

# 컴퓨터 프로그래밍2

## 배열과 포인터

---

장서윤 [pineai@cnu.ac.kr](mailto:pineai@cnu.ac.kr)

# 배열명 역할을 하는 포인터

---

## 예제 10-3 배열명처럼 사용되는 포인터

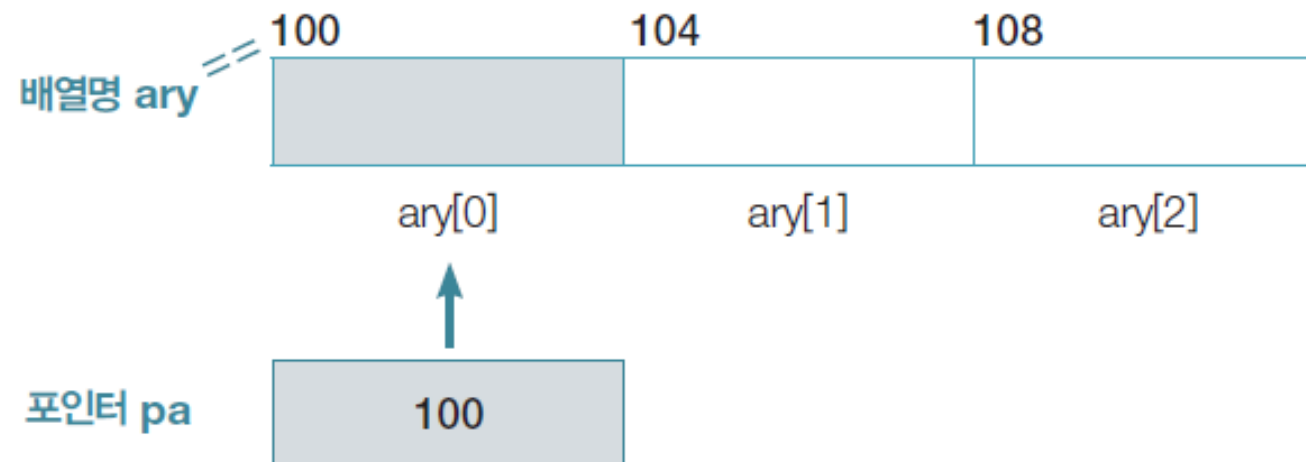
```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int ary[3];           // 배열 선언
6.     int *pa = ary;        // 포인터에 배열명 저장
7.     int i;                // 반복 제어 변수
8.
9.     *pa = 10;              // 첫 번째 배열 요소에 10 대입
10.    *(pa + 1) = 20;        // 두 번째 배열 요소에 20 대입
11.    pa[2] = pa[0] + pa[1];  // 대괄호를 써서 pa를 배열명처럼 사용
12.
13.    for(i = 0; i < 3; i++)
14.    {
15.        printf("%5d", pa[i]); // 포인터로 모든 배열 요소 출력
16.    }
17.
18.    return 0;
19. }
```

실행  
결과 10 20 30

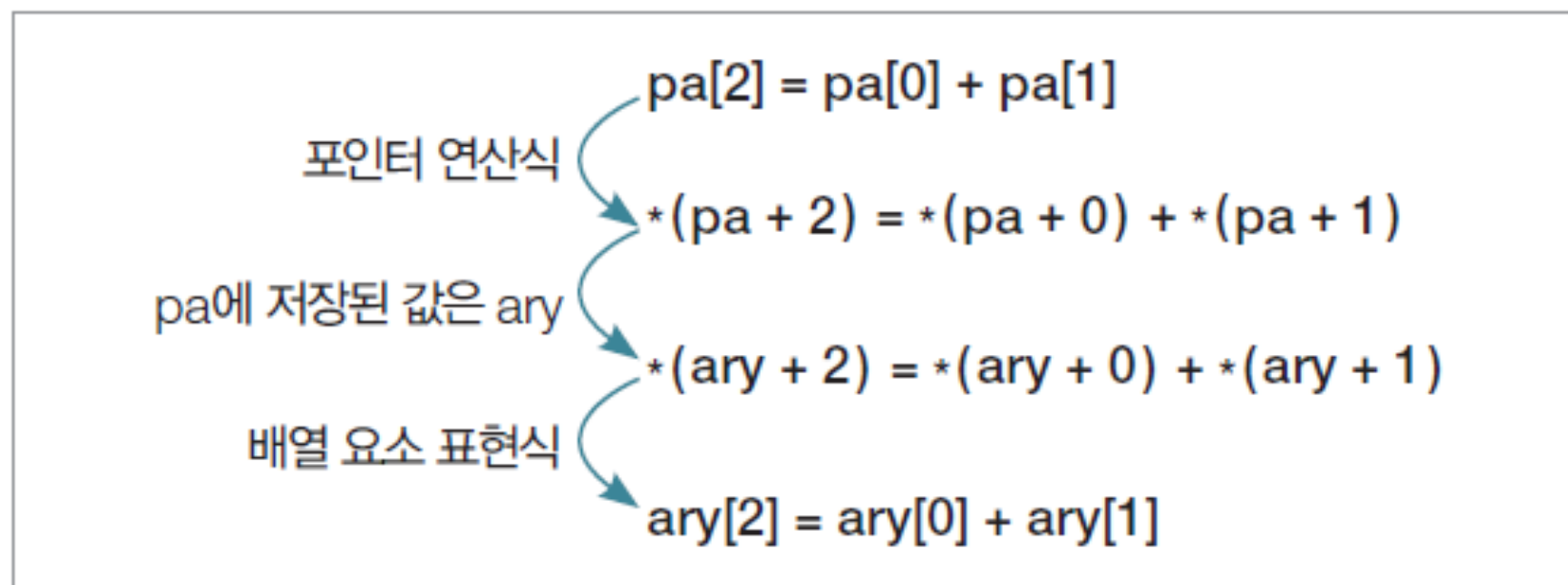
# 배열명 역할을 하는 포인터

## ▶ int형 가리키는 포인터에 저장

- ▶ 배열이 메모리 100번지부터 할당되었다면 배열명 ary는 주소값 100번지, 포인터 pa는 100 저장하여 첫 번째 배열 요소 가리키는 상태



