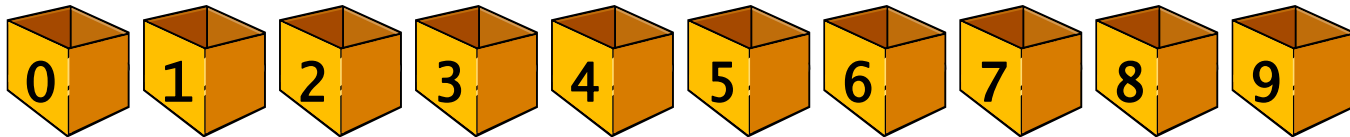


3장: 배열, 구조체, 포인터

- 목 차
 - 1. 배열
 - 2. 구조체
 - 3. 포인터

1. 배열이란?

- 같은 형의 변수를 여러 개 만드는 경우에 사용
- `int A0, A1, A2, A3, ..., A9;` → `int A[10];`



- 반복 코드 등에서 배열을 사용하면 효율적인 프로그래밍이 가능
- 예) 최대값을 구하는 프로그램: 만약 배열이 없었다면?

```
tmp=score[0];  
for(i=1;i<n;i++){  
    if( score[i] > tmp )  
        tmp = score[i];  
}
```

배열 ADT

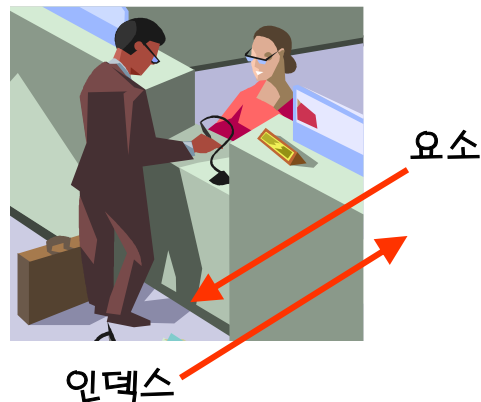
- 배열: <인덱스, 요소> 쌍의 집합
- 인덱스가 주어지면 해당되는 요소가 대응되는 구조

배열 ADT

객체: <인덱스, 요소> 쌍의 집합

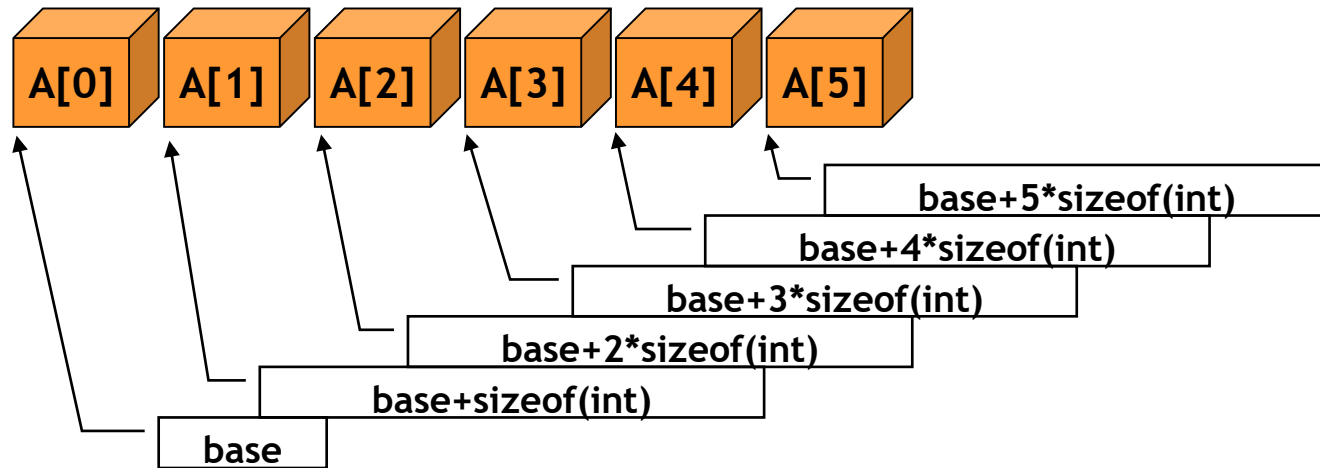
연산:

- $\text{create}(n) ::= n$ 개의 요소를 가진 배열의 생성.
- $\text{retrieve}(A, i) ::=$ 배열 A 의 i 번째 요소 반환.
- $\text{store}(A, i, \text{item}) ::=$ 배열 A 의 i 번째 위치에 item 저장.



