

컴퓨터 프로그래밍 및 실습

강의자료 5

- 포인터 (pointer)

포인터(Pointer)

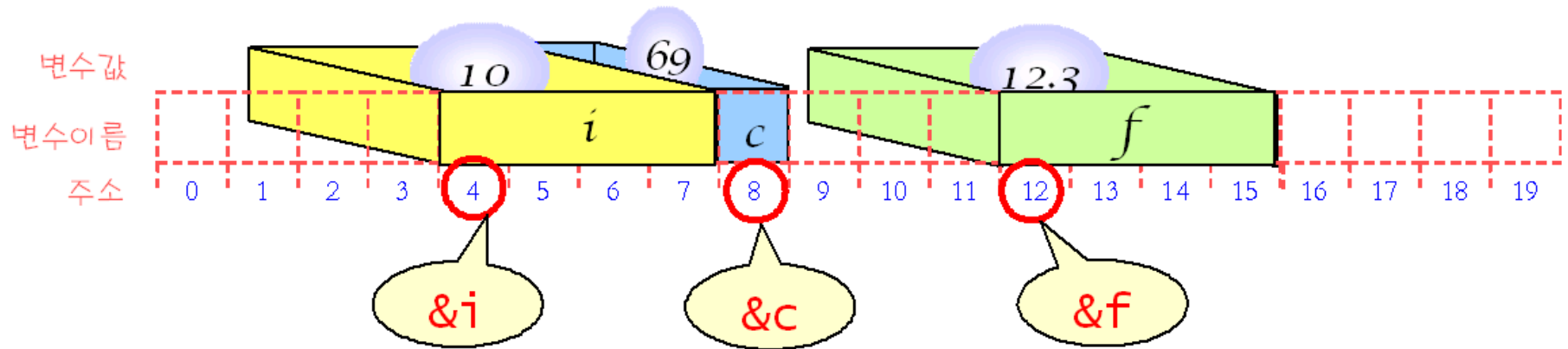
- 포인터 변수는 메모리의 주소를 저장하는 변수이다.
- Pointer 변수의 선언식

data_type * variable_name;

- Pointer 연산
 - ✓ 변수의 메모리 주소 값
&variable
 - ✓ Pointer 변수가 가리키는 기억장소
***pointer_variable**

변수의 주소

- 변수의 주소를 계산하는 연산자: &
- 변수 i의 주소: &i

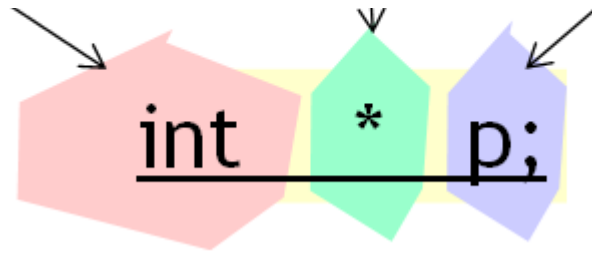


포인터의 선언

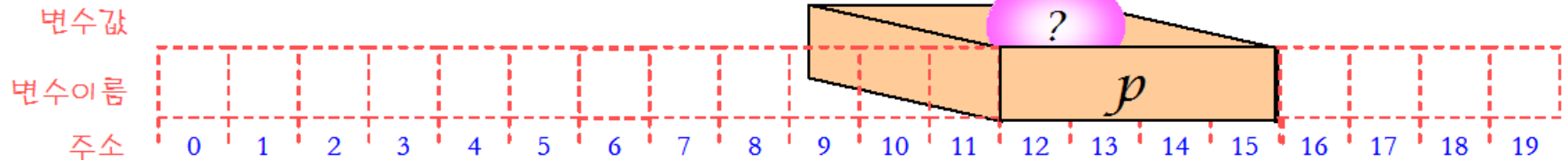
- 포인터: 변수의 주소를 가지고 있는 변수

*p가 가리키는
내용은 정수가 된다.

p는 포인터가 된다.

