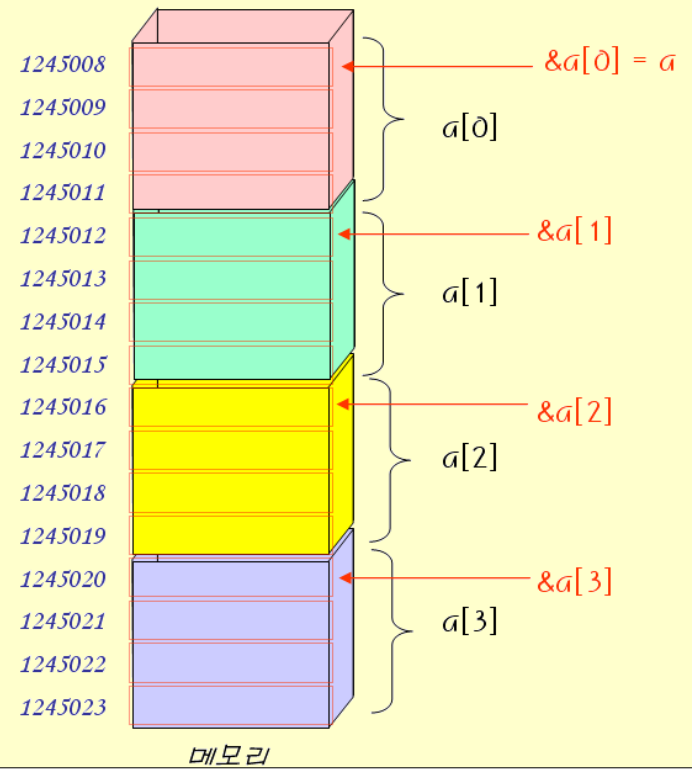
포인터 Part 2

포인터와 배열

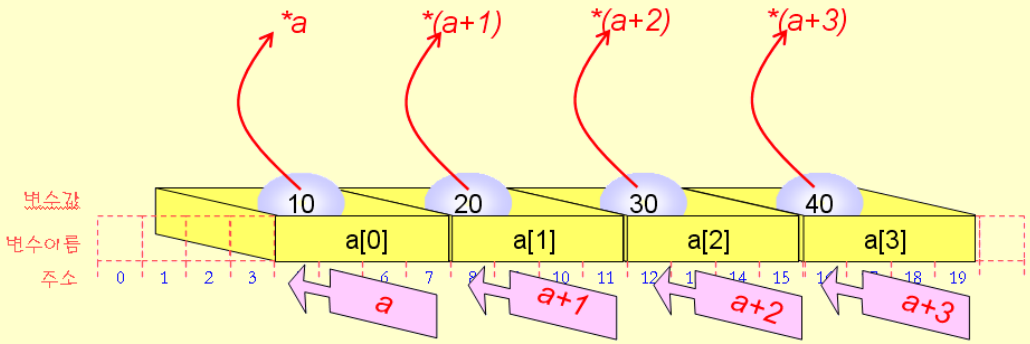
```c
// 포인터와 배열의 관계
#include <stdio.h>

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

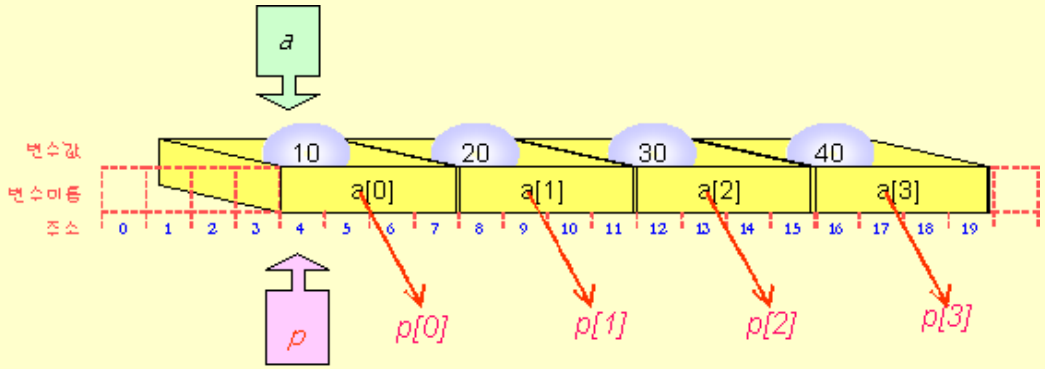
```c
// 포인터와 배열의 관계
#include <stdio.h>