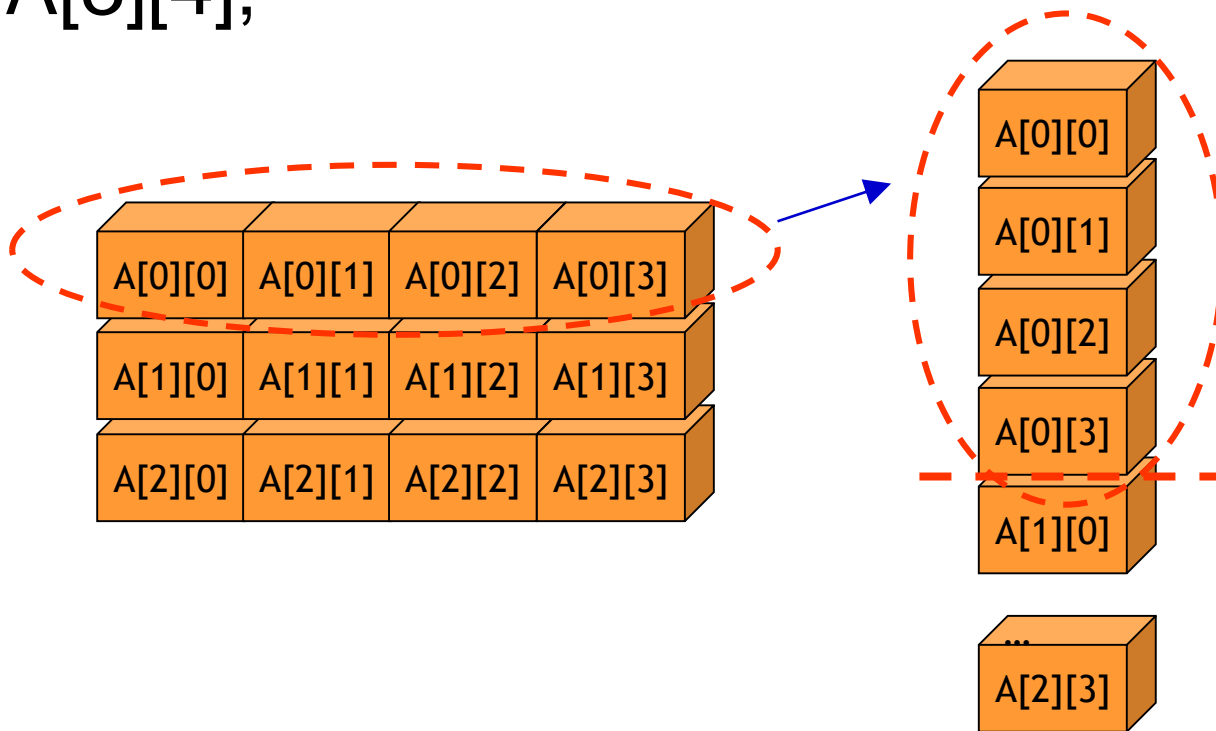
1차원 배열

- int A[6];



2차원 배열

- int A[3][4];