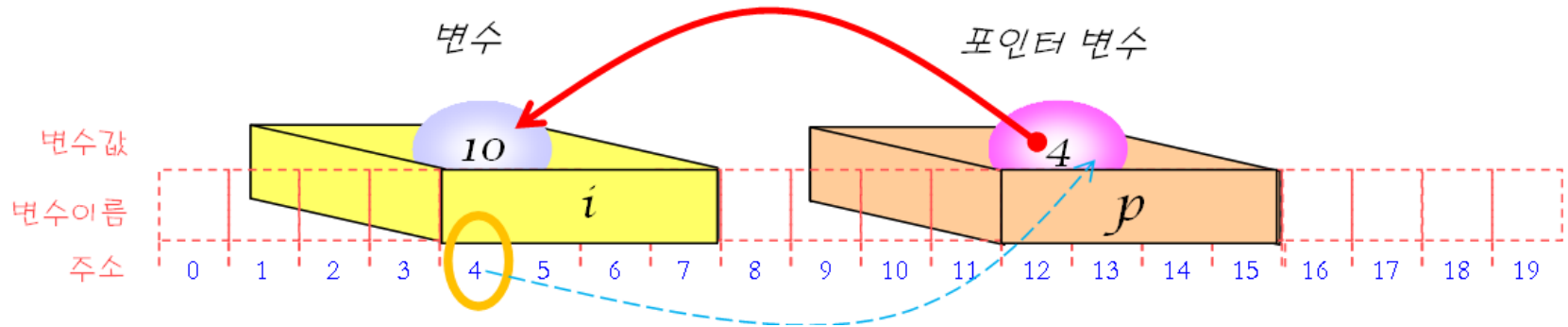


포인터 변수



포인터와 변수의 연결

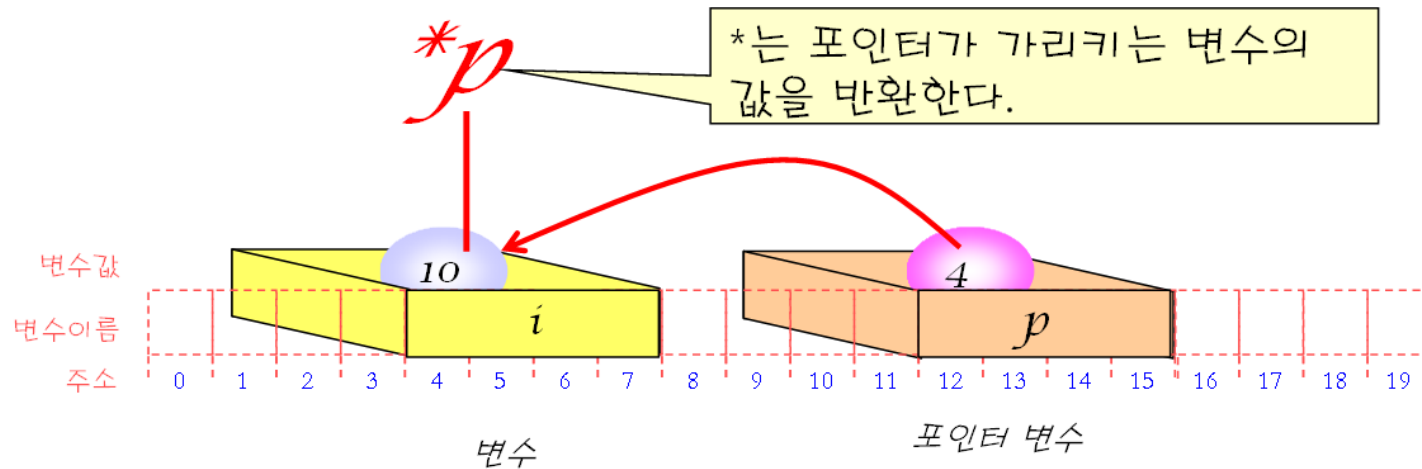
```
int i = 10;      // 정수형 변수 i 선언  
int *p;          // 포인터 변수 p 선언  
p = &i;          // 변수 i의 주소가 포인터 p로 대입
```



