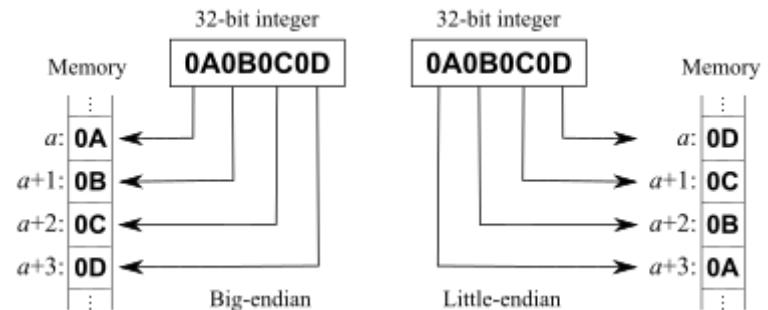
- 컴퓨터가 메모리에 값을 저장할 때, 프로세서의 아키텍처마다 서로 다른 엔디언을 사용한다.
- 엔디언은 Byte Order라고도 하며, 빅 엔디언과 리틀 엔디언으로 나누어져있다.

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	



- int 타입 변수에 0x12345678이 저장되어 있을 때, 포인터 연산으로 각 메모리 셀마다 값이 어떻게 저장되어 있는지 파악하면 본인 컴퓨터의 엔디언을 알 수 있을 것이다.
- 위 그림을 참조하여 본인 컴퓨터의 엔디언을 파악하고 출력하는 프로그램을 개발해보자