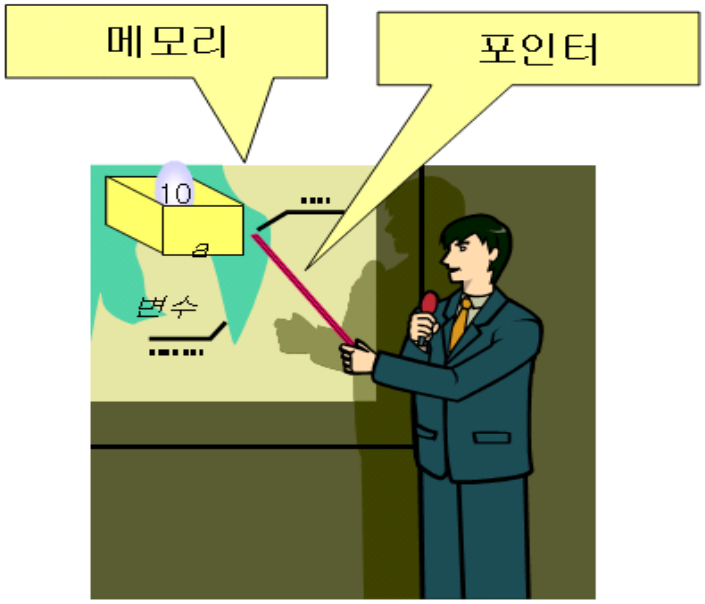
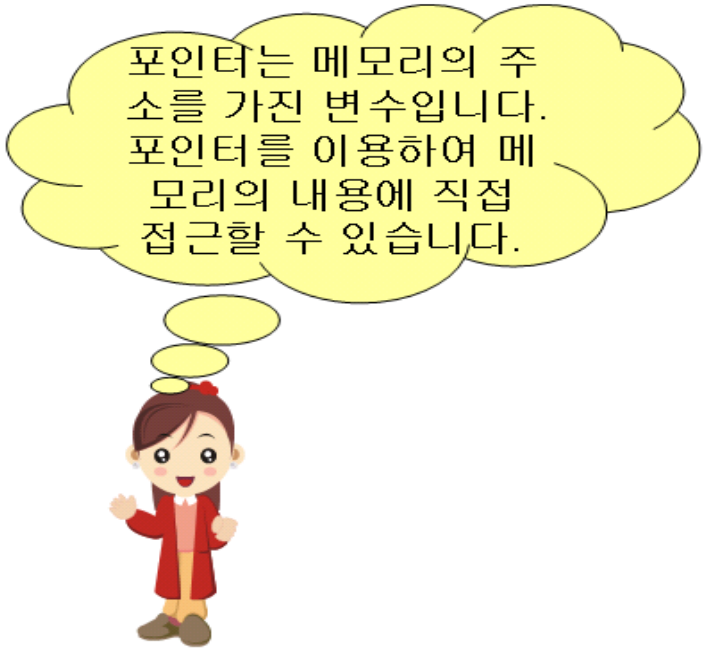
포인터 Part 1

포인터란?