간접 참조 연산자

- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자

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



포인터 변수의 선언과 사용 예(1)

```
int i = 33;
int j = 44;
int k = 0;
```

i	1000	33 -> 44
j	1004	44 -> 88
k	1008	0 -> 44
p	1012	1004
q	1016	

```
int *p, *q;    /*read as “p is a pointer to integer” – in declaration */
scanf(" %d ", &i);

p = &j;        /* p is assigned 1004 – that is address of j */
i = j;         /* i is assigned 44, that is value of j */

k = *p;        /*read as “dereference p” – in expression */
               /* value of p is 1004 . “dereference 1004” means
               /* “use 1004 as address & fetch value from address 1004 → 44 */

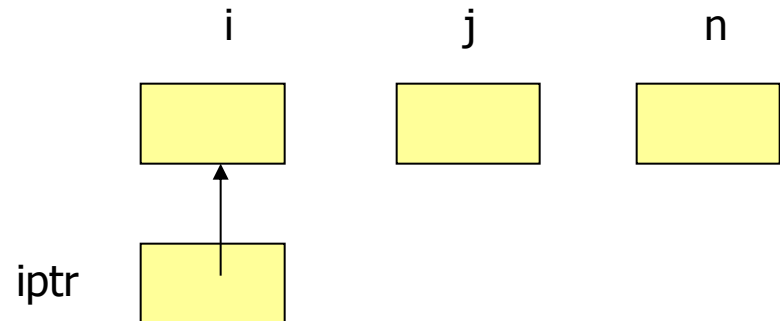
*p = 88;       /* “dereference p”. Store, NOT in p (loc. 1012),
               /* but in loc. 1004 (use p’s value as address & store there) */
```

포인터 변수의 선언과 사용 예(2)

```
int i, j, n;  
int *iptr;
```

```
i = 0;  
iptr = &i;  
n = 17;  
j = 23;  
n = *iptr;  
*iptr = j;  
*iptr = *iptr + 10;
```

		...
i	1000	0->23->33
j	1004	23
n	1008	17->0
iptr	1012	1000
	1016	...



* 의 세 가지 다른 의미

- 곱셈 연산자 (이항 연산)

$a = b * c;$

- Pointer variable 선언

- void separate (double num, char* signp, int* wholep, double* fracp)

- Expression에서

- “포인터가 가리키는 곳” (**dereferencing**)
 - *signp = '-';
 - /* signp stores the address of char. NOT char itself */
 - /* follow signp address – cell for a character */


```
char    c1, c2, c3;  
char    *signp;  
signp = &c2;  
c1 = * signp;  
* signp = 'A';
```

포인터 증감 연산

- 증가 연산의 경우 증가되는 값은 포인터가 가리키는 객체의 크기

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

포인터의 증가는
일반 변수와는 약간
다릅니다. 가리키는 객체의
크기만큼 증가합니다.


