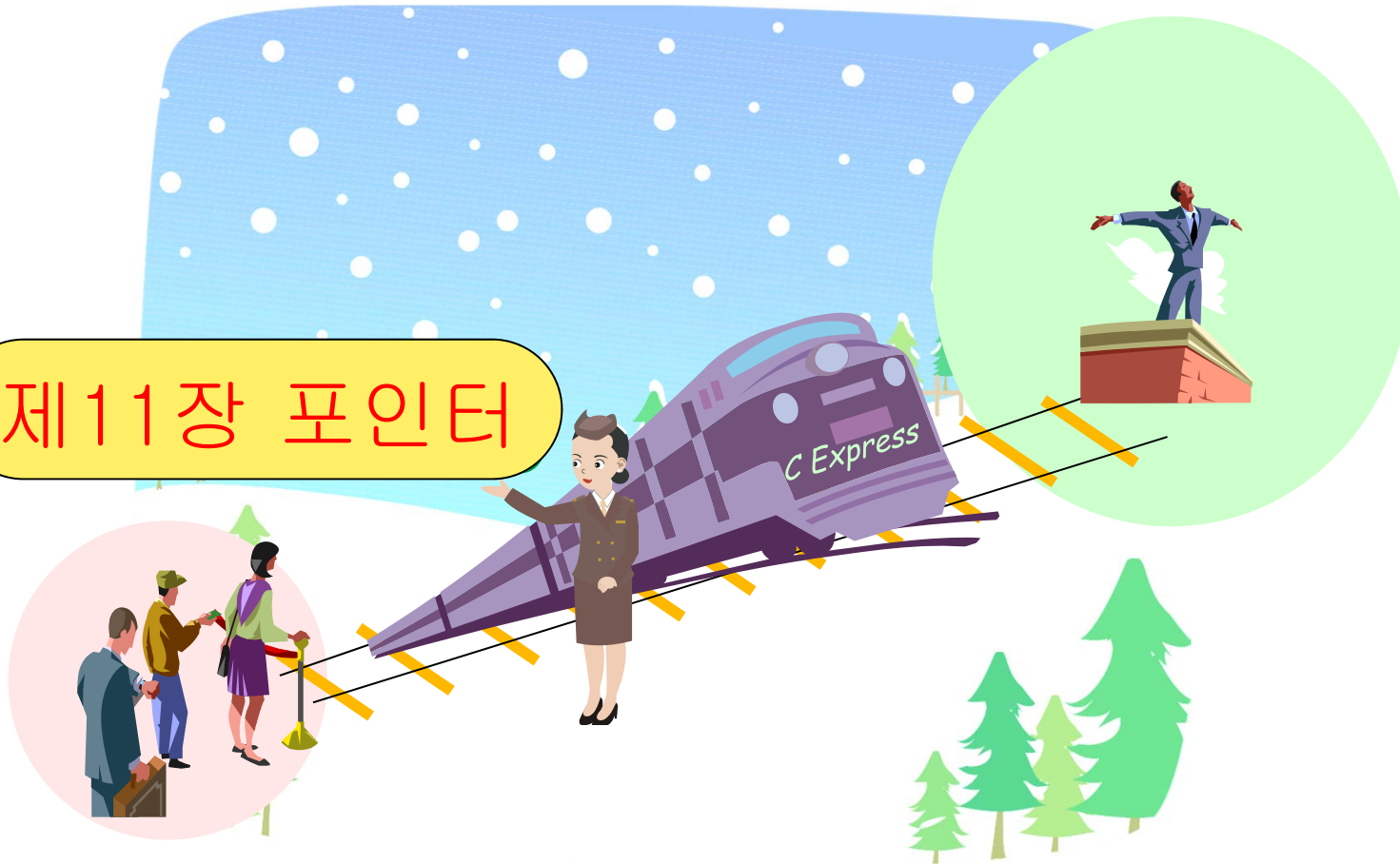





쉽게 풀어쓴 C언어 Express

제11장 포인터





이번 장에서 학습할 내용

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

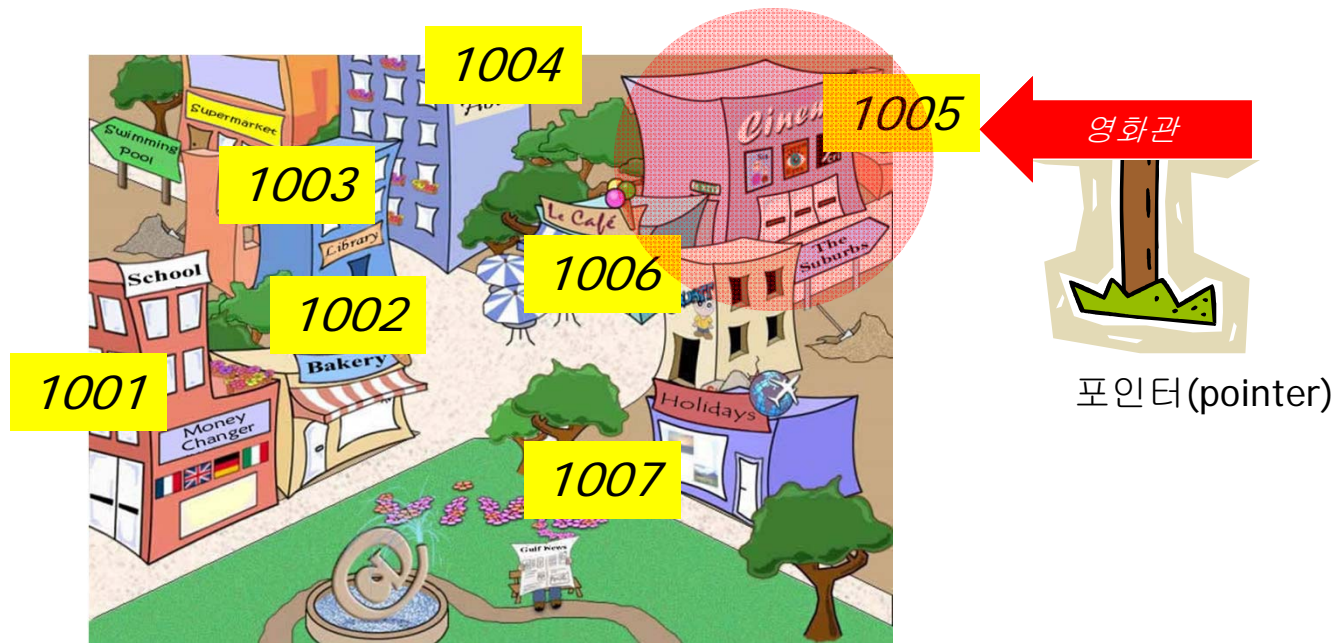


이번 장에서는
포인터의 기초적인
지식을 학습한다.



포인터란?

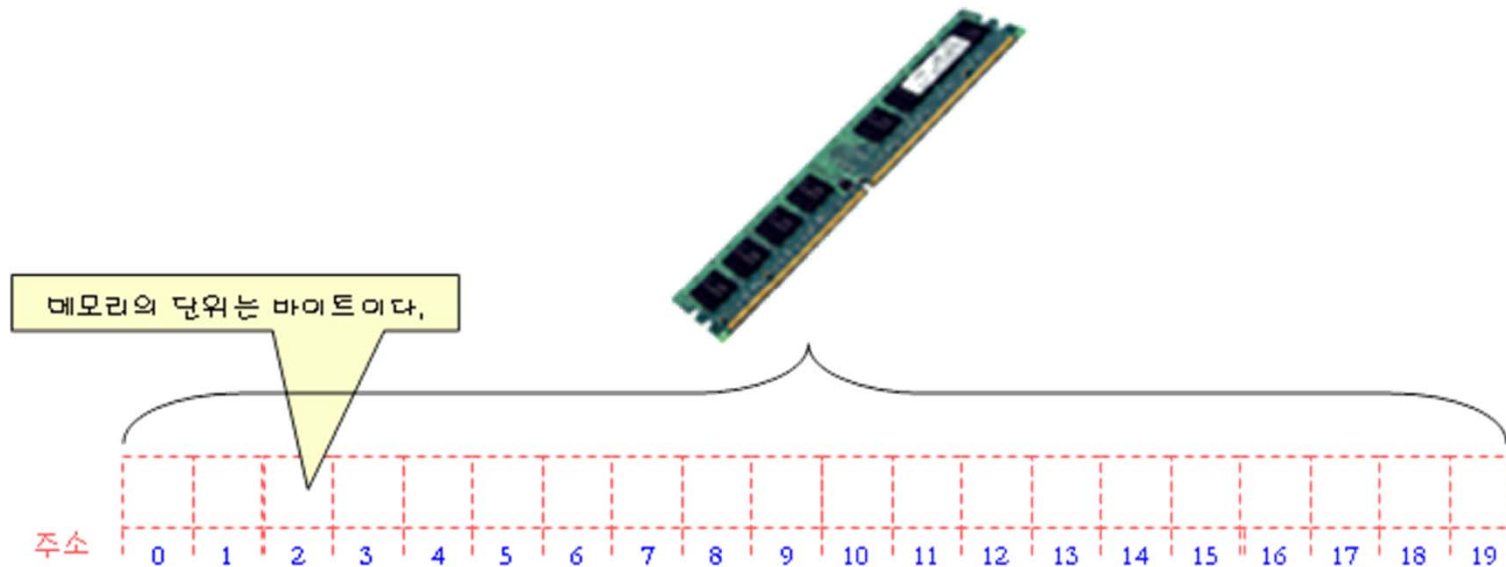
- **포인터(pointer):** 주소를 가지고 있는 변수





메모리의 구조

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

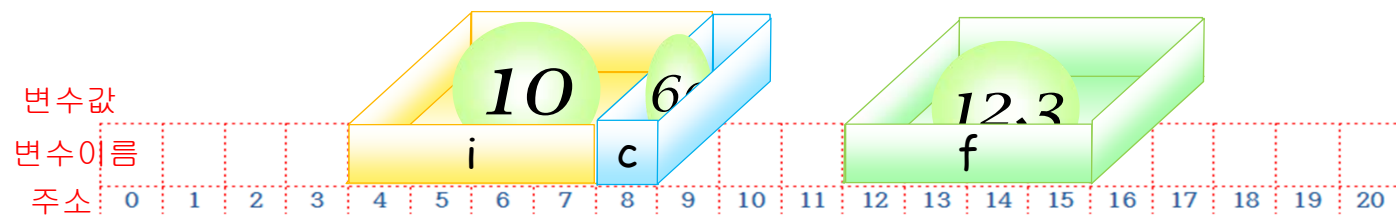




변수와 메모리

- 변수의 크기에 따라서 차지하는 메모리 공간이 달라진다.
- **char**형 변수: 1바이트, **int**형 변수: 4바이트,...

