C 언어 EXPRESS(개정3판)

제 11장 포인터

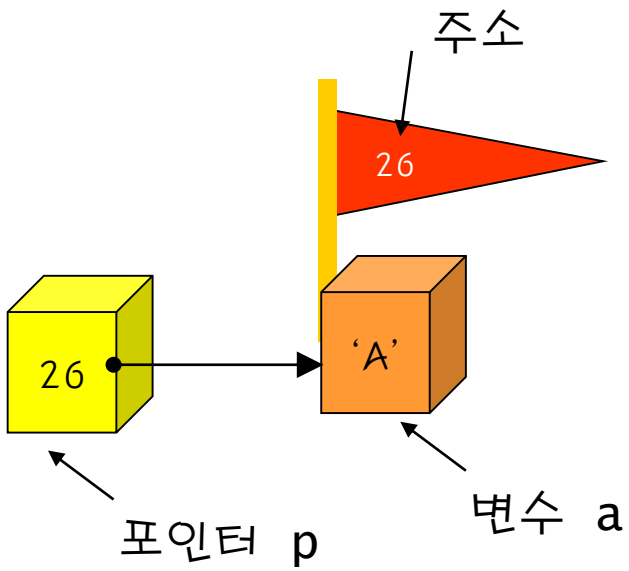
이번 장에서 학습할 내용

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

포인터(pointer)

- 포인터: 다른 변수의 주소를 가지고 있는 변수

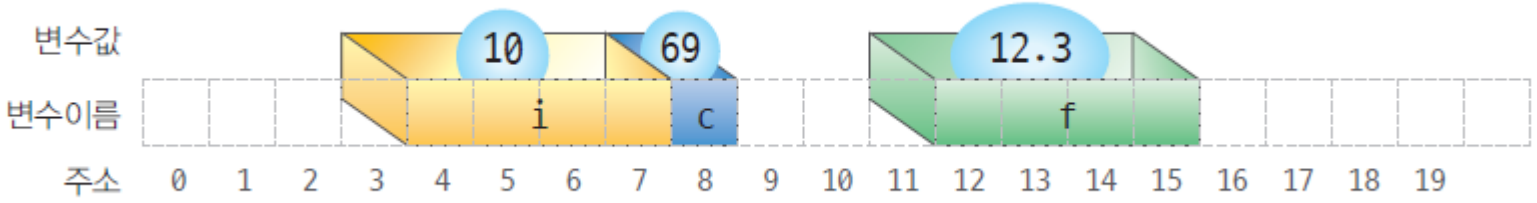
```
char a='A';
char *p;
p = &a;
```



변수와 메모리

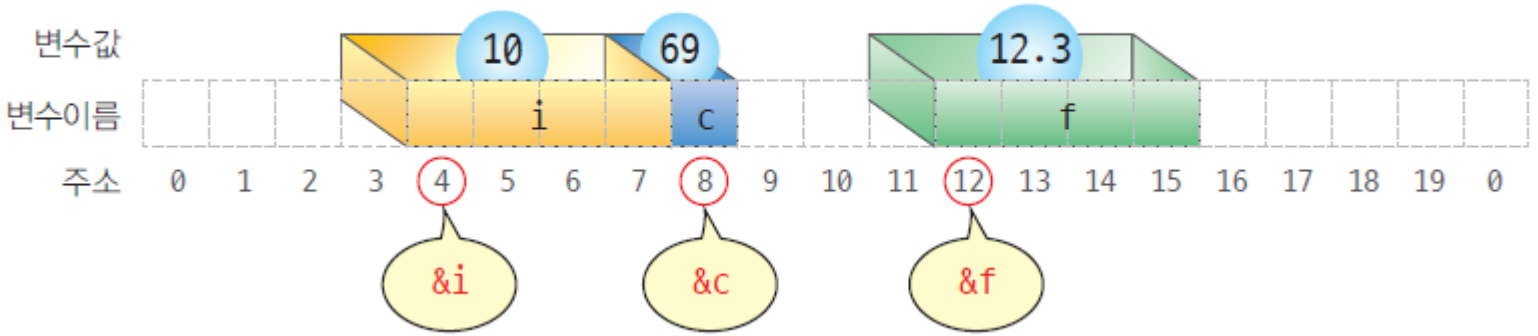
- 변수의 크기에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,...

```
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

