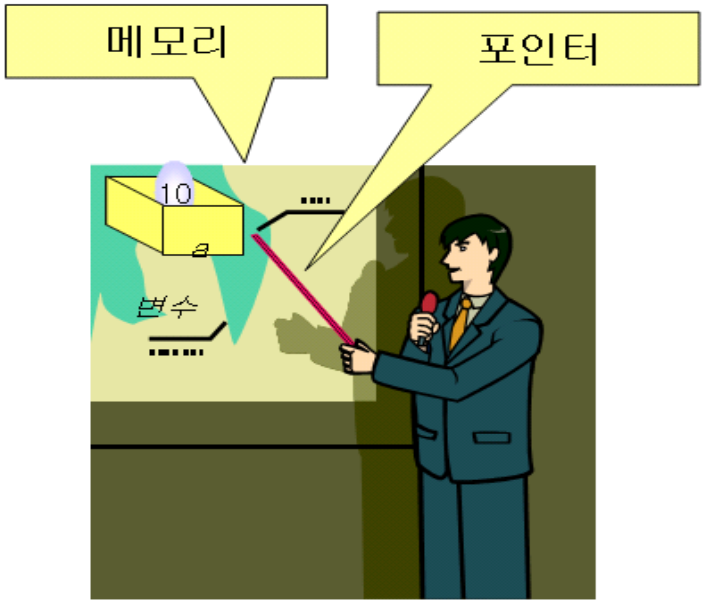
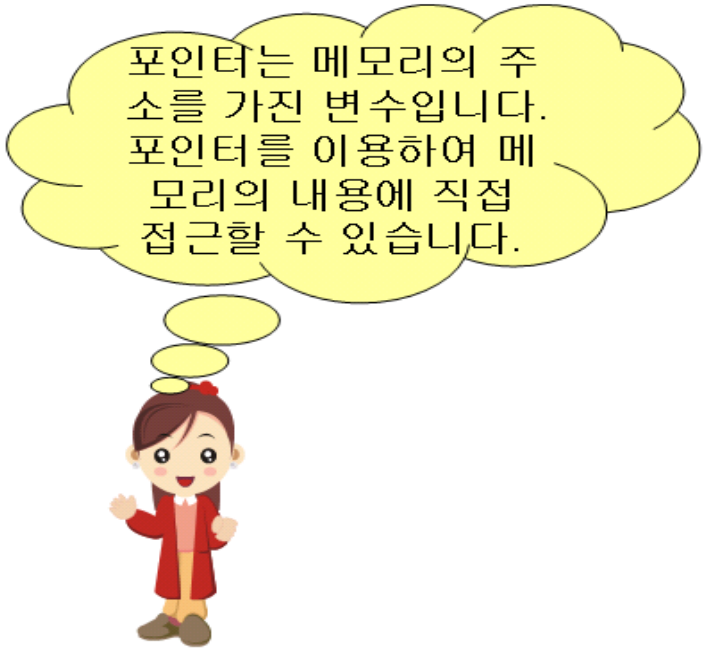
제8장 포인터

포인터란?

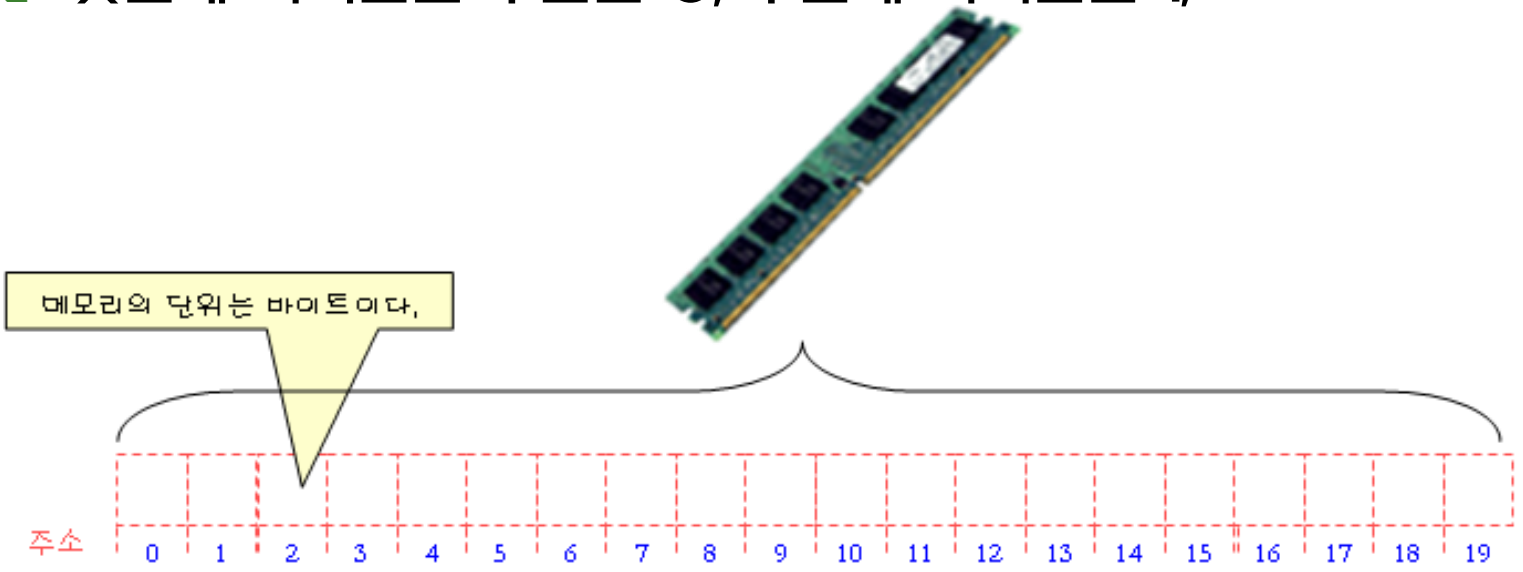
- *포인터(pointer):* 주소(address)를 값으로 가지는 변수





메모리의 구조

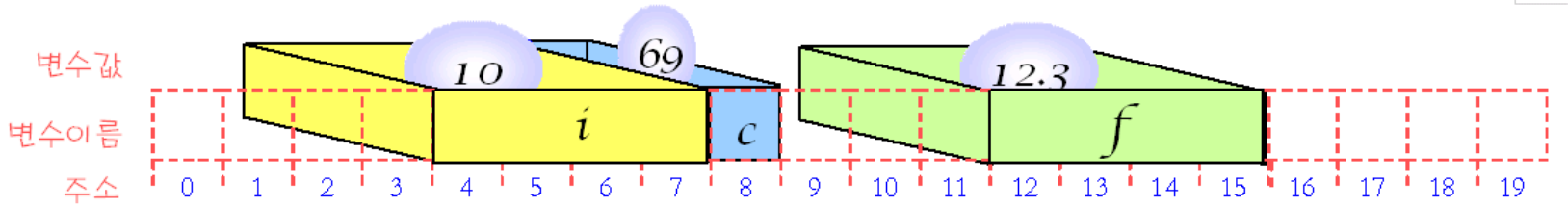
- 변수는 보통 메인 메모리에 저장된다.
- 메모리는 바이트(byte) 단위로 액세스되며, 각 바이트 당 하나의 주소가 할당된다.
  - 첫번째 바이트의 주소는 0, 두번째 바이트는 1,…



