누구나 즐기는 C언어 콘서트

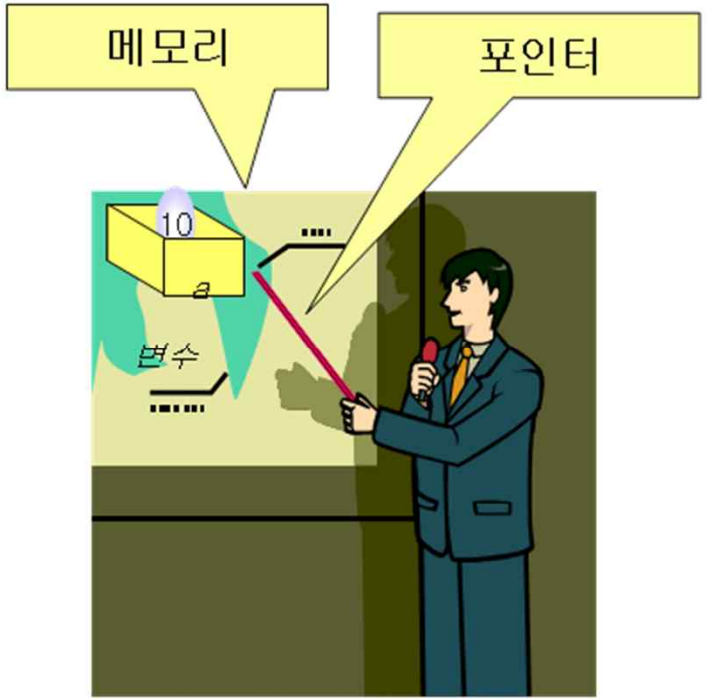
제9장 포인터

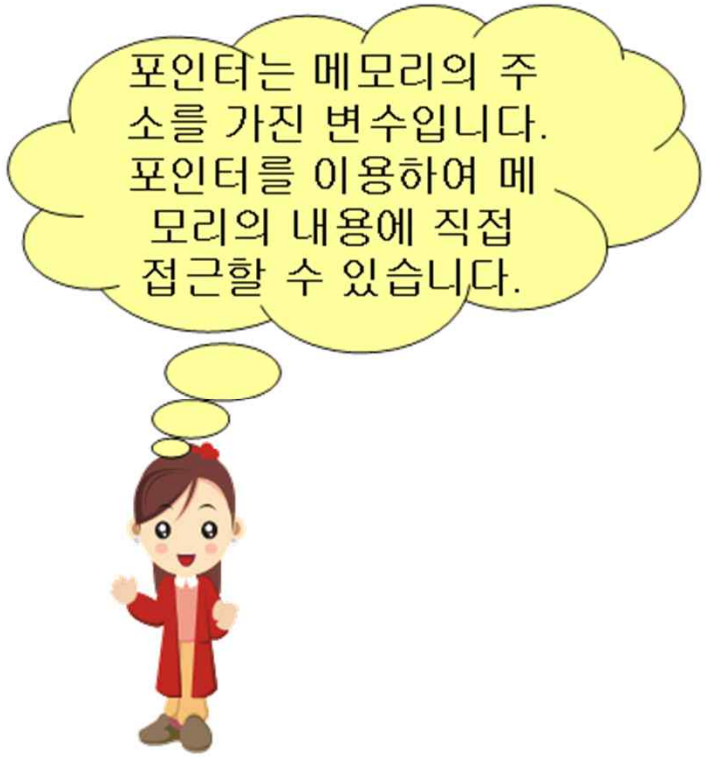
이번 장에서 학습할 내용

- 포인터이란?
- 변수의 주소
- 포인터의 선언
- 간접 참조 연산자
- 포인터 연산
- 포인터와 배열
- 포인터와 함수

포인터란?

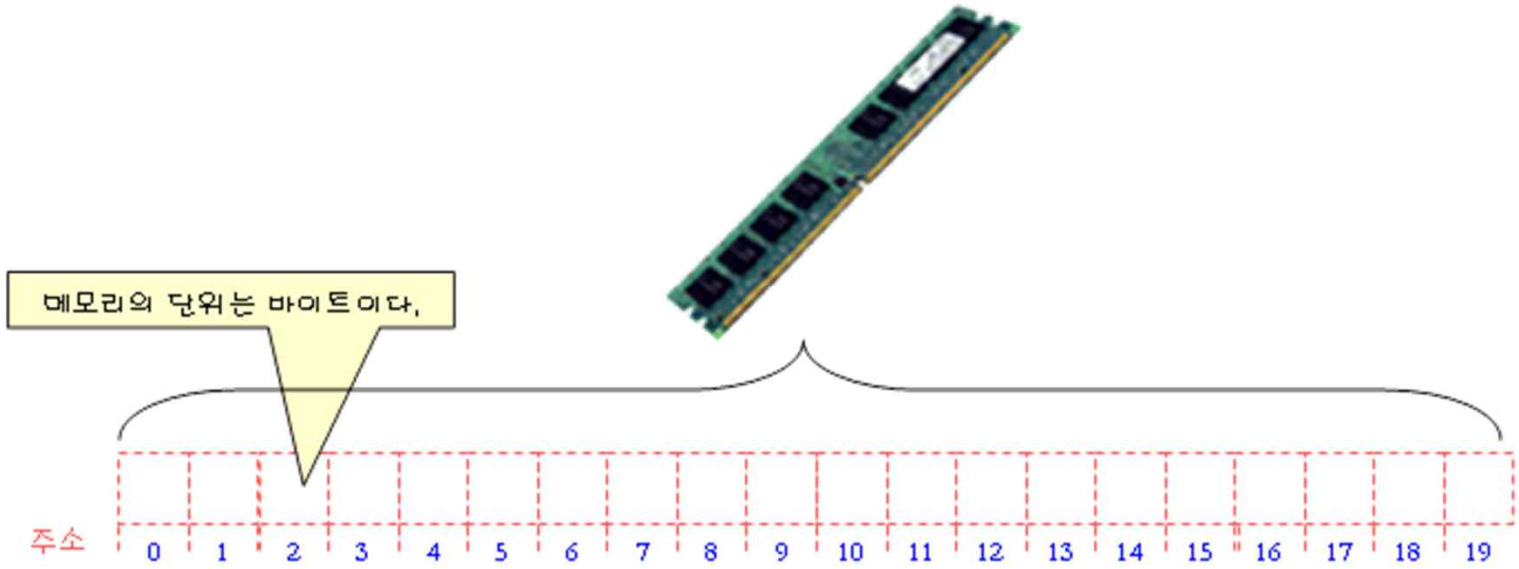
- *포인터(pointer):* 주소를 가지고 있는 변수





메모리의 구조

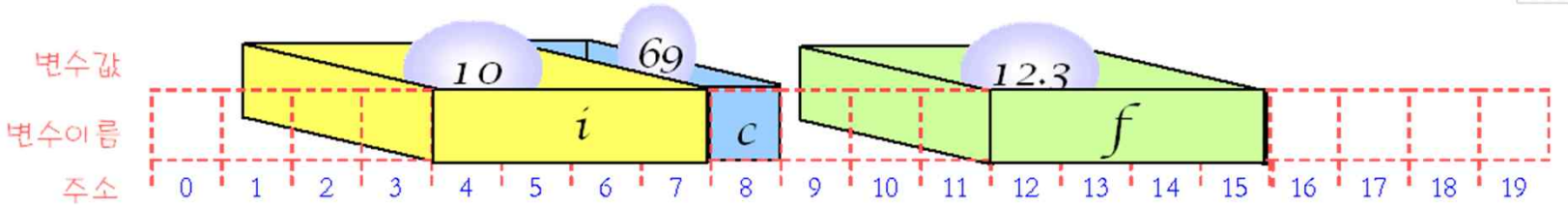
- 변수는 메모리에 저장된다.
- 메모리는 바이트 단위로 액세스된다.
  - 첫번째 바이트의 주소는 0, 두번째 바이트는 1,...



변수와 메모리

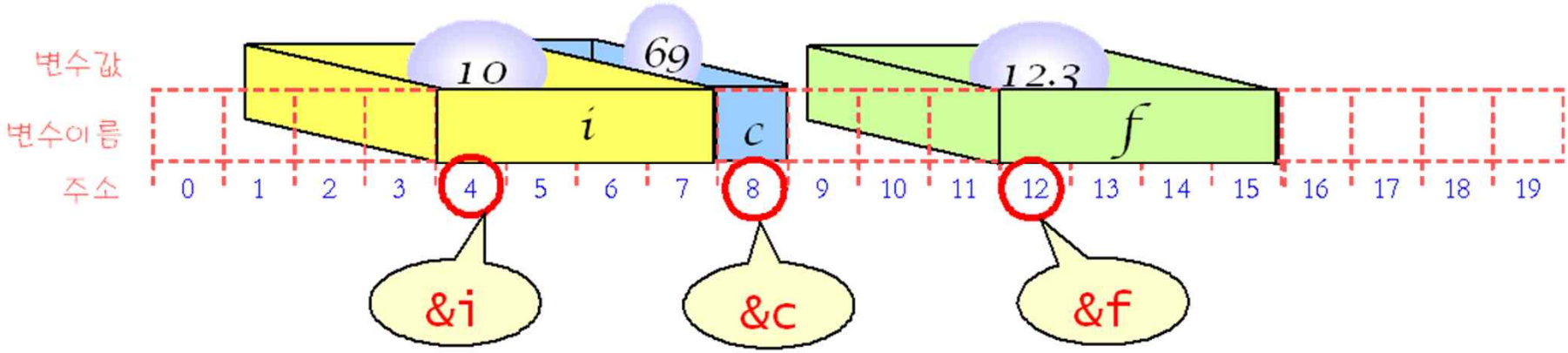
- 변수의 크기에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,...