```c
#include<stdio.h>

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

    printf("i 값 = %u , i의 주소 = %u ₩n", i, &i);
    printf("i 값 = %d , i의 주소 = %d ₩n", i, &i);


    printf("c 값 = %u , c의 주소 = %u ₩n", c &c);
    printf("c 값 = %d , c의 주소 = %d ₩n", c, &c);
    printf("c 값 = %c , c의 주소 = %d ₩n", c, &c);

    printf("f 값 = %.1f , f의 주소 = %u ₩n", f, &f);
    printf("c 값 = %.1f , f의 주소 = %d ₩n", f, &f);

    return 0;
}
```

포인터의 선언

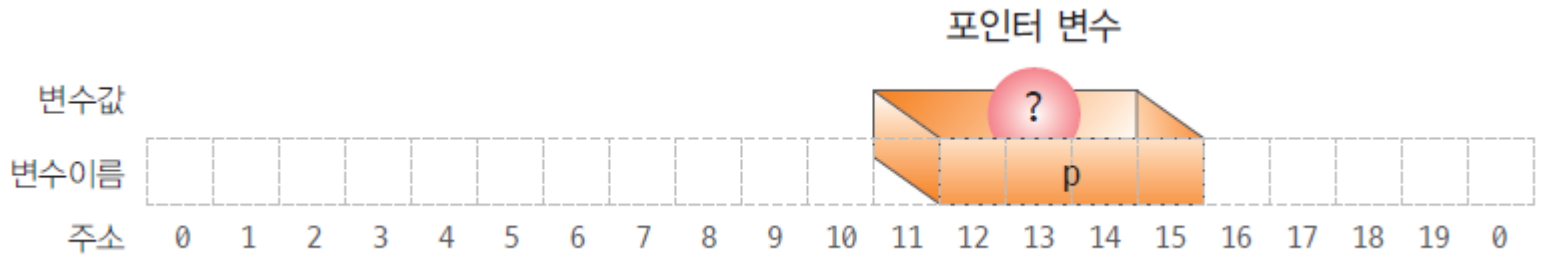
- 포인터 변수 : 변수의 주소를 가지고 있는 변수

Syntax: 포인터 선언

예

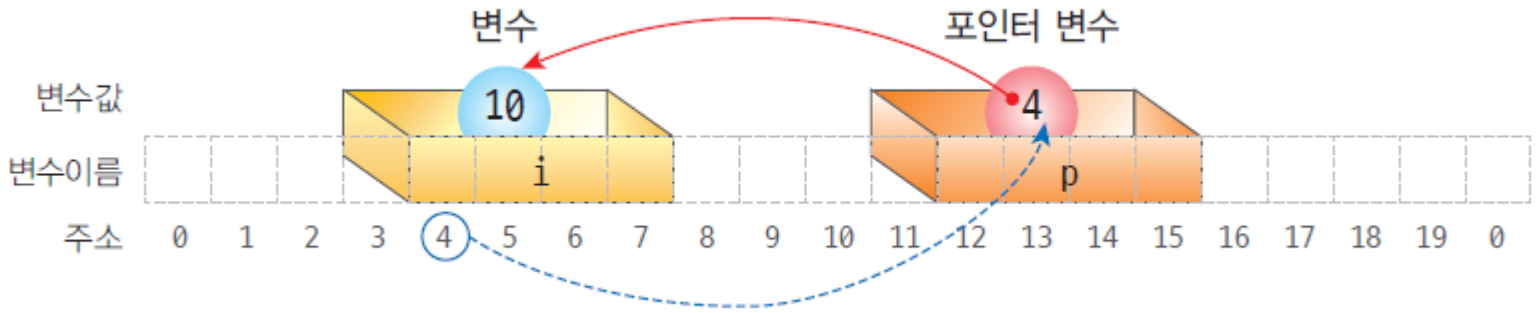
```
int * p;
```

정수를 가리키는 포인터 p



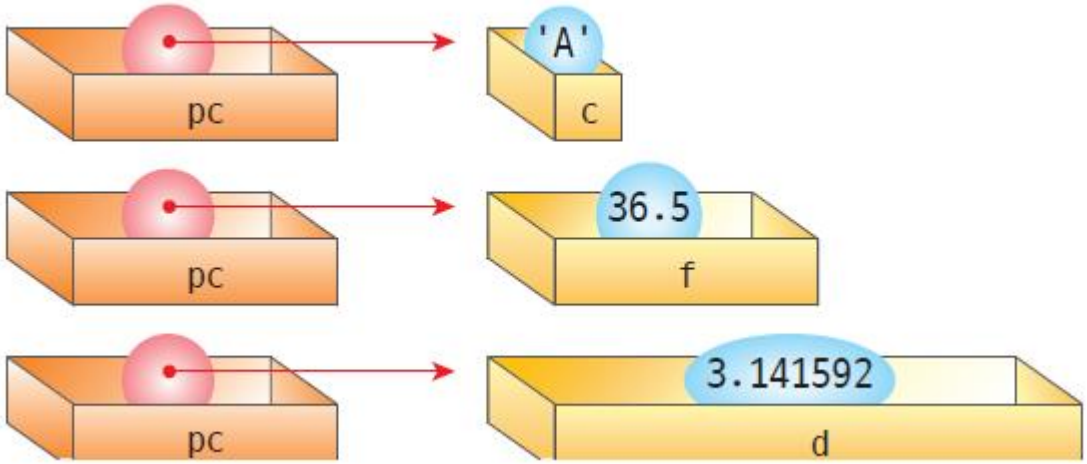
포인터와 변수의 연결