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
```

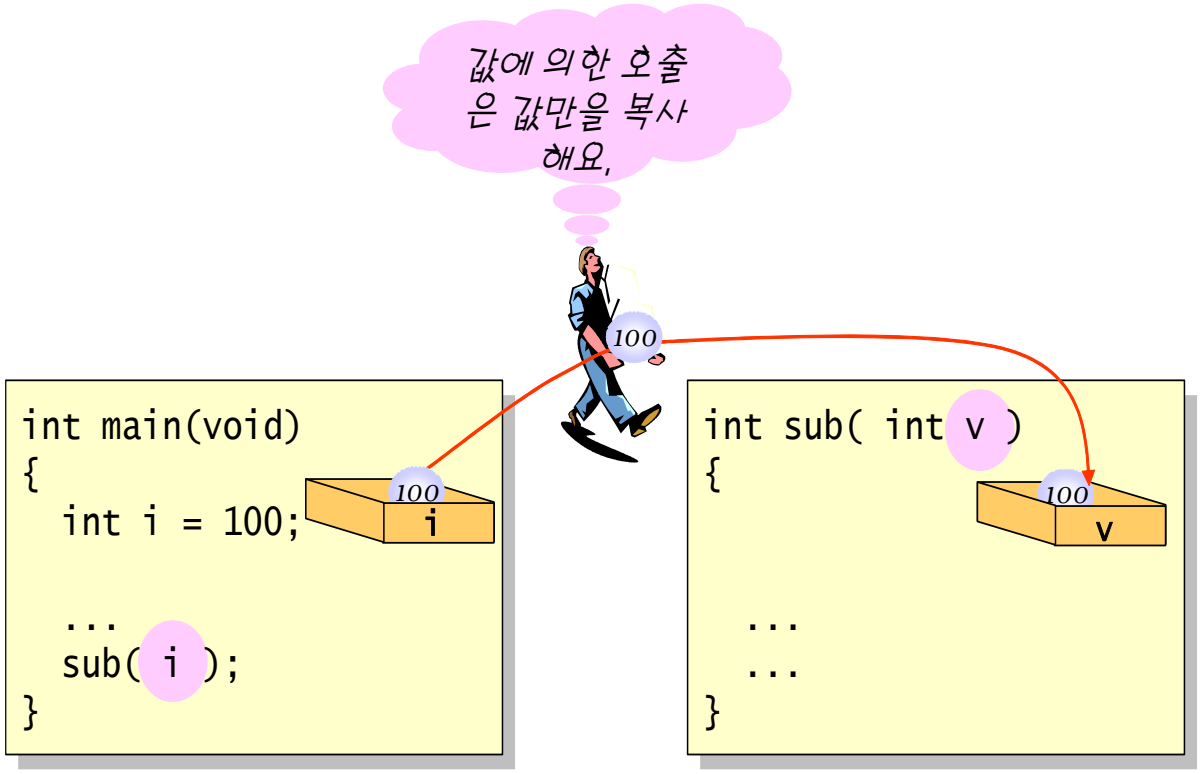
포인터 연산의 주의점

- 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