변수와 메모리

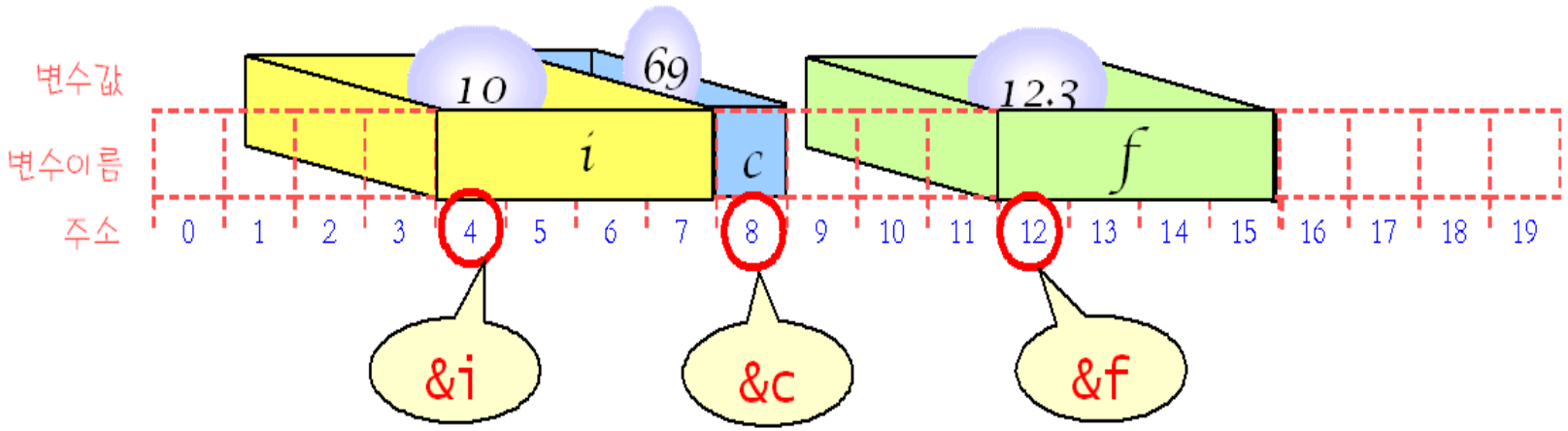
- 변수의 유형에 따라서 차지하는 메모리 공간 크기가 달라진다.
- char형 변수: 1 바이트, int형 변수: 4 바이트, …

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



변수의 주소

- 변수의 주소를 나타내는 연산자: **&**
- 변수 **v**의 주소: **&v**



변수의 주소 예

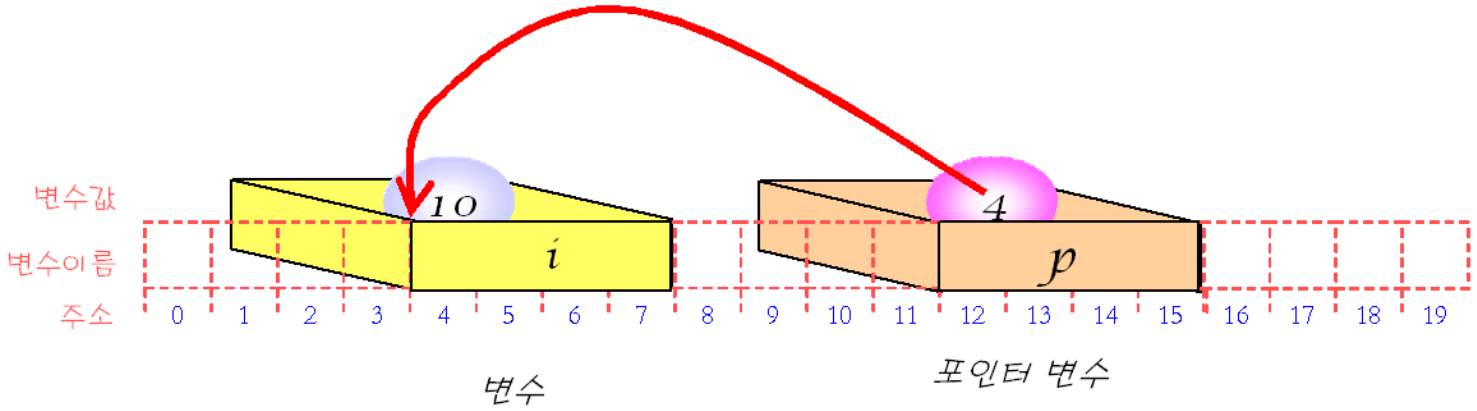
```
int main(void)
{
	int i = 10;
	char c = 69;
	float f = 12.3;

	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
```

포인터 변수의 선언

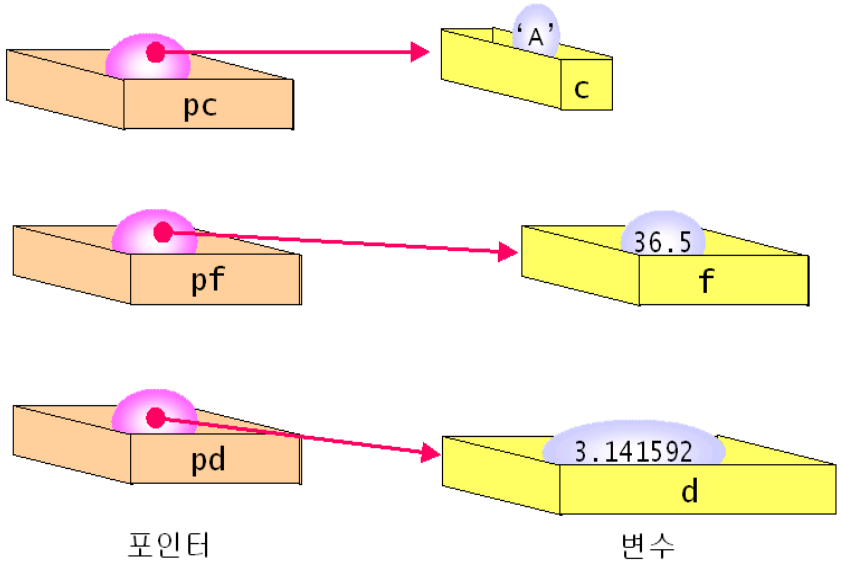
- 포인터 변수: 주소를 값으로 가지는 변수로 '*' 를 사용하여 선언한다
- int *p ; p는 포인터 변수로, 주소를 값으로 가져야하며, p가 가리키는 위치에 저장된 데이터의 유형은 정수형이어야 한다. 그러면, double *f ; 는 어떤 의미?