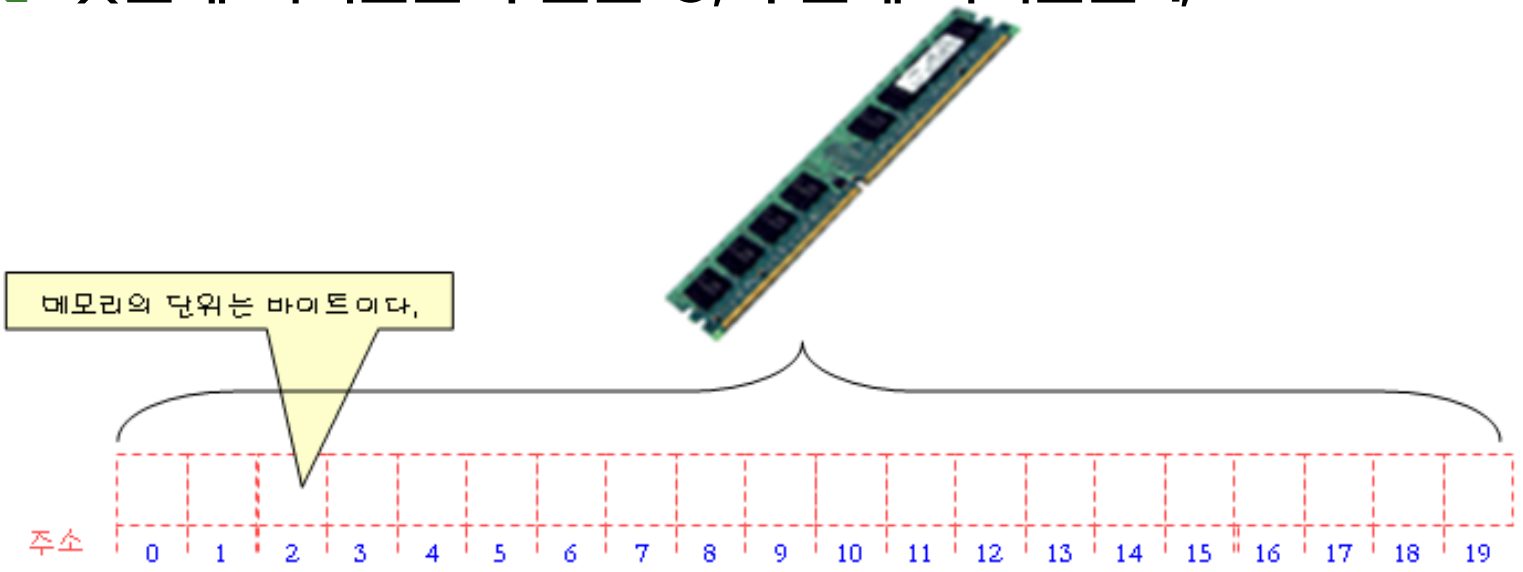
- *포인터(pointer):* 주소(address)를 값으로 가지는 변수





메모리의 구조

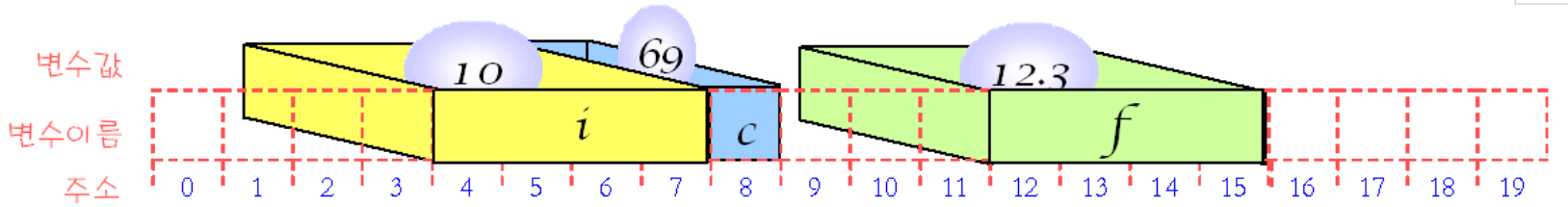
- 변수는 보통 메인 메모리에 저장된다.
- 메모리는 바이트(byte) 단위로 액세스되며, 각 바이트 당 하나의 주소가 할당된다.
  - 첫번째 바이트의 주소는 0, 두번째 바이트는 1,…