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



변수의 주소

- 변수의 주소를 계산하는 연산자: &
- 변수 i의 주소: &i



변수의 주소

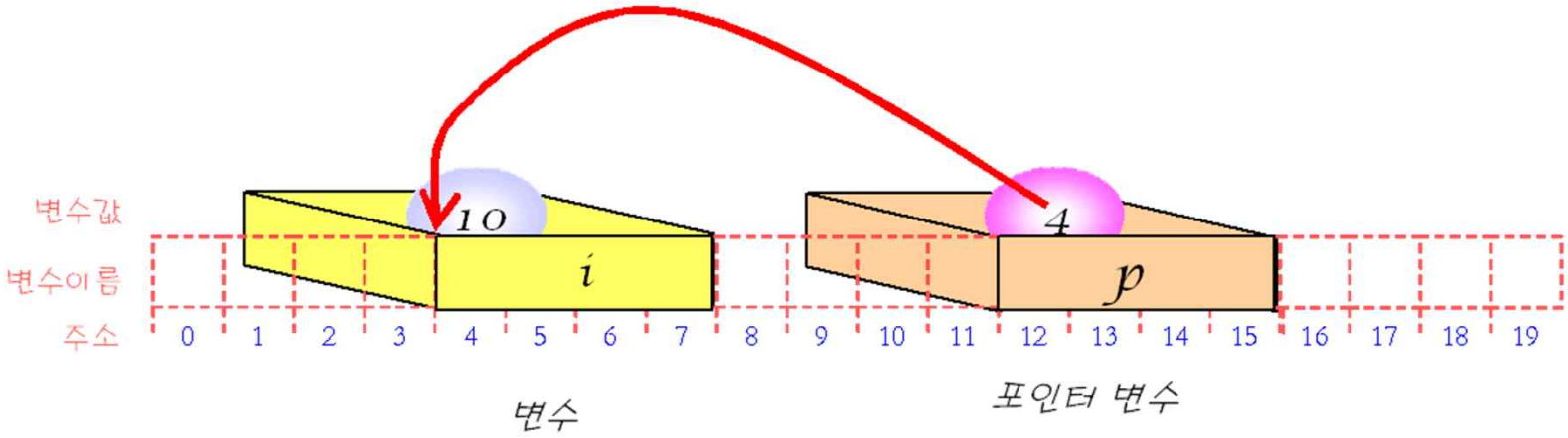
```
int main(void)
{
      int i = 10;
      char c = 69;
      float f = 12.3;

      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
```

포인터의 선언

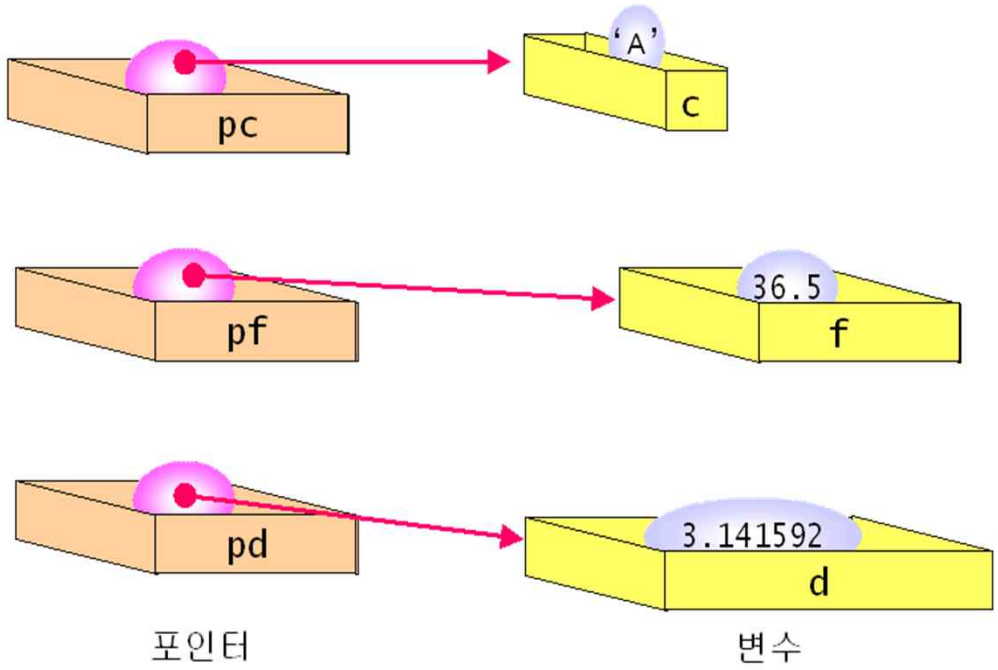
- 포인터: 변수의 주소를 가지고 있는 변수

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



다양한 포인터의 선언

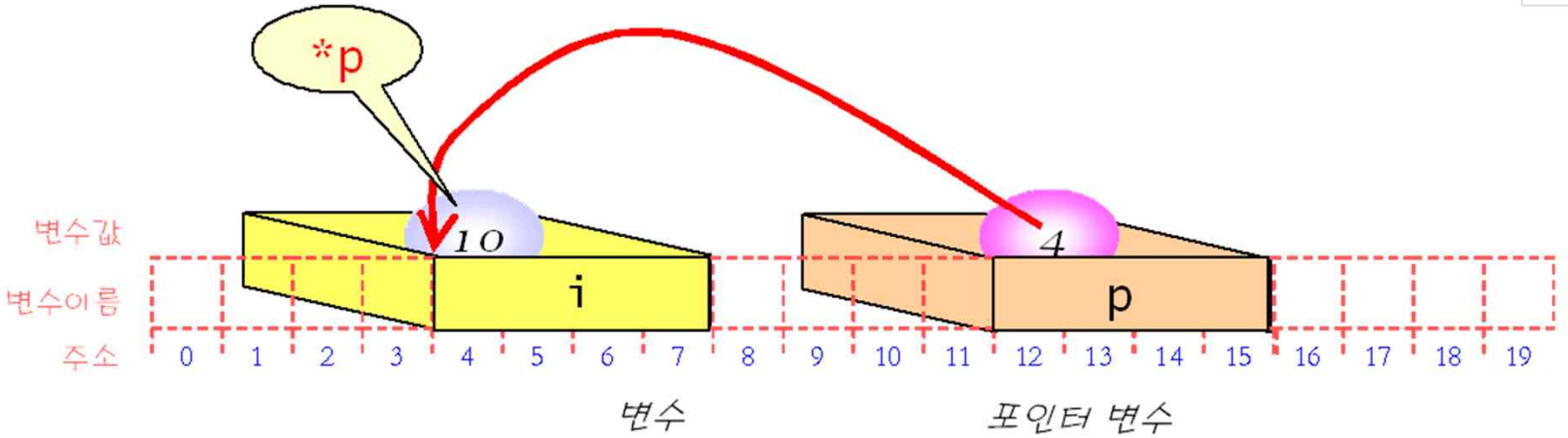
```
char c = 'A';                    // 문자형 변수 c
float f = 36.5;                  // 실수형 변수 f
double d = 3.141592;             // 실수형 변수 d

char *pc = &c;                   // 문자를 가리키는 포인터 pc
float *pf = &f;                  // 실수를 가리키는 포인터 pf
double *pd = &d;                 // 실수를 가리키는 포인터 pd
```



간접 참조 연산자

- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자

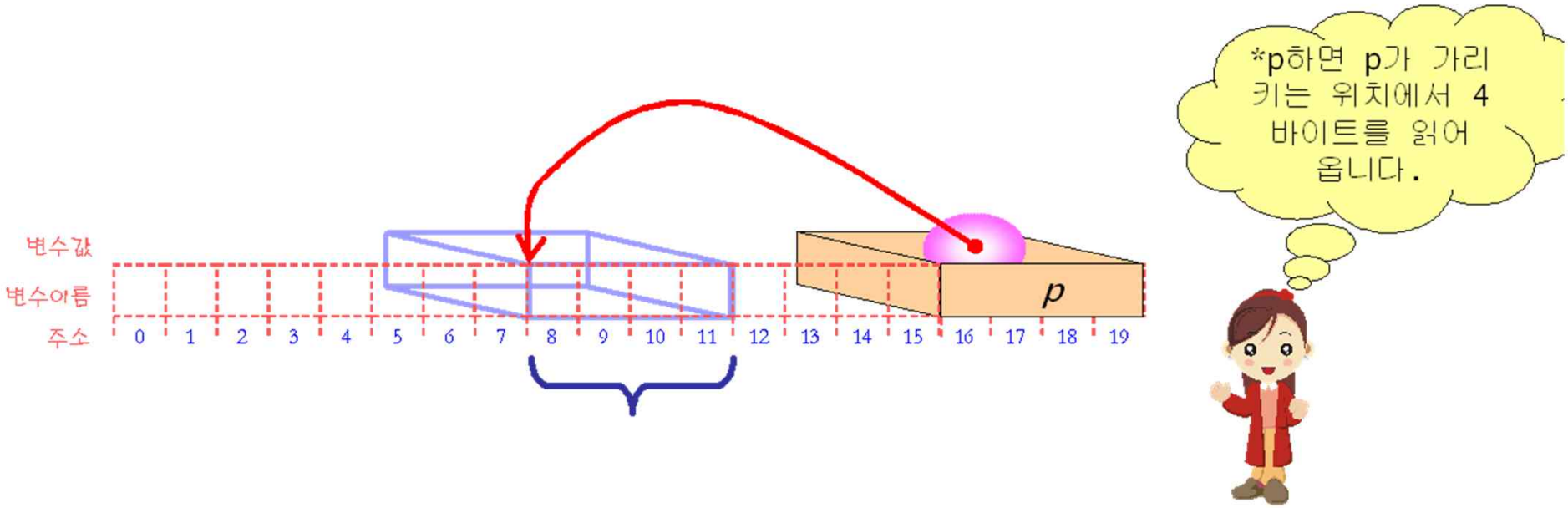
```
int i = 10;
int *p;
p = &i;

printf("%d\n", *p);    // 10이 출력된다.
```



