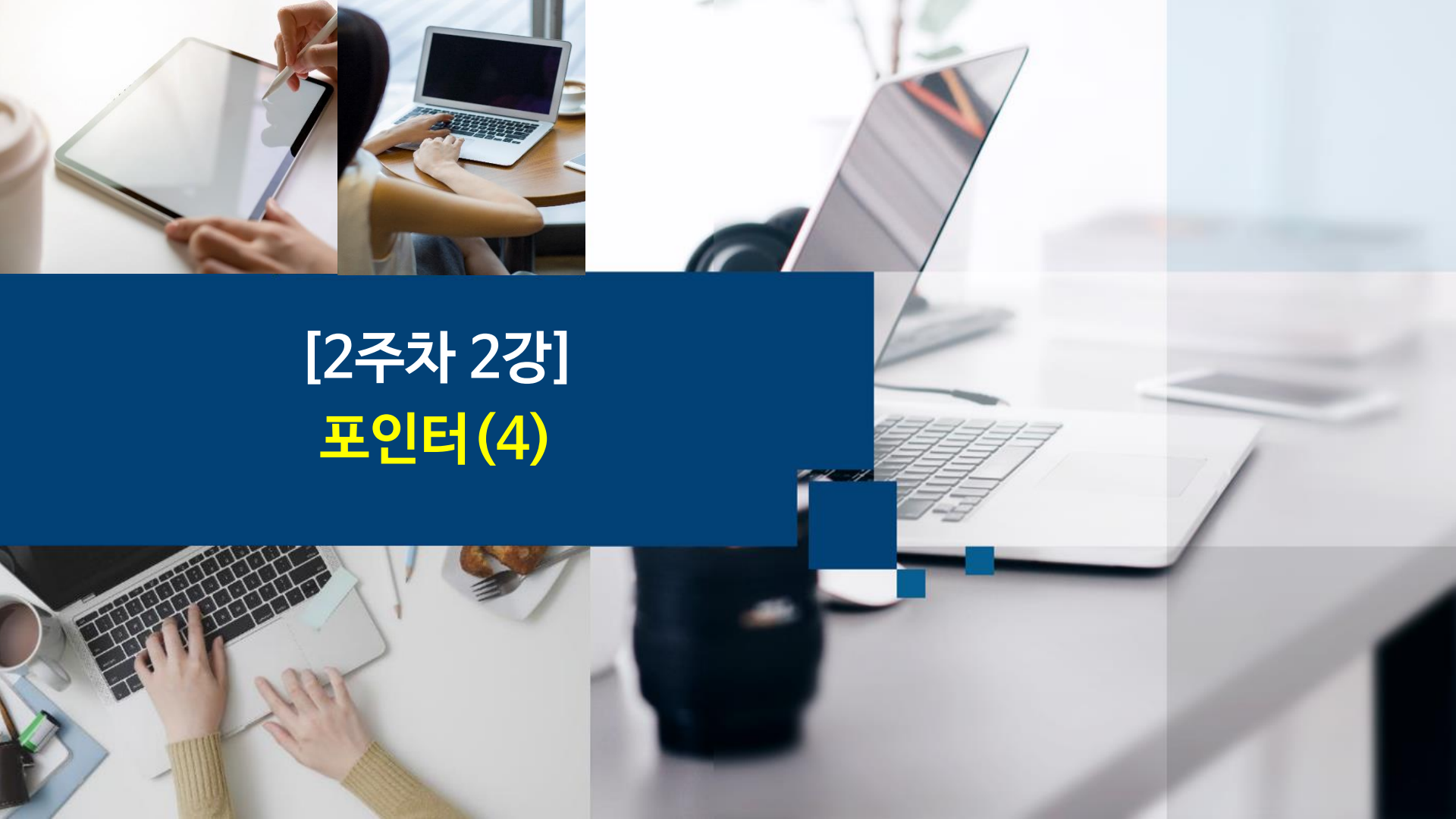


# [2주차 2강]
# 포인터(4)

# 학습 내용

- 9.4 포인터 연산
- 9.5 포인터 인자와 주소 반환(1)

# 학습 목표

- 9.4 포인터로 사용할 수 있는 연산이 무엇인지 학습한다.
- 9.5 포인터 개념이 추가된 함수 호출 과정을 이해한다.