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



다양한 포인터의 선언

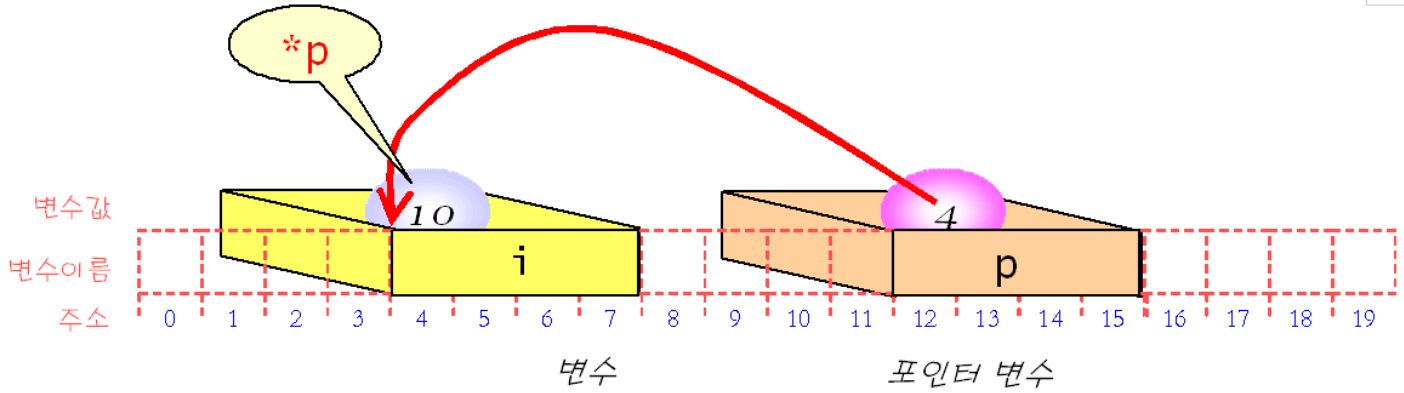
```
char c = 'A';                   // 문자형 변수 c
float f = 36.5;                 // 실수형 변수 f
double d = 3.141592;            // 실수형 변수 d

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 가리키고 있는 위치의 값을 읽거나 쓰고자 할 경우 사용하는 연산자

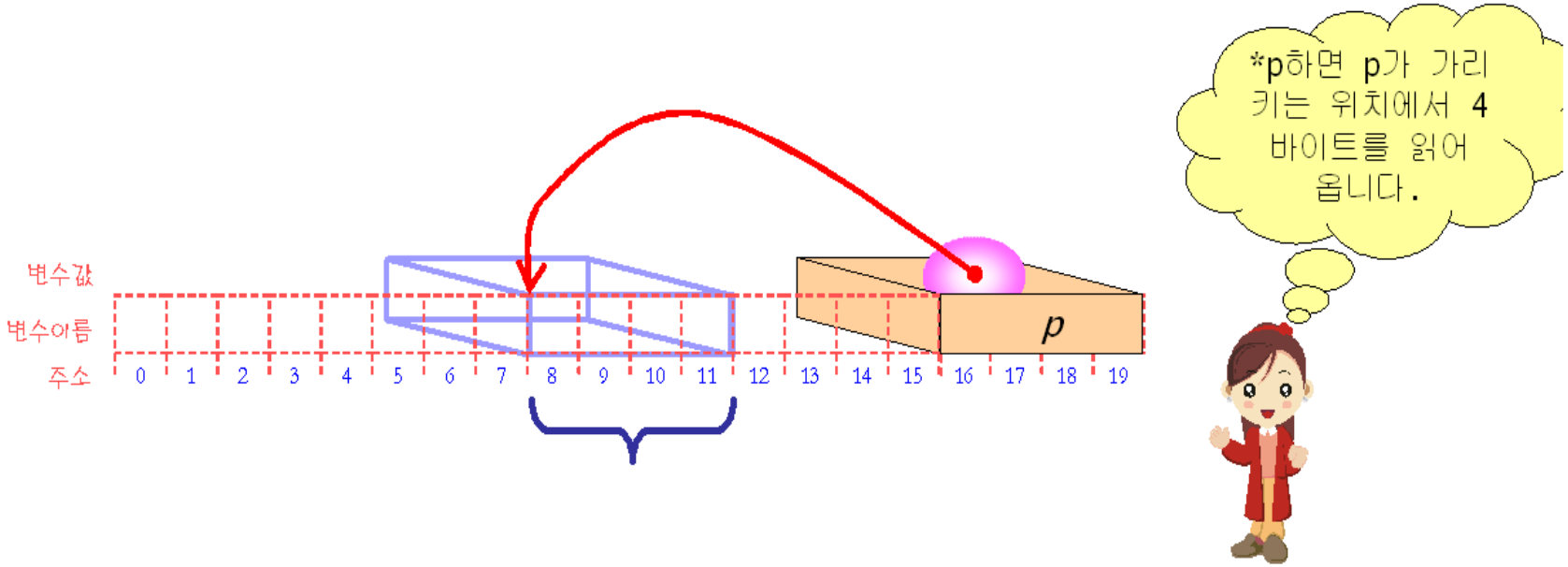
```
int i = 10;
int *p = &i;

printf("%d\n", *p);     // 10이 출력된다.

*p = 20;
printf("%d\n", *p);     // 20이 출력된다.
```



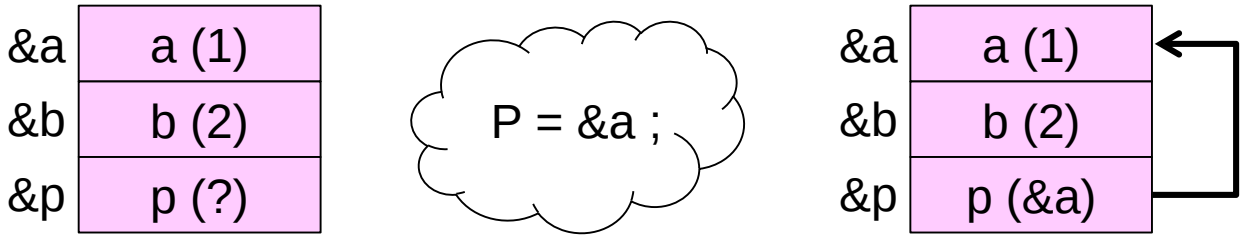
간접 참조 연산자의 해석

- 간접 참조 연산자: 지정된 위치에서 포인터의 타입에 따라 그 유형의 값을 읽거나 저장한다.

```
int *p = 8;
char *pc = 8;
double *pd = 8;
```



- int a=1, b=2, ***p** ;



- int i=3, j=5, *p=&i, *q=&j, *r ;
  double x ;
  - p == &i ; ?
  - **&p ; ?
  - r = &x ; → **오류?**
  - 7 * *p/*q + 7 = ?
  - *(r=&j) * = *p

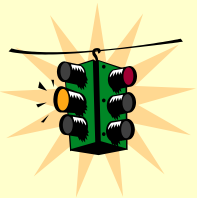
포인터 예제 #1

```c
#include <stdio.h>

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

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

    printf("i = %d\n", i);          // 변수의 값 출력
    printf("*p = %d\n", *p);        // 포인터를 통한 간접 참조 값 출력

    return 0;
}
```

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

포인터 예제 #2

```c
#include <stdio.h>

int main(void)
{
    char c = 'A';                   // 문자형 변수 정의
    int i = 10000;                  // 정수형 변수 정의
    double d = 6.78;                // 실수형 변수 정의

    char *pc = &c;                  // 문자형 포인터 정의 및 초기화
    int *pi = &i;                   // 정수형 포인터 정의 및 초기화
    double *pd = &d;                // 실수형 포인터 정의 및 초기화

    (*pc)++;                        // 간접 참조로 1 증가
    *pi = *pi + 1;                  // 간접 참조로 1 증가
    *pd += 1;                       // 간접 참조로 1 증가

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

    return 0;
}
```

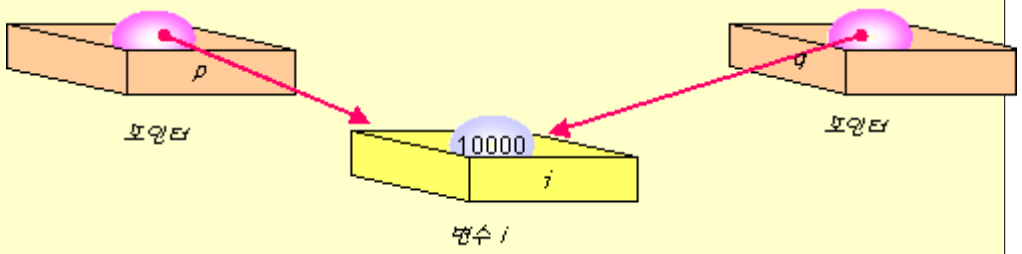


*pc++라고 하면 안됨

```
c = B
i = 10001
d = 7.780000
```

포인터 예제 #3

```c
#include <stdio.h>

int main(void)
{
    int i = 10000;              // 정수 변수 정의
    int *p, *q;                 // 정수형 포인터 정의

    p = &i;                     // 포인터 p와 변수 i를 연결
    q = &i;                     // 포인터 q와 변수 i를 연결

    *p = *p + 1;                // 포인터 p를 통하여 1 증가
    *q = *q + 1;                // 포인터 q를 통하여 1 증가