실제 메모리에서의 위치

배열의 응용: 다항식

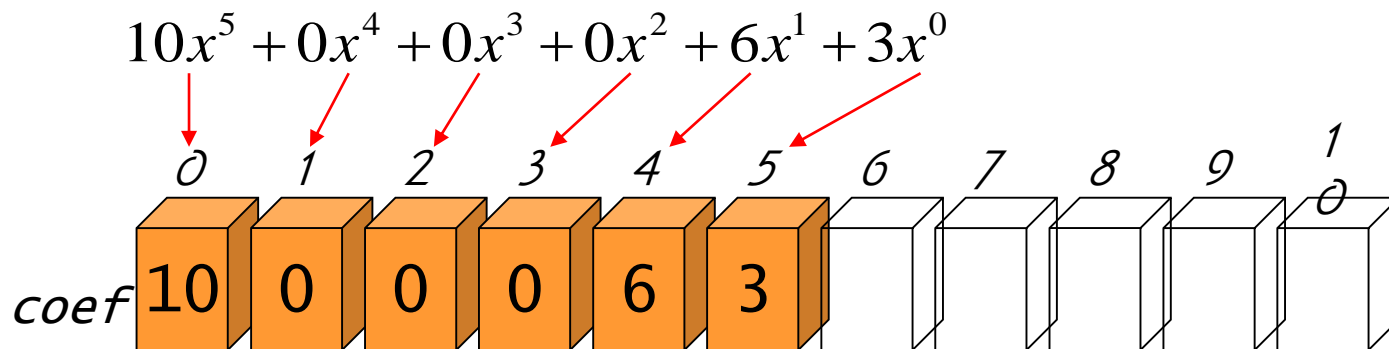
- 다항식의 일반적인 형태

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

- 프로그램에서 다항식을 처리하려면 다항식을 위한 자료구조가 필요-> 어떤 자료구조를 사용해야 다항식의 덧셈, 뺄셈, 곱셈, 나눗셈 연산을 할 때 편리하고 효율적일까?
- 배열을 사용한 2가지 방법
 - 다항식의 모든 항을 배열에 저장
 - 다항식의 0이 아닌 항만을 배열에 저장

다항식 표현 방법 #1

- 모든 차수에 대한 계수값을 배열로 저장
- 하나의 다항식을 하나의 배열로 표현



```
typedef struct {  
    int degree;  
    float coef[MAX_DEGREE];  
} polynomial;  
polynomial a = { 5, {10, 0, 0, 0, 6, 3} };
```

다항식 표현 방법 #1(계속)

- 장점: 다항식의 각종 연산이 간단해짐
- 단점: 대부분의 항의 계수가 0이면 공간의 낭비가 심함.
- 예) 다항식의 덧셈 연산

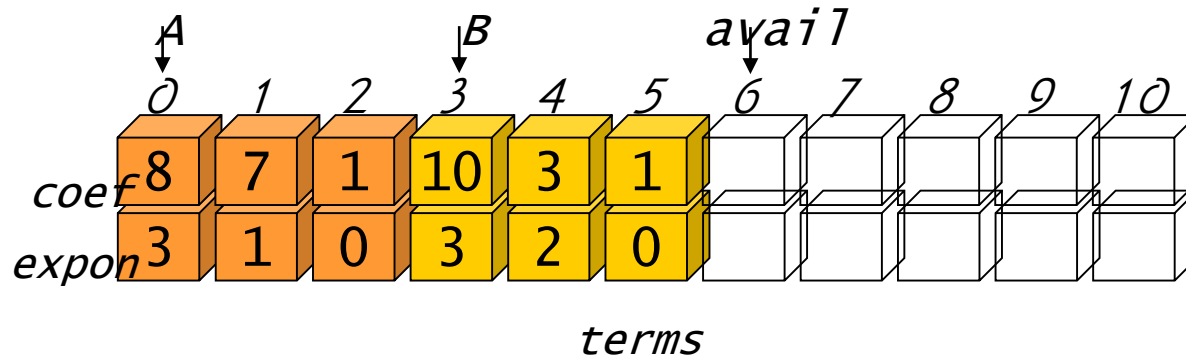
```
while( Apos<=A.degree && Bpos<=B.degree ){  
    if( degree_a > degree_b ){ // A항 > B항  
        C.coef[Cpos++]= A.coef[Apos++];  
        degree_a--;  
    }  
    else if( degree_a == degree_b ){ // A항 == B항  
        C.coef[Cpos++]=A.coef[Apos++]+B.coef[Bpos++];  
        degree_a--; degree_b--;  
    }  
    else { // B항 > A항  
        C.coef[Cpos++]= B.coef[Bpos++];  
        degree_b--;  
    }  
}
```


다항식 표현 방법 #2

- 다항식에서 0이 아닌 항만을 배열에 저장
- (계수, 차수) 형식으로 배열에 저장
 - (예) $10x^5+6x+3 \rightarrow ((10,5), (6,1), (3,0))$