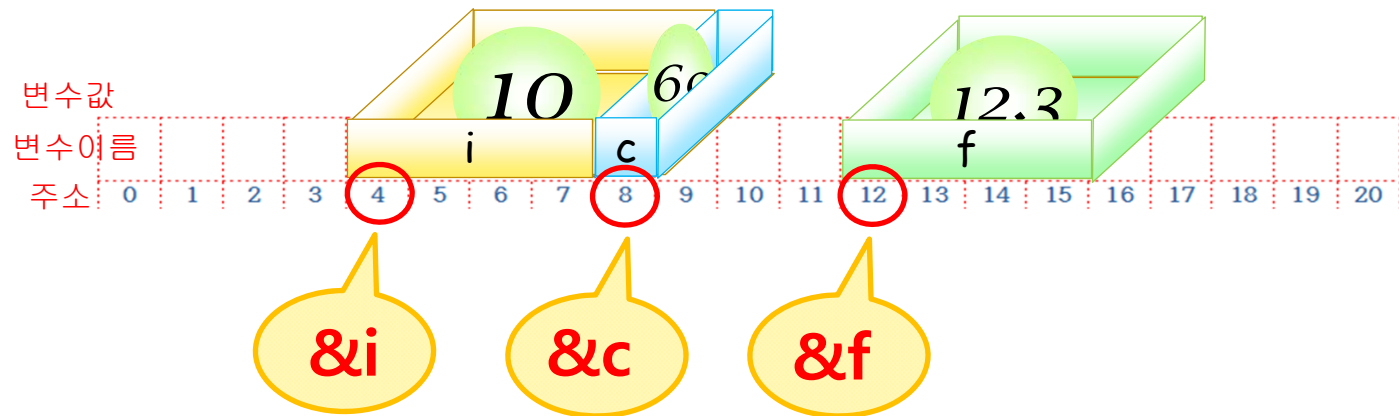
```
int main(void)
{
    int i = 10;
    char c = 69;
    float f = 12.3;
}
```





변수의 주소

- 변수의 주소를 계산하는 연산자: &
- 변수 i의 주소: &i



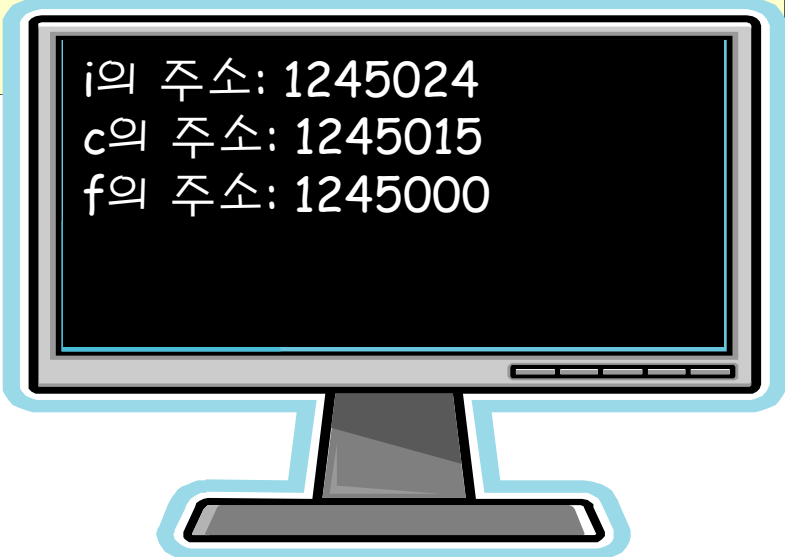


변수의 주소



```
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;
}
```

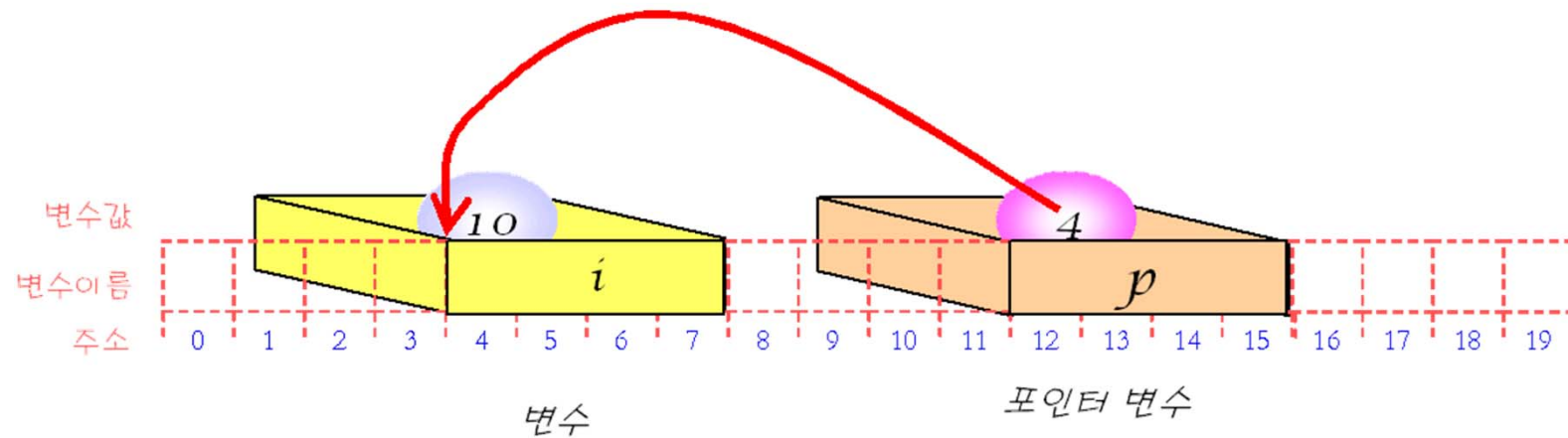


```
i의 주소: 1245024
c의 주소: 1245015
f의 주소: 1245000
```



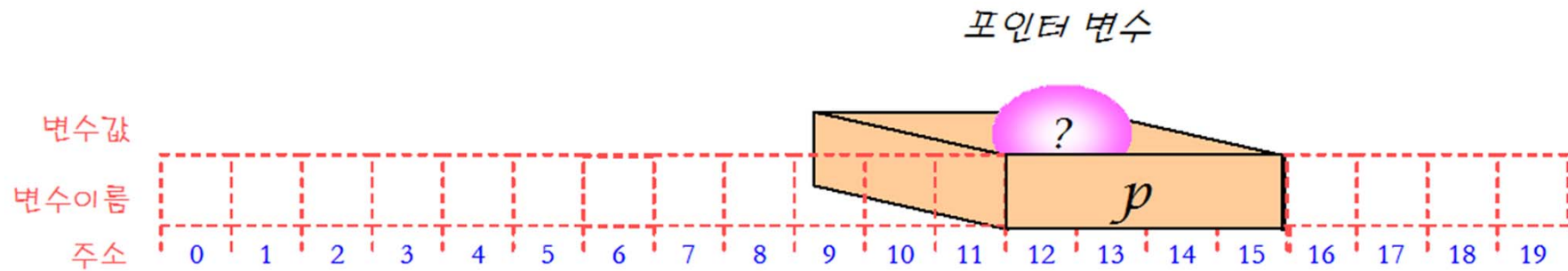
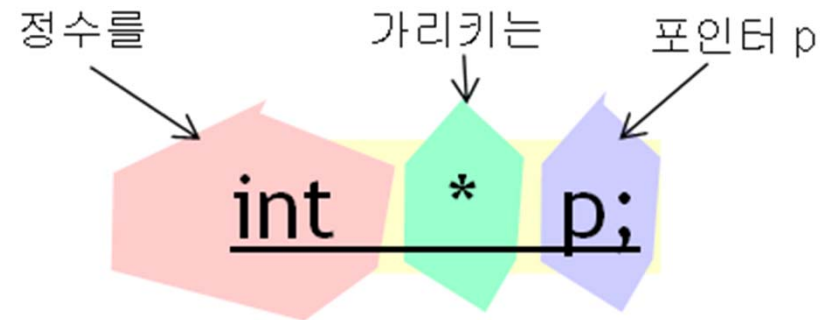
포인터의 선언