간접 참조 연산자의 해석

- 간접 참조 연산자: 지정된 위치에서 포인터의 타입에 따라 값을 읽어 들인다.

```
int *p = 8;                    // 위치 8에서 정수를 읽는다.
char *pc = 8;                  // 위치 8에서 문자를 읽는다.
double *pd = 8;                // 위치 8에서 실수를 읽는다.
```



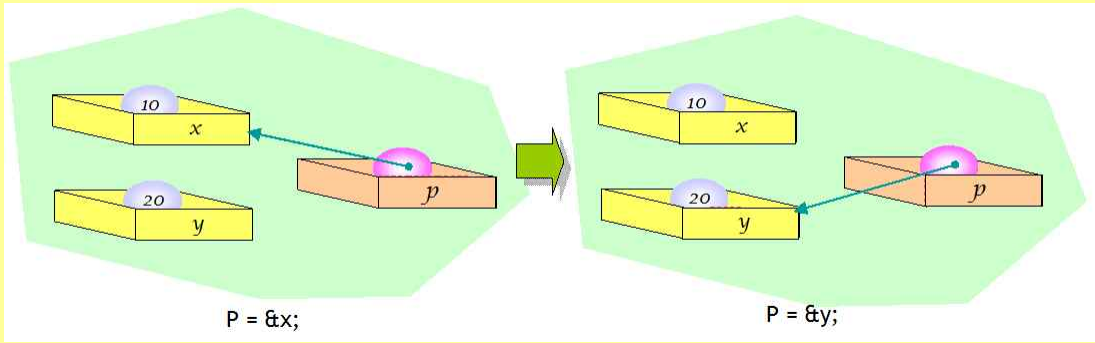
포인터 예제 #1

```c
#include <stdio.h>

int main(void)
{
    int i = 3000;
    int *p;
    p = &i;                 // 변수와 포인터 연결

    printf("i = %d\n", i);          // 변수의 값 출력
    printf("&i = %u\n", &i);        // 변수의 주소 출력

    printf("p = %u\n", p);          // 변수의 값 출력
    printf("*p = %d\n", *p);        // 포인터를 통한 간접 참조 값 출력
    return 0;
}
```

```
i = 3000
&i = 1245024
p = 1245024
*p = 3000
```

포인터 예제 #2

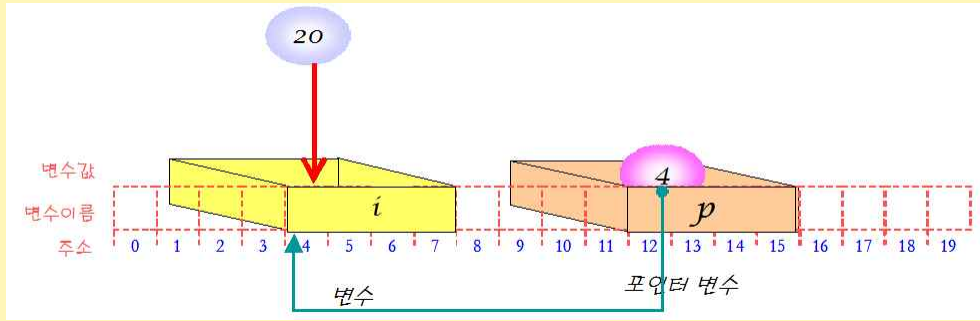
```
#include <stdio.h>
int main(void)
{
      int x=10, y=20;
      int *p;

      p = &x;
      printf("p = %d\n", p);
      printf("*p = %d\n\n", *p);

      p = &y;
      printf("p = %d\n", p);
      printf("*p = %d\n", *p);
      return 0;
}
```



```
p = 1245052
*p = 10
p = 1245048
*p = 20
```

포인터 예제 #3

```
#include <stdio.h>
int main(void)
{
    int i=10;
    int *p;

    p = &i;
    printf("i = %d\n", i);

    *p = 20;
    printf("i = %d\n", i);
    return 0;
}
```



```
i = 10
i = 20
```

중간 점검

1. 메모리는 어떤 단위를 기준으로 주소가 매겨지는가?
2. 다음의 각 자료형이 차지하는 메모리 공간의 크기를 쓰시오.

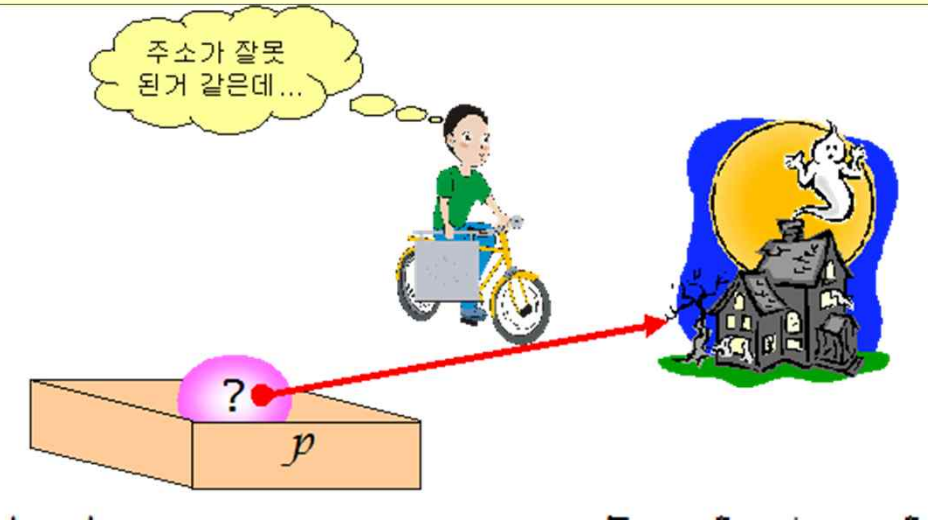
(a) char (b) short (c) int (d) long (e) float (f) double

3. 포인터도 변수인가?
4. 변수의 주소를 추출하는데 사용되는 연산자는 무엇인가?
5. 변수 x의 주소를 추출하여 변수 p에 대입하는 문장을 쓰시오.
6. 정수형 포인터 p가 가리키는 위치에 25를 저장하는 문장을 쓰시오.

포인터 사용시 주의점 #1

- 초기화가 안된 포인터를 사용하면 안된다.

```
int main(void)
{
    int *p;          // 포인터 p는 초기화가 안되어 있음

    *p = 100;        // 위험한 코드
    return 0;
}
```



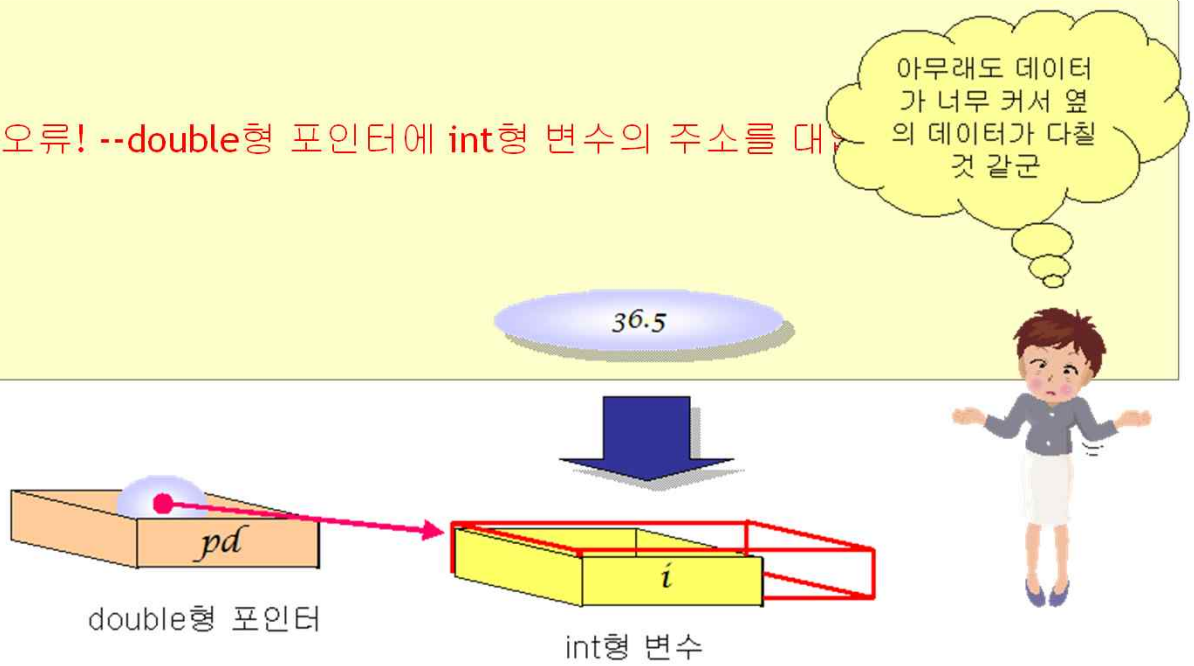
포인터 사용시 주의점 #2

- 포인터가 아무것도 가리키고 있지 않는 경우에는 NULL로 초기화
- NULL 포인터를 가지고 간접 참조하면 하드웨어로 감지할 수 있다.
- 포인터의 유효성 여부 판단이 쉽다.