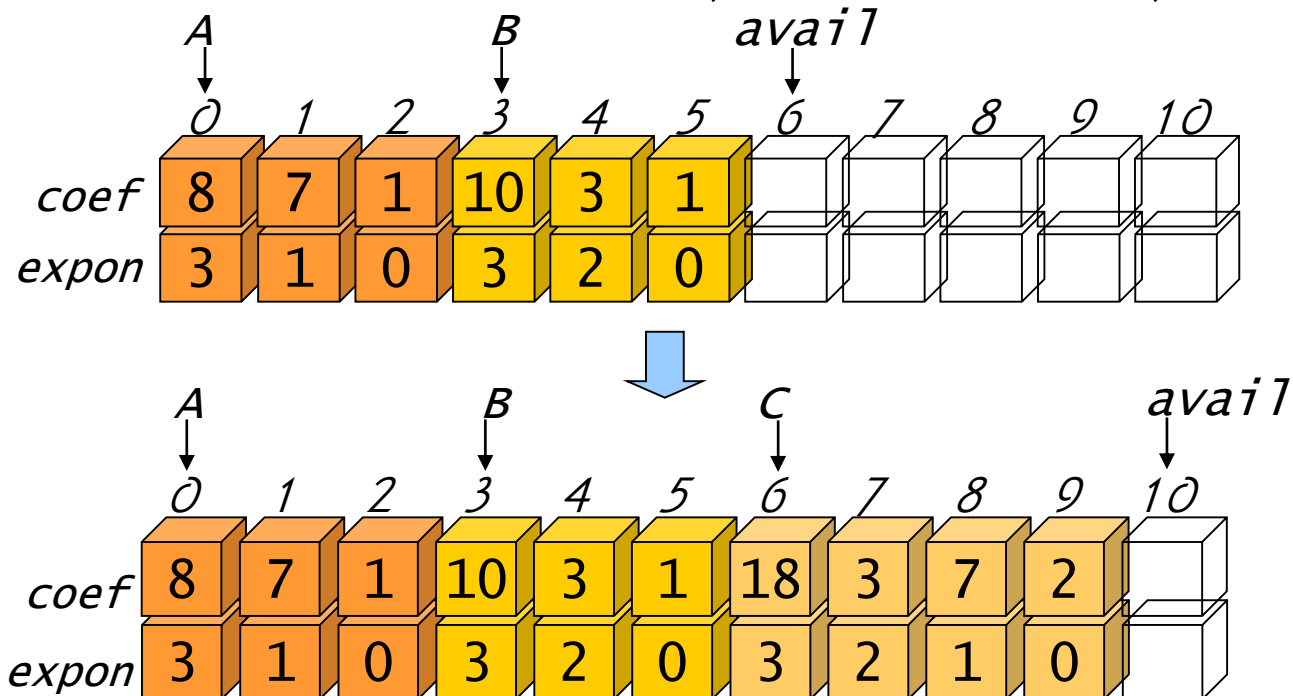
```
struct {
    float coef;
    int expon;
} terms[MAX_TERMS]={ {10,5}, {6,1}, {3,0} };
```

- 하나의 배열로 여러 개의 다항식을 나타낼 수 있음.



다항식 표현 방법 #2(계속)

- 장점: 메모리 공간의 효율적인 이용
- 단점: 다항식의 연산들이 복잡해진다(프로그램 3.3 참조).
- (예) 다항식의 덧셈 $A=8x^3+7x+1$, $B=10x^3+3x^2+1$, $C=A+B$



다항식 표현 방법 #2(계속)

```
// C = A + B
void poly_add2(int As, int Ae, int Bs, int Be, int *Cs, int *Ce)
{
    float tempcoef;
    *Cs = avail;
    while( As <= Ae && Bs <= Be )
        switch(compare(terms[As].expon, terms[Bs].expon)){
            case '>': // A의 차수 > B의 차수
                attach(terms[As].coef, terms[As].expon);
                As++; break;
            case '=': // A의 차수 == B의 차수
                tempcoef = terms[As].coef + terms[Bs].coef;
                if( tempcoef )
                    attach(tempcoef, terms[As].expon);
                As++; Bs++; break;
            case '<': // A의 차수 < B의 차수
                attach(terms[Bs].coef, terms[Bs].expon);
                Bs++; break;
        }
    // A의 나머지 항들을 이동함
    for(; As <= Ae; As++)
        attach(terms[As].coef, terms[As].expon);
    // B의 나머지 항들을 이동함
    for(; Bs <= Be; Bs++)
        attach(terms[Bs].coef, terms[Bs].expon);
    *Ce = avail - 1;
}
```