- 포인터: 변수의 주소를 가지고 있는 변수





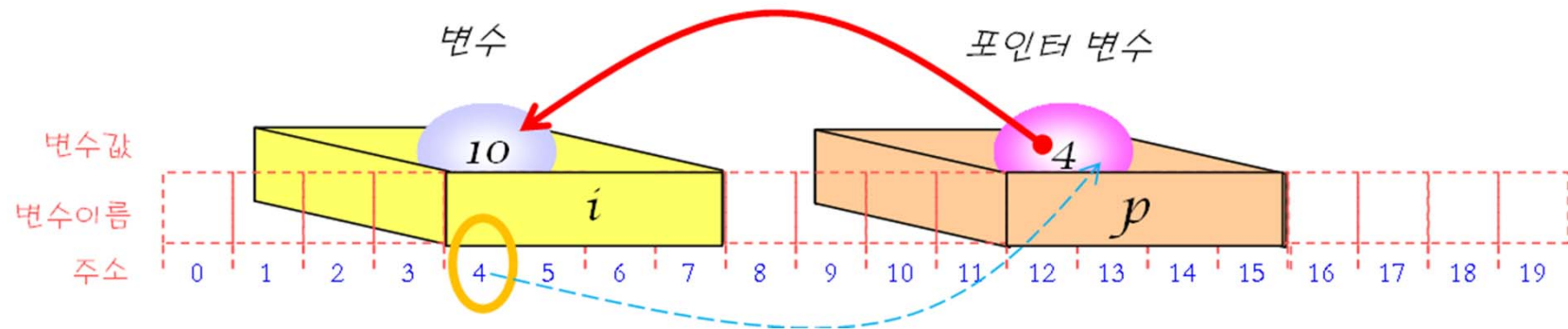
포인터의 선언





포인터와 변수의 연결

- `int i = 10;` // 정수형 변수 `i` 선언
- `int *p = &i;` // 변수 `i`의 주소가 포인터 `p`로 대입

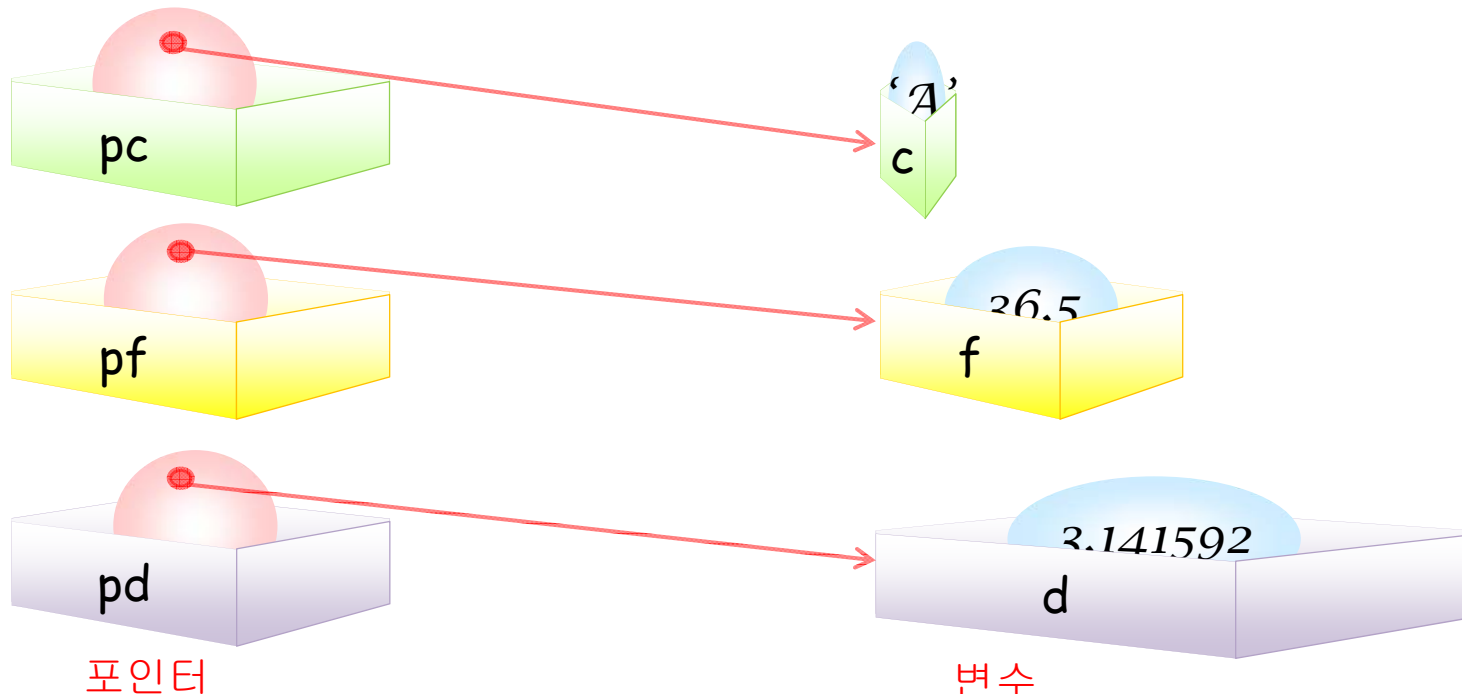




다양한 포인터의 선언

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

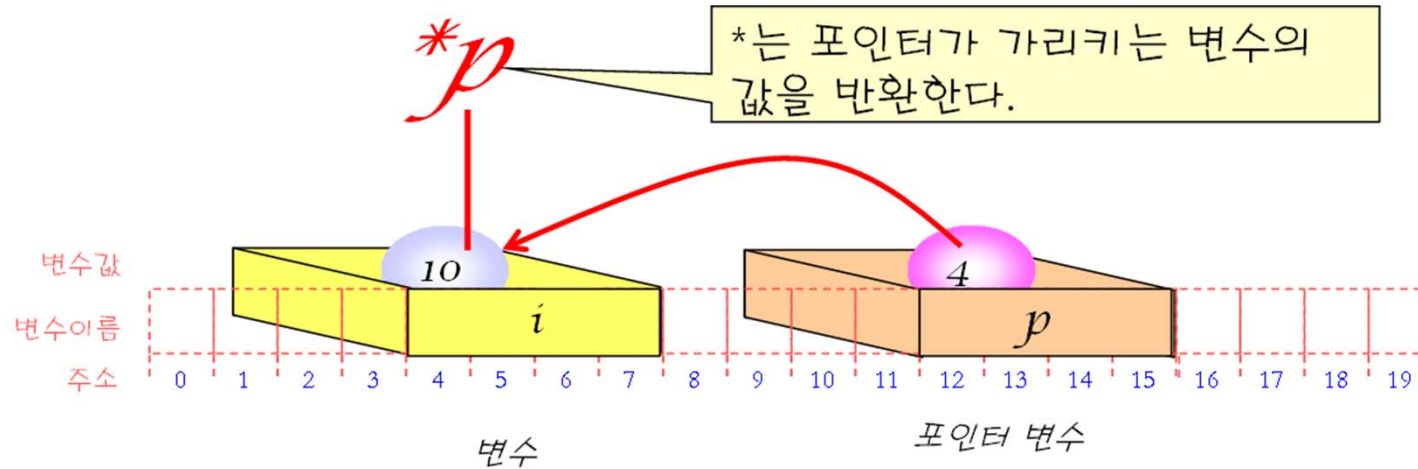
char *pc = &c;           // 문자를 가리키는 포인터 pc
float *pf = &f;           // 실수를 가리키는 포인터 pf
double *pd = &d;          // 실수를 가리키는 포인터 pd
```





간접 참조 연산자

- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자
- ```
int i;
int *p=&i;
printf("%d", *p);
```

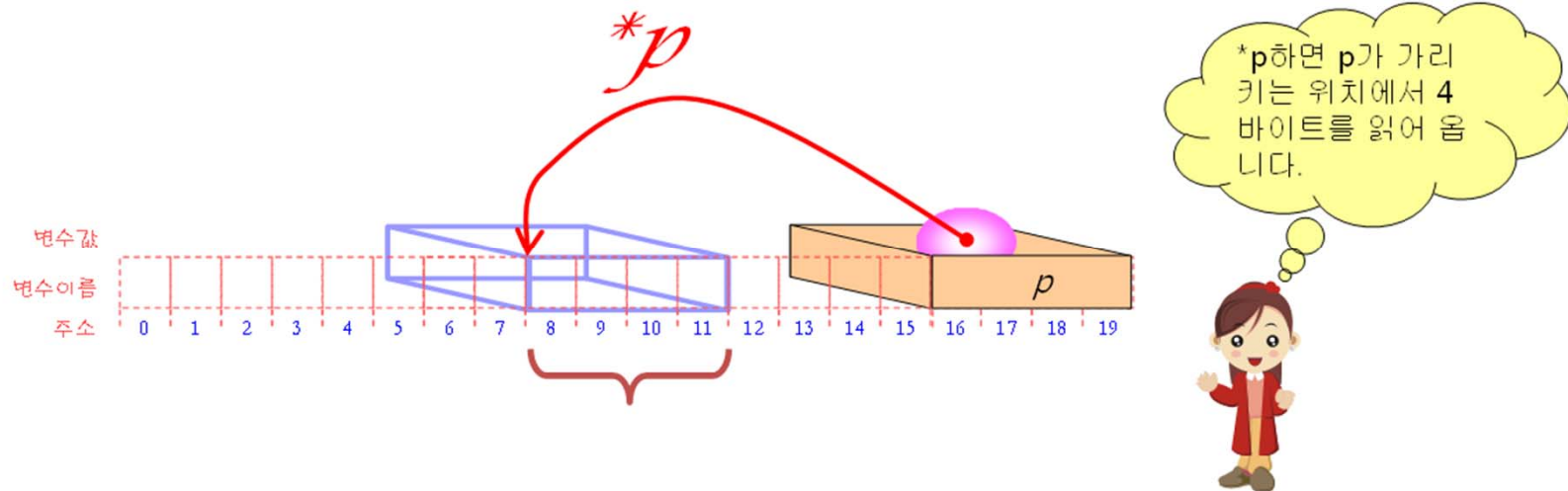




# 간접 참조 연산자의 해석

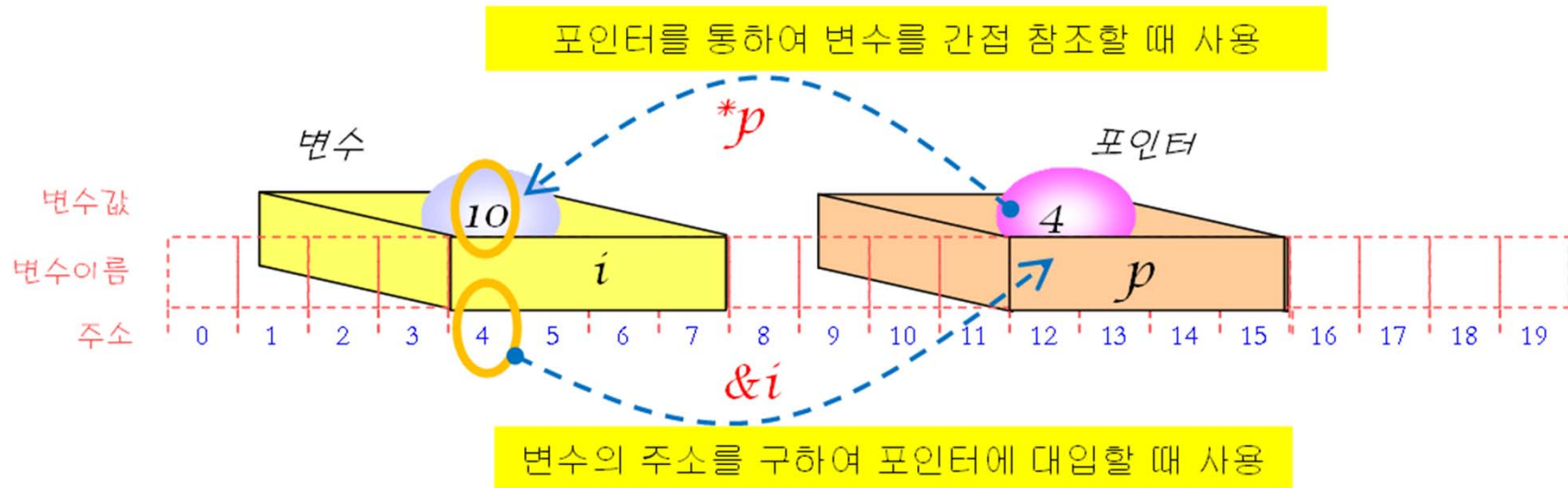
- 간접 참조 연산자: 지정된 위치에서 포인터의 타입에 따라 값을 읽어 들인다.

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





# & 연산자와 \* 연산자





# 포인터 예제 #1

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