C 프로그래밍 및 실습

1. 포인터 개요
2. 포인터 선언과 사용
3. 배열과 포인터
4. **포인터 연산**
5. 포인터 인자와 주소 반환
6. 포인터 배열
7. 다중 포인터

# 4. 포인터 연산

## 주소에 정수를 더하거나 빼기

- ++, --, +=, -= 와 같이 덧셈, 뺄셈에 대한 연산자는 모두 가능

```
int ar[5] = {2, 3, 5, 7, -1}, *p = ar;
int i = 4;

printf("%p %d\n", p+2, *(p+2));    ⇨ &ar[2], ar[2]
printf("%p %d\n", p+i, *(p+i));    ⇨ ar[4]
printf("%p %d\n", ++p, *p);        ⇨ ar[1]
```

실행 결과

```
001E40BC 5
001E40C4 -1
001E40B8 3
```

# ※ 실습하기


[예제 9.6] 다음에 해당하는 문장을 차례로 작성하고, p1과 p2의 값이 얼마나 증가하는지 확인해보자 (배열과 포인터의 주의사항 3 참고).

① int 포인터 p1을 선언하고, NULL로 초기화

② char 포인터 p2를 선언하고, NULL로 초기화

③ p1과 p2를 출력 (즉, p1과 p2에 저장된 값 출력)

④ p1과 p2를 1만큼 증가 (++ 연산자 사용)

⑤ p1과 p2를 출력

⑥ p1과 p2를 2만큼 증가 (+= 연산자 사용)

⑦ p1과 p2를 출력

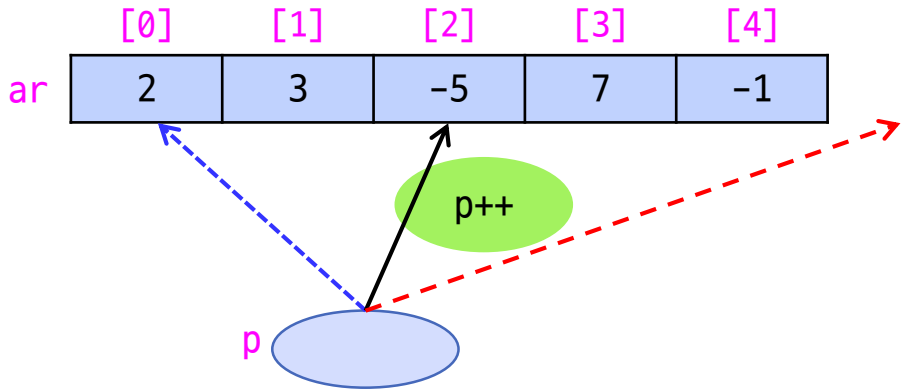
# ※ 실습하기

[예제 9.7]
포인터 연산을 이용하여 배열 전체 훑어보기

① 정수 배열 ar[5]를 선언하고 {2, 3, 5, 7, -1}로 초기화
② 정수 변수 i와 정수 포인터 p 선언 후, p에 배열 이름 ar 연결
③ for문: 다음을 5회 반복 (i는 반복 제어 변수)
  ✓ p가 가리키는 변수의 값 출력
  ✓ p의 값 1만큼 증가

실행결과

```
2
3
5
7
-1
```



# 4. 포인터 연산

## 주소 비교하기

- 비교 연산자(==, !=, 〈, 〉, 〉=, 〈=) 사용 가능

```
int ar[5] = {2, 3, 5, 7, -1}, *p1 = &ar[1], *p2 = &ar[4];

printf("%p %p\n", p1, p2);   ⇨ ar[1]과  ar[4]의 주소

printf("%d\n", p1 < p2);     ⇨ ar[1]과 ar[4]의 주소 값 비교
printf("%d\n", *p1 < *p2);   ⇨ ar[1]과 ar[4]의 원소 값 비교
```