포인터 사용시 주의점 #3

- 포인터의 타입과 변수의 타입은 일치하여야 한다.

```
#include <stdio.h>

int main(void)
{
      int i;
      double *pd;

      pd = &i;            // 오류! --double형 포인터에 int형 변수의 주소를 대
      *pd = 36.5;

      return 0;
}
```



포인터 연산

- 가능한 연산: 증가, 감소, 덧셈, 뺄셈 연산
- 증가 연산의 경우 증가되는 값은 포인터가 가리키는 객체의 크기

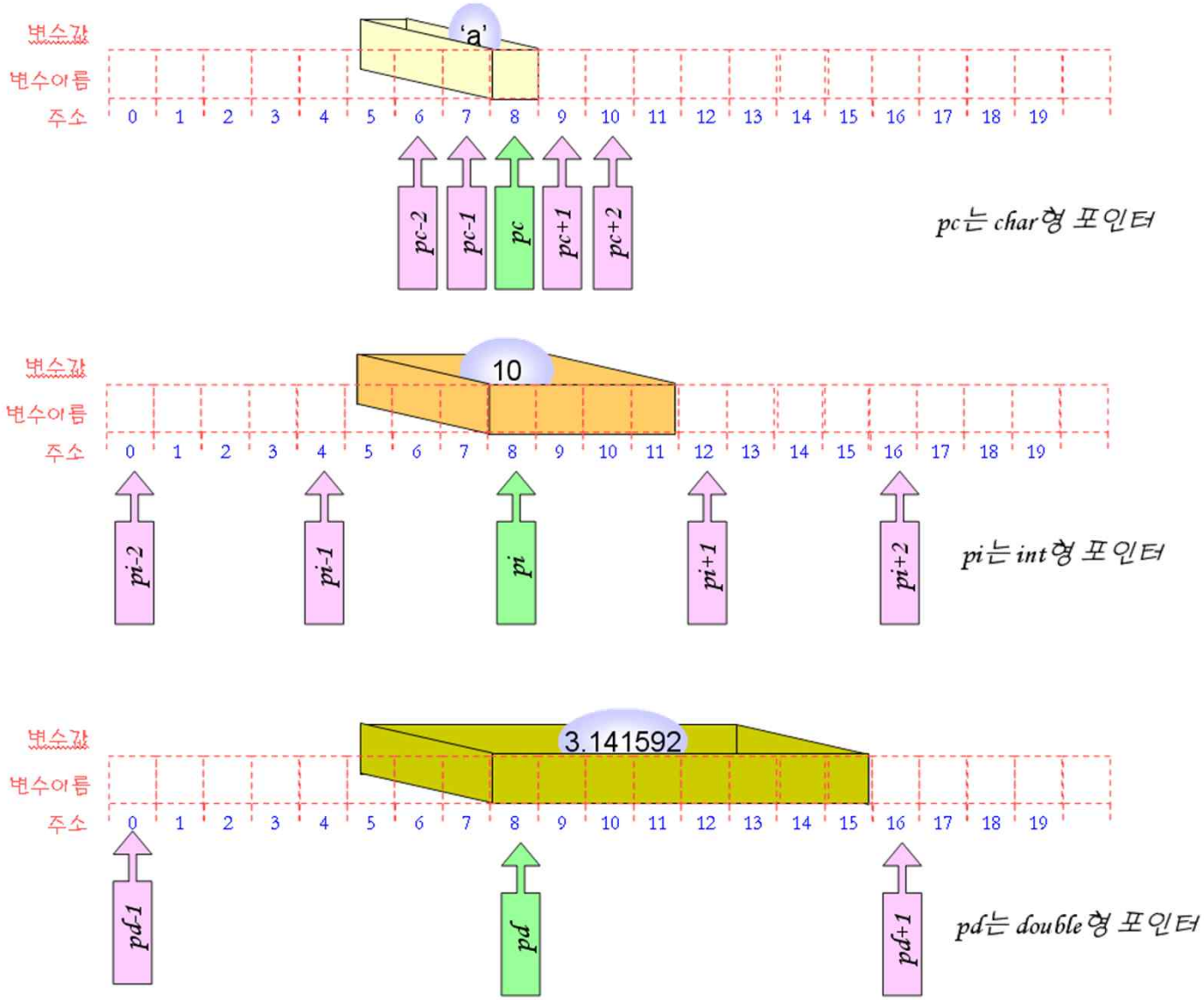
포인터 타입	++연산후 증가되는값
char	1
short	2
int	4
float	4
double	8

증가 연산 예제

```c
#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
```

포인터의 증감 연산



간접 참조 연산자와 증감 연산자

수식	의미
v = *p++	p가 가리키는 값을 v에 대입한 후에 p를 증가한다.
v = (*p)++	p가 가리키는 값을 v에 대입한 후에 가리키는 값을 증가한다.
v = *++p	p를 증가시킨 후에 p가 가리키는 값을 v에 대입한다.
v = ++*p	p가 가리키는 값을 가져온 후에 그 값을 증가하여 v에 대입한다.

간접 참조 연산자와 증감 연산자

```
// 포인터의 증감 연산
#include <stdio.h>

int main(void)
{
    int i = 10;
    int *pi = &i;

    printf("i = %d,  pi = %p\n", i, pi);
    (*pi)++;
    printf("i = %d,  pi = %p\n", i, pi);

    printf("i = %d,  pi = %p\n", i, pi);
    *pi++;
    printf("i = %d,  pi = %p\n", i, pi);

    return 0;
}
```

```
i = 10, pi = 0012FF60
i = 11, pi = 0012FF60
i = 11, pi = 0012FF60
i = 11, pi = 0012FF64
```

중간 점검

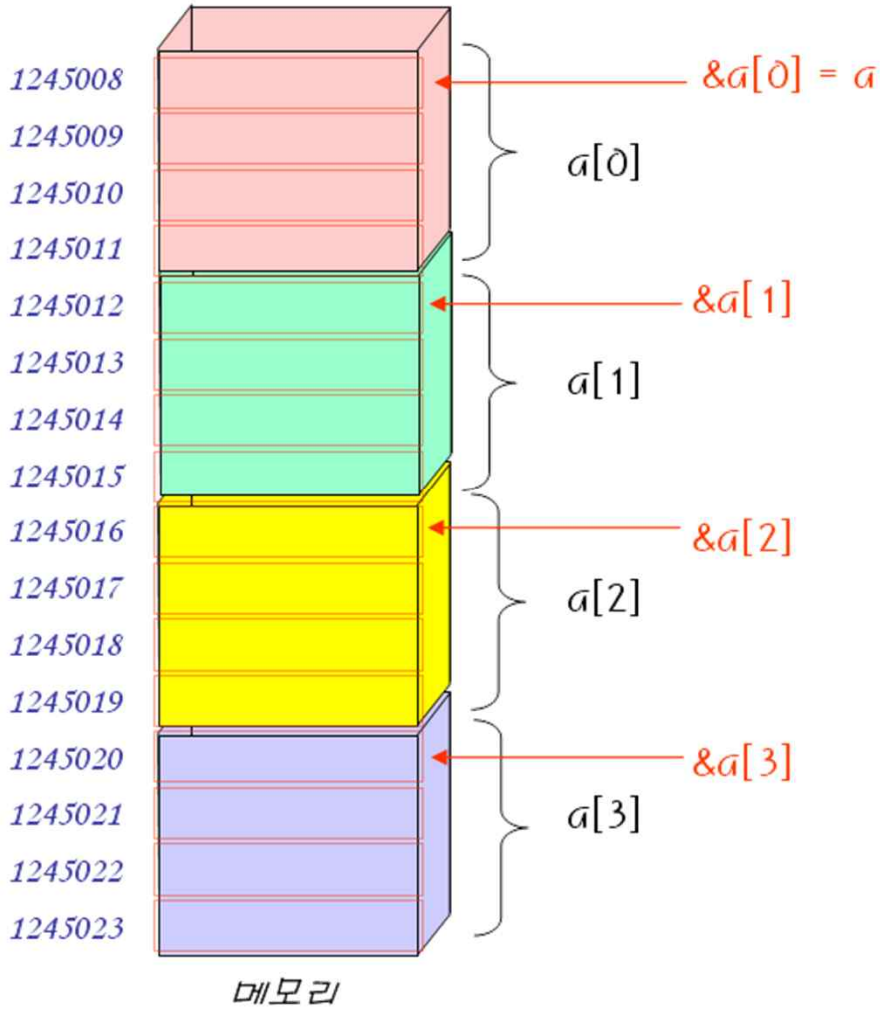
1. 포인터에 대하여 적용할 수 있는 연산에는 어떤 것들이 있는가?
2. int형 포인터 p가 80번지를 가리키고 있었다면 (p+1)은 몇 번지를 가리키는가?
3. p가 포인터라고 하면 *p++와 (*p)++의 차이점은 무엇인가?
4. p가 포인터라고 하면 *(p+3)의 의미는 무엇인가?