```
int main(void)
```

```
{
```

```
→ int i = 3000;
```

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

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

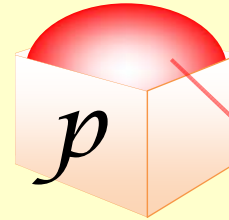
```
printf("i = %d\n", i);
```

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

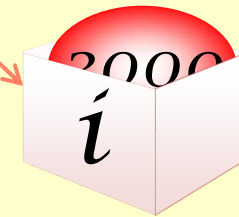
```
printf("p = %u\n", p);
```

```
return 0;
```

```
}
```



// 변수와 포인터 연결



// 변수의 주소 출력

// 변수의 값 출력

// 포인터를 통한 간접 참조 값 출력

// 포인터의 값 출력

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



## 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);
```

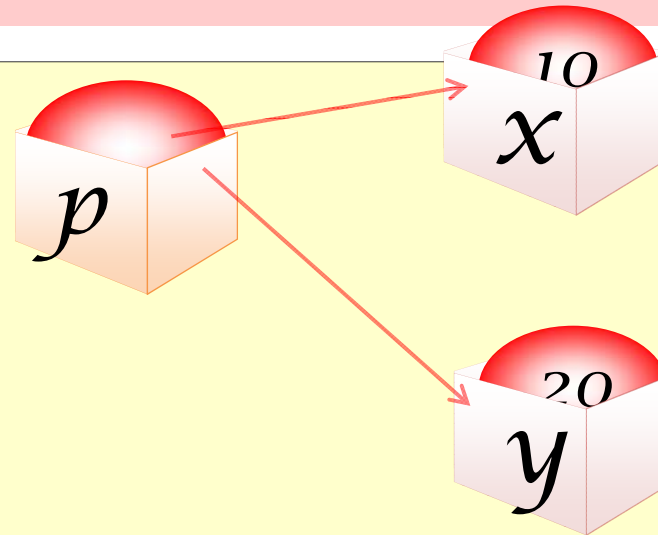
```
 p = &y;
```

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

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

```
 return 0;
```

```
}
```



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



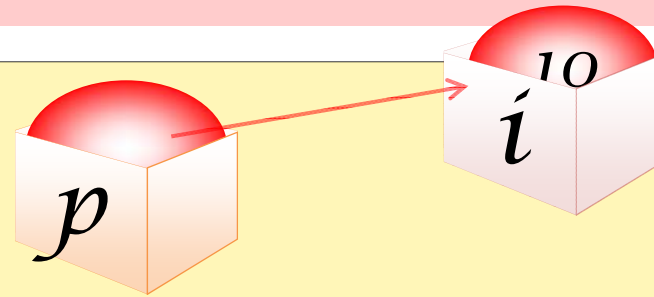


## 포인터 예제 #3

```
#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;
}
```





## 중간 점검

- 메모리는 어떤 단위를 기준으로 주소가 매겨지는가?
- 다음의 각 자료형이 차지하는 메모리 공간의 크기를 쓰시오.
- (a) char (b) short (c) int (d) long (e) float (f) double
- 포인터도 변수인가?
- 변수의 주소를 추출하는데 사용되는 연산자는 무엇인가?
- 변수 **x**의 주소를 추출하여 변수 **p**에 대입하는 문장을 쓰시오.
- 정수형 포인터 **p**가 가리키는 위치에 **25**를 저장하는 문장을 쓰시오.





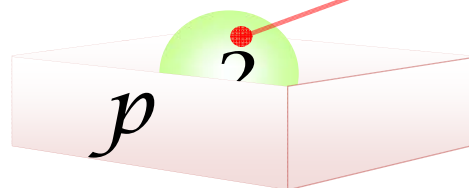
# 포인터 사용시 주의점

- 초기화가 안된 포인터를 사용하면 안된다.

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

 *p = 100; // 위험한 코드
 return 0;
}
```

주소가  
잘못된것 같은데...

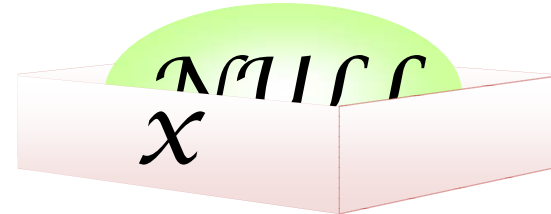
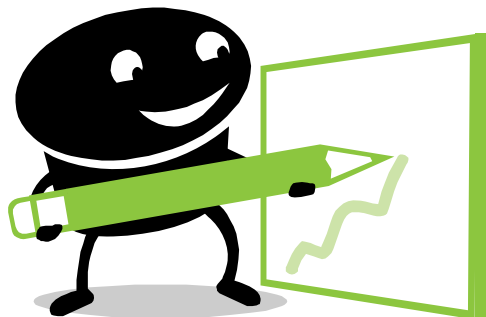




# 포인터 사용시 주의점

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

포인터가 아무것도  
가리키지 않을때는  
반드시 NULL로  
설정하세요.





# 포인터 사용시 주의점

- 포인터의 타입과 변수의 타입은 일치하여야 한다.

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

```
int main(void)
```

```
{
```

```
 int i;
```

```
 → double *pd;
```

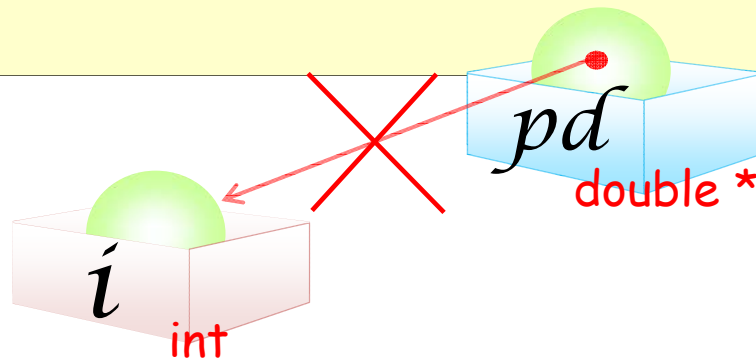
```
 pd = &i;
```

```
 *pd = 36.5;
```

```
 return 0;
```

```
}
```

// 오류! double형 포인터에 int형 변수의 주소를 대입





## 중간 점검


- 초기값이 결정되지 않은 포인터에는 어떤 값을 넣어두는 것이 안전한가?
- **char**형 변수에 **double**형 포인터로 값을 저장한다면 어떤 문제가 발생하는가?



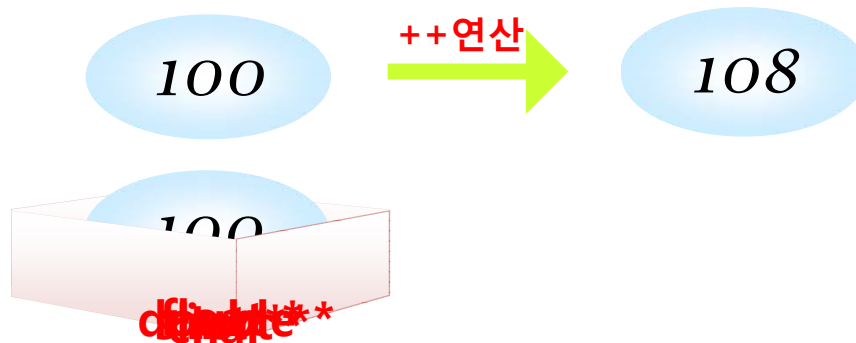


# 포인터 연산

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

| 포인터 타입                                                                                        | ++연산후 증가되는값 |
|-----------------------------------------------------------------------------------------------|-------------|
|  <i>char</i> | 1           |
| <i>short</i>                                                                                  | 2           |
| <i>int</i>                                                                                    | 4           |
| <i>float</i>                                                                                  | 4           |
| <i>double</i>                                                                                 | 8           |

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





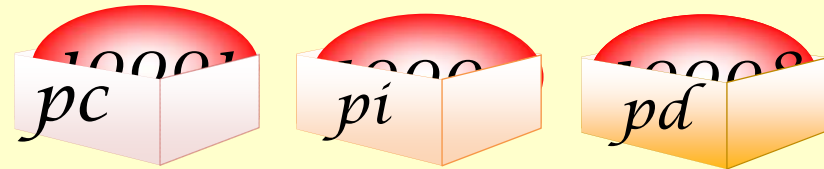
# 증가 연산 예제