p++

포인터 이용

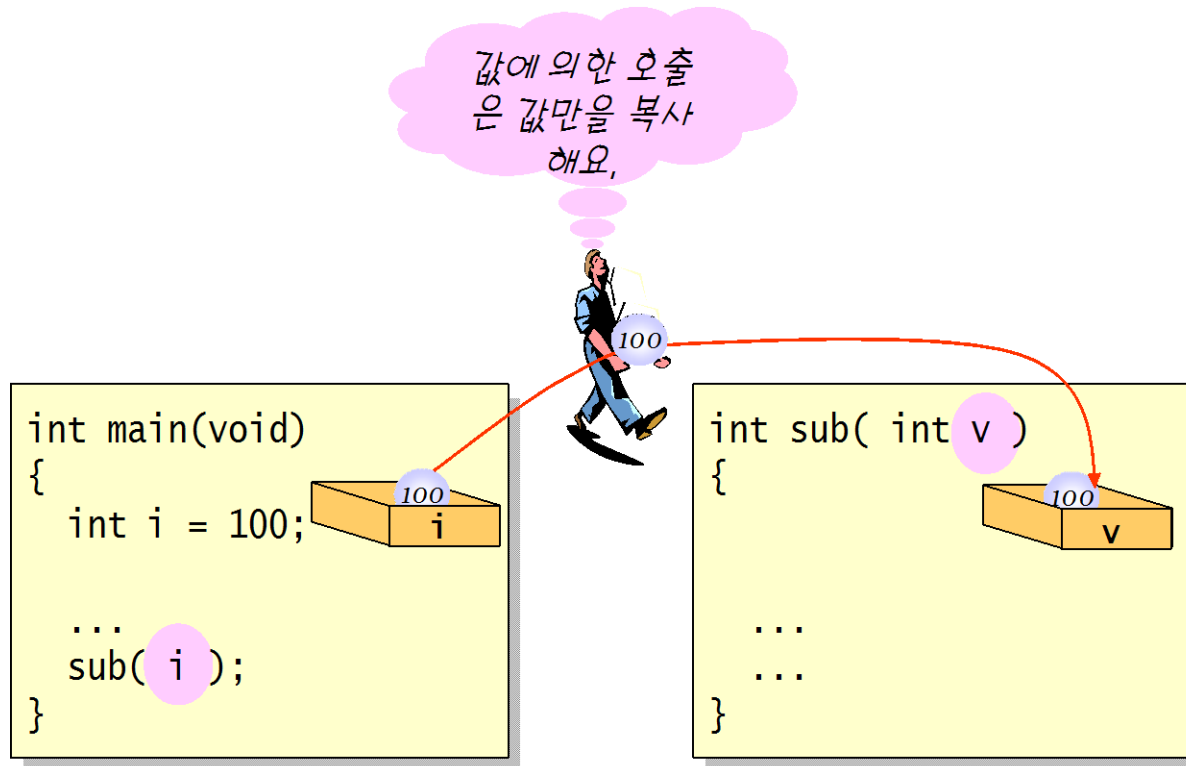
- 함수의 포인터 형식 인자를 통한 실제 인자 접근(access)
 - 실제인자 배열을 형식인자로 받을 때, 포인터를 이용할 수 있다
- 동적할당

인수 전달 방법

- 함수 호출 시에 인수 전달 방법
 - 값에 의한 호출(call by value)
 - C에서 기본적인 방법
 - 참조에 의한 호출(call by reference)
 - C에서는 포인터를 이용하여 흉내 낼 수 있다.