희소행렬 (Sparse matrix)

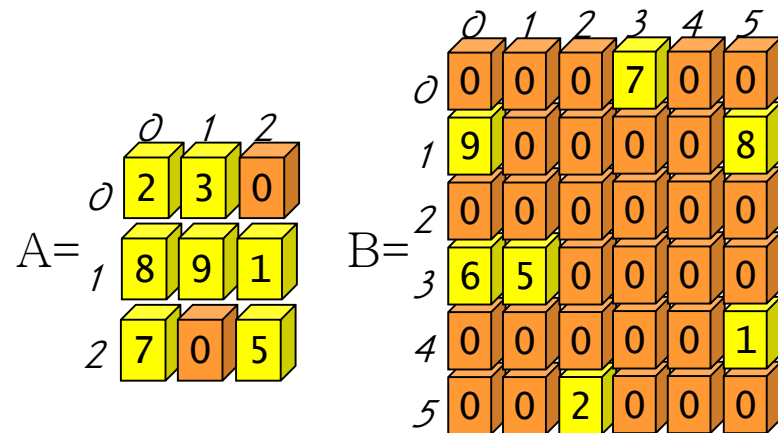
- 배열을 이용하여 행렬(matrix)를 표현하는 2가지 방법
- (1) 2차원 배열을 이용하여 배열의 전체 요소를 저장하는 방법
- (2) 0이 아닌 요소들만 저장하는 방법
- 희소행렬: 대부분의 항들이 0인 배열

$$A = \begin{bmatrix} 2 & 3 & 0 \\ 8 & 9 & 1 \\ 7 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 & 7 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

희소행렬 표현방법 #1

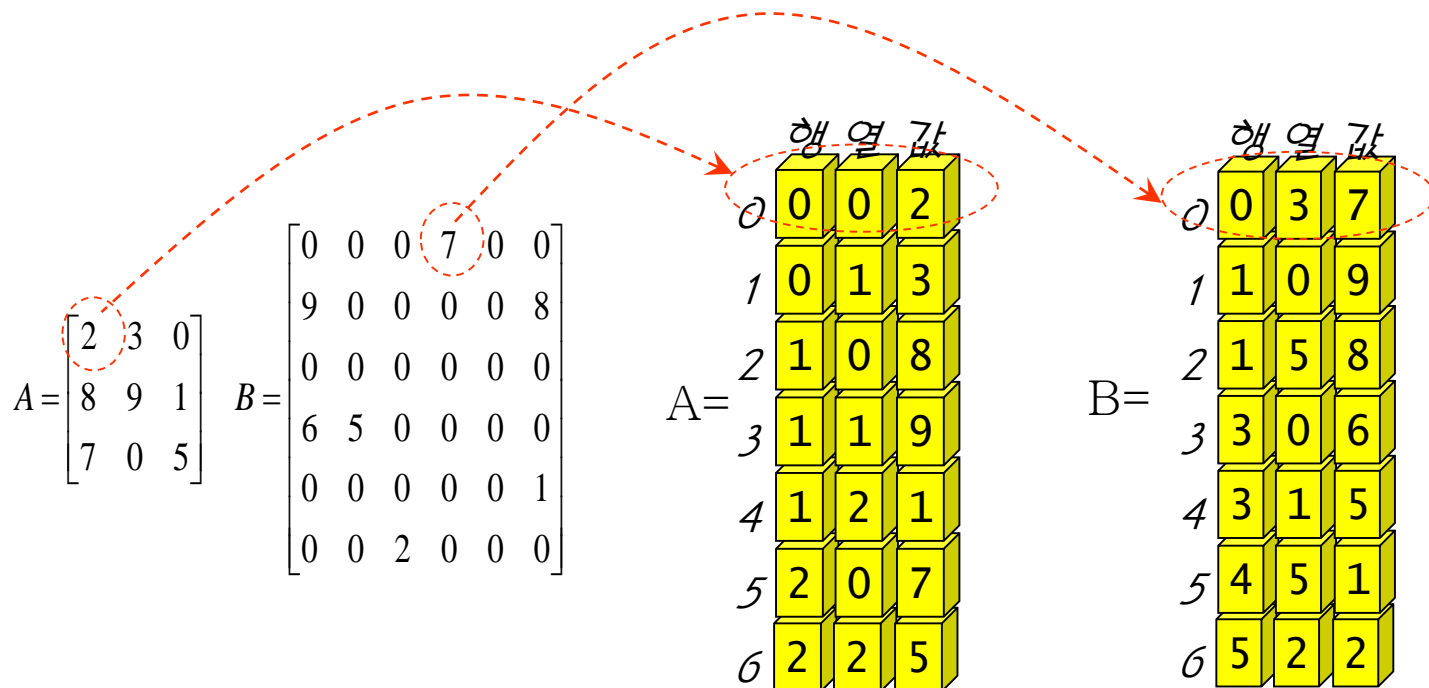
- 2차원 배열을 이용하여 배열의 전체 요소를 저장하는 방법
- 장점: 행렬의 연산들을 간단하게 구현할 수 있다.
- 단점: 대부분의 항들이 0인 희소 행렬의 경우 많은 메모리 공간 낭비