```
// 포인터의 증감 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,
```

```
return 0;
```

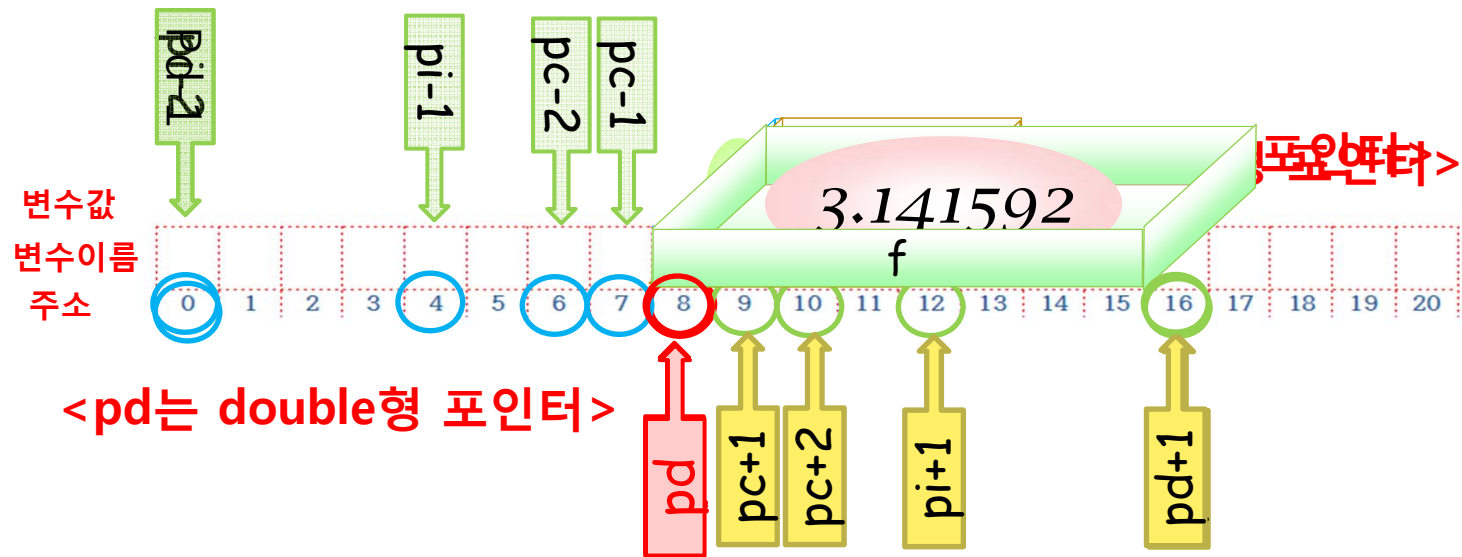
```
}
```

```
증가 전 pc = 10000, pi = 10000, pd = 10000
증가 후 pc = 10001, pi = 10004, pd = 10008
```





# 포인터의 증감 연산





# 간접 참조 연산자와 증감 연산자

- $*p++;$ 
  - $p$ 가 가리키는 위치에서 값을 가져온 후에  $p$ 를 증가한다.
- $(*p)++;$ 
  - $p$ 가 가리키는 위치의 값을 증가한다.

| 수식           | 의미                                         |
|--------------|--------------------------------------------|
| $v = *p++$   | $p$ 가 가리키는 값을 $v$ 에 대입한 후에 $p$ 를 증가한다.     |
| $v = (*p)++$ | $p$ 가 가리키는 값을 $v$ 에 대입한 후에 가리키는 값을 증가한다.   |
| $v = *++p$   | $p$ 를 증가시킨 후에 $p$ 가 가리키는 값을 $v$ 에 대입한다.    |
| $v = ++*p$   | $p$ 가 가리키는 값을 가져온 후에 그 값을 증가하여 $v$ 에 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);

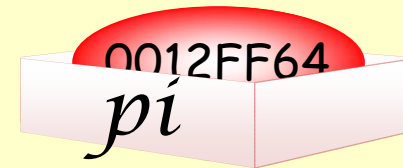
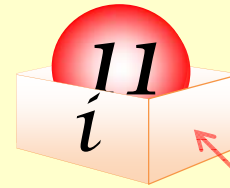
printf("i = %d, pi = %p\n", i, pi);

\*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
```



# 포인터의 형변환

- C언어에서는 꼭 필요한 경우에, 명시적으로 포인터의 타입을 변경할 수 있다.

```
double *pd = &f;
```

```
int *pi;
```

```
pi = (int *)pd;
```



# 간접 참조 연산자와 증감 연산자

```
#include <stdio.h>
int main(void)
{
 char buffer[8];
 double *pd;
 int *pi;

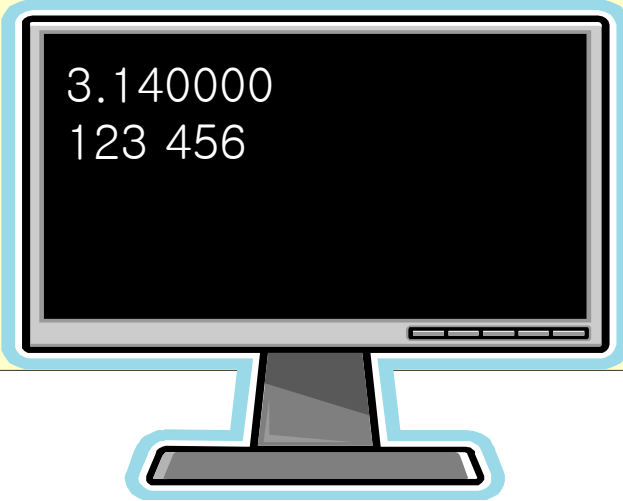
 pd = (double *)buffer;
 *pd = 3.14;

 printf("%f\n", *pd);
 pi = (int *)buffer;
 *pi = 123;
 *(pi+1) = 456;

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

char형 포인터를 double형 포인터로 변환  
배열의 이름은 char형 포인터이다.

char형 포인터를 int형 포인터로 변환



3.140000  
123 456



## 중간 점검

- 포인터에 대하여 적용할 수 있는 연산에는 어떤 것들이 있는가?
- `int`형 포인터 `p`가 80번지를 가리키고 있었다면 `(p+1)`은 몇 번지를 가리키는가?
- `p`가 포인터라고 하면 `*p++`와 `(*p)++`의 차이점은 무엇인가?
- `p`가 포인터라고 하면 `*(p+3)`의 의미는 무엇인가?





# 포인터와 배열

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





# 포인터와 배열

// 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]);
```

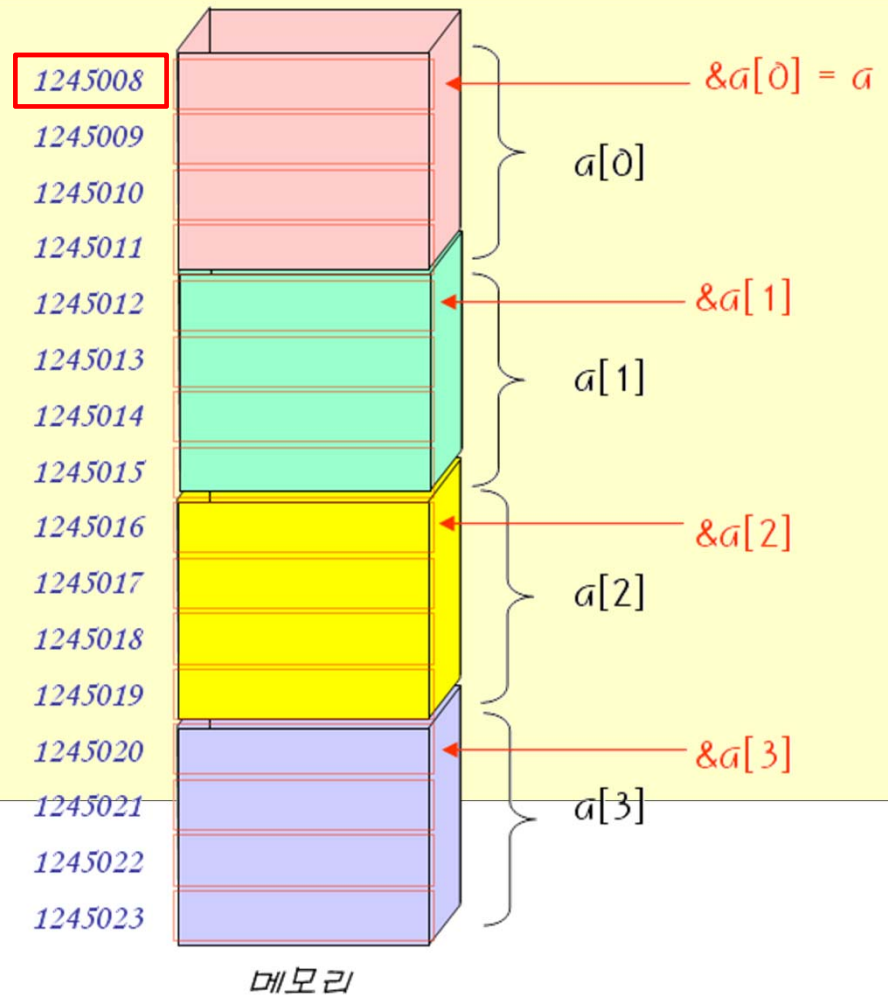
```
printf("&a[2] = %u\n", &a[2]);
```

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

```
return 0;
```

```
}
```