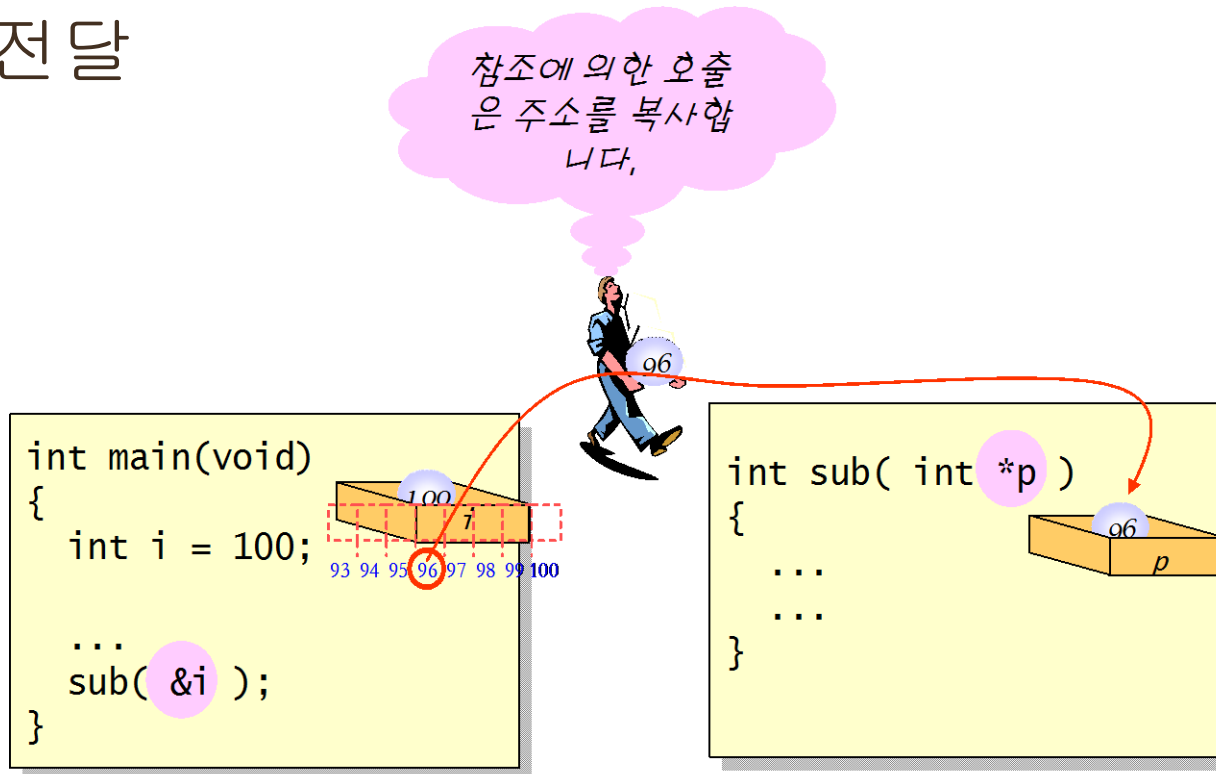
값에 의한 호출

- 함수 호출시에 변수의 값을 함수에 전달



포인터를 사용한 인자전달

- 함수 호출시에 변수의 주소를 함수의 매개 변수로 전달



swap() 함수 #1

- 변수 2개의 값을 바꾸는 작업을 함수

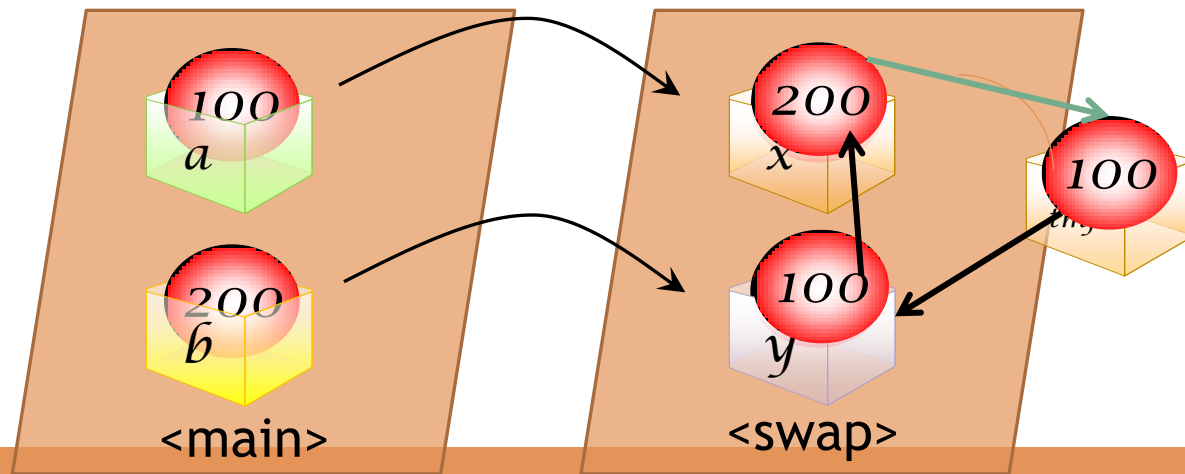
```
#include <stdio.h>
void swap(int x, int y);
int main(void)
{
    int a = 100, b = 200;

    swap(a, b);
    return 0;
}
```

```
void swap(int x, int y)
{
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
}
```