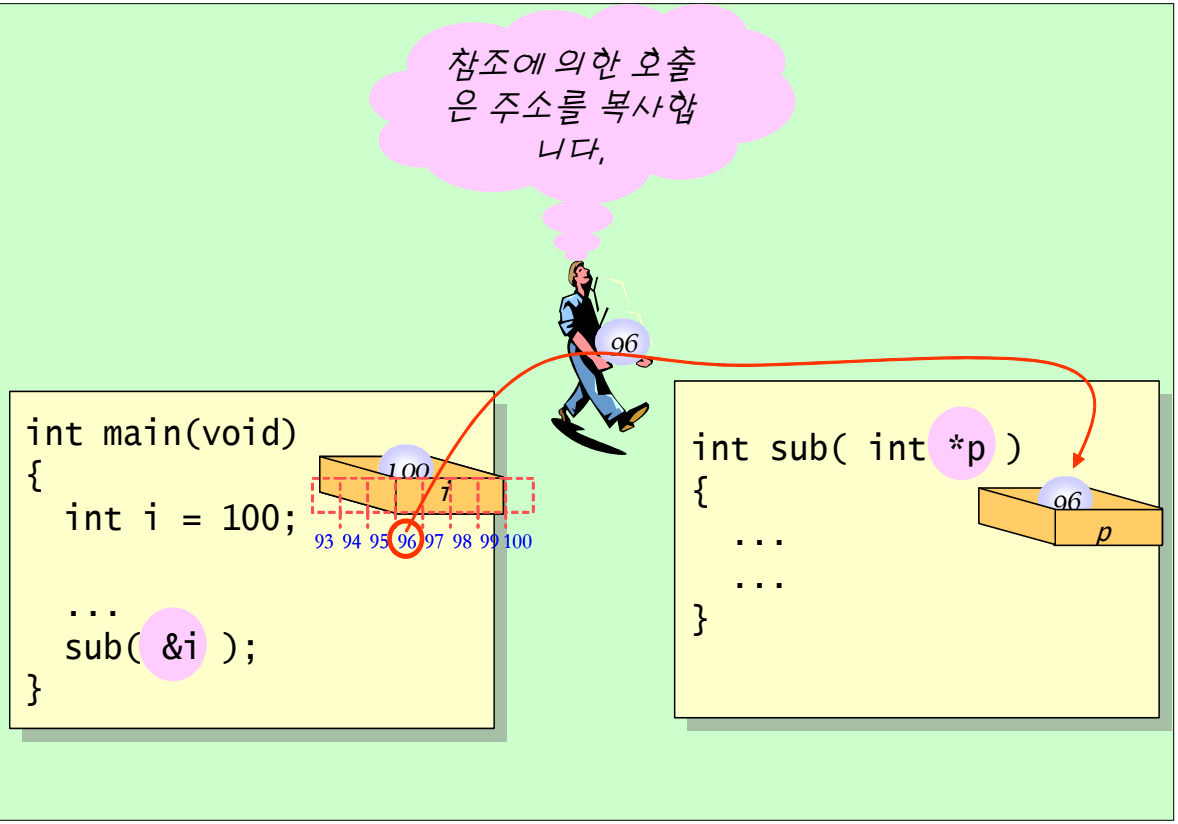
- C에서의 인수 전달 방법
  - 값에 의한 호출 (call by value): 기본적인 방법
  - 참조에 의한 호출 (call by reference) : 포인터 이용



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;
```