변수와 메모리

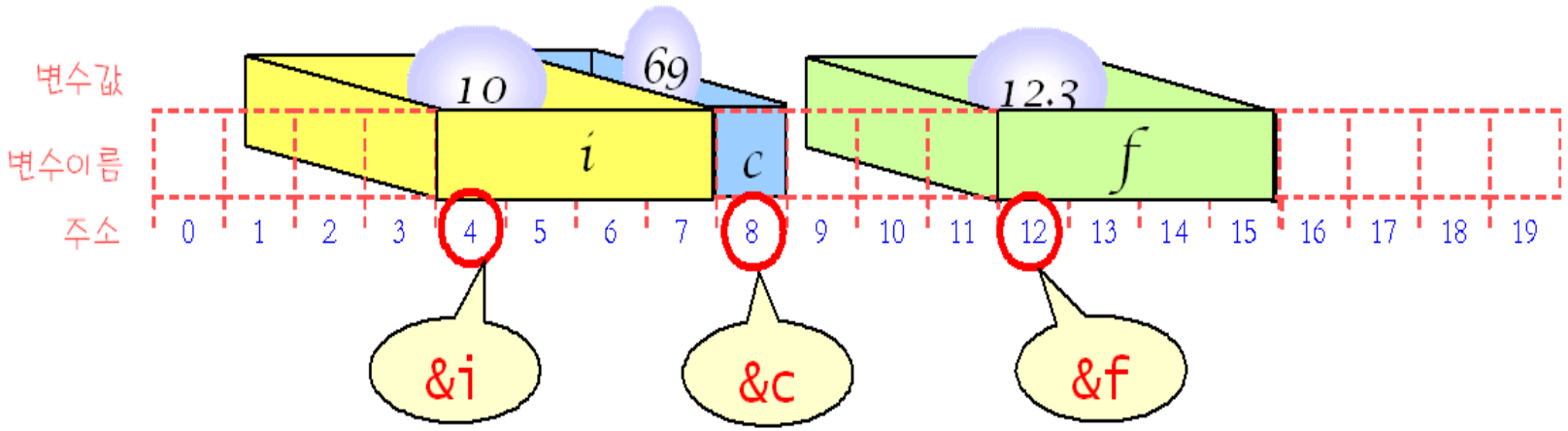
- 변수의 유형에 따라서 차지하는 메모리 공간 크기가 달라진다.
- char형 변수: 1 바이트, int형 변수: 4 바이트, …

```
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

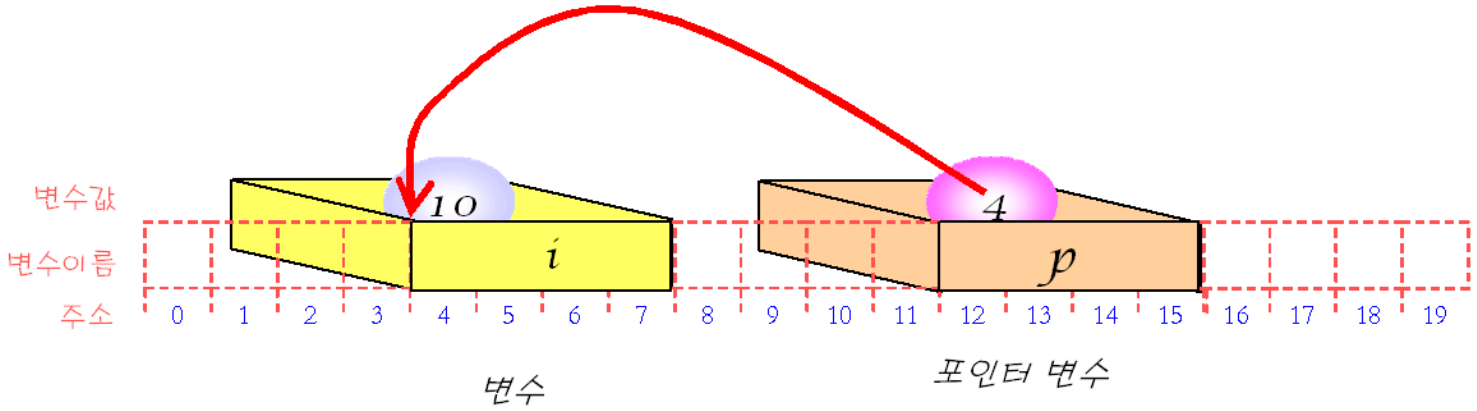
```
int main(void)
{
	int i = 10;
	char c = 69;
	float f = 12.3;