함수 호출시에 값만 복사된다.



swap() 함수 #2

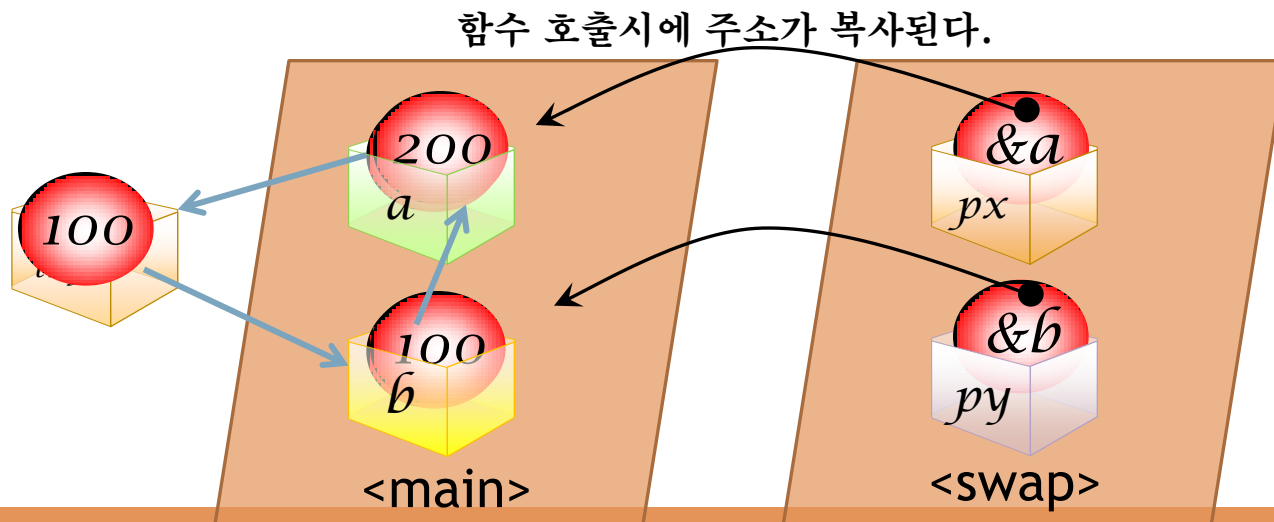
● 포인터를 이용

```
#include <stdio.h>
void swap(int x, int y);
int main(void)
{
    int a = 100, b = 200;
    swap(&a, &b);

    return 0;
}
```

```
void swap(int *px, int *py)
{
    int tmp;

    tmp = *px;
    *px = *py;
    *py = tmp;
}
```



포인터 사용시 주의점

- 포인터가 아무것도 가리키고 있지 않는 경우에는 NULL로 초기화

포인터와 배열

- 배열과 포인터는 아주 밀접한 관계를 가지고 있다.
- 배열 이름이 바로 포인터이다.
- 포인터는 배열처럼 사용이 가능하다.