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

    return 0;
}
```




```
i = 10002
```

포인터 사용시 주의점 #1

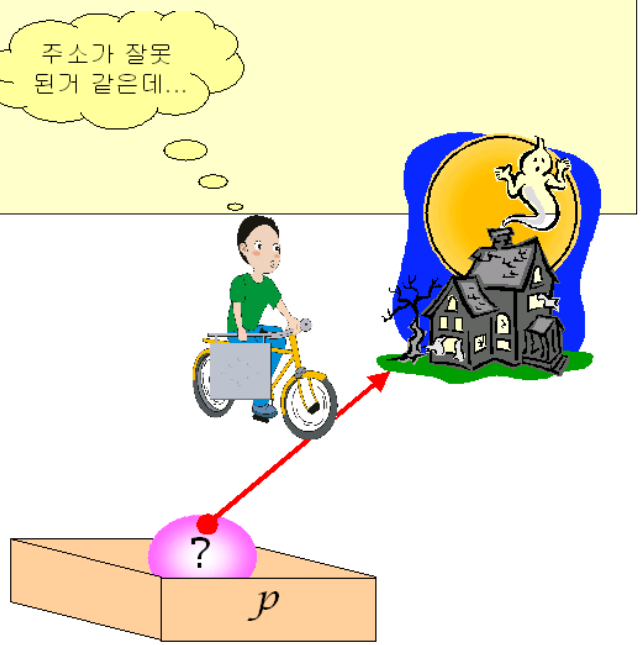
- 포인터가 가리키는 곳에 저장될 변수의 타입과 실제 저장하는 변수의 타입은 서로 일치하여야 한다.

```
#include <stdio.h>

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

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

    return 0;
}
```

포인터 사용시 주의점 #2

- 초기화가 되지 않은 포인터를 사용하지 말 것.

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

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



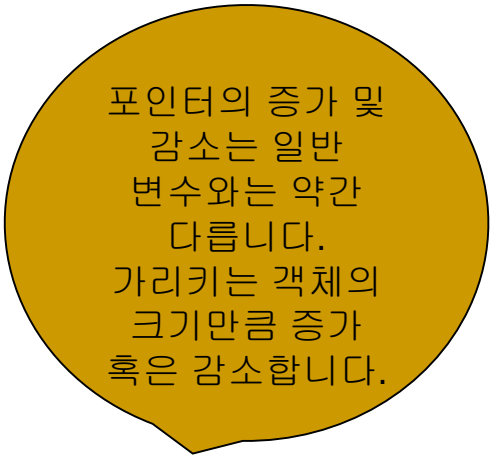
포인터 사용시 주의점 #3

- 포인터가 아무것도 가리키고 있지 않는 경우에는 NULL로 초기화, 즉 p = NULL ; 혹은 p = 0 ;
  - NULL 포인터를 가지고 간접 참조하면 하드웨어로 감지 가능.
  - 포인터의 유효성 여부 판단이 쉽다.
- 상수를 가리키지 않도록 주의
  - 즉, p = &3 ;
- 배열의 이름을 가리키지 않도록 주의
  - 즉, int a[10] ; int *p = &a ; → 오류
- 연산식을 가리키지 않도록 주의
  - 즉, p = &(x + 99) ; → 오류
- 레지스터 변수를 가리키지 않도록 주의
  - 즉, register int x ; int *p = &x ;

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

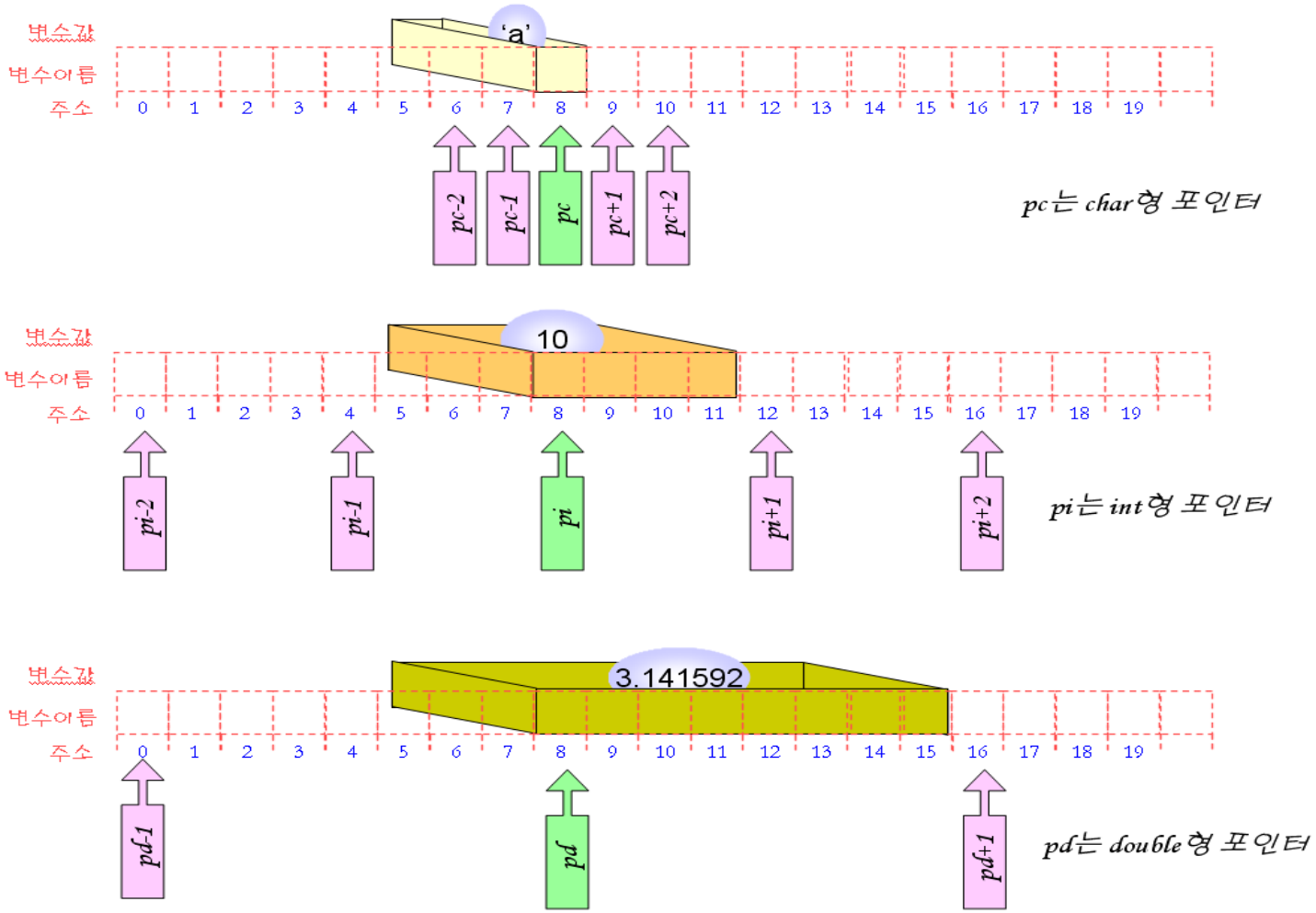


증가 연산 예제

```c
// 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, pi, pd);
	return 0;
}
```

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

포인터의 증감 연산


변수값
변수이름
주소
'a'
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
pc-2
pc-1
pc
pc+1
pc+2
pc는 char형 포인터
변수값
변수이름
주소
10
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
pi-2
pi-1
pi
pi+1
pi+2
pi는 int형 포인터
변수값
변수이름
주소
3.141592
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
pd-1
pd
pd+1
pd는 double형 포인터

- double a[2], *p, *q ;
  - p = a ;
  - q = p+1 ;
  - printf(“%d\n”, q – p) ;
  - printf(“%d\n”, (int)q – (int)p) ;

포인터간의 비교