- ▶ 대괄호는 포인터 연산식으로 바뀌므로 결국 각 배열 요소 사용 가능



# 배열명과 포인터의 차이

---

## ▶ sizeof 함수 사용 결과의 차이

- ▶ 배열명에 사용시 배열 전체의 크기
- ▶ 포인터에 사용하면 포인터 하나의 크기
  - ▶ 배열명을 포인터에 저장하면 포인터로 배열 전체 크기 확인 불가

---

```
int ary[3];  
int *pa = ary;
```

sizeof (ary)	12바이트	// 배열 전체 크기
sizeof (pa)	4바이트	// 포인터 하나의 크기

---

# 배열명과 포인터의 차이

---

## ▶ 변수와 상수의 차이

- ▶ 포인터는 그 값을 바꿀 수 있음
- ▶ 배열명은 상수이므로 값을 바꿀 수 없음

---

<code>pa = pa + 1</code>	가능	// pa에 1을 더하여 다시 pa에 저장할 수 있다.
<code>ary = ary + 1</code>	불가능	// ary에 1을 더하는 것은 가능하나 그 값을 다시 저장할 수 없다.

---

# 배열명과 포인터의 차이

---

## 예제 10-4 포인터를 이용한 배열의 입출력

---

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int ary[3] = {10,20,30};
6.     int *pa = ary;
7.     int i;
8.
9.     printf("배열의 값 : ");
10.    for(i = 0; i < 3; i++)
11.    {
12.        printf("%d ", *pa);
13.        pa++;
14.    }
15.
16.    return 0;
17. }
```

실행  
결과  
배열의 값 : 10 20 30

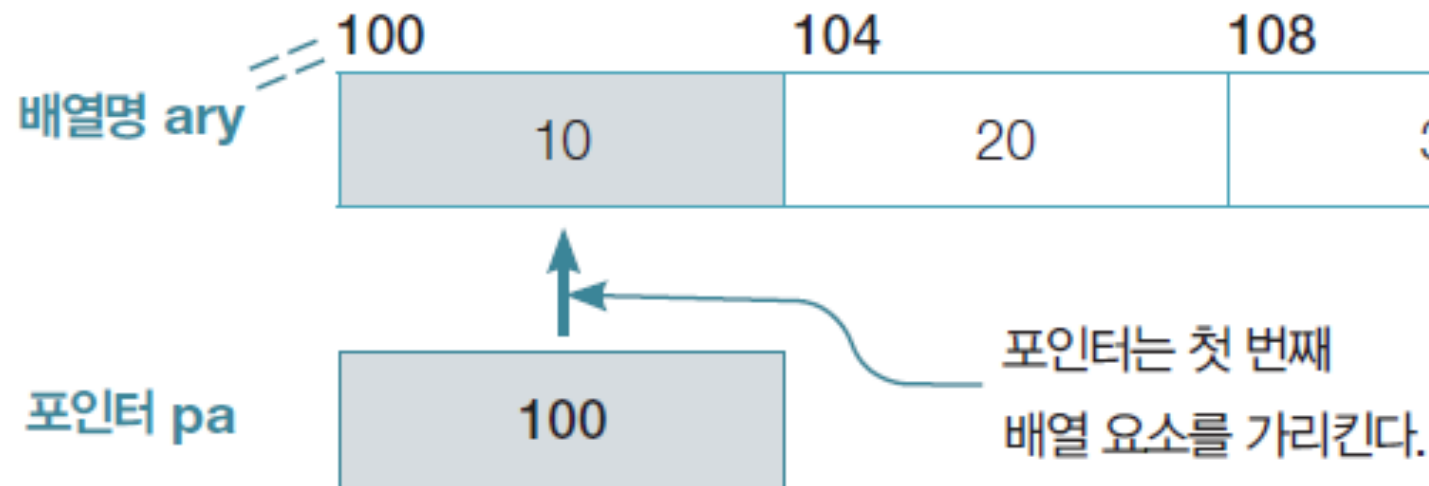
// pa가 가리키는 배열 요소 출력

// 다음 배열 요소를 가리키도록 pa값 증가

# 배열명과 포인터의 차이

---

- ▶ 배열이 메모리 100번지부터 할당
  - ▶ 포인터가 초기화된 상태



- ▶ 포인터 pa로 첫 번째 배열 요소 출력하는 방법

---

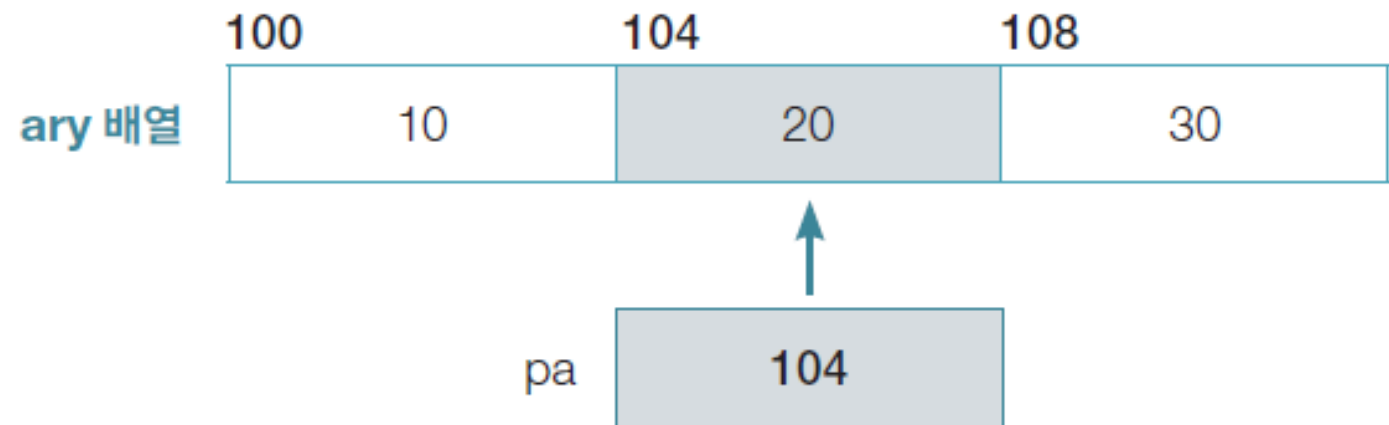
<code>printf("%d", pa[0]);</code>	<code>// pa를 배열명처럼 사용하여 첫 번째 배열 요소 출력</code>
<code>printf("%d", *(pa + 0));</code>	<code>// pa[0]를 그대로 포인터 연산식으로 바꿈</code>
<code>printf("%d", *pa);</code>	<code>// *(pa + 0)에서 의미 없는 0과 괄호를 제거한 표현</code>

---

# 배열명과 포인터의 차이

---

- ▶ pa가 두 번째 배열 요소를 가리키도록 하는 방법
  - ▶ pa에 1을 더하면 두 번째 배열 요소의 주소 104번지
  - ▶ 이 값을 다시 pa에 저장



- ▶ 포인터 pa가 변수이므로 그 값 바꿀 수 있기 때문에 가능
  - ▶ 배열명 ary는 주소 상수라 값을 바꿀 수 없음

---