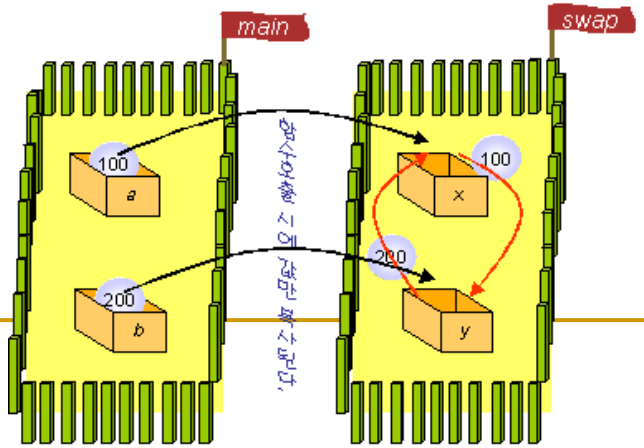
swap() 함수 #1 (call by value)

- 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
```



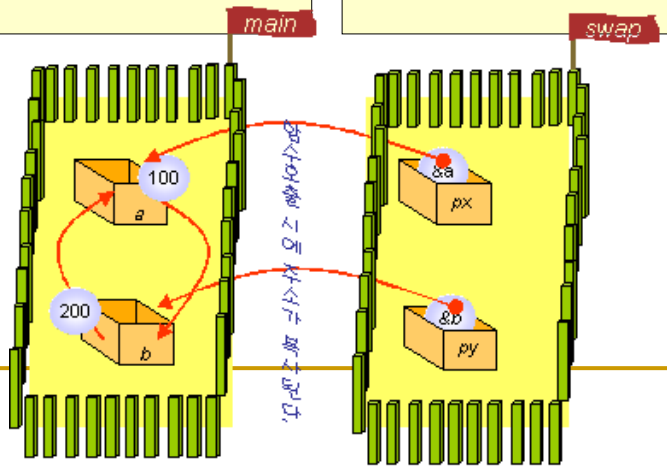
swap() 함수 #2(call by reference)

- 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
```