```c
#include <stdio.h>

int main(void)
{
    int i, j, *p1, *p2;

    p1 = &i;
    p2 = &j;

    if( p1 != NULL )
        printf("p1이 NULL이 아님\n");

    if( p1 != p2 )
        printf("p1과 p2가 같지 않음\n");

    if( p1 < p2 )
        printf("p1이 p2보다 앞에 있음\n");
    else
        printf("p1이 p2보다 앞에 있음\n");

    return 0;
}
```

포인터와 다른 포인터 비교 가능

```
p1이 NULL이 아님
p1과 p2가 같지 않음
p1이 p2보다 앞에 있음
```

간접 참조 연산자와 증감 연산자

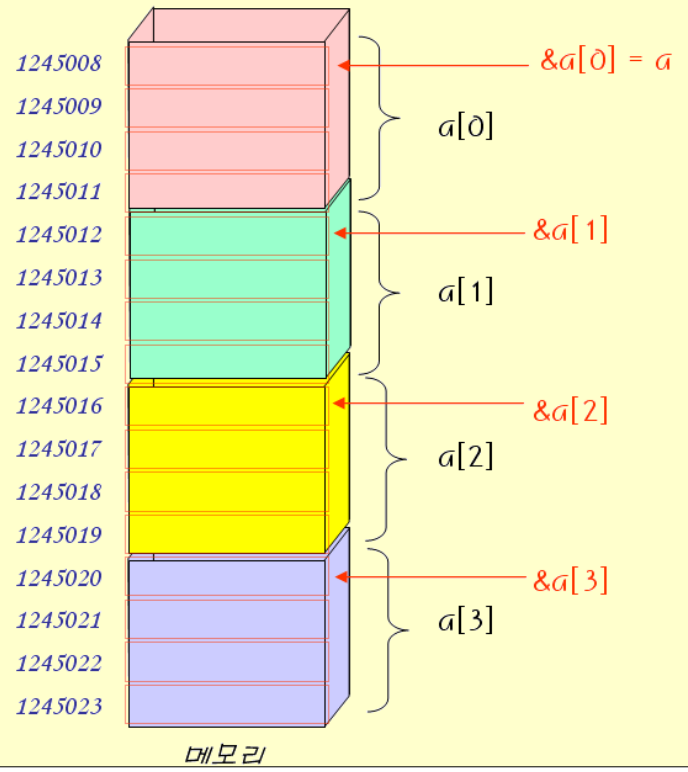
수식	의미
v = *p++	p가 가리키는 값을 v에 대입한 후에 p를 증가한다.
v = (*p)++	p가 가리키는 값을 v에 대입한 후에 가리키는 값을 증가한다.
v = *++p	p를 증가시킨 후에 p가 가리키는 값을 v에 대입한다.
v = ++*p	p가 가리키는 값을 가져온 후에 그 값을 증가하여 v에 대입한다.

```
// 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
```

포인터와 배열

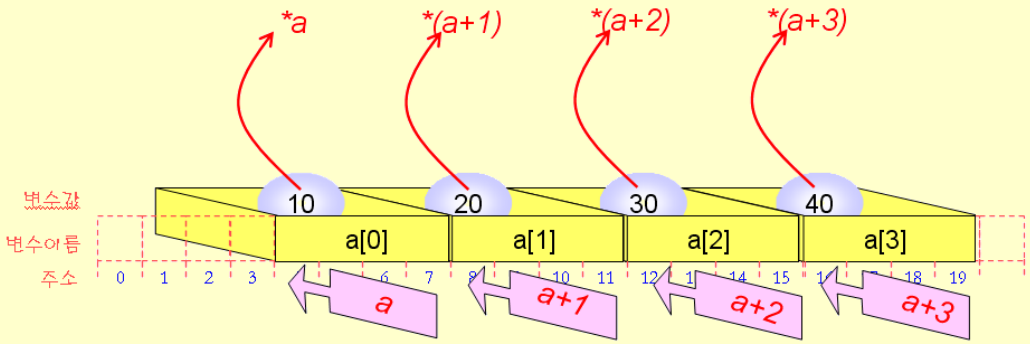
```c
// 포인터와 배열의 관계
#include <stdio.h>

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

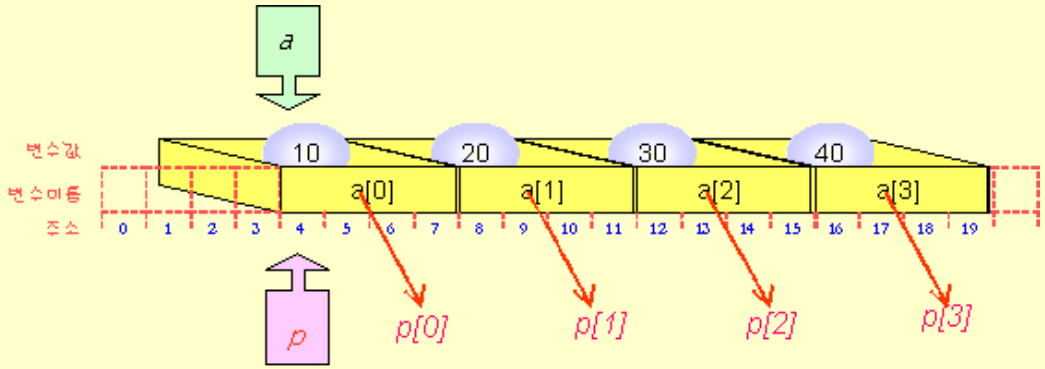
```c
// 포인터와 배열의 관계
#include <stdio.h>

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
```

포인터 → 배열의 역할

```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]);

	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 사용한 방법의 장점

- 인덱스 표기법보다 빠르다.
  - 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

배열의 원소를 역순으로 출력

```c
#include <stdio.h>

void print_reverse(int a[], int n);

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

	print_reverse(a, 5);

	return 0;
}

void print_reverse(int a[], int n)
{
	int *p = a + n - 1;			// 마지막 노드를 가리킨다.

	while(p >= a)				// 첫번째 노드까지 반복
		printf("%d\n", *p--);		// p가 가리키는 위치를 출력하고 감소
}
```