포인터와 배열

```
#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
```

포인터와 배열

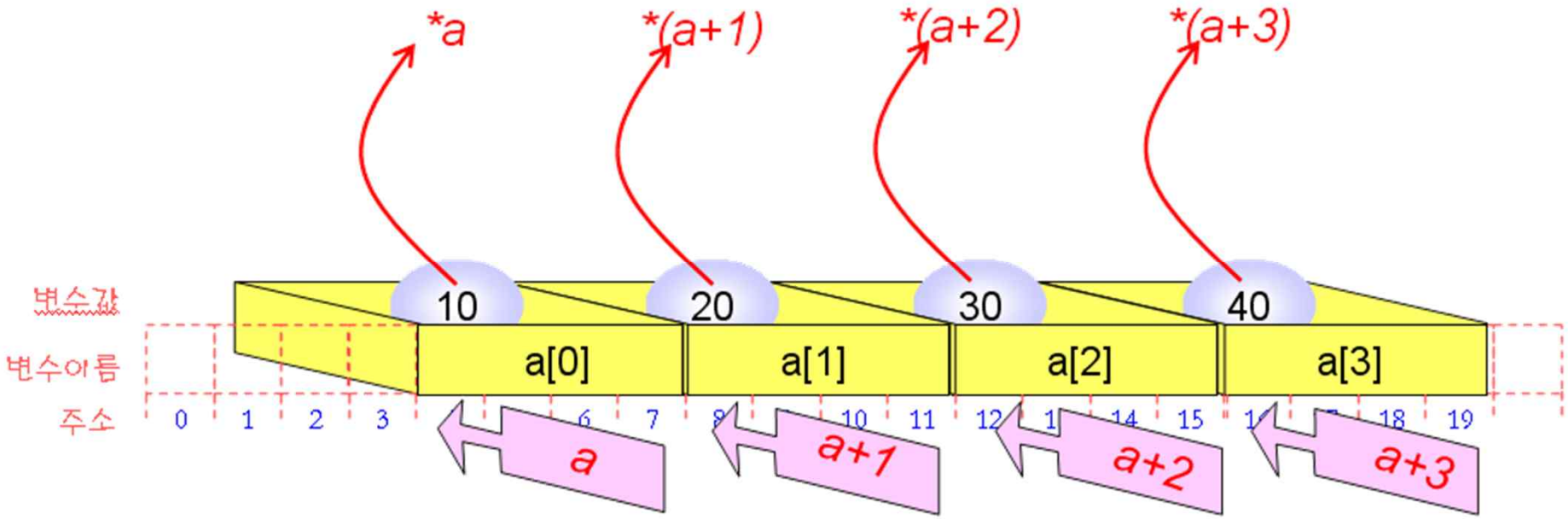


포인터와 배열

```
// 포인터와 배열의 관계
#include <stdio.h>

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

    printf("a = %u\n", a);
    printf("a + 1 = %u\n", a + 1);
    printf("*a = %d\n", *a);
    printf("*(a+1) = %d\n", *(a+1));
    return 0;
}
```

```
a = 1245008
a + 1 = 1245012
*a = 10
*(a+1) = 20
```

포인터와 배열



포인터를 배열처럼 사용

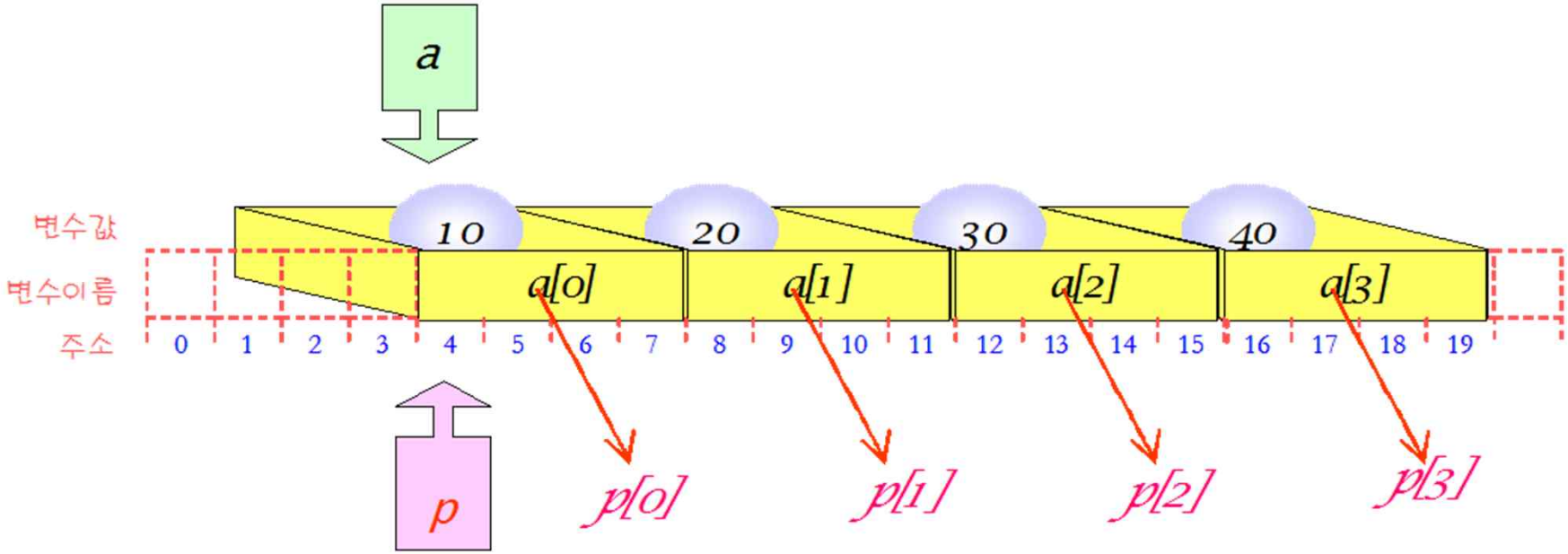
```c
// 포인터를 배열 이름처럼 사용
#include <stdio.h>

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]);
```

포인터를 배열처럼 사용

```
    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
```

포인터를 배열처럼 사용



중간 점검

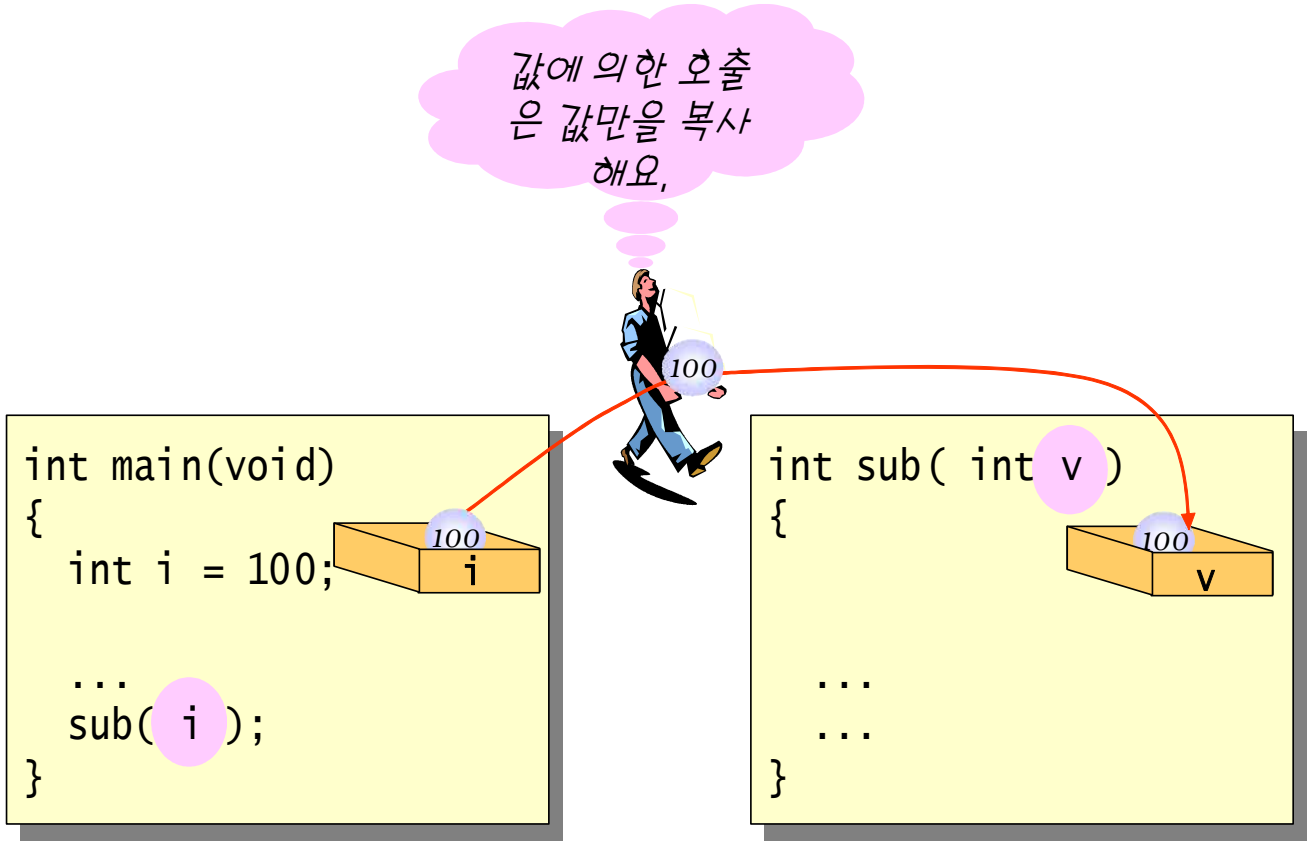
1. 배열의 첫 번째 원소의 주소를 계산하는 2가지 방법을 설명하라.
2. 배열 a[]에서 *a의 의미는 무엇인가?
3. 배열의 이름에 다른 변수의 주소를 대입할 수 있는가?
4. 포인터를 이용하여 배열의 원소들을 참조할 수 있는가?
5. 포인터를 배열의 이름처럼 사용할 수 있는가?