```
int i = 10;          // 정수형 변수 i 선언
int *p;              // 포인터 변수 p 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
```



```c
#include<stdio.h>

int main(void)
{
    int i = 10; int* pi = &i;
    char c = 69; char* pc = &c;
    float f = 12.3; float* pf = &f;

    printf("i 값 = %u , i의 주소 = %u ₩n", i, pi);
    printf("i 값 = %d , i의 주소 = %d ₩n", i, pi);

    printf("c 값 = %u , c의 주소 = %u ₩n", c , pc);
    printf("c 값 = %d , c의 주소 = %d ₩n", c, pc);
    printf("c 값 = %c , c의 주소 = %d ₩n", c, pc);

    printf("f 값 = %.1f , f의 주소 = %u ₩n", f, pf);
    printf("c 값 = %.1f , f의 주소 = %d ₩n", f, pf);

    return 0;
}
```

예제

```c
#include <stdio.h>

int main(void)
{
    int i = 10;
    double f = 12.3;
    int* pi = NULL;

    double* pf = NULL;
    pi = &i;
    pf = &f;

    printf("%u %u\n", pi, &i);
    printf("%u %u\n", pf, &f);
    return 0;
}
```

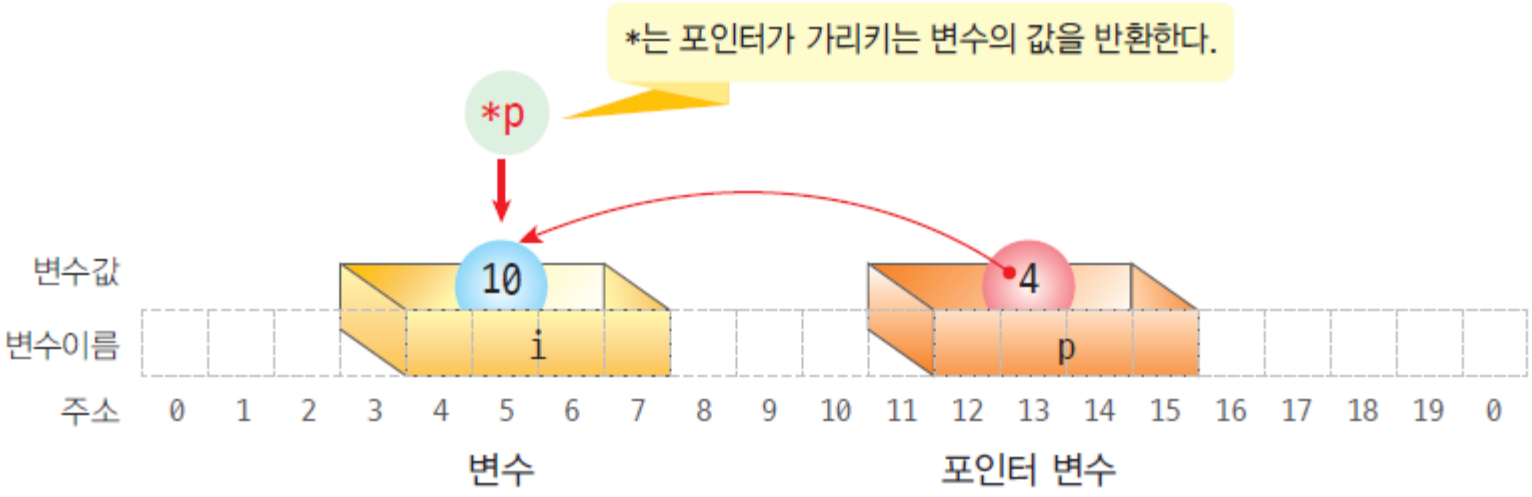
참고

- #define NULL ((void *)0)
- 0번지는 일반적으로는 사용할 수 없다(CPU가 인터럽트를 위하여 사용한다). 따라서 포인터 변수의 값이 0이면 아무 것도 가리키고 있지 않다고 판단할 수 있다.

간접 참조 연산자

- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자

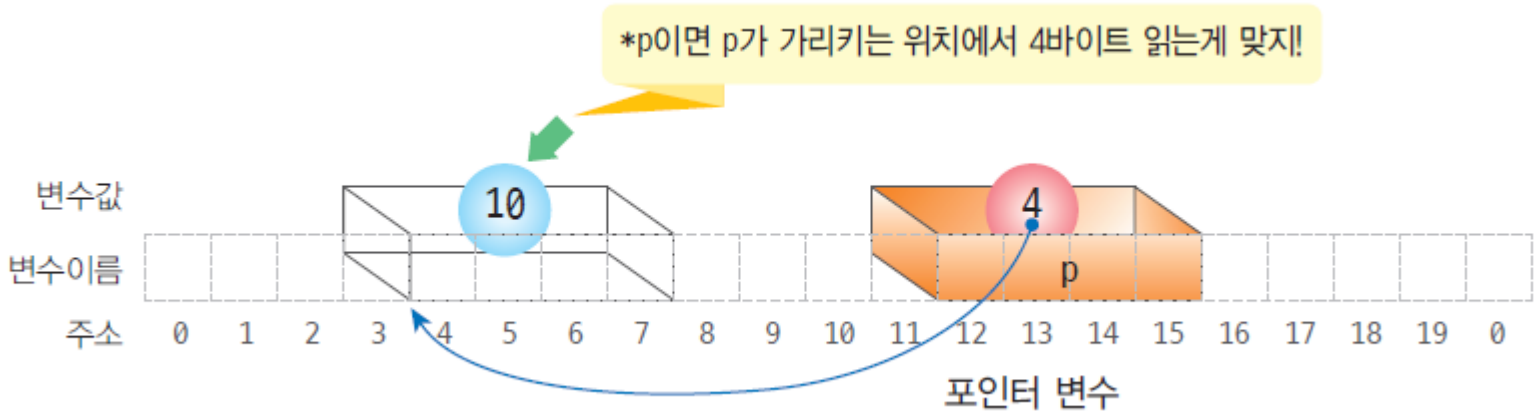
```
int i = 10;