```
50
40
30
20
10
```

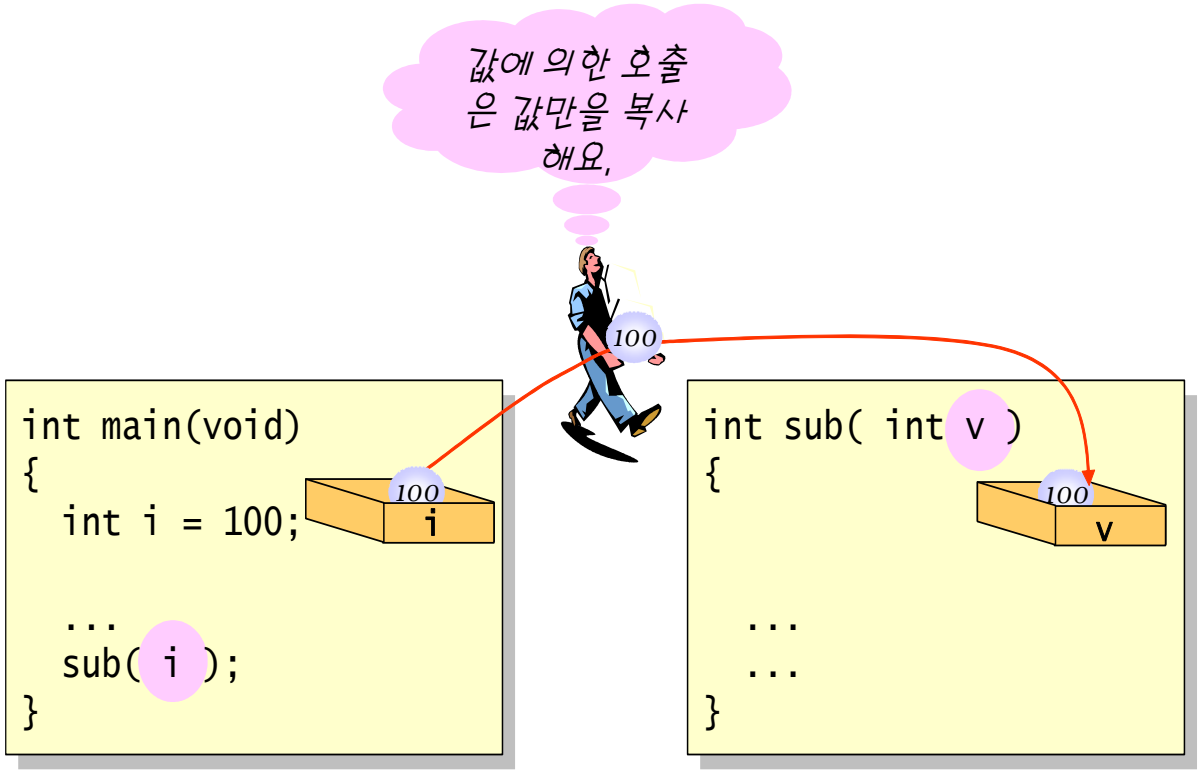
포인터 연산의 주의점

- Legal operations (정상 연산)
  - 포인터와 정수의 덧셈 혹은 뺄셈
  - 포인터 간의 비교관계 연산
  - 포인터 간의 뺄셈 : 두 포인터 사의의 객체의 개수

- Illegal operations (비정상 연산)
  - 포인터 간의 덧셈
  - 곱셈, 나눗셈, 쉬프트 연산
  - 포인터와 실수(float, double)의 덧셈

포인터와 함수

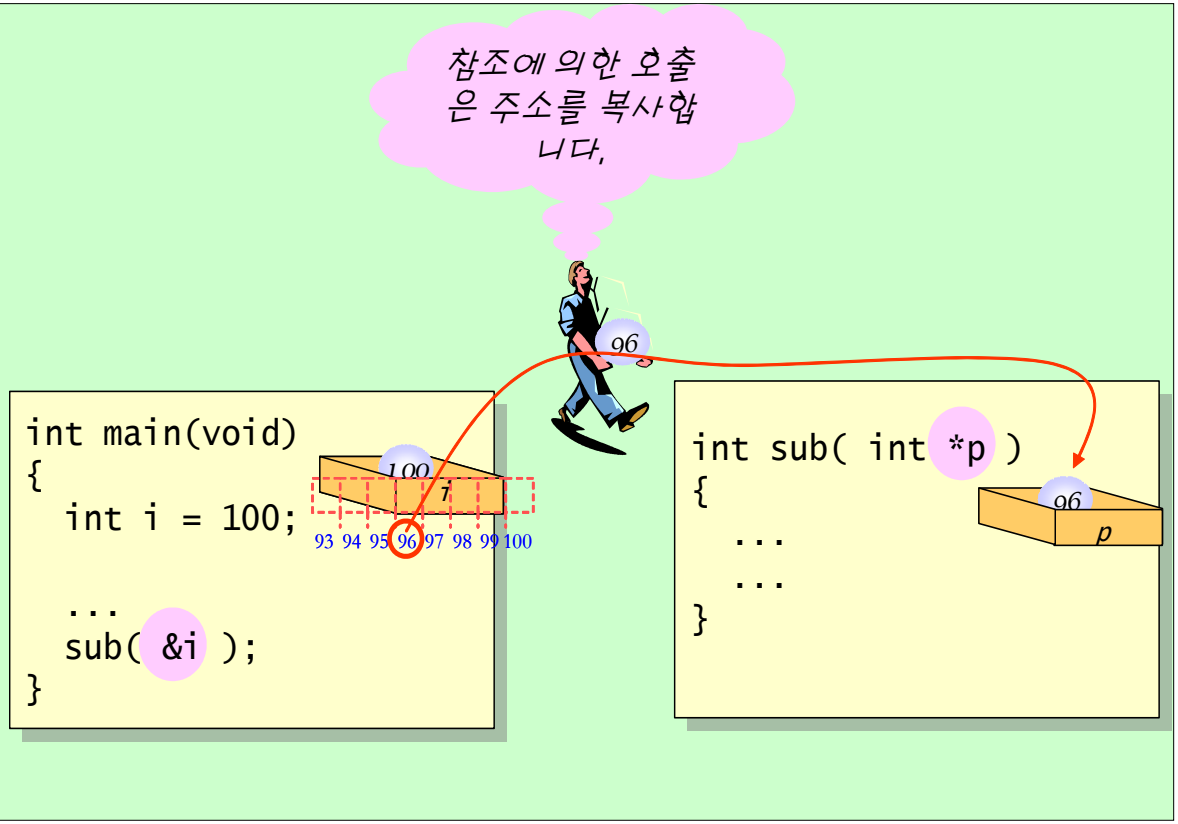
- C에서의 인수 전달 방법
  - 값에 의한 호출 (call by value): 기본적인 방법
  - 참조에 의한 호출 (call by reference) : 포인터 이용



참조에 의한 호출(Call by Referenc)

- 함수 호출시에 **포인터**를 함수의 **매개 변수로** 전달한다

```
#include <stdio.h>
void sub(int *p);

int main(void)
{
      int i = 100;

      sub(&i);
      return 0;
}

void sub(int *p)
{
      *p = 200;
```



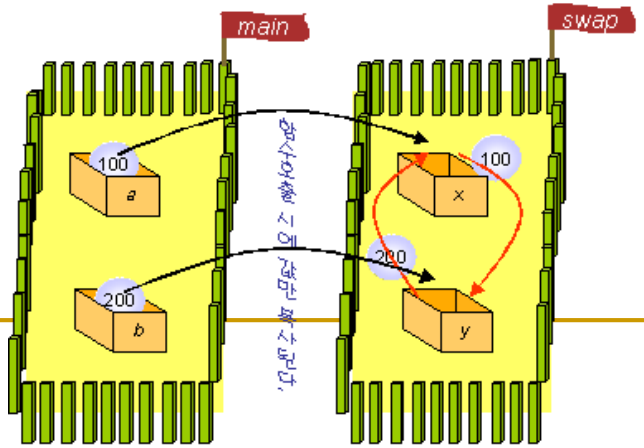
swap() 함수 #1 (call by value)

- 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("main() a=%d b=%d\n",a, b);

        swap(a, b);

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

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

        printf("swap() x=%d y=%d\n",x, y);

        tmp = x;
        x = y;
        y = tmp;

        printf("swap() x=%d y=%d\n",x, y);
}
```

```
main() a=100 b=200
swap() x=100 y=200
swap() x=200 y=100
main() a=100 b=200
```



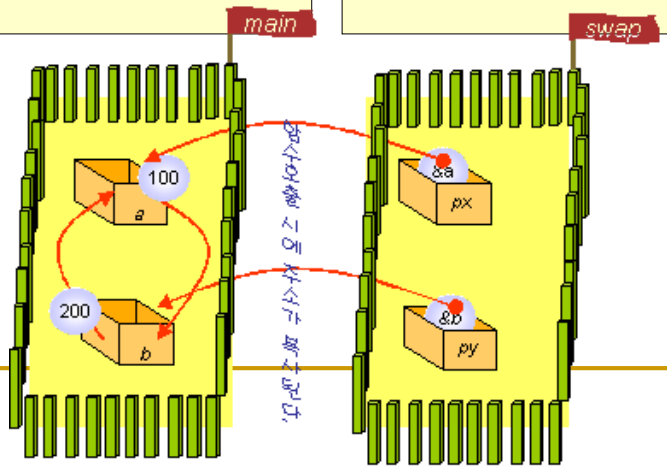
swap() 함수 #2(call by reference)

- 포인터를 매개변수로 사용한다

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

        swap(&a, &b);

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

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

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

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

```
main() a=100 b=200
swap() *px=100 *py=200
swap() *px=200 *py=100
main() a=200 b=100
```



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;
      }
}
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 y-절편을 인수로 전달

```
기울기는 1.000000, y절편은 0.000000
```