	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

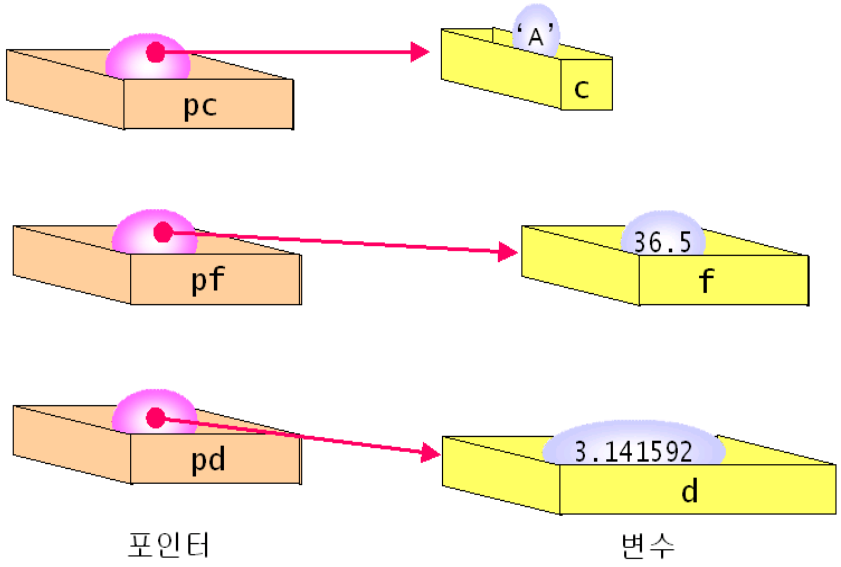
- 포인터 변수: 주소를 값으로 가지는 변수로 '*' 를 사용하여 선언한다
- int *p ; p는 포인터 변수로, 주소를 값으로 가져야하며, p가 가리키는 위치에 저장된 데이터의 유형은 정수형이어야 한다. 그러면, double *f ; 는 어떤 의미?