실행결과

```
001E40B8 001E40C4
1   ⇨ p1과 p2에 저장된 값 비교
0   ⇨ p1과 p2가 가리키는 변수의 값 비교
```

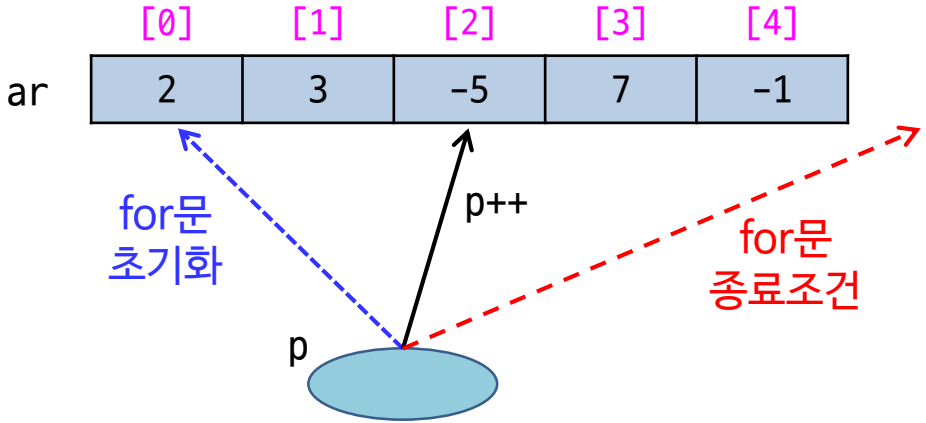
# 4. 포인터 연산

## 주소 비교를 이용하여 배열 훑어보기

```
int ar[5]={2, 3, 5, 7, -1}, *p;

for( p = ar ; p < &ar[5] ; p++ )
    printf("%d\n", *p);
```