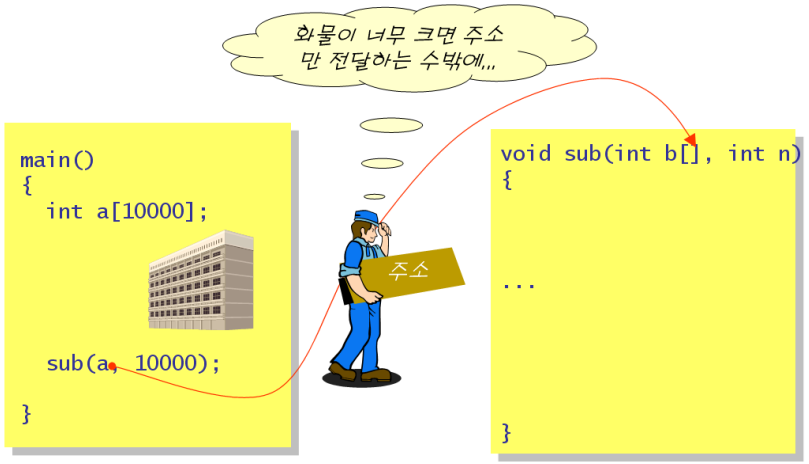
배열이 함수의 인수인 경우

- 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

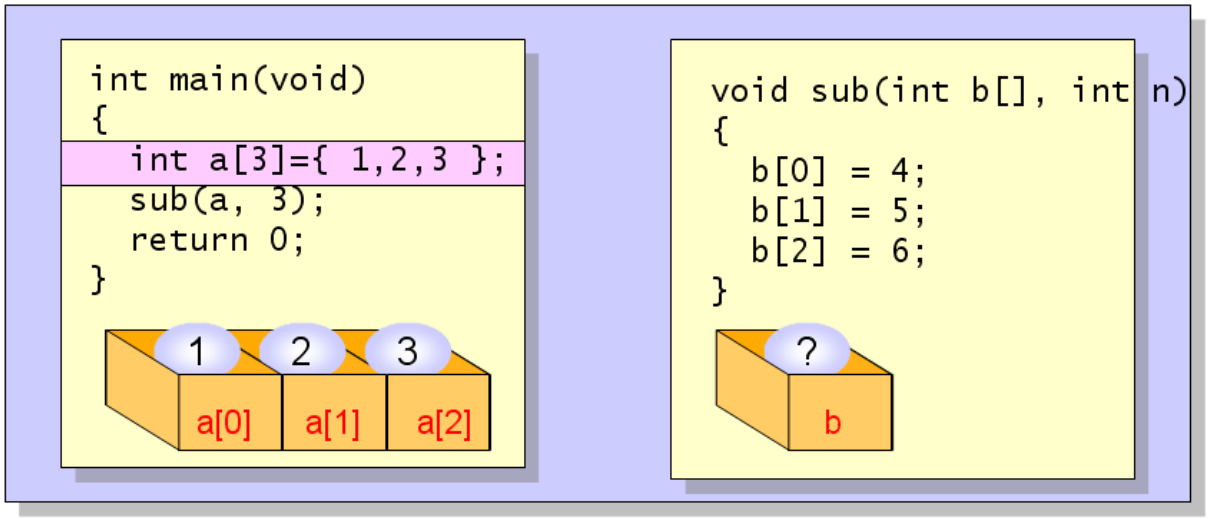


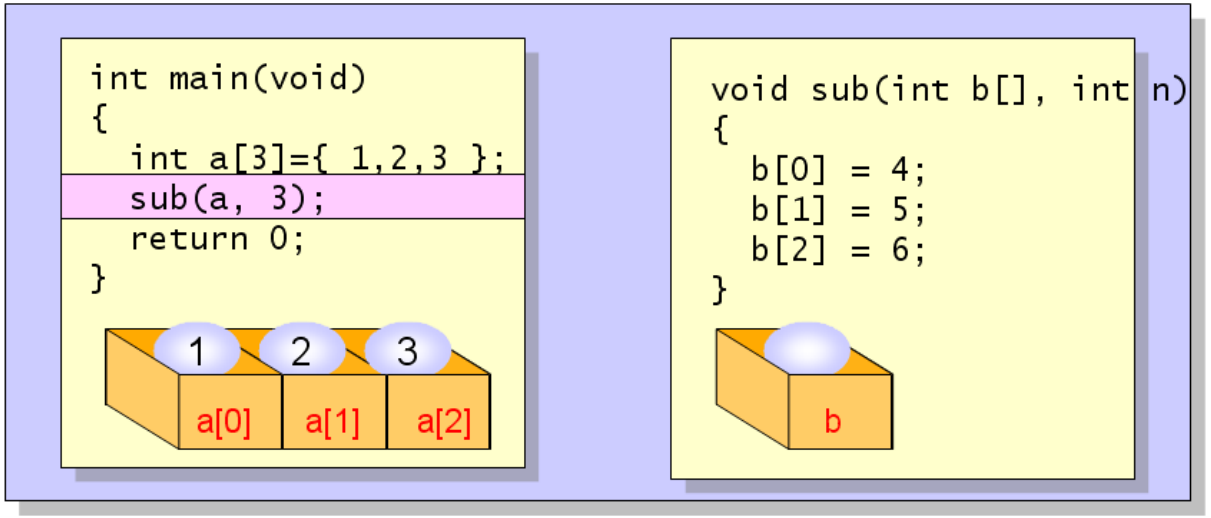
예제

```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;
}
```