```
for(i = 0; i < 3; i++)  
{  
    printf("%d ", *ary);    // 배열명 ary가 가리키는 첫 번째 배열 요소 출력  
    ary++;                 // ( X ) ary는 상수이므로 증가시킬 수가 없다!  
}
```

---



# 배열명과 포인터의 차이

---

- ▶ 포인터로 배열 데이터 처리시 주의할 점
  - ▶ 특정 배열 요소의 위치 기억할 수 있는 이점
- ▶ 포인터의 값이 변할 수 있으므로 유효한 값인지 확인하는 습관 필요
  - ▶ 간접참조 연산을 통해 그 공간이나 저장된 값 사용 불가
- ▶ pa로 다시 배열의 처음부터 데이터를 처리해야 한다면?
  - ▶ 배열명으로 다시 초기화
- ▶ 주소 상수로 그 값이 바뀌지 않으므로 언제든지 배열의 시작 위치 찾아갈 때 사용 가능

# 배열명과 포인터의 차이

---

- ▶ 배열에 입력 받을 때
  - ▶ 간접참조 연산 없이 포인터만 사용
  - ▶ scanf 함수 - 입력할 배열 요소의 주소가 필요하므로 그 값을 갖고 있는 포인터 그대로 사용

---

```
for(i = 0; i < 3; i++)  
{  
    scanf("%d ", pa);           // pa가 가리키는 배열 요소에 입력, 간접참조 연산 없음  
    pa++;                       // 다음 배열 요소를 가리키도록 pa 증가  
}
```

---

# 배열명과 포인터의 차이

---

- ▶ 포인터로 배열 요소 차례로 출력
  - ▶ 포인터에 증가 연산자와 간접참조 연산자 함께 사용 가능

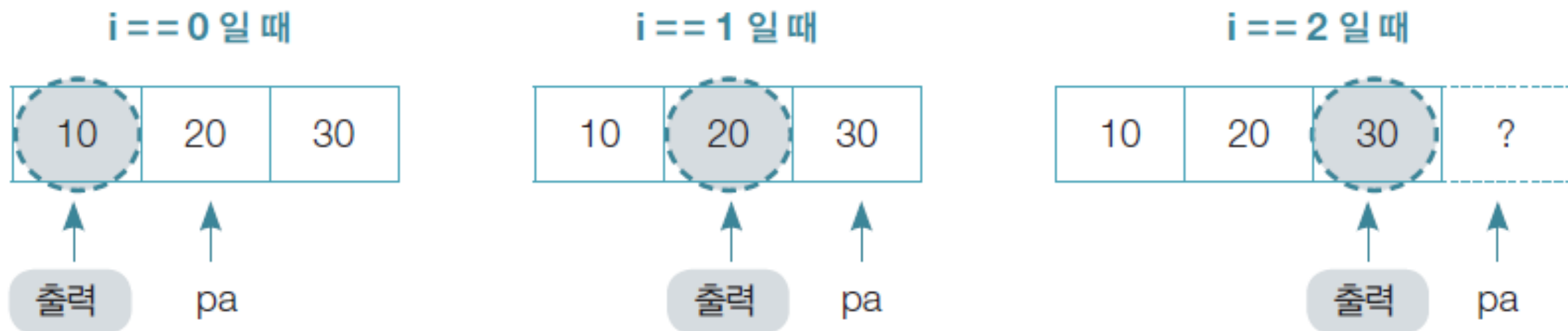
---