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





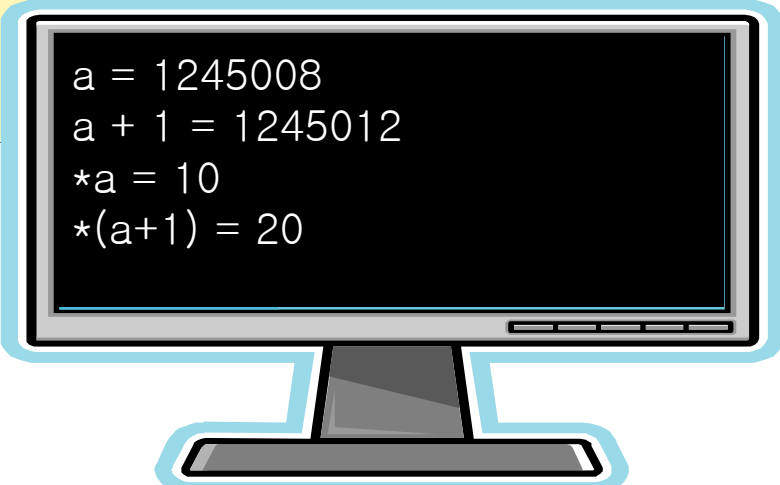


# 포인터와 배열

```
// 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;
}
```

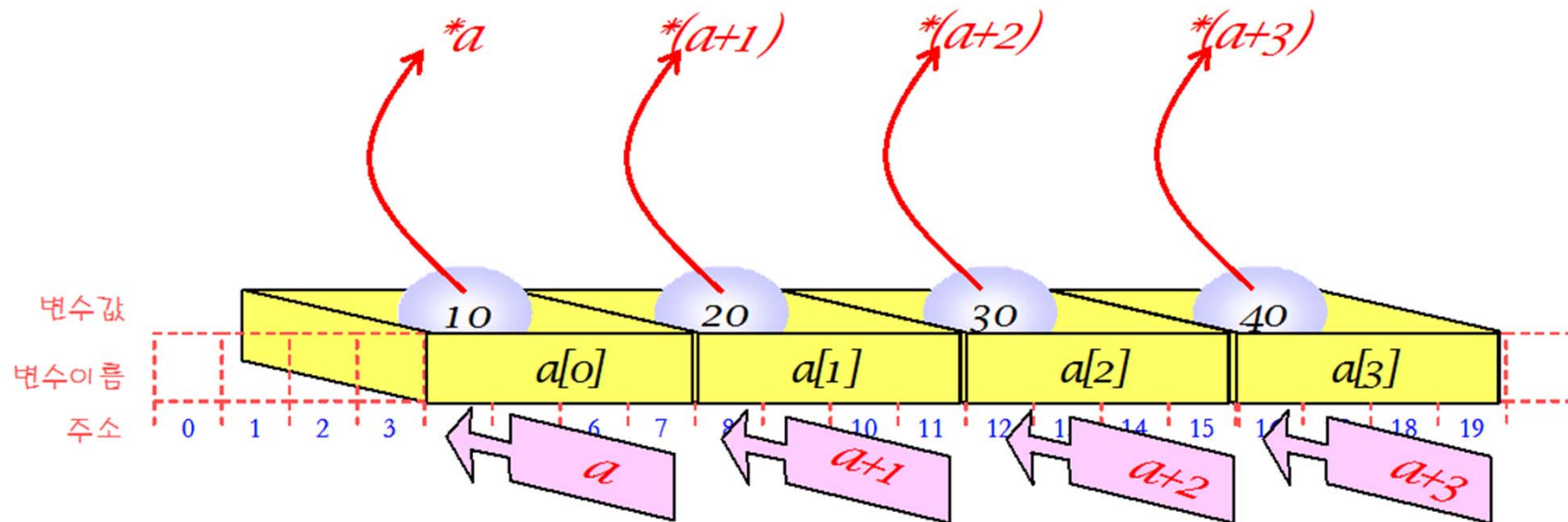


```
a = 1245008
a + 1 = 1245012
*a = 10
*(a+1) = 20
```



# 포인터와 배열

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





# 포인터를 배열처럼 사용

// 포인터를 배열 이름처럼 사용

#include <stdio.h>

int main(void)

{

→ int a[] = { 10, 20, 30, 40 };

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;

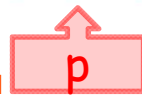
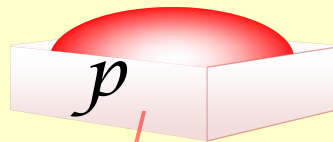
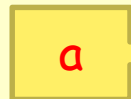
}

변수값

변수이름

주소

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20



p[0]

p[1]

p[2]

p[3]

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 사용한 방법의 장점

- 포인터가 인덱스 표기법보다 빠르다.
  - Why?: 원소의 주소를 계산할 필요가 없다.

```
int get_sum1(int a[], int n)
{
 int i;
 int sum = 0;

 for(i = 0; i < n; i++)
 sum += a[i];
 return sum;
}
```

인덱스 표기법 사용



```
int get_sum2(int a[], int n)
{
 int i;
 int *p;
 int sum = 0;

 p = a;
 for(i = 0; i < n; i++)
 sum += *p++;
 return sum;
}
```

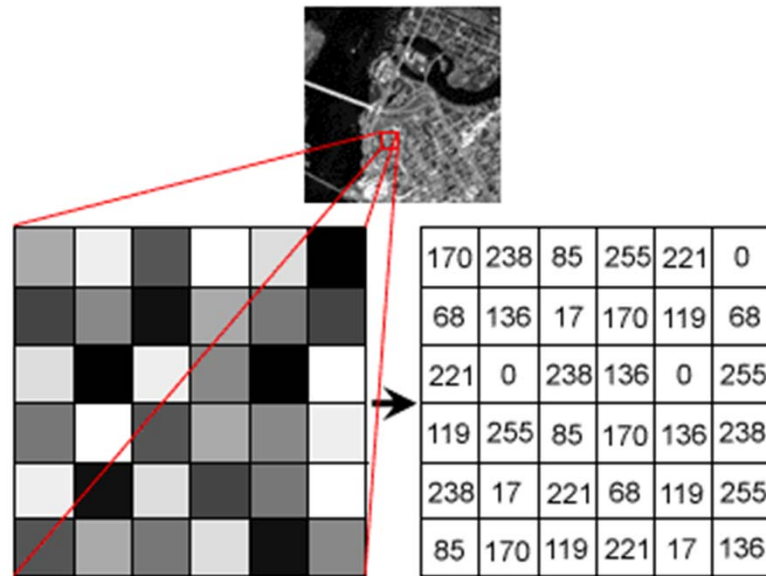
포인터 사용





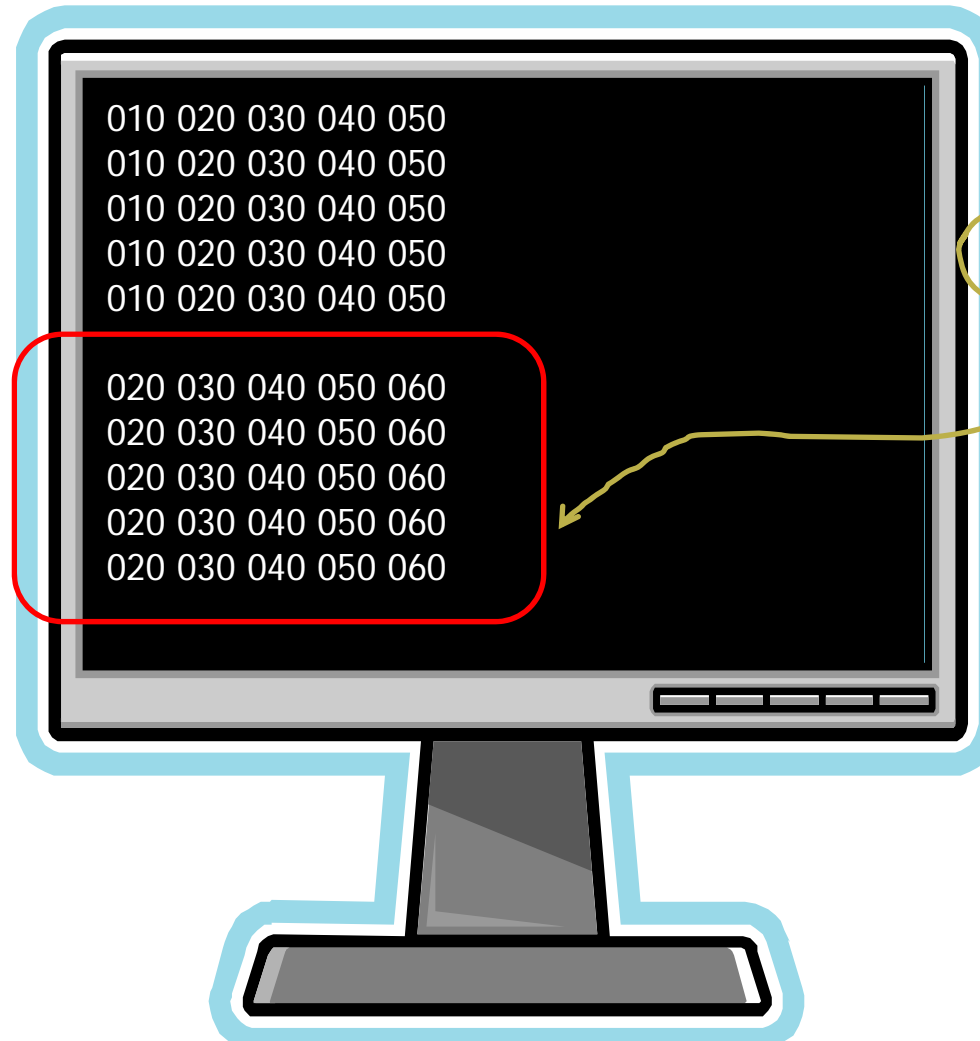
# 실습: 영상 처리

- 디지털 이미지는 배열을 사용하여 저장된다.
- 이미지 처리를 할 때 속도를 빠르게 하기 위하여 포인터를 사용한다.
- 이미지 내의 모든 픽셀의 값을 **10**씩 증가시켜보자.





## 실행 결과



모든 픽셀  
의 값이  
10씩 증가  
되었다.



## 실습: 영상 처리

```
#include <stdio.h>
#define SIZE 5
void print_image(int image[][SIZE])
{
 int r,c;
 for(r=0;r<SIZE;r++){
 for(c=0;c<SIZE;c++){
 printf("%03d ", image[r][c]);
 }
 printf("\n");
 }
 printf("\n");
}
```



## 실습: 영상 처리

```
void brighten_image(int image[][SIZE])
{
 int r,c;
 int *p;
 p = &image[0][0];
 for(r=0;r<SIZE;r++){
 for(c=0;c<SIZE;c++){
 *p += 10;
 p++;
 }
 }
}
```