- `int a[9];`

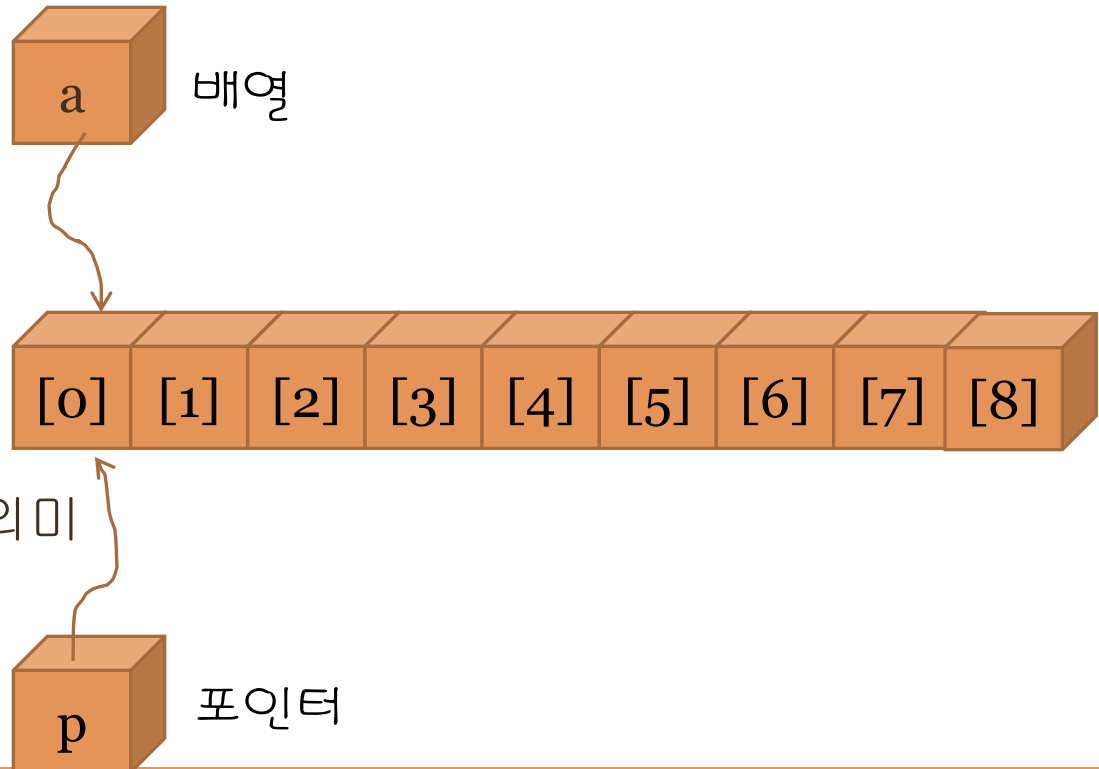
// a는 `&a[0]` 값을 가진다.