int* p;
p = &i;

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



간접 참조 연산자의 해석

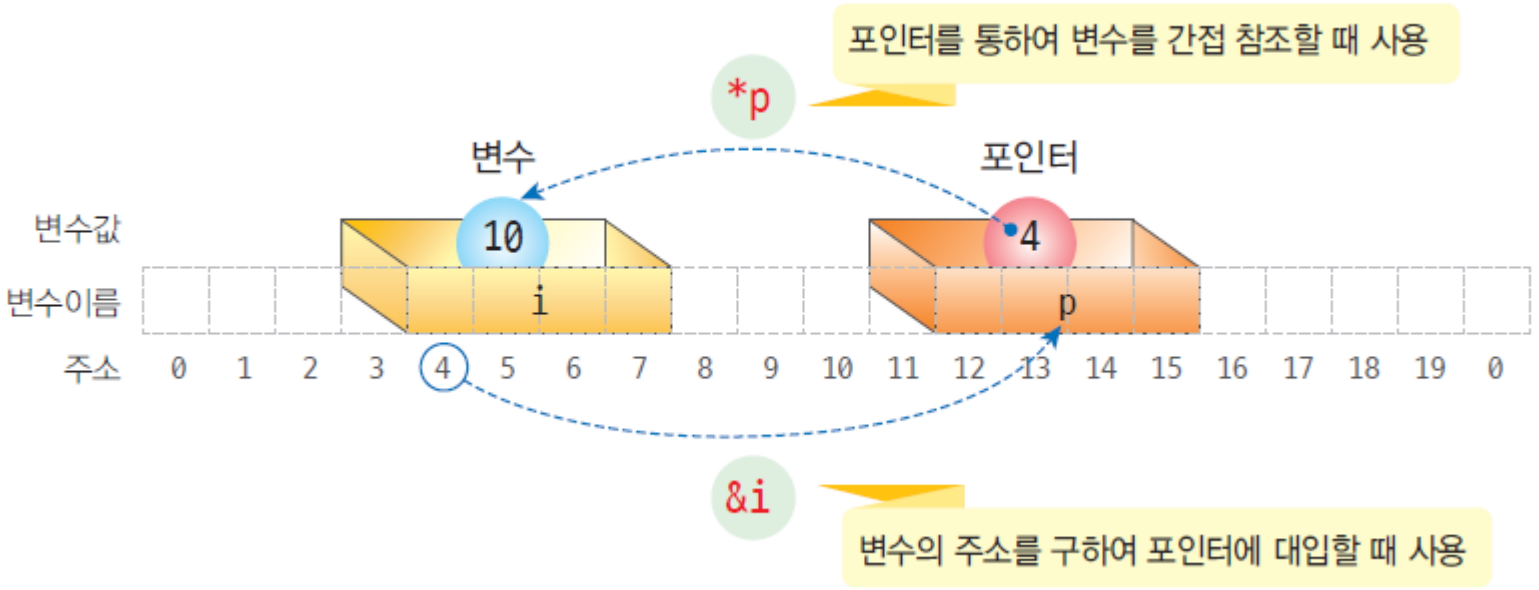
- 간접 참조 연산자: 지정된 위치에서 포인터의 타입에 따라 값을 읽어 들인다.

```
int *pi = (int *)10000;
char *pc = (char *)10000;
double *pd = (double *)10000;
```