## 실습: 영상 처리

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

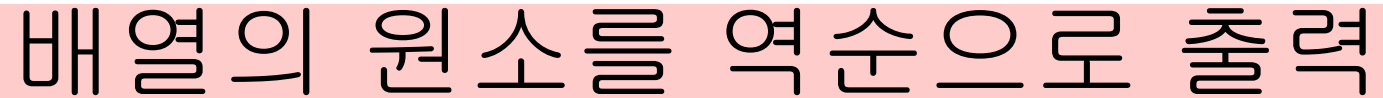
 print_image(image);
 brighten_image(image);
 print_image(image);
 return 0;
}
```



# 도전문제

- 포인터를 이용하지 않는 버전도 작성하여 보자. 즉 배열의 인덱스 표기법으로 위의 프로그램을 변환하여 보자.





The diagram shows an array with 5 elements, labeled  $a[0]$  through  $a[4]$ . Each element is represented by a green circle containing a number: 10, 20, 30, 40, and 50. Above each element is a blue box containing a red letter 'p', with a blue arrow pointing down to the element. Below the array, a horizontal axis is marked with numbers 1 through 20, with red dashed lines separating the groups of 5.



## 중간 점검

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





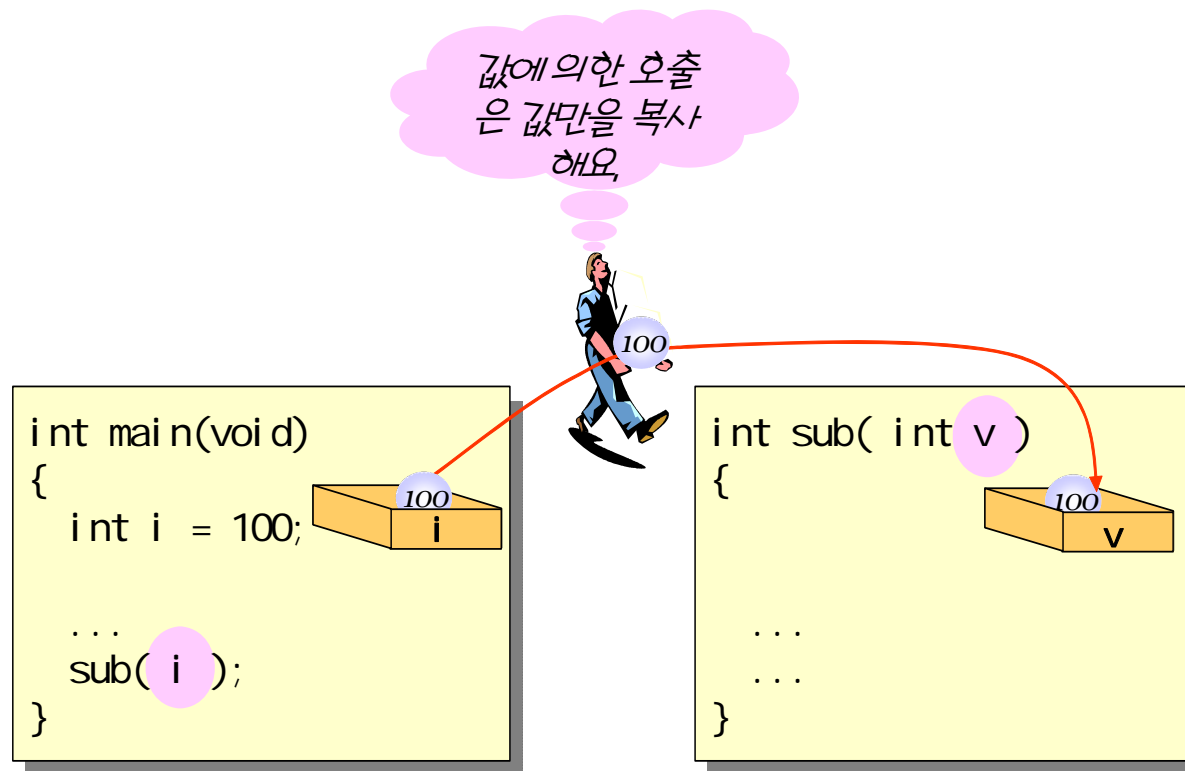
# 인수 전달 방법

- C에서의 인수 전달 방법
  - 값에 의한 호출: 기본적인 방법
  - 참조에 의한 호출: 포인터 이용



# 값에 의한 호출

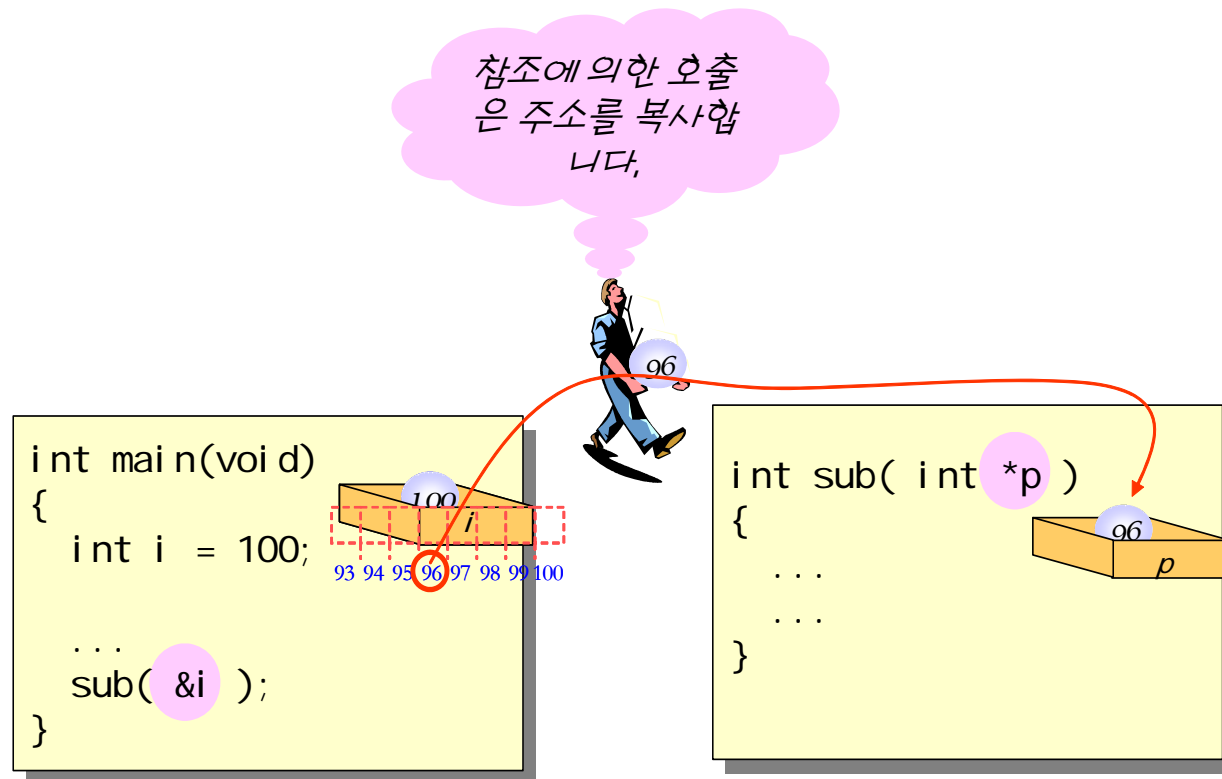
- 함수 호출시에 변수의 값을 함수에 전달





# 참조에 의한 호출

- 함수 호출시에 변수의 주소를 함수의 매개 변수로 전달





# swap() 함수 #1

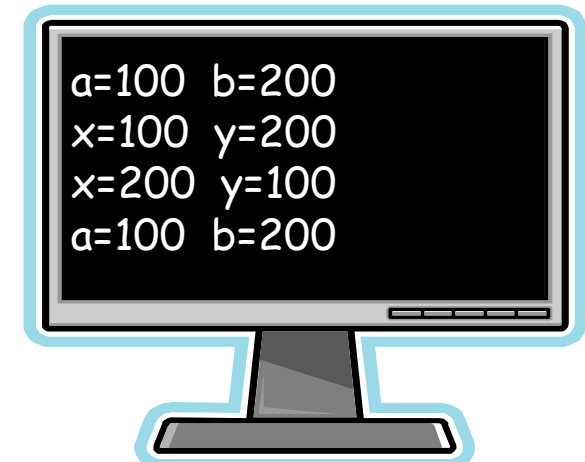
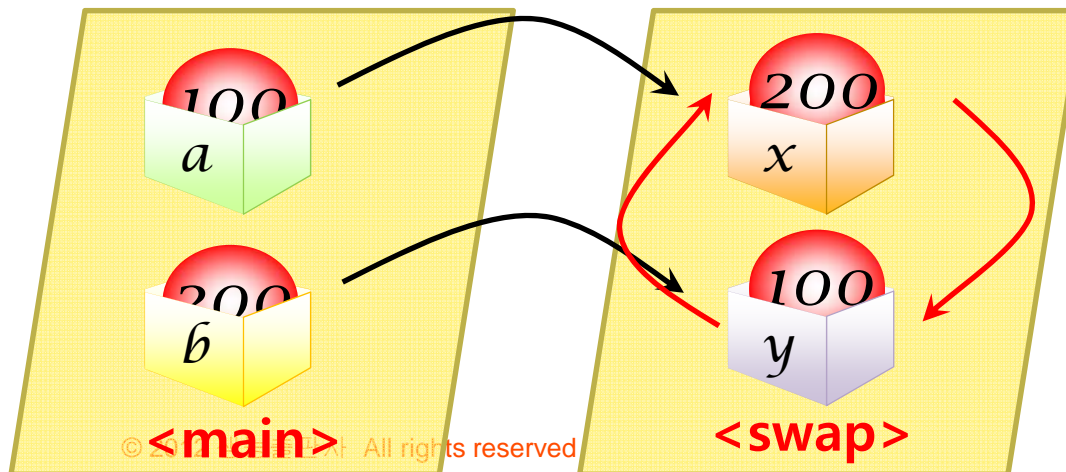
- 변수 2개의 값을 바꾸는 작업을 함수로 작성

```
#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;
}
```

함수 호출시에 값만 복사된다.

```
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);
}
```







# 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);

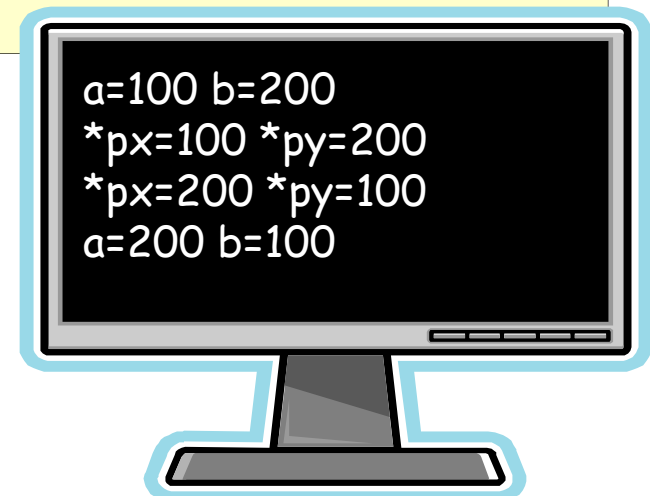
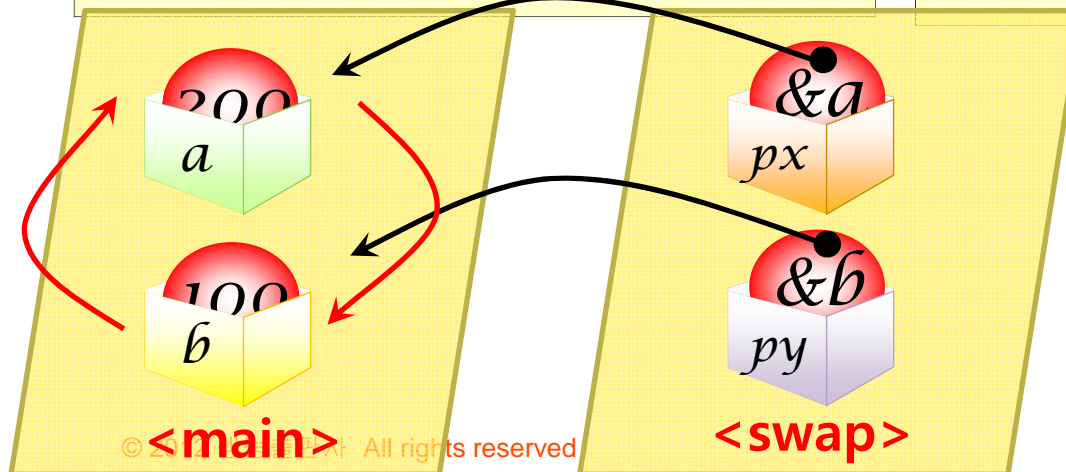
 printf("a=%d b=%d\n", a, b);
 return 0;
}
```