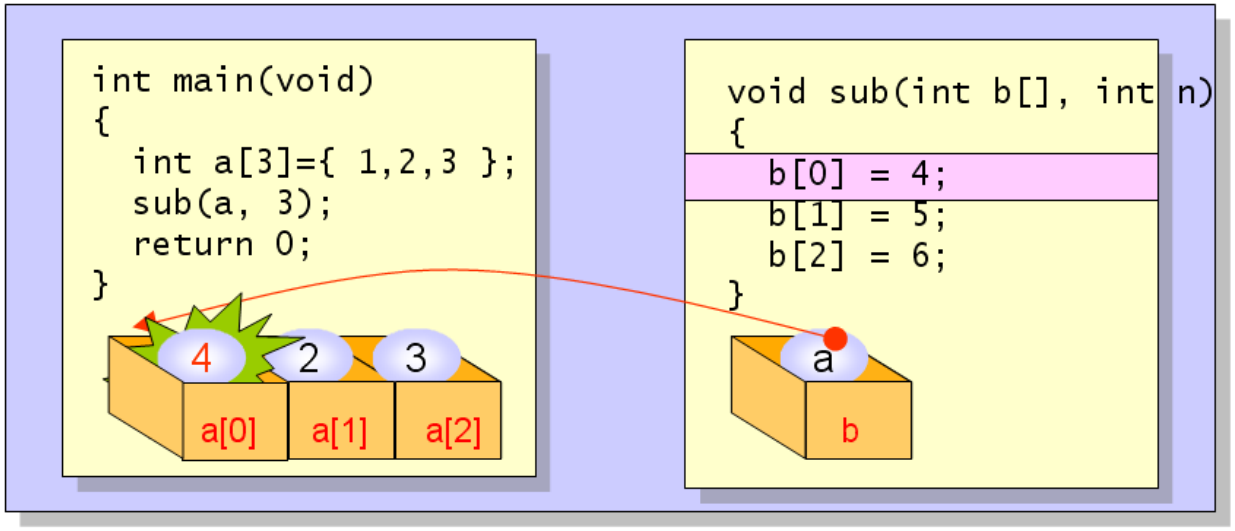
```
1 2 3
4 5 6
```

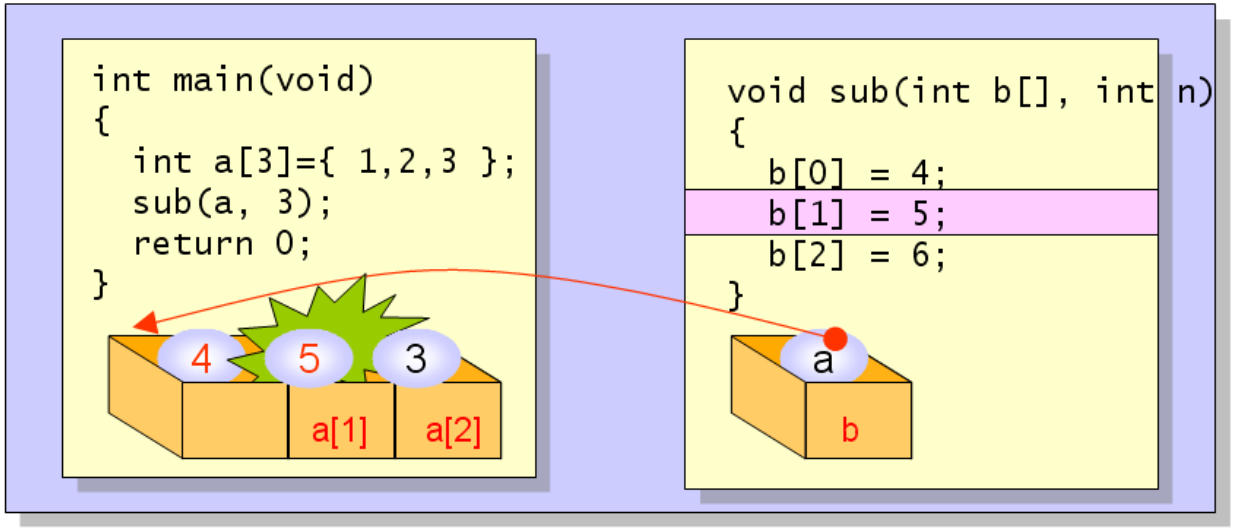
배열이 함수의 인수인 예 1/3



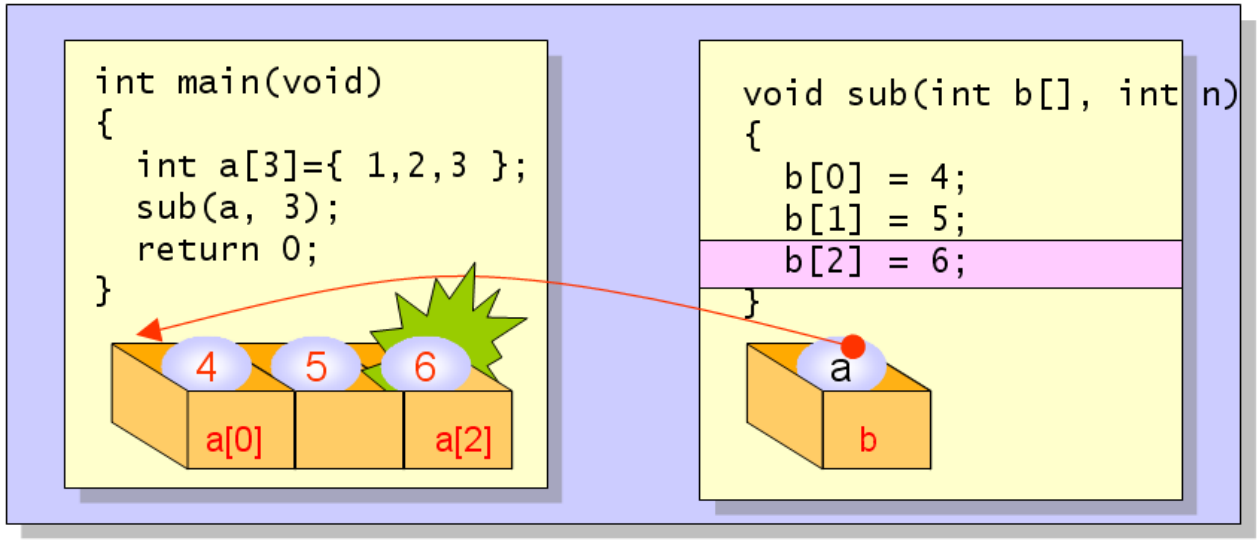


배열이 함수의 인수인 예 2/3





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