& 연산자와 * 연산자

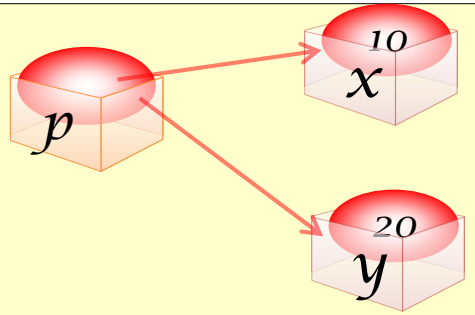
- & 연산자: 변수의 주소를 반환한다
- * 연산자: 포인터가 가리키는 곳의 내용을 반환한다.



```c
#include<stdio.h>

int main(void)
{
    int i = 10; int* pi = &i;
    char c = 69; char* pc = &c;
    float f = 12.3; float* pf = &f;

    printf("i 값 = %u , i의 주소 = %u , 주소의 내용 = %d₩n", i, pi, *pi);
    printf("i 값 = %d , i의 주소 = %d ₩n", i, pi);

    printf("c 값 = %u , c의 주소 = %u , 주소의 내용 = %d ₩n", c , pc, *pc);
    printf("c 값 = %d , c의 주소 = %d ,  주소의 내용 = %c ₩n", c, pc, *pc);
    printf("c 값 = %c , c의 주소 = %d ₩n", c, pc);

    printf("f 값 = %.1f , f의 주소 = %u ,  주소의 내용 = %.1f₩n", f, pf, *pf);
    printf("c 값 = %.1f , f의 주소 = %d ₩n", f, pf);

    return 0;
}
```

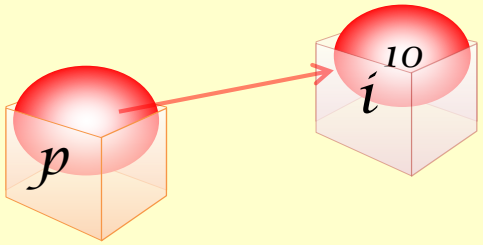
포인터 예제

```
#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;
}
```



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;
}
```



포인터를 통하여 변수의 값을 변경한다.

```c
#include<stdio.h>

int main(void)
{
    int i = 10; int* pi = &i;
    printf("i 값 = %u , i의 주소 = %u , 주소의 내용 = %d₩n", i, pi, *pi);

    *pi = 20;
    printf("i 값 = %u , i의 주소 = %u , 주소의 내용 = %d₩n", i, pi, *pi);

    return 0;
}
```

포인터 사용시 주의점

- 초기화가 안된 포인터를 사용하면 안된다.

```
int main(void)
{
    int *p;             // 포인터 p는 초기화가 안되어 있음
    *p = 100;           // 위험한 코드
    return 0;
}
```

포인터 사용시 주의점

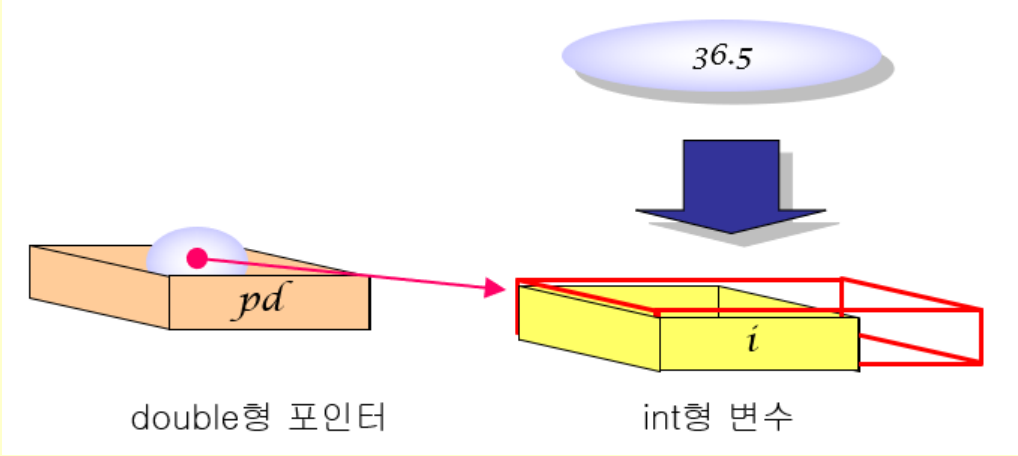
- 포인터가 아무것도 가리키고 있지 않는 경우에는 **NULL**로 초기화
- **NULL** 포인터를 가지고 간접 참조하면 하드웨어로 감지할 수 있다.