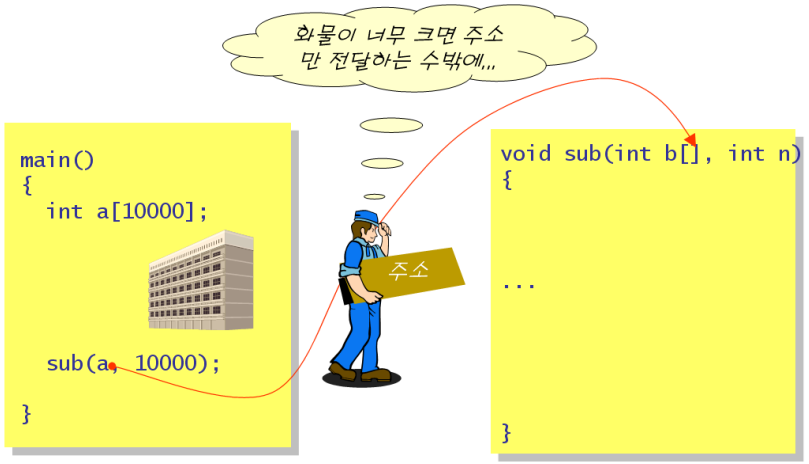
배열이 함수의 인수인 경우

- 일반 변수 vs. 배열

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

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

- 배열의 경우 call by value시 복사에 많은 시간 소모
- 그러므로, 배열의 경우에는 배열의 주소를 전달

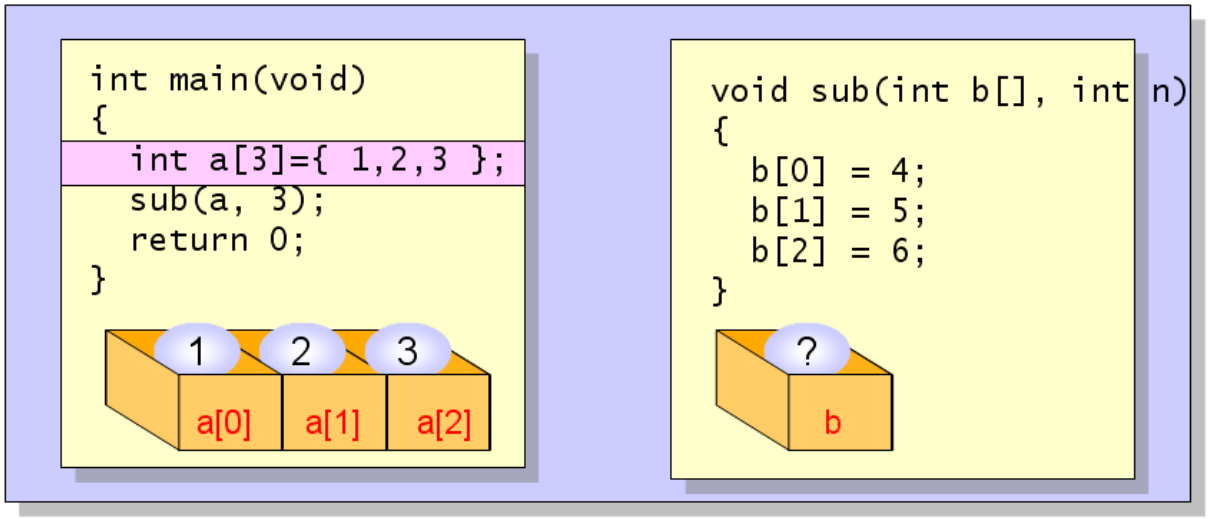


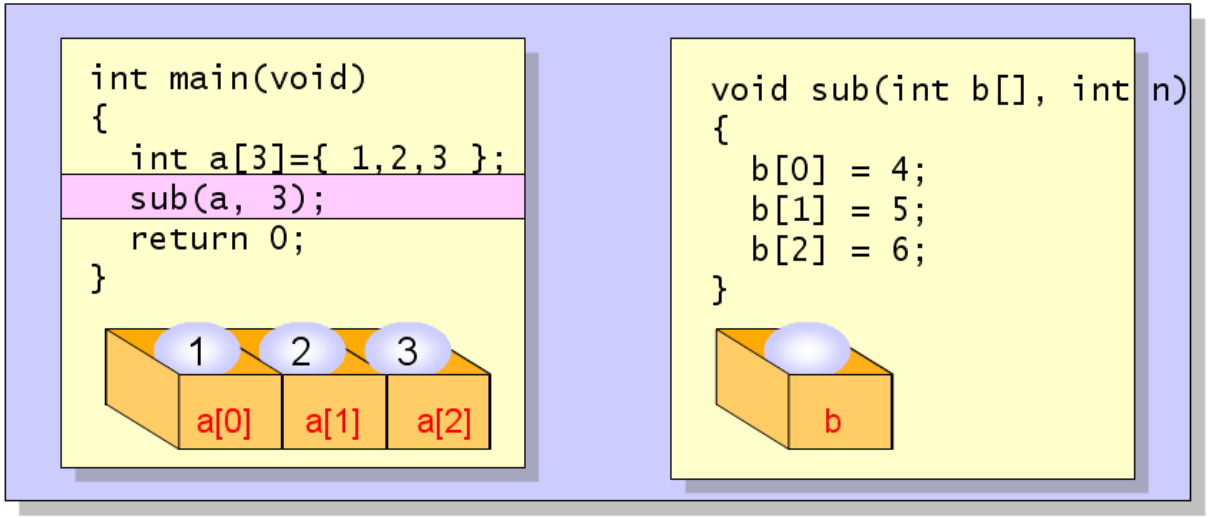
예제

```c
// 포인터와 함수의 관계
#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;
}
```