함수 호출시에 주소가 복사된다.

```
void swap(int *px, int *py)
{
 int tmp;
 printf("*px=%d *py=%d\n", *px, *py);

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

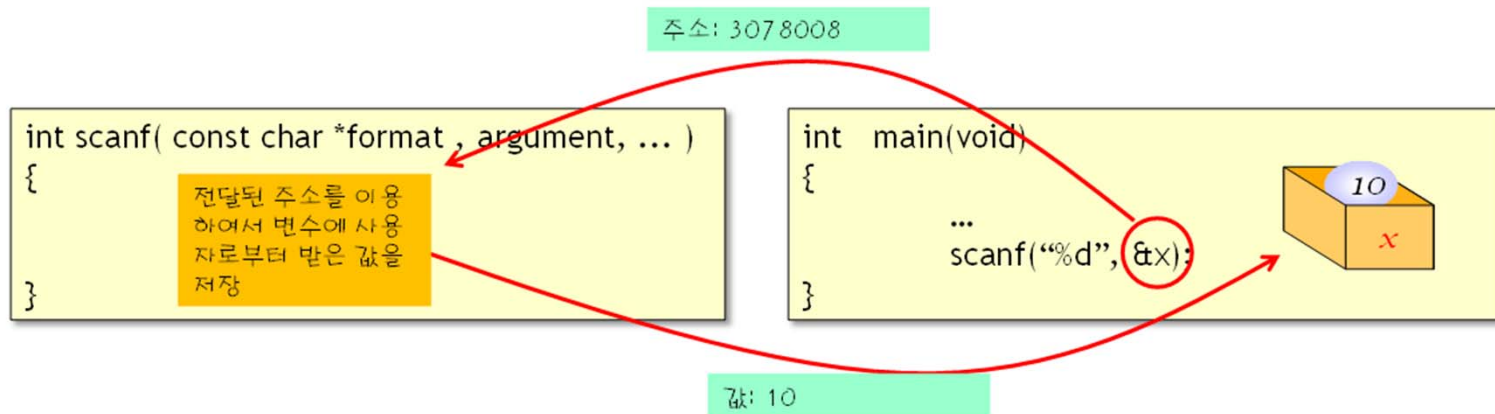
 printf("*px=%d *py=%d\n", *px, *py);
}
```





# scanf() 함수

- 변수에 값을 저장하기 위하여 변수의 주소를 받는다.



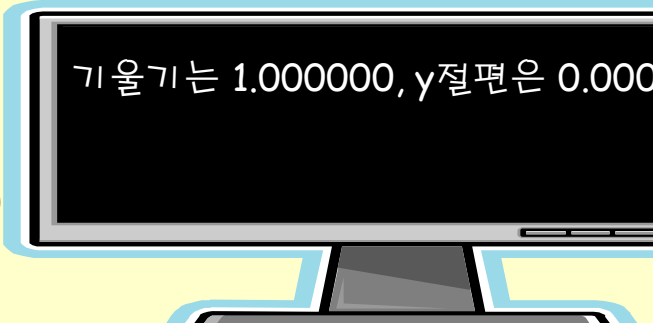


## 2개 이상의 결과를 반환

```
#include <stdio.h>
// 기울기와 y절편을 계산
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;
 }
}

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;
}
```

기울기와 Y절편을 인수로 전달



기울기는 1.000000, y절편은 0.000000



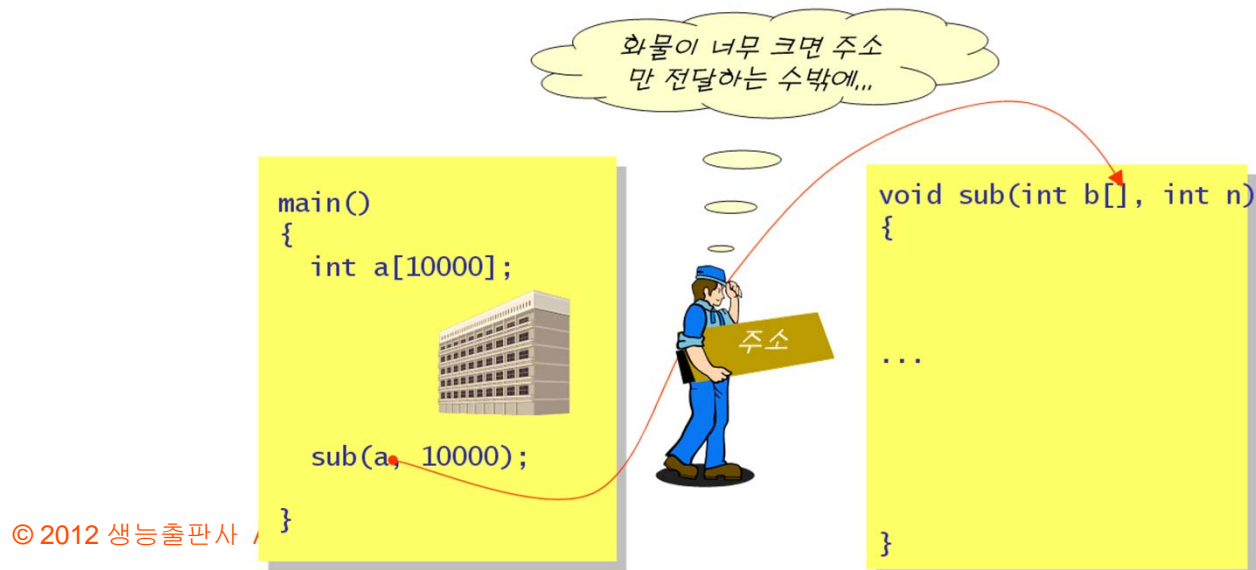
# 배열이 함수 인수인 경우

- 일반 변수 vs 배열

```
// 매개 변수 x에 기억 장소가 할당
void sub(int x)
{
 ...
}
```

```
// b[]에 기억 장소가 할당되지 않는다.
void sub(int b[], int n)
{
 ...
}
```

- 배열의 경우, 크기가 큰 경우에 복사하려면 많은 시간 소모
- 배열의 경우, 배열의 주소를 전달





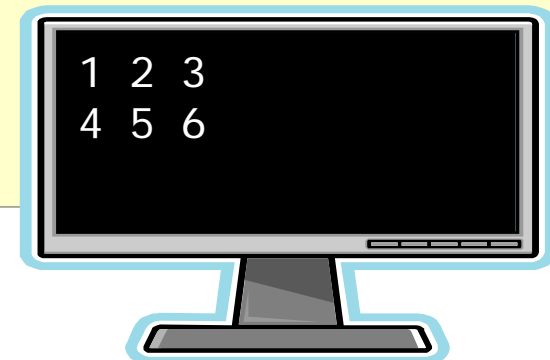
# 예제

```
#include <stdio.h>
void sub(int b[], int n);
```

```
int main(void)
{
 int a[3] = { 1, 2, 3 };

 printf("%d %d %d\n", a[0], a[1], a[2]);
 sub(a, 3);
 printf("%d %d %d\n", a[0], a[1], a[2]);
 return 0;
}
```

```
void sub(int b[], int n)
{
 b[0] = 4;
 b[1] = 5;
 b[2] = 6;
}
```





# 포인터를 반환할 때 주의점

- 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;
}
```

지역변수 result는  
함수가 종료되면 소멸되므로 그  
주소를 반환하면 안된다!





## 중간 점검

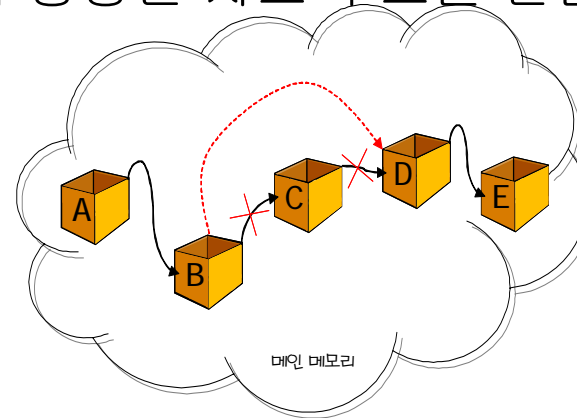
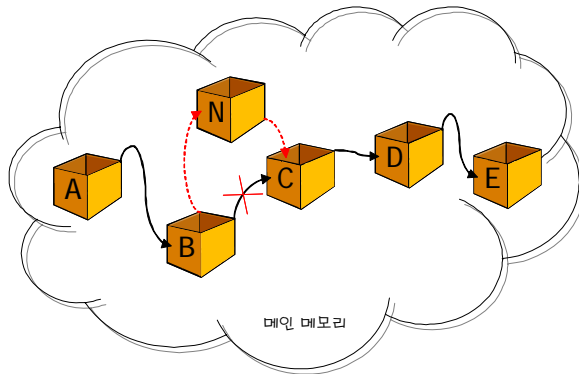
- 함수에 매개 변수로 변수의 복사본이 전달되는 것을 \_\_\_\_\_라고 한다.
- 함수에 매개 변수로 변수의 원본이 전달되는 것을 \_\_\_\_\_라고 한다.
- 배열을 함수의 매개 변수로 지정하는 경우, 배열의 복사가 일어나는가?





# 포인터 사용의 장점

- 연결 리스트나 이진 트리 등의 향상된 자료 구조를 만들 수 있다.



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





# Q & A