```
for(i = 0; i < 3; i++)  
{  
    printf("%d ", *(pa++) );    // 후위형 사용  
}
```

---

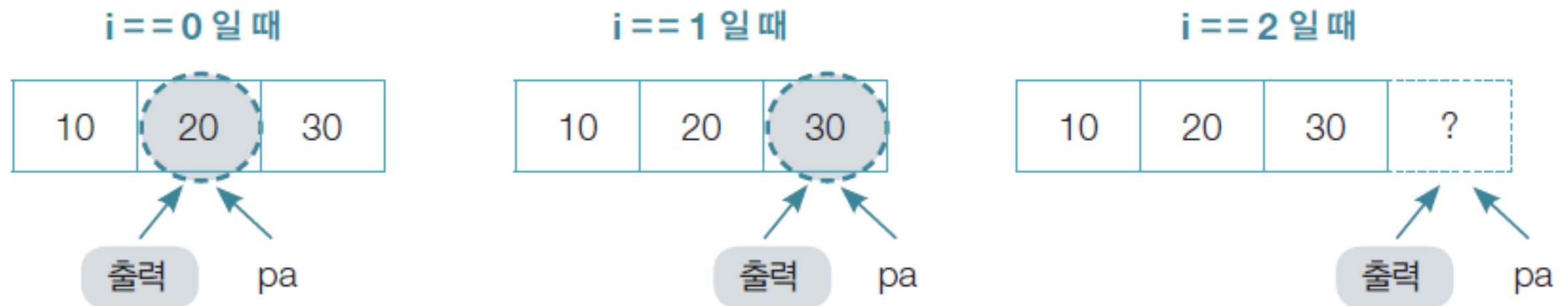
# 배열명과 포인터의 차이

- ▶ 포인터로 배열 요소 차례로 출력
  - ▶ 포인터에 증가 연산자 후위형 사용했을 때
    - ▶ 연산자 우선순위에 따라  $pa++$  먼저 수행 ( $pa$  값 증가)
      - ▶ 후위형이므로 다음 연산인 간접참조 연산을 수행할 때는 증가되기 이전 값 사용
  - ▶  $pa$ 가 가리키던 배열 요소의 값이 먼저 출력되고  $pa$ 가 다음 배열 요소를 가리키는 것과 결과 같음



# 배열명과 포인터의 차이

- ▶ 포인터로 배열 요소 차례로 출력
  - ▶ 포인터에 증가 연산자 전위형 사용했을 때
    - ▶  $*(++pa)$ 는  $pa$ 의 값 먼저 증가된 후 증가된  $pa$ 가 가리키는 배열 요소 간접참조
    - ▶ 두 번째 배열 요소부터 출력
      - ▶ 마지막에 출력되는 값은 배열 값이 아닌 쓰레기값



# 배열명과 포인터의 차이

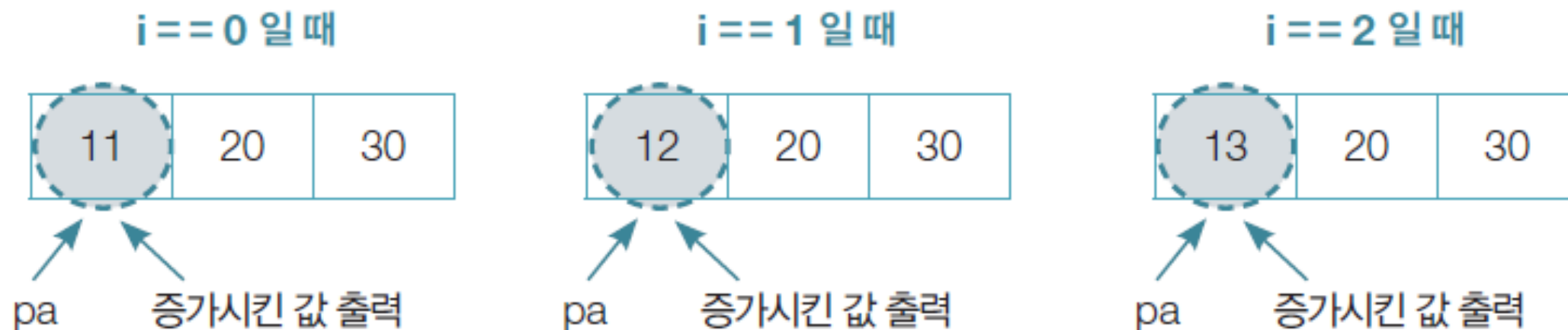
---

- ▶ 포인터로 배열 요소 차례로 출력