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



다양한 포인터의 선언

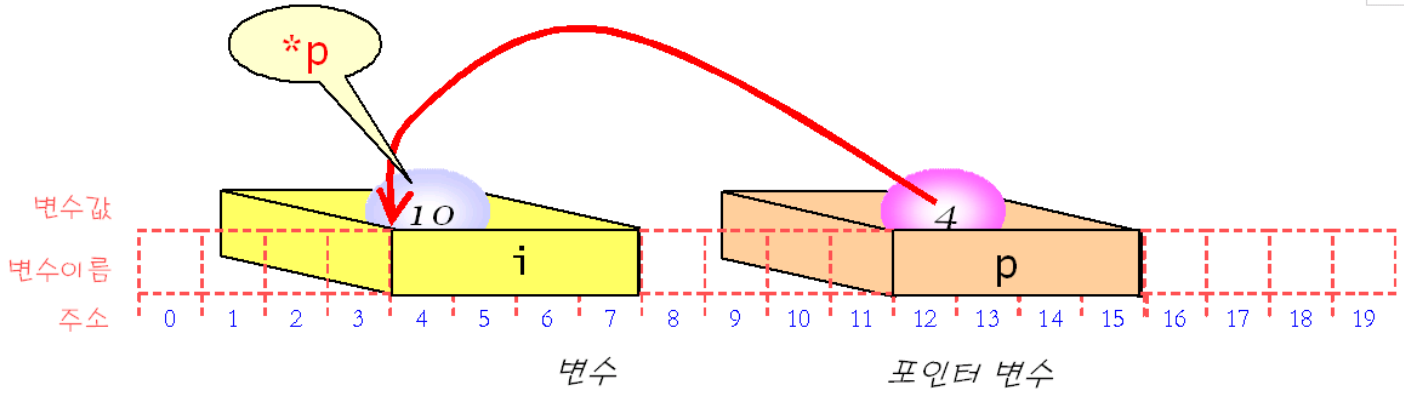
```
char c = 'A';                   // 문자형 변수 c
float f = 36.5;                 // 실수형 변수 f
double d = 3.141592;            // 실수형 변수 d

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

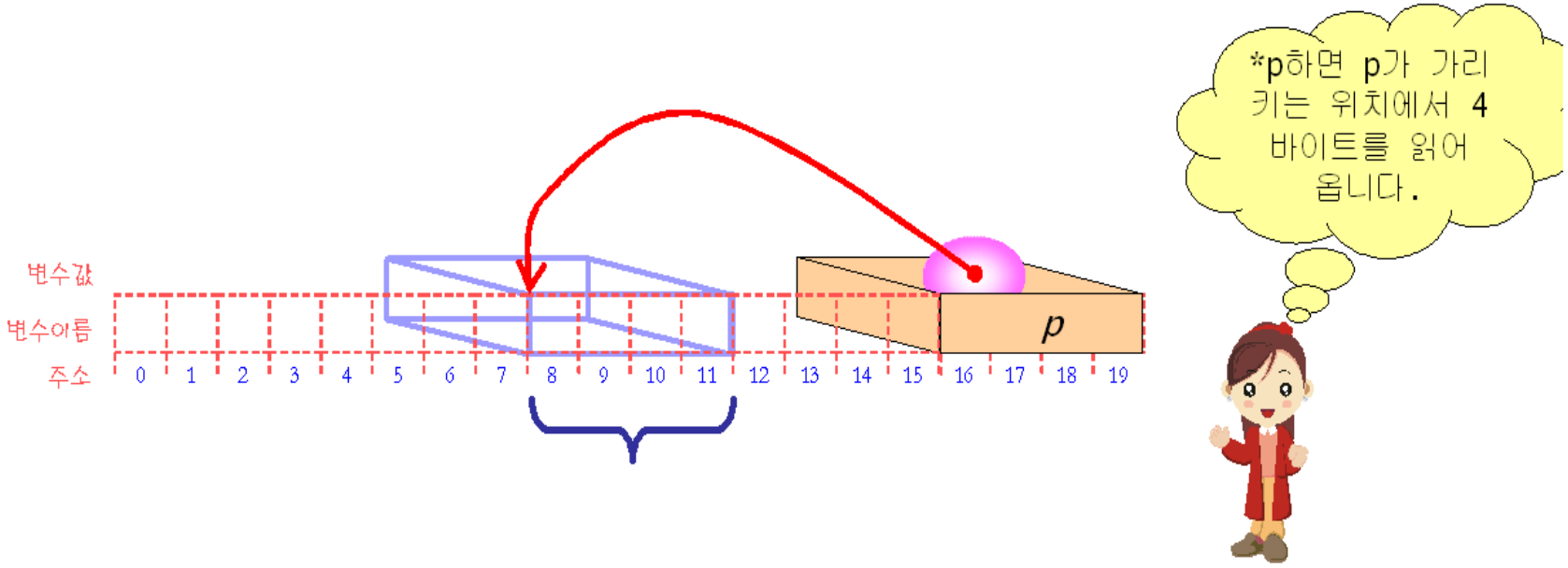
```
int i = 10;
int *p = &i;

printf("%d\n", *p);     // 10이 출력된다.

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



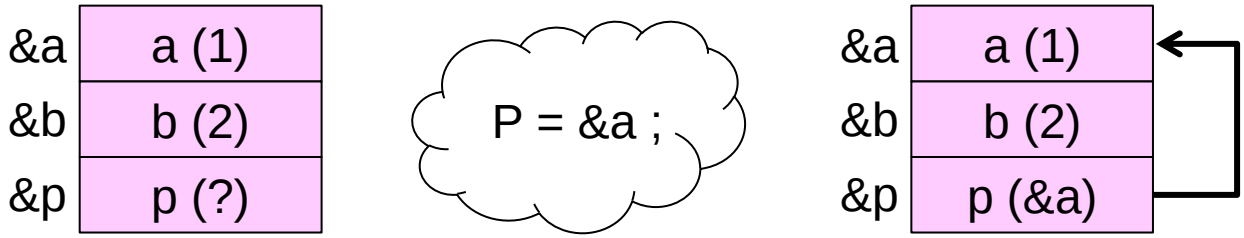
간접 참조 연산자의 해석

- 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

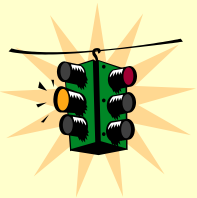
포인터 예제 #1

```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;
}
```