(참고) 포인터 학습을 위한 연습용 코드,
보통 배열은 반복 제어 변수 사용



C 프로그래밍 및 실습

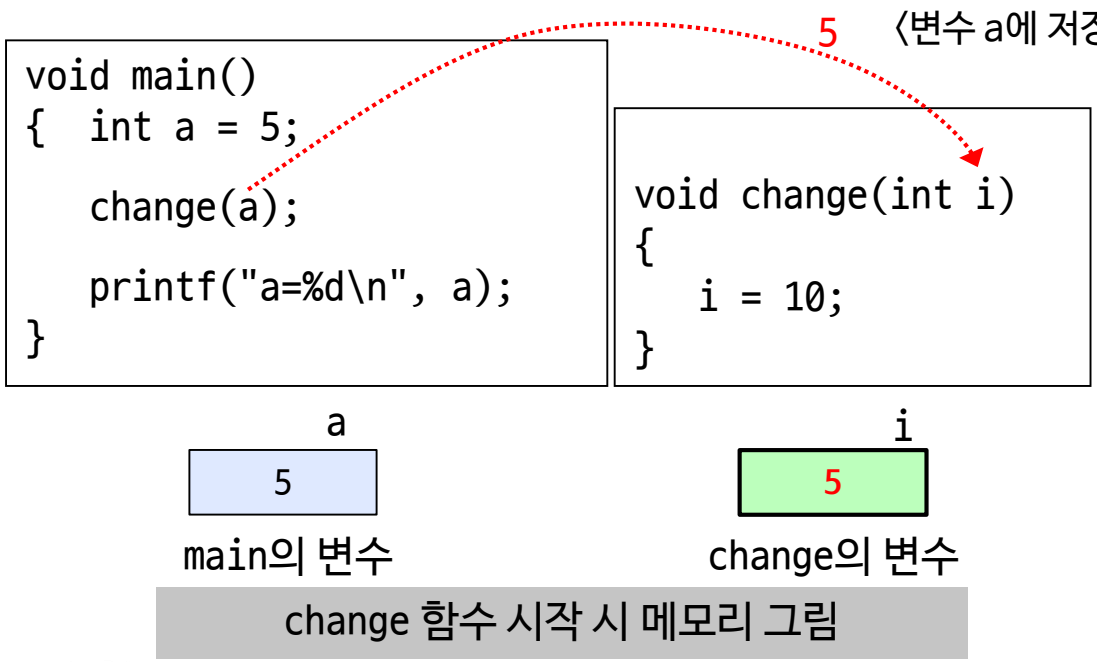
9.4 포인터 연산

# 9.5 포인터 인자와 주소 반환(1)

# 5. 포인터 인자와 주소 반환(1)

## 함수 수행 과정 (복습) - 정수 인자

- A. 함수 시작(호출) : 형식 인자(변수)에 공간이 할당되고, 각 인자에 전달된 정수 값이 대입됨.



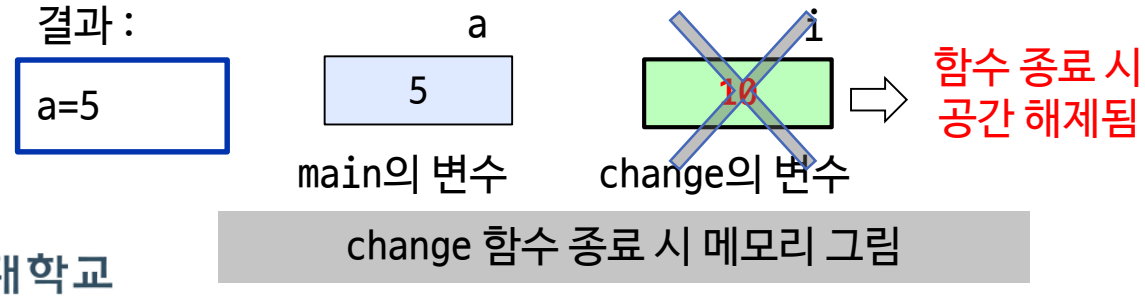
# 5. 포인터 인자와 주소 반환(1)

## 함수 수행 과정 (복습) - 정수 인자

- B. 함수 본체 수행 : 지역변수 i에 10 대입
- C. 함수 종료 : 함수의 지역변수(인자 포함)가 없어짐 (할당된 메모리 공간 반환)

```
void main()
{  int a = 5;
   change(a);
   printf("a=%d\n", a);
}
```

```
void change(int i)
{
   i = 10;
}
```