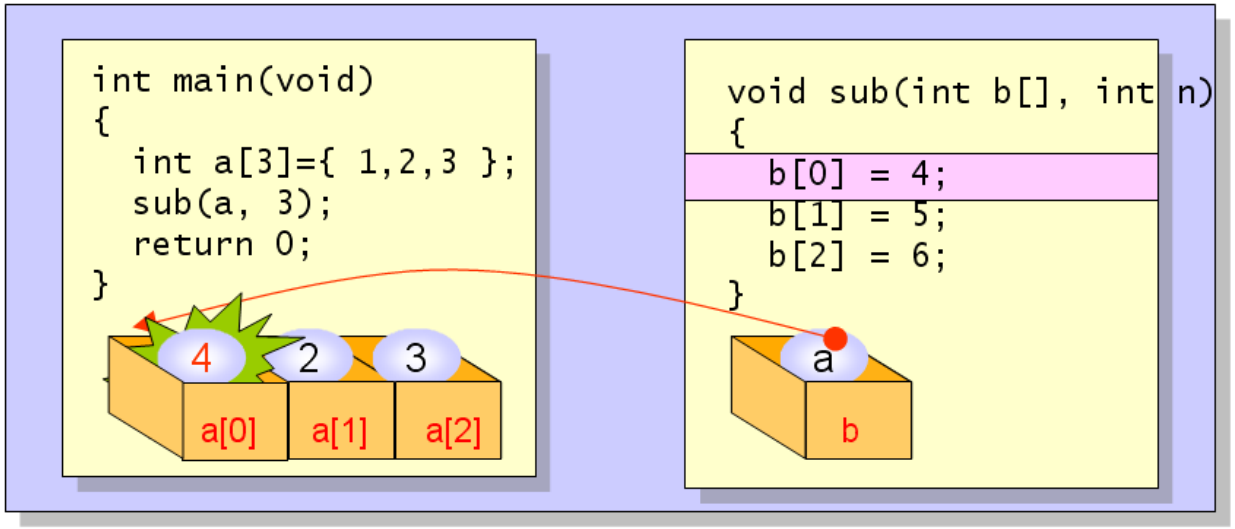
```
1 2 3
4 5 6
```

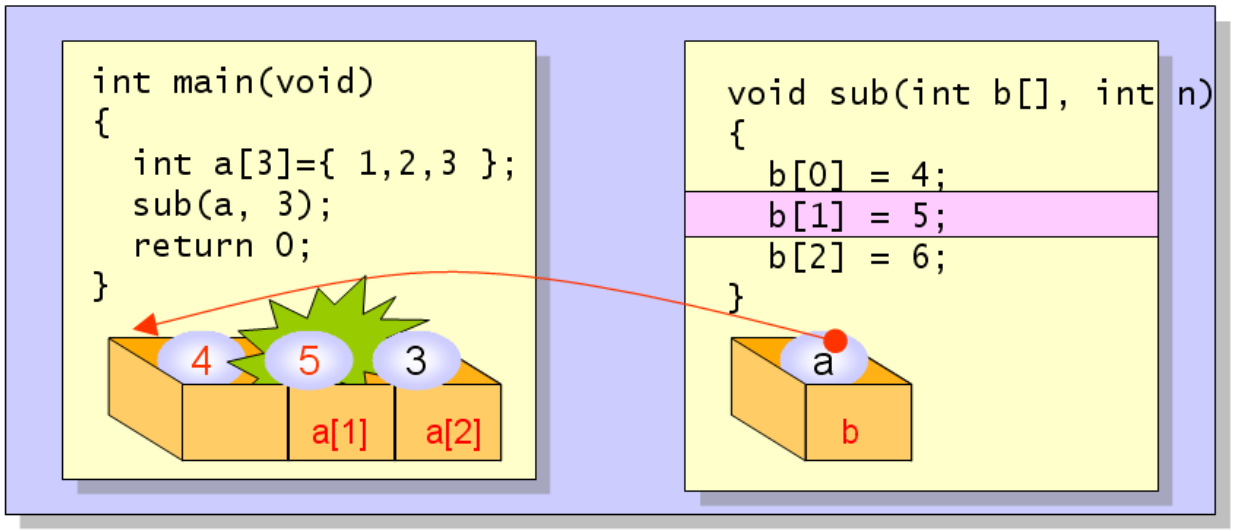
배열이 함수의 인수인 예 1/3



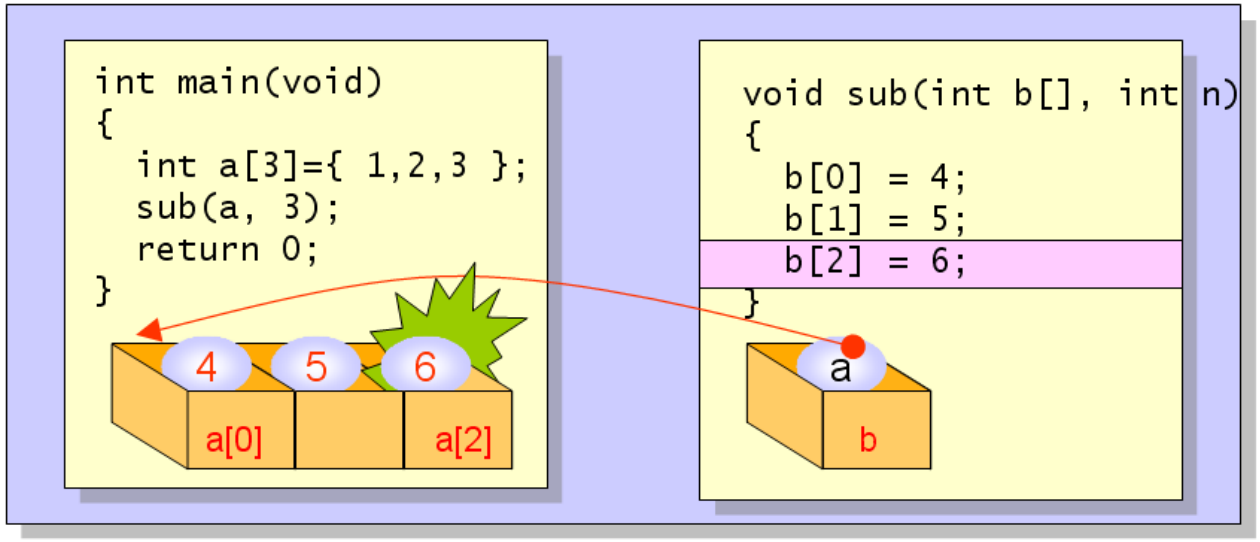


배열이 함수의 인수인 예 2/3





배열이 함수의 인수인 예 3/3



주의

- 함수가 종료되더라도 사라지지 않고 남아 있는 변수의 주소를 반환하여야 한다.
- 지역 변수의 주소를 반환하면, 함수가 종료되면 사라지기 때문에 오류

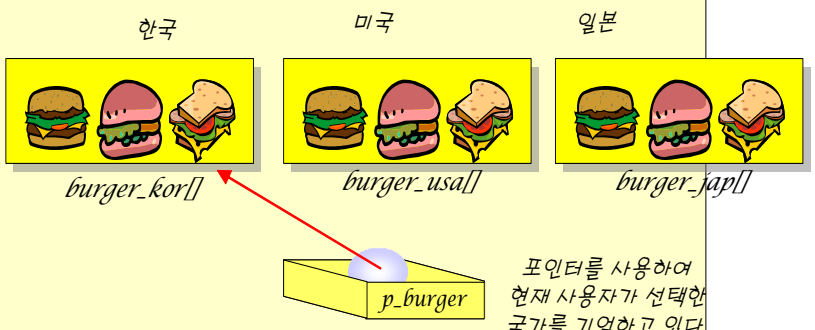
지역 변수 result는 함수가 종료되면 소멸되므로 그 주소를 반환하면 안된다.!!

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

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

응용 예제 #1

- 포인터를 통한 간접 접근의 장점
- 현재 설정된 나라의 햄버거의 가격을 출력

```
#include <stdio.h>
int main(void)
{
	int burger_kor[3]={ 3000, 2000, 4000 };
	int burger_usa[3]={ 3500, 2600, 5000 };
	int burger_jap[3]={ 3200, 2700, 4500 };
	int country;
	int *p_burger=NULL;

	printf("지역을 입력하시요:");
	scanf("%d", &country);

	if( country == 0 ) p_burger = burger_kor;
	else if( country == 1 ) p_burger = burger_usa;
	else p_burger = burger_jap;

	printf("현지역에서의 햄버거 가격:");
	printf("%d %d %d\n", p_burger[0], p_burger[1], p_burger[2]);
	return 0;
}
```



버블 정렬

```
void bubble_sort(int *p, int n)
{
	int i, scan;
	// 스캔 회수를 제어하기 위한 루프
	for(scan = 0; scan < n-1; scan++)
	{
		// 인접값 비교 회수를 제어하기 위한 루프
		for(i = 0; i < n-1; i++)
		{
			// 인접값 비교 및 교환
			if( p[i] > p[i+1] )
				swap(&p[i], &p[i+1]);
		}
	}
}
void swap(int *px, int *py)
{
	int tmp;
	tmp = *px;
	*px = *py;
	*py = tmp;
}
```

포인터를 통하여 배열 원소 교환

배열의 최소값과 최대값

```c
#include <stdio.h>
#define SIZE 10

void get_max_min(int list[], int size, int *pmax, int *pmin);

int main(void)
{
	int max, min;
	int grade[SIZE] = { 3, 2, 9, 7, 1, 4, 8, 0, 6, 5 };

	get_max_min(grade, SIZE, &max, &min);
	printf("최대값은 %d, 최소값은 %d입니다.\n", max, min);

	return 0;
}
```

배열의 최소값과 최대값

```
void get_max_min(int list[], int size, int *pmax, int *pmin)
{
	int i, max, min;

	max = min = list[0];				// 첫번째 원소를 최대, 최소값으로가정
	for(i = 1;i < size; i++)			// 두번째 원소부터 최대, 최소값과 비교
	{
		if( list[i] > max)			// list[i]가 최대값보다 크면
			max = list[i];			// list[i]를 최대값으로 설정
		if( list[i] < min)			// list[i]가 최소값보다 작으면
			min = list[i];			// list[i]를 최소값으로 설정
	}
	*pmax = max;
	*pmin = min;
}
```

```
최대값은 9, 최소값은 0입니다.
```