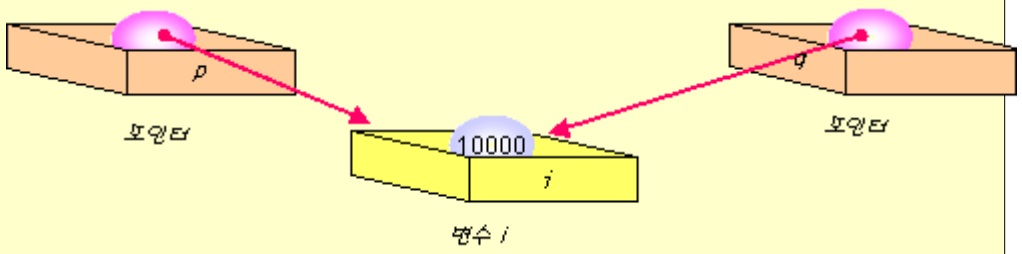


*pc++라고 하면 안됨

```
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

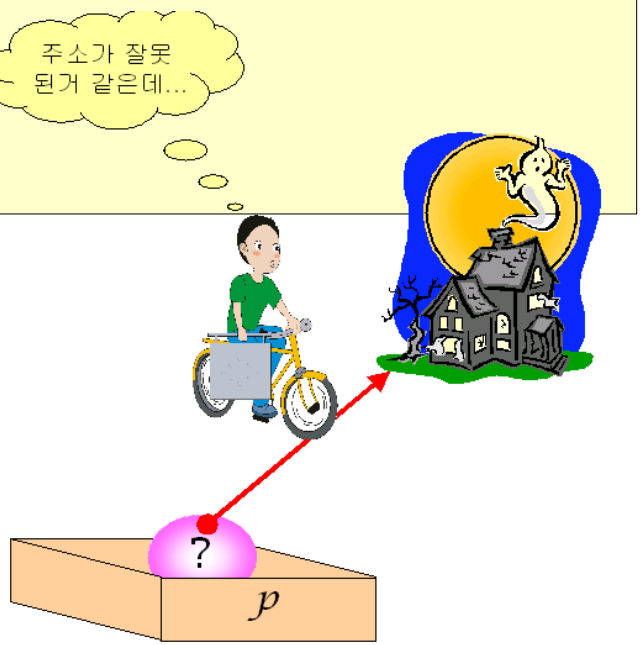
- 포인터가 가리키는 곳에 저장될 변수의 타입과 실제 저장하는 변수의 타입은 서로 일치하여야 한다.