$$A = \begin{bmatrix} 2 & 3 & 0 \\ 8 & 9 & 1 \\ 7 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 & 7 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$



희소행렬 표현방법 #2

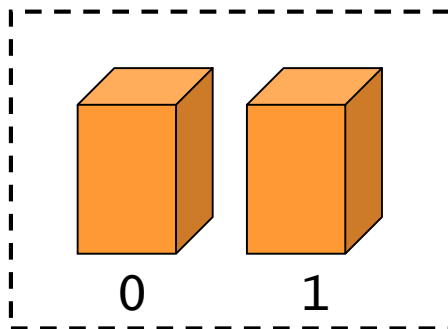
- 0이 아닌 요소들만 저장하는 방법
- 장점: 희소 행렬의 경우, 메모리 공간의 절약
- 단점: 각종 행렬 연산들의 구현이 복잡해진다.



2. 구조체란?

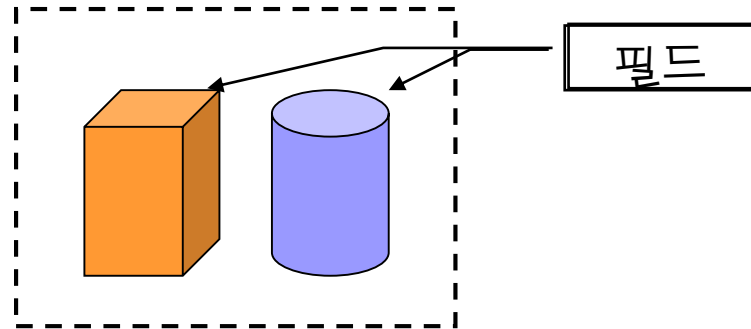
- 구조체(structure): 타입이 다른 데이터를 하나로 묶는 방법
- 배열(array): 타입이 같은 데이터들을 하나로 묶는 방법

배열



```
char carray[100];
```

구조체



```
struct example {
    char cfield;
    int ifield;
    float ffield;
    double dfield;
};
struct example s1;
```

구조체의 사용예

- 구조체의 선언과 구조체 변수의 생성

```
struct person {  
    char name[10];    // 문자배열로 된 이름  
    int age;           // 나이를 나타내는 정수값  
    float height;     // 키를 나타내는 실수값  
};  
struct person a;      // 구조체 변수 선언
```

- typedef을 이용한 구조체의 선언과 구조체 변수의 생성

```
typedef struct person {  
    char name[10];    // 문자배열로 된 이름  
    int age;           // 나이를 나타내는 정수값  
    float height;     // 키를 나타내는 실수값  
} person;  
person a;             // 구조체 변수 선언
```


구조체의 대입과 비교 연산

- 구조체 변수의 대입: 가능

```
typedef struct person {  
    char name[10];    // 문자배열로 된 이름  
    int age;           // 나이를 나타내는 정수값  
    float height;      // 키를 나타내는 실수값  
} person;  
main()  
{  
    person a, b;  
    b = a;             // 가능  
}
```