```
int *p = NULL;
```

포인터 사용시 주의점

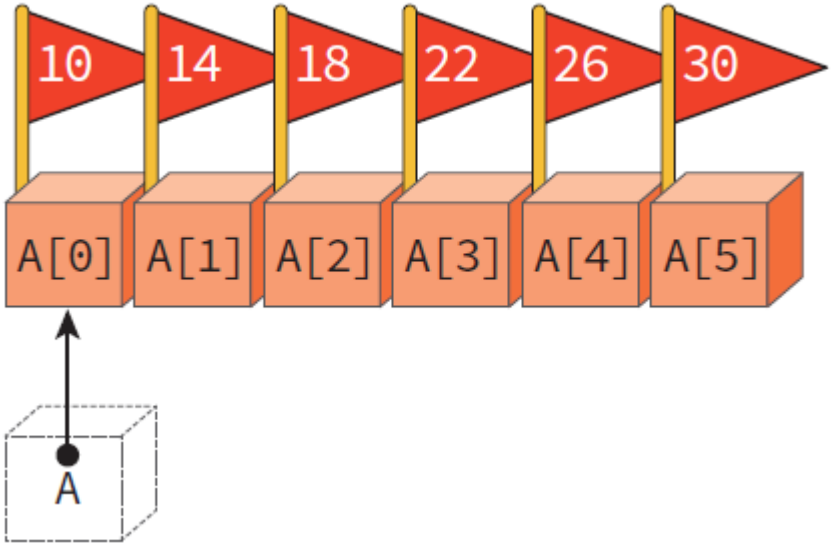
- 포인터의 타입과 변수의 타입은 일치하여야 한다.

```
#include <stdio.h>

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

    pd = &i;          // 오류!
    *pd = 36.5;

    return 0;
}
```



배열과 포인터

- 배열의 이름: 사실상의 포인터와 같은 역할



예제

```c
#include<stdio.h>
#define SIZE 6

void get_integer(int list[])
{
    printf("6개의 정수를 입력하시오:");
    for (int i = 0; i < SIZE; i++)
        scanf_s("%d", &list[i]);
}

int cal_sum(int list[])
{
    int sum = 0;
    for (int i = 0; i < SIZE; i++)
        sum = sum + *(list + i);

    return sum;
}

int main(void)
{
    int list[SIZE];
    get_integer(list);
    printf("합 = %d ₩n", cal_sum(list));
    return 0;
}
```

동적 메모리 할당

- **동적(Dynamic) 메모리 할당**
  - 프로그램의 실행 도중에 메모리를 할당 받는 것
  - 필요한 만큼만 할당을 받고 또 필요한 때에 사용하고 반납
  - 메모리를 매우 효율적으로 사용가능
  - 임베디드 시스템에서 메모리 매핑시 사용

- **정적(Static)메모리**
  - 컴파일 단계에서 메모리 할당 받는 것

동적 메모리 할당 : malloc()

- 전형적인 동적 메모리 할당 코드

```
main()
{
    int *pi;
    pi = (int *)malloc(sizeof(int));          // 동적 메모리 할당
    …
    …                                         // 동적 메모리  사용
    …
    free(pi);                                 // 동적 메모리 반납
}
```

동적 메모리 할당 예제