```
#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;
}
```



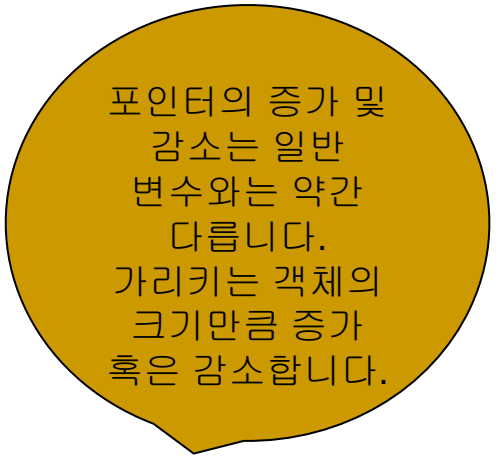
포인터 사용시 주의점 #3

- 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

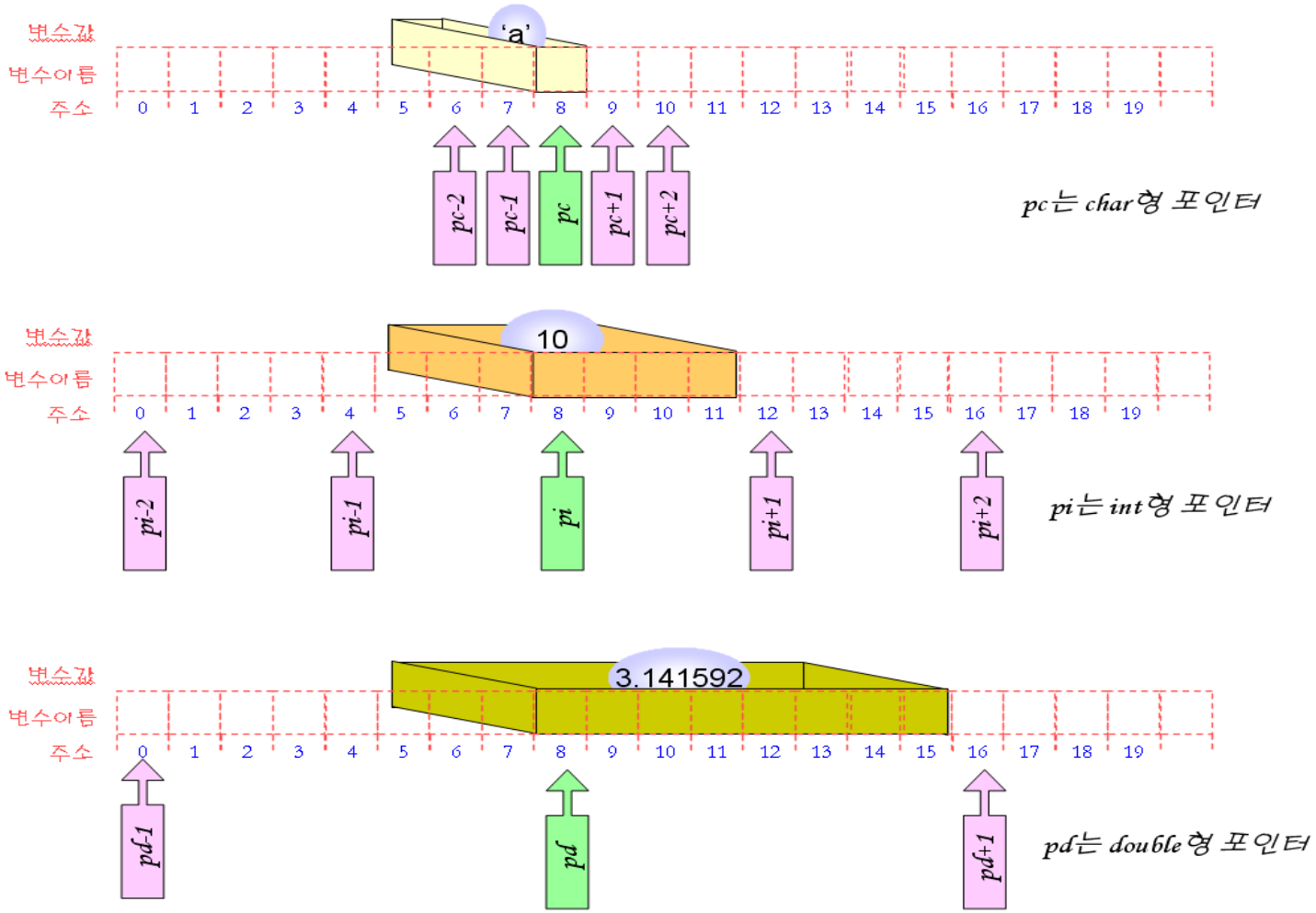


증가 연산 예제

```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) ; // print 1
  - printf(“%d\n”, (int)q – (int)p) ; // print 4

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.

```c
// 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
```