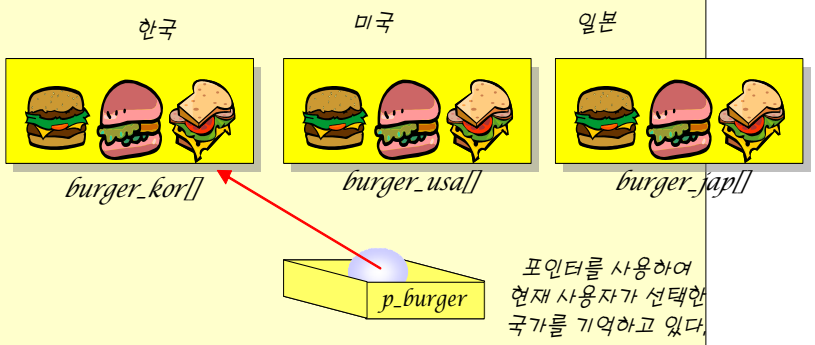
지역 변수 result는 함수가 종료되면 소멸되므로 그 주소를 반환하면 안된다.!!

```
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

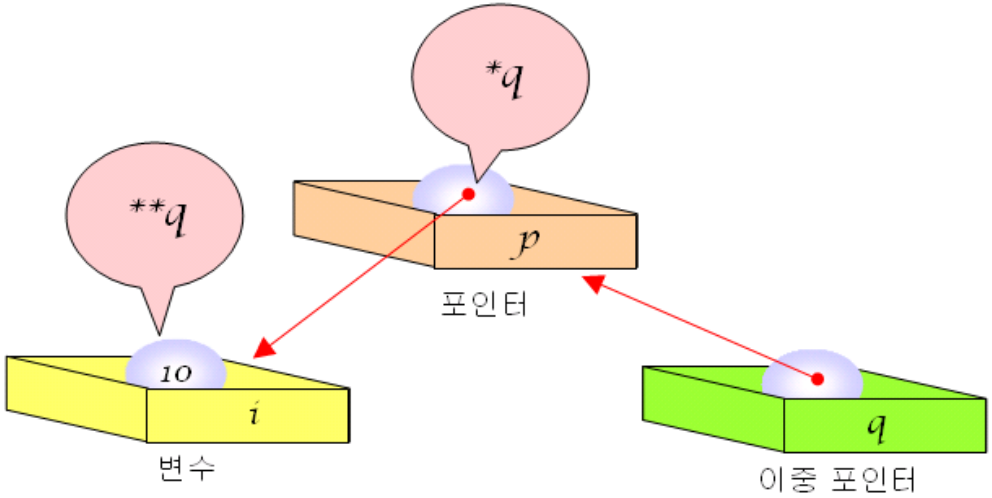
```
void get_max_min(int list[], int size, int *pmax, int *pmin)
{
	int i, max, min;