```
// MALLOC.C: malloc을 이용하여 정수 10를 저장할 수 있는 동적 메모리를
// 할당하고 free를 이용하여 메모리를 반납한다.
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#define SIZE 10
int main(void)
{
        int *p;
        p = (int *)malloc(SIZE * sizeof(int));
        if (p == NULL) {
                fprintf(stderr, "메모리가 부족해서 할당할 수 없습니다.\n");
                exit(1);
        }
        for (int i = 0; i<SIZE; i++)
                p[i] = i;
        for (int i = 0; i<SIZE; i++)
                printf("%d ", p[i]);
        free(p);
        return 0;
}
```

포인터 연산

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

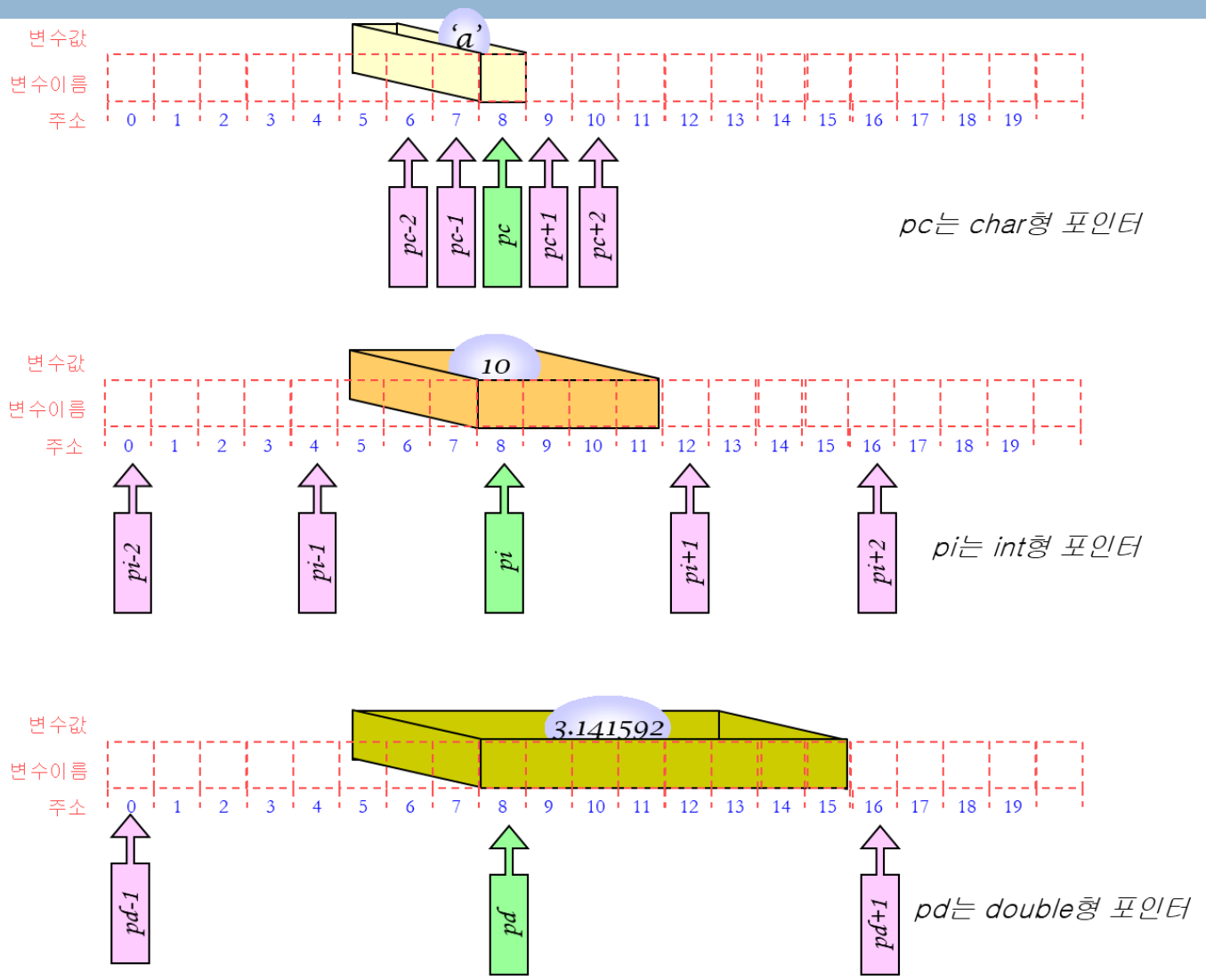
포인터 타입	++연산후 증가되는값
char	1
short	2
int	4
float	4
double	8

증가 연산 예제

```
#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


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

간접 참조 연산자와 증감 연산자

- *p++;
  - p가 가리키는 위치에서 값을 가져온 후에 p를 증가한다.
- (*p)++;
  - p가 가리키는 위치의 값을 증가한다.