  - ▶ 전위형이나 후위형 모두 괄호 생략해도 결과 동일
    - ▶ 간접참조 연산자와 증가 연산자는 모두 단항 연산자
      - ▶ 우선순위 같음
    - ▶ 연산 방향은 오른쪽에서 왼쪽
    - ▶ 항상 증가 연산자 먼저 수행

# 배열명과 포인터의 차이

- ▶ 포인터로 배열 요소 차례로 출력
  - ▶ 괄호를 간접참조 연산자에 먼저 사용하면?
    - ▶ pa 값 자체는 바뀌지 않으며 첫 번째 배열 요소를 가리키는 상태로 고정
    - ▶ pa가 가리키는 배열 요소의 값이 증가하면서 차례로 출력
    - ▶ 전위형은 pa가 가리키는 배열 요소의 값 증가시킨 후 출력

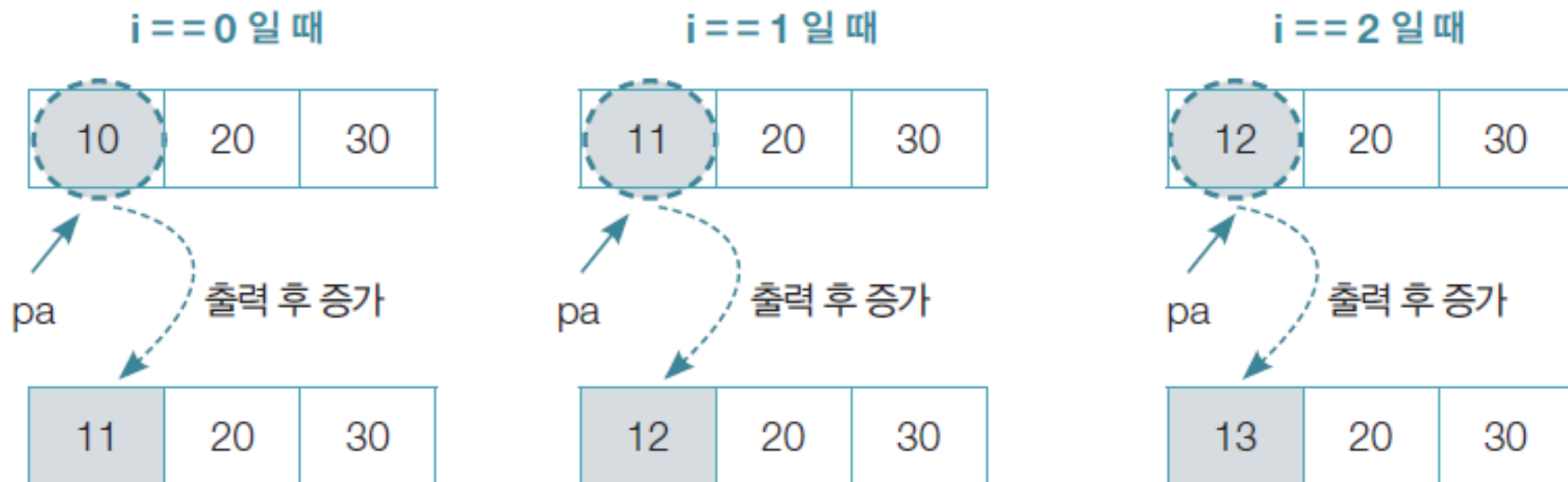
`++(*pa)` // 전위형, 결과는 11, 12, 13 출력



# 배열명과 포인터의 차이

- ▶ 포인터로 배열 요소 차례로 출력
  - ▶ 괄호를 간접참조 연산자에 먼저 사용하면?
    - ▶ 후위형은 먼저 출력하고 나중에 값 증가

**`(*pa)++` // 후위형, 결과는 10, 11, 12 출력**





# 포인터의 뱀셈과 관계 연산

---

- ▶ 포인터의 뱀셈

- ▶ 포인터 - 포인터 값의 차 / 가리키는 자료형의 크기

- ▶ 관계 연산자로 포인터의 대소관계 확인 가능