`int *p = a;`

// p와 a, `&a[0]`는 같은 값

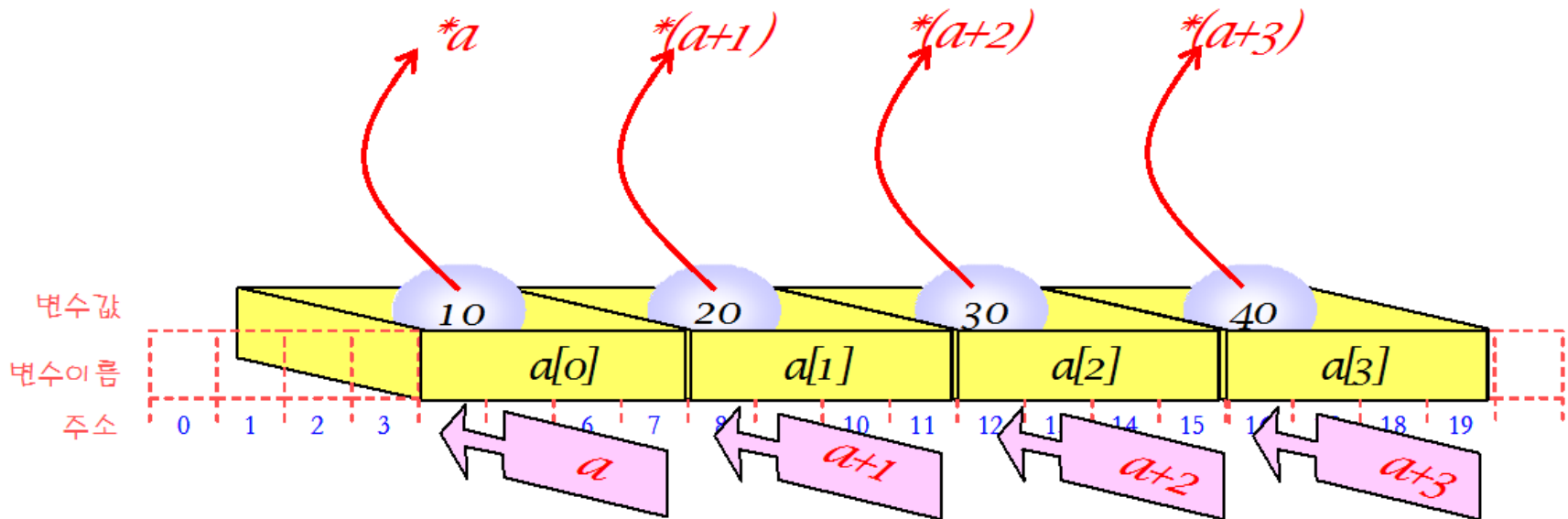
`*(p + 1) = 3;`

// `*(p + 1)`는 `p[1]`과 같은 의미



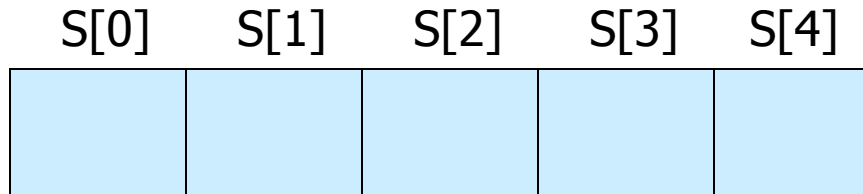
포인터와 배열

- 포인터는 배열처럼 사용할 수 있다.
- 인덱스 표기법을 포인터에 사용할 수 있다.



배열과 포인터

- int S[5];



배열 이름 S는 &S[0]와 같음

```
*S = 10;      // S[0] = 10;  
*(S+1) = 20 ; // S[1] = 20;  
*(S+i) = 30;  // S[i] = 30;
```

배열 매개변수 vs. Pointer 매개변수

- Array 변수는 pointer 변수로 취급되기 때문에 함수 정의에서 array 변수를 인자로 사용하는 것과 pointer 변수를 인자로 사용하는 것은 동등함

보기:

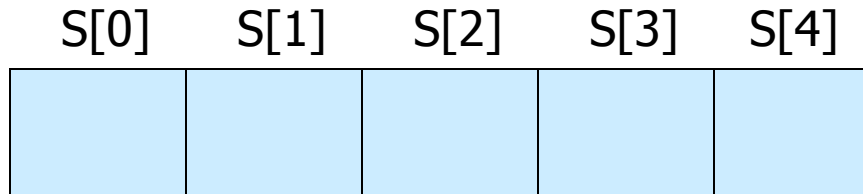
/* 아래의 두 prototype은 동등함 */

```
double sum(double a[], int n);
```

```
double sum(double *a, int n);
```

배열과 포인터

- int S[5];



배열 이름 S는 &S[0]와 같음

```
*S = 10;      // S[0] = 10;  
*(S+1) = 20;  // S[1] = 20;  
*(S+i) = 30;  // S[i] = 30;
```

배열 매개변수 예 - 배열의 원소들의 합을 구하는 함수

방법 1

```
double sum(double a[], int n)
{ double sum = 0.0;
  int i;
  for (i = 0; i < n; i++)
    sum += a[i];

  return sum;
}
```

방법 2

```
double sum(double* a, int n)
{ double sum = 0.0;
  int i;
  for (i = 0; i < n; i++)
    sum += a[i]; // sum += *(a+i);

  return sum;
}
```

함수 사용 예

sum(v, 100); // sum(&v[0],100);

sum(v, 88);

sum(&v[7], k-7);