함수 호출시 인수 전달 방법

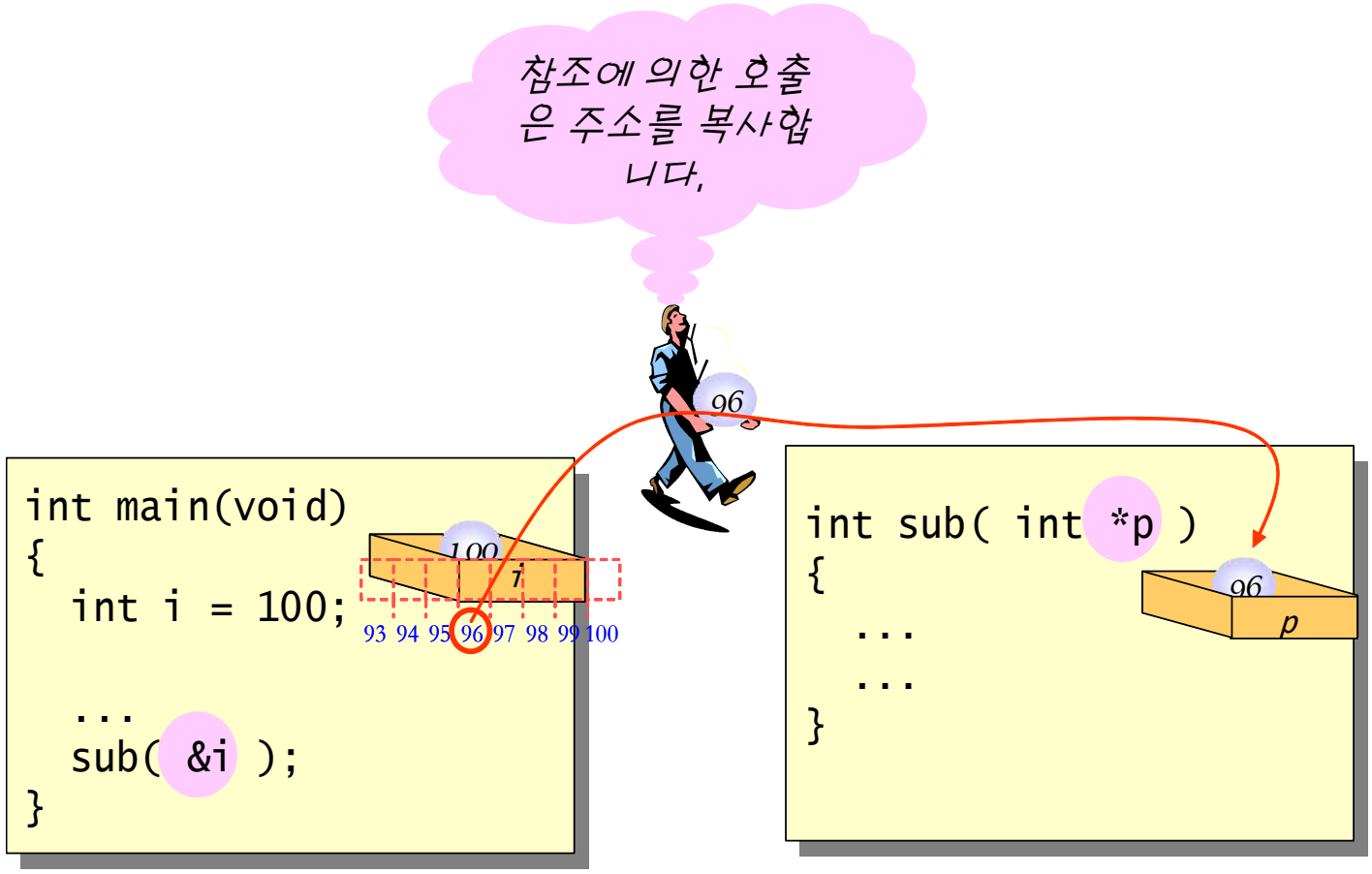
- 값에 의한 호출(call-by-value)
  - C의 기본적인 방법
  - 인수의 값만이 함수로 복사된다.
  - 복사본이 전달된다고 생각하면 된다.

- 참조에 의한 호출(call-by-reference)
  - C에서는 포인터를 이용하여 흉내낼 수 있다.
  - 인수의 주소가 함수로 복사된다.
  - 원본이 전달된다고 생각하면 된다.

값에 의한 호출



참조에 의한 호출

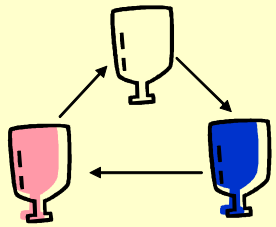


swap() 함수 #1

- 변수 2개의 값을 바꾸는 작업을 함수로 작성

```c
#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;
}
```

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

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

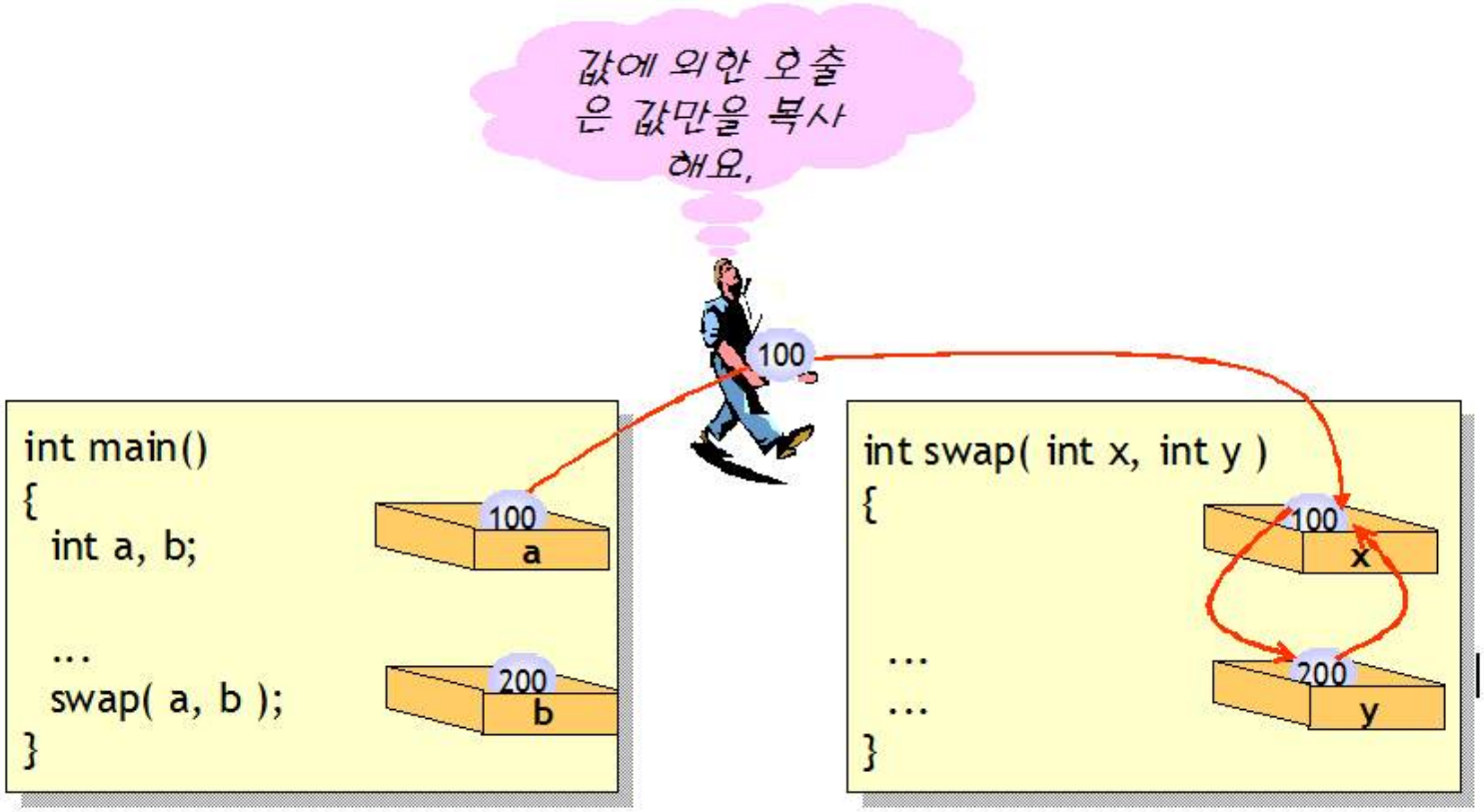


```
a=100 b=200
a=100 b=200
```

변경되지 않음!!
Why?

swap() 함수 #1

- 값들이 복사되었고 원본 변수에는 아무런 영향이 없다.