change 함수 종료 시 메모리 그림

# 5. 포인터 인자와 주소 반환(1)

## 함수 인자가 정수가 아니라 주소라면?

- 코드에서 변경되는 부분
  - ✓ 주소를 저장하기 위해 함수의 형식 인수는 포인터로 선언
  - ✓ 포인터 변수를 이용한 간접 참조

```
void main()
{  int a = 5;

   change(&a);

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

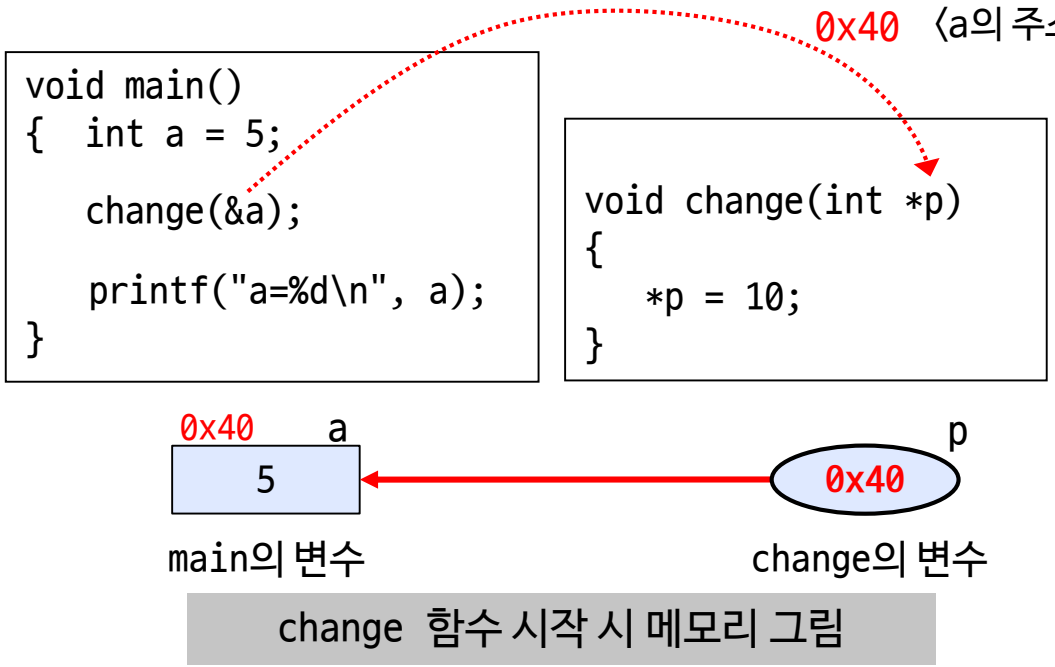
```
void change(int *p)
{
   *p = 10;
}
```

- 위 함수의 수행 과정은? 다음슬라이드

# 5. 포인터 인자와 주소 반환(1)

## 함수 수행과정

- A. 함수 시작(호출) : 형식 인자(변수)에 공간이 할당되고, 각 인자에 전달된 주소가 대입됨.




change 함수 시작 시 메모리 그림

# 5. 포인터 인자와 주소 반환(1)

## 함수 수행과정

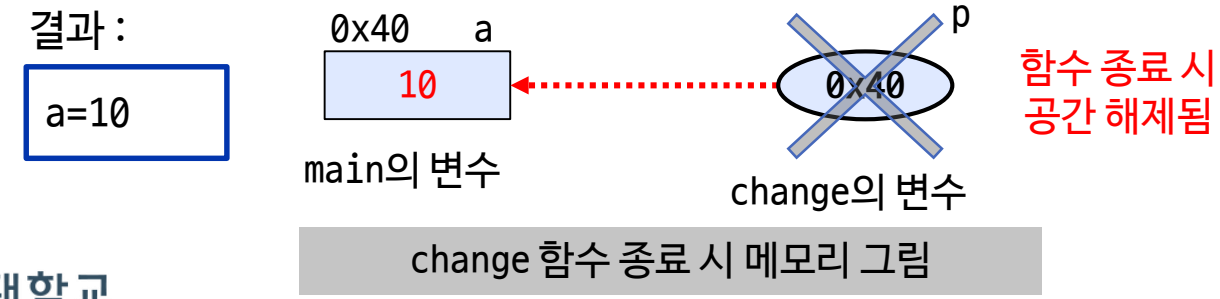
- B. 함수 본체 수행 : p가 가리키는 변수에 10 대입
- C. 함수 종료 : 함수의 지역변수(인자 포함)가 없어짐 (할당된 메모리 공간 반환)

```
void main()
{  int a = 5;

   change(&a);

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

```
void change(int *p)
{
   *p = 10;
}
```




change 함수 종료 시 메모리 그림

# 5. 포인터 인자와 주소 반환(1)

## 함수 인자 비교

- 정수 값이나 문자 등을 인자로 함수를 호출하는 것을 '값에 의한 호출(call-by-value)'이라 함
  - ✓ 호출하는 함수의 변수에 영향을 못 미침
- 주소를 인자로 함수를 호출하는 것을 '주소에 의한 호출(call-by-reference)'이라 함
  - ✓ 간접 참조로 인해 호출하는 함수의 변수 값을 변경시킬 수 있음
- 하지만, 두 호출 방식의 함수의 수행 과정(**인자 전달** 및 제어 흐름)은 **완전히 동일**
  - ✓ 값이 전달되느냐, 주소가 전달되느냐에 따라오는 부수적인 효과일 뿐

# 학습 정리

- 포인터는 **정수를 더하거나 빼는 연산**과 **비교 연산** 정도만 가능함
- 주소를 인자로 함수를 호출하는 것을 **'주소에 의한 호출'**이라 함
- 정수나 문자를 인자로 함수를 호출하는 것을 **'값에 의한 호출'**이라 함
- 두 호출 방식의 **함수 수행 과정**은 완전히 **동일**함