```

1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int ary[5] = {10, 20, 30, 40, 50};
6.     int *pa = ary;                // 첫 번째 배열 요소 주소
7.     int *pb = pa + 3;            // 네 번째 배열 요소 주소
8.
9.     printf("pa : %u\n", pa);
10.    printf("pb : %u\n", pb);
11.    pa++;                        // pa를 다음 배열 요소로 이동
12.    printf("pb - pa : %u\n", pb - pa);    // 두 포인터의 뺄셈
13.
14.    printf("앞에 있는 배열 요소의 값 출력 : ");
15.    if(pa < pb) printf("%d\n", *pa);    // pa가 배열의 앞에 있으면 *pa 출력
16.    else printf("%d\n", *pb);          // pb가 배열의 앞에 있으면 *pb 출력
17.
18.    return 0;
19. }

```

실행  
결과

```

pa : 3799428
pb : 3799440
pb - pa : 2
앞에 있는 배열 요소의 값 출력 : 20

```

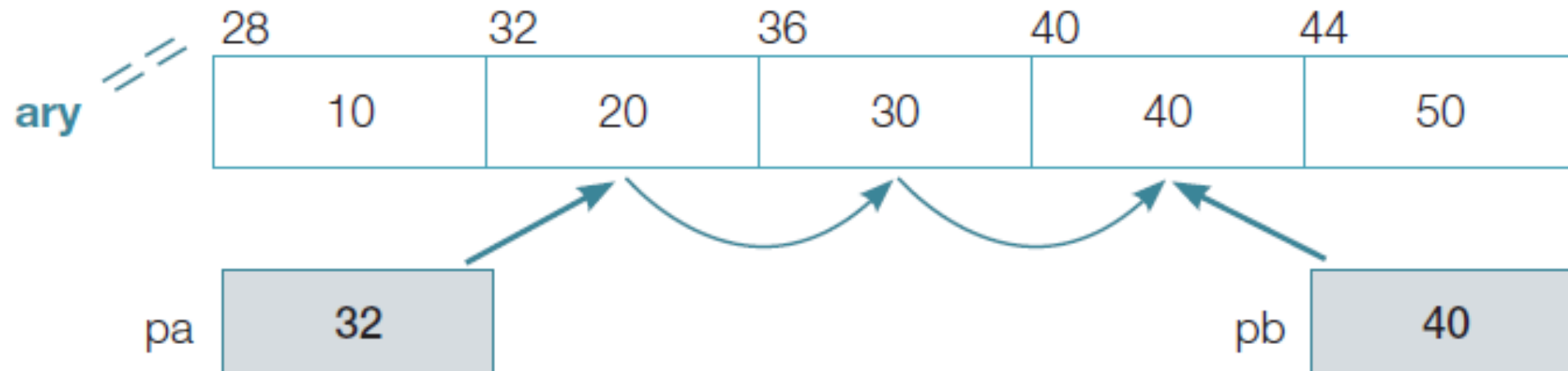
# 포인터의 뺄셈과 관계 연산

---

- ▶  $pb - pa$ 의 연산
  - ▶ 뺄셈 결과는 배열 요소 간의 간격 차이

$$pb - pa \rightarrow (40 - 32) / \text{sizeof}(\text{int}) \rightarrow 8 / 4 \rightarrow 2$$

값의 차      가리키는 자료형의 크기



# 배열의 입출력을 처리하는 함수

---

## ➤ 주소를 데이터로 주면 해결

표 10-2 배열에 데이터를 입출력하는 함수

구분	배열을 출력하는 함수	배열에 입력하는 함수
호출	<pre>int ary[5] = {10, 20, 30, 40, 50}; print_ary(ary, 5);</pre>	<pre>int ary[5]; input_ary(ary, 5);</pre>
정의	<pre>void print_ary(int *pa, int size) {     int i ;     for(i = 0; i &lt; size; i++)     {         printf( "%d ", pa[i]) ;     } }</pre>	<pre>void input_ary(int *pa, int size) {     int i;     for(i = 0; i &lt; size; i++)     {         scanf( "%d", pa + i) ;     } }</pre>

# 배열의 값을 출력하는 함수

---

- ▶ 배열의 값 확인하기 위해 수시로 출력해야 한다면?
  - ▶ 그 기능을 함수로 만들어 호출
  - ▶ 함수 호출할 때는 배열명을 주고, 함수의 매개변수로 포인터 선언
  - ▶ 함수 안에서 포인터를 배열명처럼 사용

# 배열의 값을 출력하는 함수

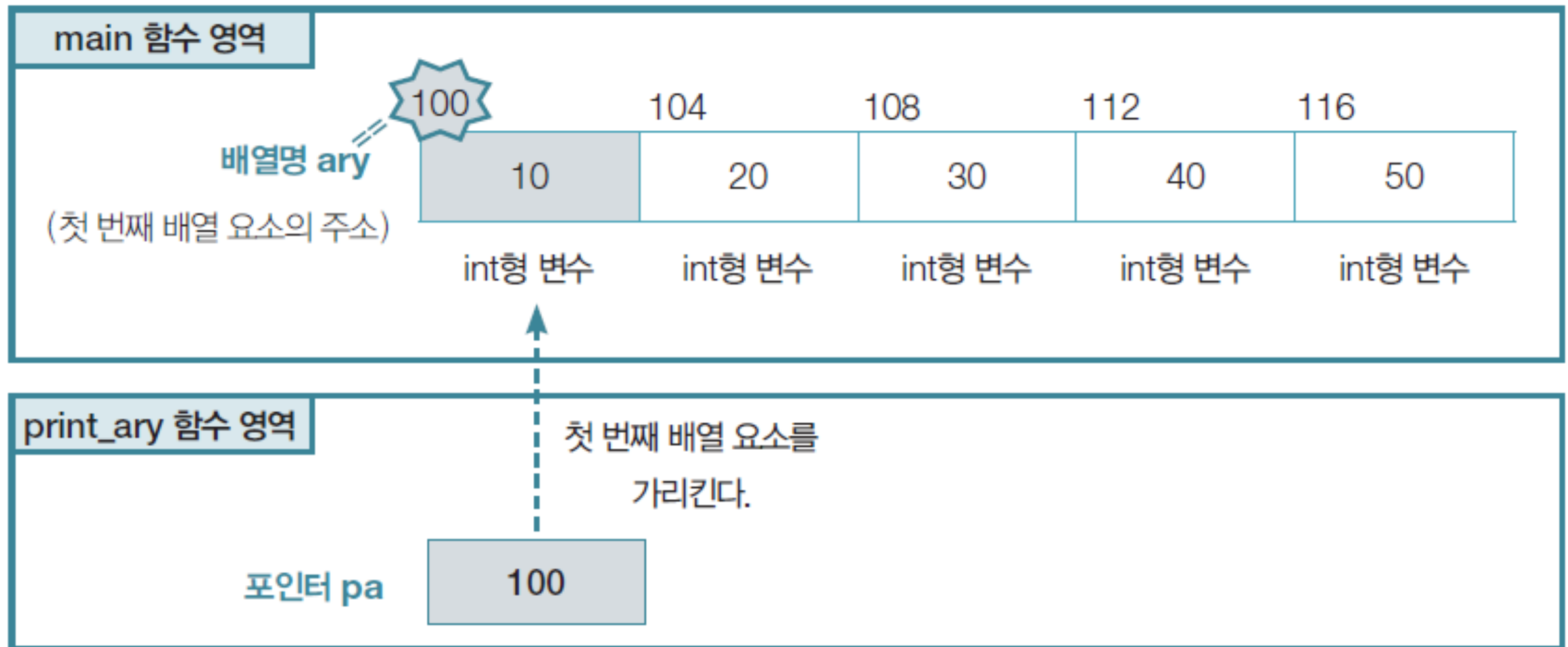
## 예제 10-6 배열의 값을 출력하는 함수