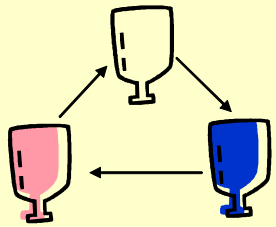


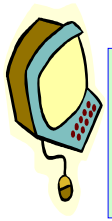
swap() 함수 #2

- 포인터를 이용

```
#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 *px, int *py)
{
    int   tmp;

    tmp = *px;
    *px = *py;
    *py = tmp;

}
```

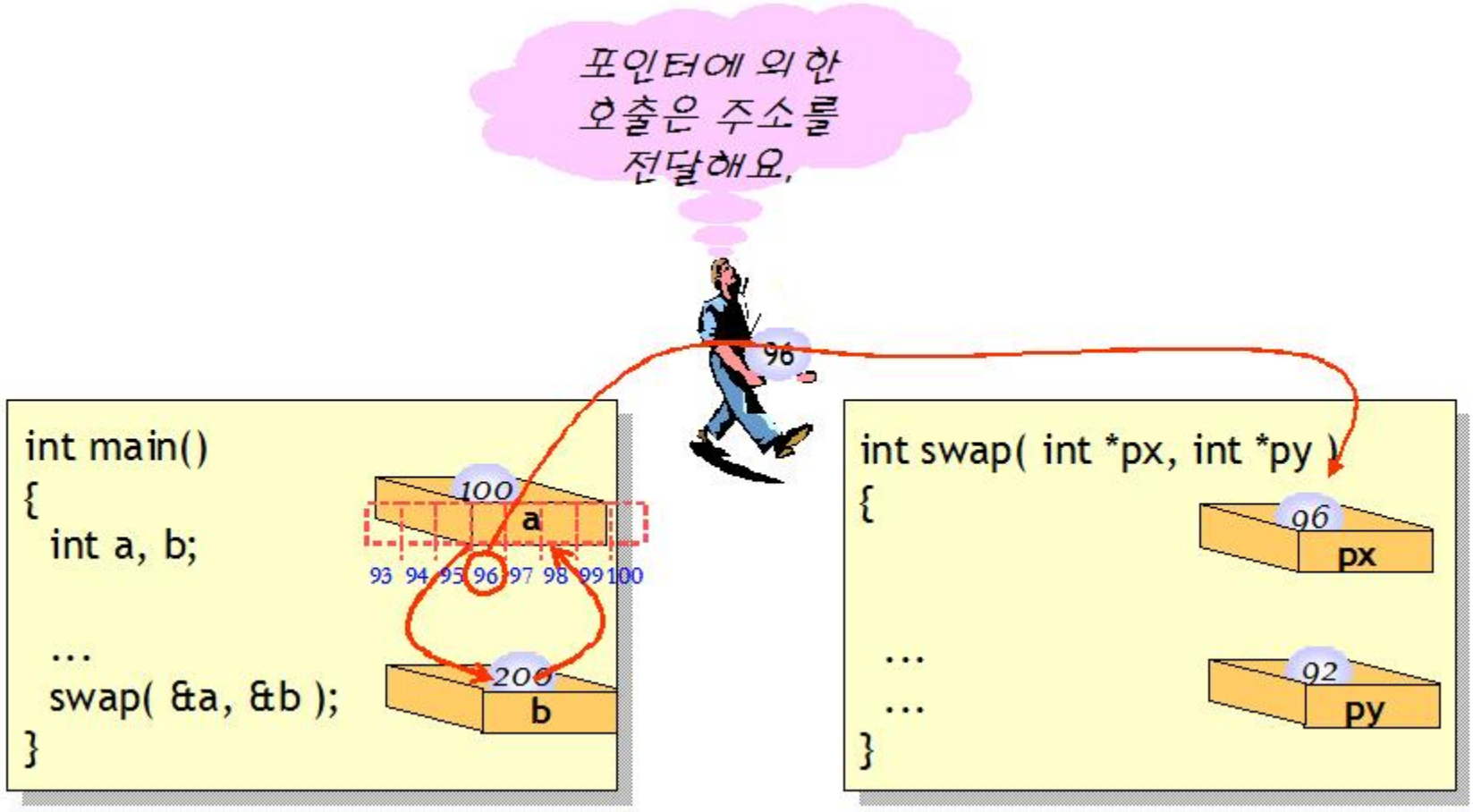




```
a=100 b=200
a=200 b=100
```

변경되었음!!

swap() 함수 #2



2개 이상의 결과를 반환

```
#include <stdio.h>
// 기울기와 y절편을계산
int get_line_parameter(int x1, int y1, int x2, int y2, float *slope, float *yintercept)
{
    if( x1 == x2 )
            return -1;
    else {
      *slope = (float)(y2 - y1)/(float)(x2 - x1);
      *yintercept = y1 - (*slope)*x1;
      return 0;
    }
}
```

기울기와 y-절편을 인수로 전달

2개 이상의 결과를 반환

```
int main(void)
{
    float s, y;
    if( get_line_parameter(3, 3, 6, 6, &s, &y) == -1 )
        printf("에러\n");
    else
        printf("기울기는 %f, y절편은 %f\n", s, y);
    return 0;
}
```

```
기울기는 1.000000, y절편은 0.000000
```

배열이 함수 인수인 경우

- 일반 변수 vs 배열

```
// 매개 변수 x에 기억 장소가
    할당된다.

void sub(int  x)
{
    ...
}
```

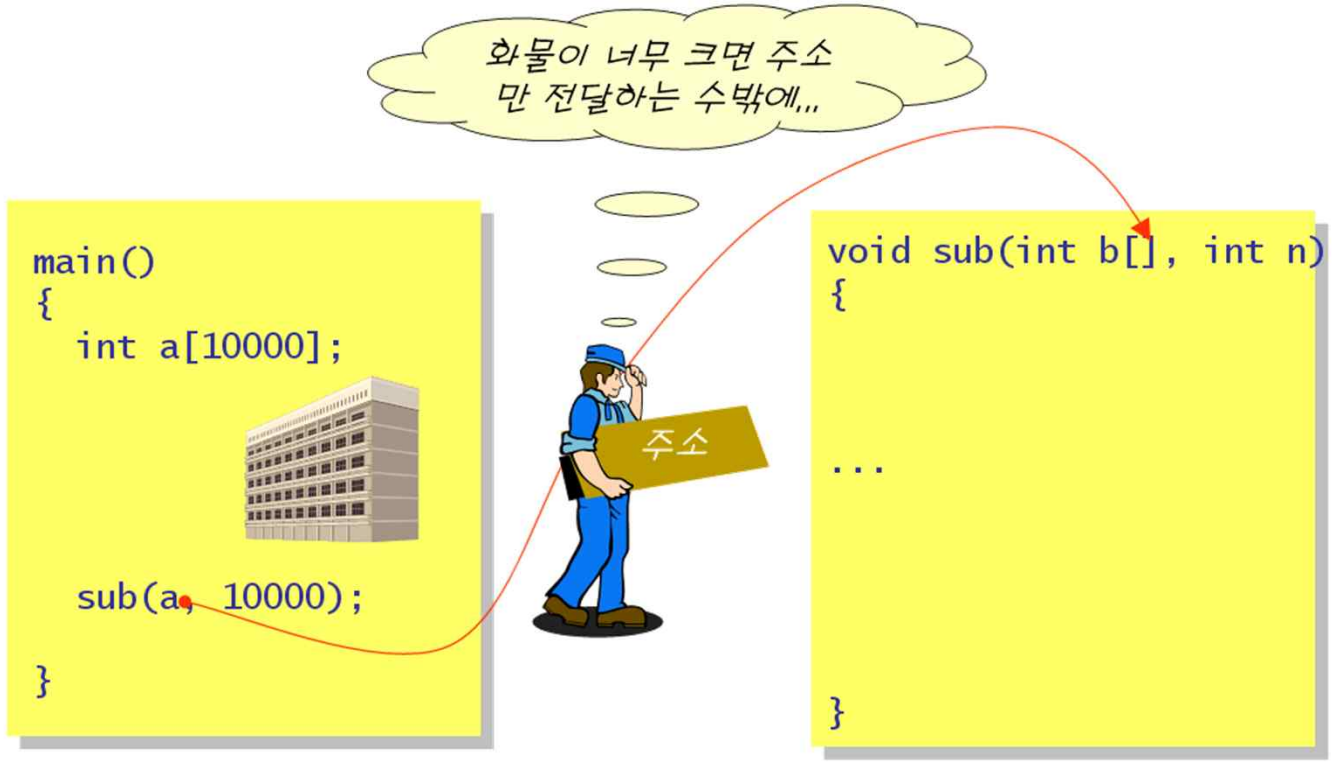
값에 의한 호출

```
// 매개 변수 b에 기억 장소가 할당되
    지 않는다.

void sub(int array[])
{
    ...
}
```

참조에 의한 호출

배열이 함수 인수인 경우

- 배열의 원본을 함수로 전달하는 이유: 크기가 큰 배열을 복사하려면 많은 CPU 시간 소모
- 따라서 배열의 경우, 배열의 주소를 전달하여서 원본을 직접 전달한다.