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

```
#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);


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

    return 0;
}
```

pi가 가리키는 위치의 값을 증가한다.

pi가 가리키는 위치에서 값을 가져온 후에 pi를 증가한다.

```
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;
```

예제

```c
#include <stdio.h>

int main(void)
{
    int data = 0x0A0B0C0D;
    char* pc;
    pc = (char*)&data;

    for (int i = 0; i < 4; i++) {
        printf("*(pc + %d) = %02X \n", i, *(pc + i));
    }
    return 0;
}
```

```
*(pc + 0) = 0D
*(pc + 1) = 0C
*(pc + 2) = 0B
*(pc + 3) = 0A
```

인수 전달 방법

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

두 값을 교환하는 프로그램 예제1(call by value)

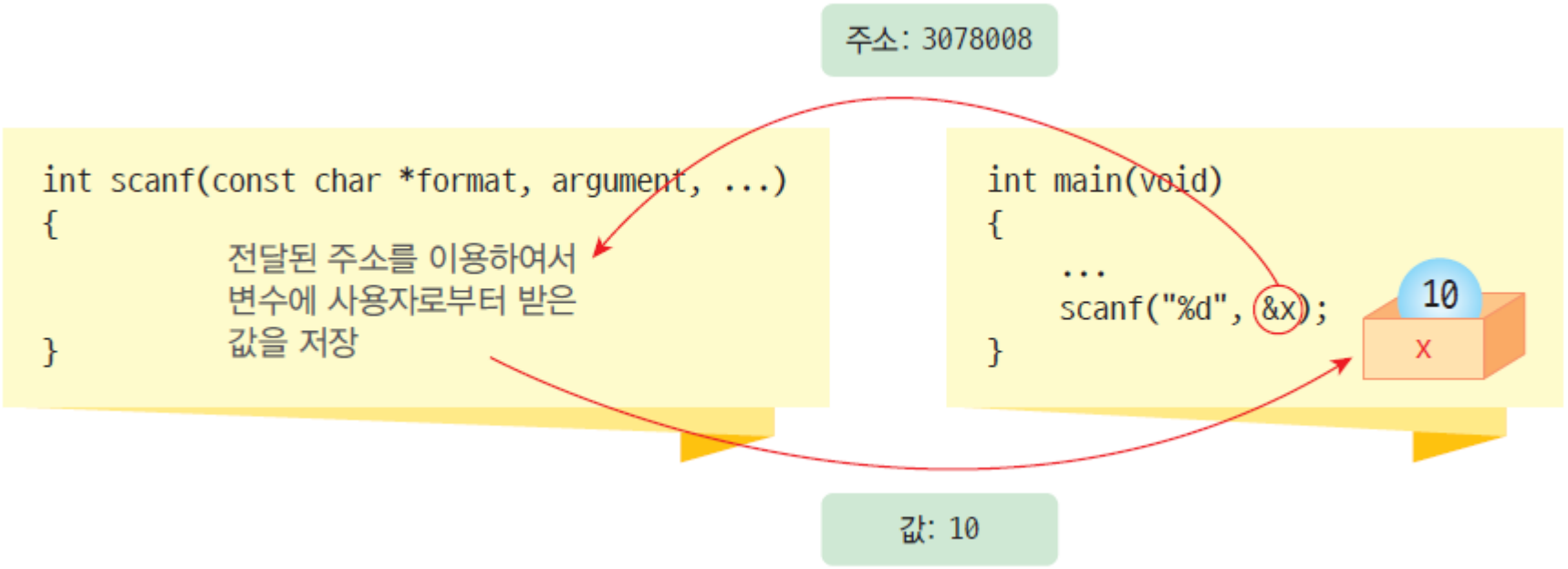
```c
#include<stdio.h>
void swap(int a, int b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;

    printf("swap 함수에서: a = %d, b = %d ₩n", a, b);
    return 0;
}
int main(void)
{
    int a = 1, b = 2;
    printf("swap 호출전: a = %d, b = %d ₩n", a, b);
    swap(a, b);
    printf("swap 호출후: a = %d, b = %d ₩n", a, b);
    return 0;
}
```

두 값을 교환하는 프로그램 예제1(call by reference)

```c
#include<stdio.h>
void swap(int *a, int *b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;

    printf("swap 함수에서: a = %d, b = %d ₩n", *a, *b);
    return 0;
}
int main(void)
{
    int a = 1, b = 2;
    printf("swap 호출전: a = %d, b = %d ₩n", a, b);
    swap(&a, &b);
    printf("swap 호출후: a = %d, b = %d ₩n", a, b);
    return 0;
}
```

scanf() 함수

- 변수에 값을 저장하기 위하여 변수의 주소를 받는다.



참고:함수가 포인터를 통하여 값을 변경할 수 없게 하려면?

- 함수의 매개 변수를 선언할 때 앞에 const를 붙이면 된다. const를 앞에 붙이면 포인터가 가리키는 내용이 변경 불가능한 상수라는 뜻이 된다.

```
void sub(const int *p)
{
        *p = 0;     // 오류!!
}
```