```
1. #include <stdio.h>
2.
3. void print_ary(int *pa) ;           // 함수 선언
4.
5. int main(void)
6. {
7.     int ary[5] = {10,20,30,40,50};
8.
9.     print_ary(ary);                 // 배열명을 주고 함수 호출
10.
11.     return 0;
12. }
13.
14. void print_ary(int *pa)             // 매개변수로 포인터 선언
15. {
16.     int i;
17.
18.     for(i = 0; i < 5; i++)
19.     {
20.         printf("%d ", pa[i]);       // pa로 배열 요소 표현식 사용
21.     }
22. }
```

실행  
결과 10 20 30 40 50

# 배열의 값을 출력하는 함수

- ▶ 매개변수로 int형을 가리키는 포인터 pa 선언
- ▶ 배열은 메모리 100번지부터 할당되었다고 가정



# 배열의 값을 출력하는 함수

---

- ▶ `print_ary` 함수에서 `ary` 배열에 관해 알고 있는 유일한 정보
  - ▶ `pa`에 받은 첫 번째 배열 요소의 주소 100번지
  - ▶ 배열의 크기를 알고 있다고 가정하면 그걸로 충분
- ▶ 정수 연산 - `pa + 1` -> 두 번째 배열 요소의 주소 104번지
- ▶ 간접 참조 연산 - `*(pa + 1)` -> 두 번째 배열 요소
- ▶ 배열 요소 표현식 - `pa[1]` -> 두 번째 배열 요소
- ▶ 동일한 배열의 데이터를 2개의 함수가 공유
  - ▶ 배열에 있는 대량의 데이터를 다른 함수로 복사하지 않고 접근하므로 더 효율적
  - ▶ 주소만 알면 해당 위치의 값 바꿀 수도 있으므로 의도치 않게 값 변형하는 일이 없도록 주의해야 함



# 배열의 값을 출력하는 함수

- ▶ 함수 안에 선언된 변수나 배열 이름은 사용 범위가 중괄호 블록 {}으로 제한
  - ▶ print\_ary 함수에서는 main 함수에 있는 배열명 ary 직접 사용 불가

```
int main(void)
{
    int ary [5] = {10, 20, 30, 40, 50};
```

배열명 ary는  
main 함수 안에서만  
사용할 수 있다.

```
}
```

```
void print_ary(void)
{
    int i;

    for(i = 0; i < 5; i++)
    {
        printf("%d ", ary[i] );
    }
}
```

다른 함수가 가진 이름을  
직접 사용할 수 없다.

### 예제 10-7 크기가 다른 배열을 출력하는 함수

실행  
결과

10	20	30	40	50		
10	20	30	40	50	60	70

```
1. #include <stdio.h>
2.
3. void print_ary(int *pa, int size) ;           // 함수 선언, 매개변수 2개
4.
5. int main(void)
6. {
7.     int ary1[5] = {10, 20, 30, 40, 50};      // 배열 요소 수가 5개인 배열
8.     int ary2[7] = {10, 20, 30, 40, 50, 60, 70}; // 요소 수가 7개인 배열
9.
10.    print_ary(ary1, 5);                       // ary1 배열 출력, 배열 요소 수 전달
11.    printf("\n");
12.    print_ary(ary2, 7);                       // ary2 배열 출력, 배열 요소 수 전달
13.
14.    return 0;
15. }
16.
17. void print_ary(int *pa, int size)           // 배열 요소 수를 받는 매개변수 선언
18. {
19.     int i;
20.
21.     for(i = 0; i < size; i++)                // size의 값에 따라 반복 횟수 결정
22.     {
23.         printf("%d ", pa[i]);
24.     }
25. }
```

# 배열 요소의 개수가 다른 배열도 출력하는 함수

---

- ▶ 함수 호출할 때 주는 배열 요소 수는 sizeof 연산자로 구할 수도 있음
  - ▶ sizeof 연산자에 배열명을 사용하면 배열 전체의 크기를 계산하므로 이 값을 배열 요소 하나의 크기로 나누어 배열 요소 수 구함

---

```
print_ary ( ary2, sizeof(ary2) / sizeof(ary2[0]) );
```

---

# 배열 요소의 개수가 다른 배열도 출력하는 함수

---

- ▶ `print _ary` 함수 안에서 `sizeof` 연산자로 배열의 크기를 알 수 없나요?
  - ▶ 알 수 없음.
  - ▶ `sizeof` 연산자
    - ▶ 배열명에 사용하면 배열 전체 크기 구함
    - ▶ 포인터에 사용하면 포인터 자체의 크기만 계산
    - ▶ 포인터가 배열명을 저장한 경우도 포함

---

`sizeof(pa) / sizeof(pa[0])` → 포인터의 크기 / 첫 번째 배열 요소의 크기 → 4 / 4

---

# 배열에 값을 입력하는 함수

---

- ▶ 입력 함수는 데이터를 저장할 배열의 위치가 필요
  - ▶ 배열에 값 입력하는 함수는 함수 안에서 포인터 직접 사용
  - ▶ 실수 배열에 값 입력 함수와 최대값 찾는 함수 예제 작성

.....

### 예제 10-8 배열에 값을 입력하는 함수

---


```
1. #include <stdio.h>
2.
3. void input_ary(double *pa, int size);
4. double find_max(double *pa, int size);
5.
6. int main(void)
7. {
8.     double ary[5];
9.     double max;                                // 최댓값을 저장할 변수
10.    int size = sizeof(ary) / sizeof(ary[0]);    // 배열 요소 수 계산
11.
12.    input_ary(ary, size);                        // 배열에 값 입력
13.    max = find_max(ary, size);                    // 배열의 최댓값 반환
14.    printf("배열의 최댓값 : %.1lf\n", max);
15.
16.    return 0;
17. }
```