	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.
```

이중 포인터

- **이중 포인터**(double pointer): **포인터를 가리키는 포인터**
  - **Pointer to Pointer**

```
int i = 100;                    // i는 int형 변수
int *p = &i;                    // p는 i를 가리키는 포인터
int **q = &p;      // q는 포인터 p를 가리키는 이중 포인터
```

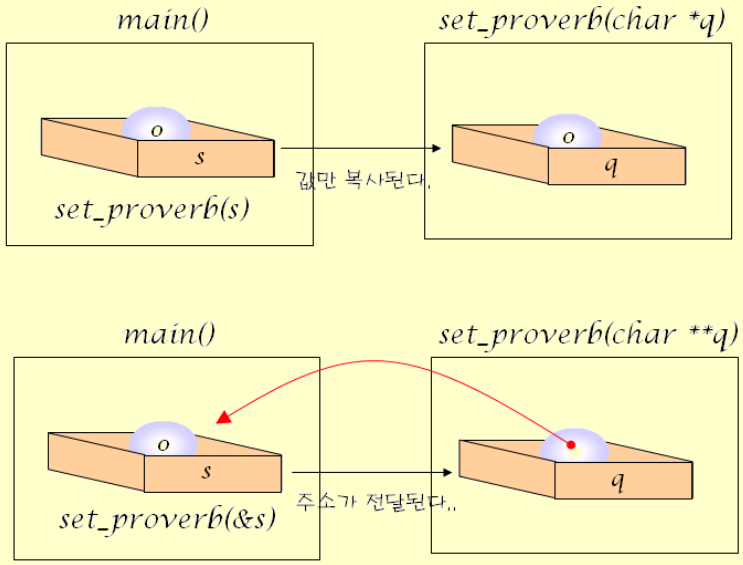


```c
// 이중 포인터 프로그램
#include <stdio.h>

int main(void)
{
	int i = 100;
	int *p = &i;
	int **q = &p;

	*p = 200;
	printf("i=%d	*p=%d	**q=%d ₩n", i, *p, **q);

	**q = 300;
	printf("i=%d	*p=%d	**q=%d ₩n", i, *p, **q);

	return 0;
}
```

```
i=200   *p=200    **q=200
i=300   *p=300    **q=300
```

예제 #2

```
// 이중 포인터 프로그램
#include <stdio.h>

void set_proverb(char **q);

int main(void)
{
        char *s = NULL;

        set_proverb(&s);
        printf("selected proverb = %s₩n",s);

        return 0;
}

void set_proverb(char **q)
{
        static char *str1="A friend in need is a friend indeed";
        static char *str2="A little knowledge is a dangerous thing";

        *q = str1;
}
```




```
selected proverb = A friend in need is a friend indeed
```

포인터 배열 (Array of Pointers)

- **포인터 배열**(array of pointers): **포인터들의 배열**

```
int a = 10, b = 20, c = 30, d = 40, e = 50;
int *pa[5] = { &a, &b, &c, &d, &e };
```