예제

```
#include <stdio.h>
#define SIZE 5
double get_avg(int values[], int n);
void check_values(int values[], int n);

int main(void)
{
        int i;
        int data[5];
       double result;

        for(i=0; i<SIZE; i++) {
                printf("값을 입력하시오: ");
                scanf("%d", &data[i]);
        }
        check_values(data, SIZE);
        result = get_avg(data, SIZE);
        printf("값들의 평균은 %f\n", result);
        return 0;
}
```

i 번째 배열 원소에 저장

배열의 경우, 원본이 전달된다.

```
void check_values(int values[], int n)
{
        int i;
        for(i = 0; i < n ; i++)
                if( values[i] < 0 )
                        values[i] = 0;
}
double get_avg(int values[], int n)
{
        int i;
        double sum=0.0;
        for(i = 0; i < n ; i++)
                sum += values[i];
        return sum/n;
}
```

원본 배열이 변경된다.

실행 결과

```
값을 입력하시오: 10
값을 입력하시오: 20
값을 입력하시오: 30
값을 입력하시오: 40
값을 입력하시오: -99
값들의 평균은 20.000000
```

함수가 포인터를 반환하는 경우

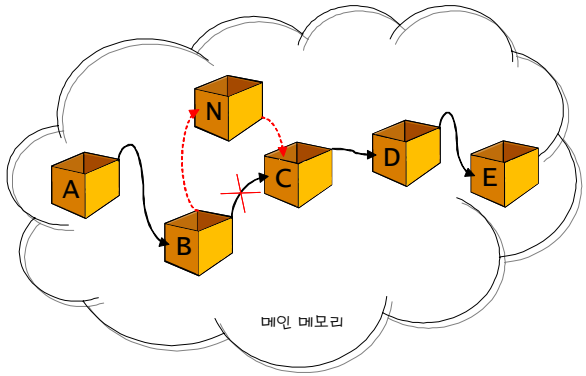
- 함수는 포인터도 반환할 수 있다.
- 함수가 종료되더라도 남아 있는 변수의 주소를 반환하여야 한다.
- 지역 변수의 주소를 반환하면 , 함수가 종료되면 사라지기 때문에 오류

```
int *add(int x, int y)
{
    int result;

    result = x + y;
    return &result;
}
```

지역 변수 result는 함수가 종료되면 소멸되므로 그 주소를 반환하면 안된다.!!

포인터 사용의 장점

- 연결 리스트나 이진 트리 등의 향상된 자료 구조를 만들 수 있다.
  - 14장에서 간단하게 학습



- 참조에 의한 호출
  - 포인터를 매개 변수로 이용하여 함수 외부의 변수의 값을 변경할 수 있다.
- 동적 메모리 할당
  - 14장에서 학습

중간 점검

1. 함수에 매개 변수로 변수의 복사본이 전달되는 것을 _____ 라고 한다.
2. 함수에 매개 변수로 변수의 원본이 전달되는 것을 _____ 라고 한다.
3. 배열을 함수의 매개 변수로 지정하는 경우, 배열의 복사가 일어나는가?

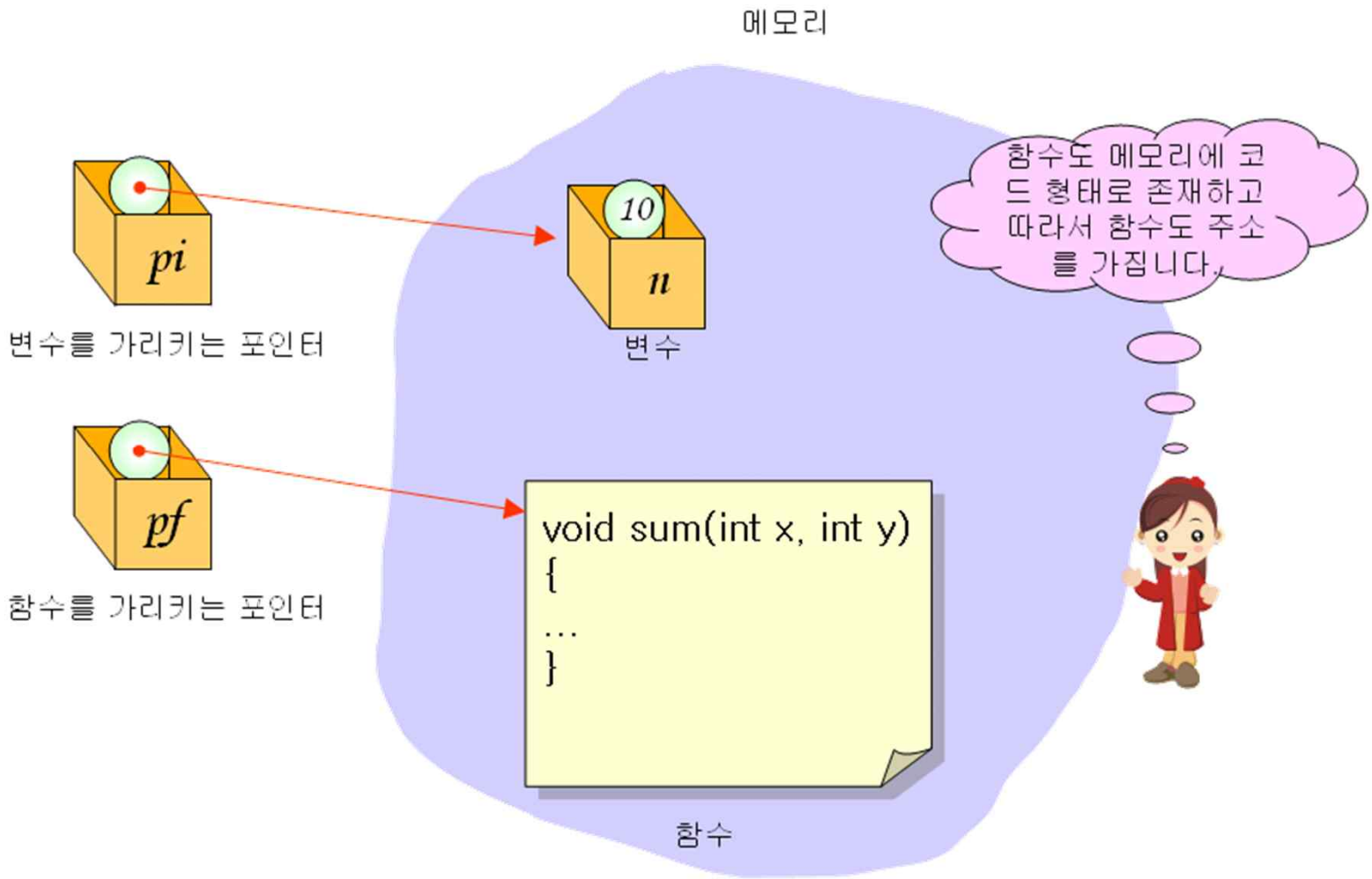
함수 포인터

- *함수 포인터(function pointer)*: 함수를 가리키는 포인터

```
반환형   (*함수포인터이름)  (매개변수1,   매개변수2, ...);
```

- (예) int (*pf)(int, int);

함수 포인터



fp1.c

```c
#include <stdio.h>
// 함수 원형 정의
int add(int, int);
int sub(int, int);

int main(void)
{
    int result;
    int (*pf)(int, int);                    // 함수 포인터 정의

    pf = add;                               // 함수 포인터에 함수 add()의 주소 대입
    result = pf(10, 20);                    // 함수 포인터를 통한 함수 add() 호출
    printf("10+20은 %d\n", result);

    pf = sub;                               // 함수 포인터에 함수 sub()의 주소 대입
    result = pf(10, 20);                    // 함수 포인터를 통한 함수 sub() 호출
    printf("10-20은 %d\n", result);

    return 0;
}
```

fp1.c

```
int add(int x, int y)
{
    return x+y;
}

int sub(int x, int y)
{
    return x-y;
}
```

```
10+20은 30
10-20은 -10
```

중간 점검

1. double 형 매개 변수를 가지며 double형의 값을 반환하는 함수 포인터 pf를 선언하여 보자.

Q & A