```

18.
19. void input_ary(double *pa, int size)           // double 포인터를 매개변수로 선언
20. {
21.     int i;
22.
23.     printf("%d개의 실수값 입력 : ", size);
24.     for(i = 0; i < size; i++)                 // size의 값에 따라 반복 횟수 결정
25.     {
26.         scanf("%lf", pa + i);    // &pa[i]도 가능, 입력할 배열 요소의 주소를 전달
27.     }
28. }
29.
30. double find_max(double *pa, int size)
31. {
32.     double max;
33.     int i;
34.
35.     max = pa[0];                               // 첫 번째 배열 요소의 값을 최댓값으로 설정
36.     for(i = 1; i < size; i++)                 // 두 번째 배열 요소부터 max와 비교
37.     {
38.         if(pa[i] > max) max = pa[i];    // 새로운 배열 요소의 값이 max보다 크면 대입
39.     }
40.
41.     return max;                               // 최댓값 반환
42. }

```

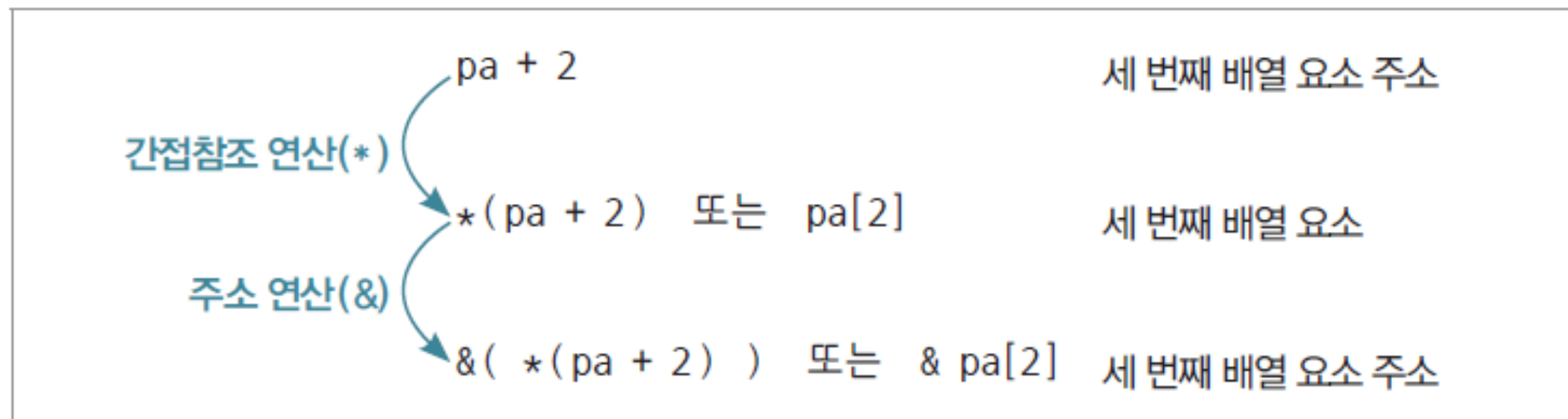
.....

실행 결과 5개의 실수값 입력 : 3.4 0.5 1.7 5.2 2.0   
배열의 최댓값 : 5.2

# 실수 배열에 값 입력 함수와 최댓값 찾는 함수

---

- ▶ scanf 함수 - 입력한 값을 저장할 배열의 위치를 알아야 함
  - ▶ 인수로 받은 pa의 값 그대로 사용
- ▶  $*(pa+i)$ 의 연산으로 배열 요소 구하고 다시  $\&(*(pa+i))$ 와 같이 주소 연산 사용하는 방법도 가능 -> 복잡하므로 불필요
- ▶ 대괄호를 사용한 배열 요소 표현식이 익숙하다면  $\&pa[i]$





# 실수 배열에 값 입력 함수와 최댓값 찾는 함수

---

- ▶ 함수의 매개변수 자리에 배열 선언
  - ▶ 배열명은 포인터로 바뀜
    - ▶ 배열의 저장 공간이 할당되지 않음
    - ▶ 배열명은 컴파일 과정에서 첫 번째 배열 요소 포인터

---

<code>void func( int pa[5] ) { ... }</code>	<code>→</code>	<code>void func( int * pa ) { ... }</code>
<code>void func( int pa[10] ) { ... }</code>	<code>→</code>	<code>void func( int *pa ) { ... }</code>
<code>void func( int pa[] ) { ... }</code>	<code>→</code>	<code>void func( int *pa ) { ... }</code>
<code>void func( double pa[5] ) { ... }</code>	<code>→</code>	<code>void func( double *pa ) { ... }</code>

---

# 실수 배열에 값 입력 함수와 최댓값 찾는 함수

---

- ▶ 매개변수 자리에 선언된 배열
  - ▶ 배열 요소 수는 의미 없음 (생략가능)
- ▶ 함수의 매개변수에 처리할 배열과 같은 배열 선언
  - ▶ 함수 안에서 배열처럼 사용
- ▶ 컴파일러는 배열명을 자동으로 포인터로 바꾸고 모든 배열 요소를 사용할 수 있도록 포인터 연산 수행

# 실수 배열에 값 입력 함수와 최댓값 찾는 함수

---

## ➤ 매개변수 자리에 선언된 배열

---

```
// 배열 선언과 함수 호출
int ary[5] = {1, 2, 3, 4, 5};
print_ary(ary);                // 배열명을 주고 함수 호출

// 함수 정의
void print_ary(int pa[5])      // 매개변수 자리에 ary 배열과 같은 배열 선언
{
    int i;
    for(i = 0; i < 5; i++)
    {
        printf("%d ", pa[i]);  // *(pa + i) 연산으로 배열 요소의 값 출력
    }
}
```

---