- 구조체 변수끼리의 비교: 불가능

```
main()  
{  
    if ( a > b )  
        printf("a가 b보다 나이가 많음"); // 불가능  
}
```

자체참조 구조체

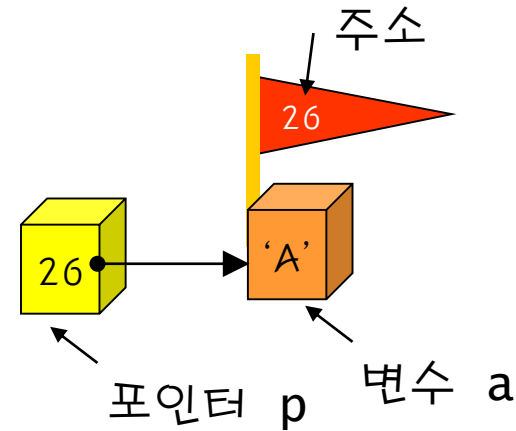
- 자체 참조 구조체(self-referential structure): 필드중에 자기 자신을 가리키는 포인터가 한 개 이상 존재하는 구조체
- 연결 리스트나 트리에 많이 등장

```
typedef struct ListNode {  
    char        data[10];  
    struct ListNode *link;  
} ListNode;
```

3. 포인터(Pointer)란?

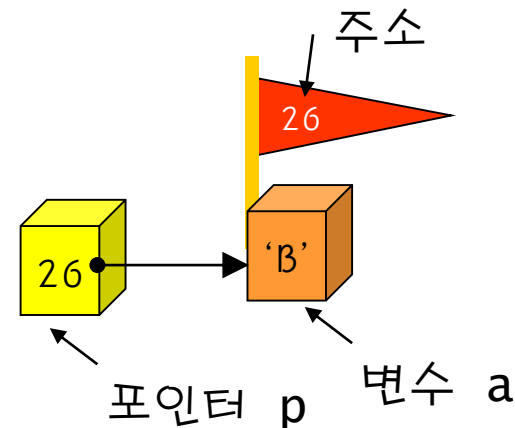
- 포인터: 다른 변수의 주소를 가지고 있는 변수

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



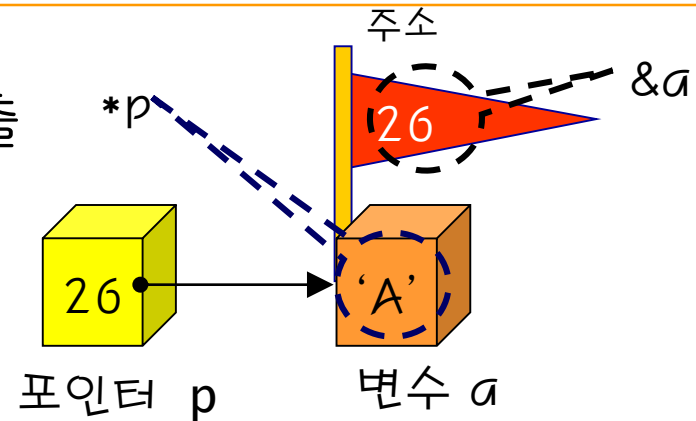
- 포인터가 가리키는 내용의 변경: * 연산자 사용

```
*p= 'B';
```



포인터와 관련된 연산자

- & 연산자: 변수의 주소를 추출
- * 연산자: 포인터가 가리키는 곳의 내용을 추출



```
p          // 포인터
*p         // 포인터가 가리키는 값
*p++      // 포인터가 가리키는 값을 가져온 다음, 포인터를 한칸 증가한다.
*p--      // 포인터가 가리키는 값을 가져온 다음, 포인터를 한칸 감소한다.
(*p)++    // 포인터가 가리키는 값을 증가시킨다.
```

```
int a;      // 정수 변수 선언
int *p;     // 정수 포인터 선언
int **pp;   // 정수 포인터의 포인터 선언
p = &a;     // 변수 a와 포인터 p를 연결
pp = &p;    // 포인터 p와 포인터의 포인터 pp를 연결
```

다양한 포인터

- 포인터의 종류

```
void *p; // p는 아무것도 가리키지 않는 포인터
int *pi; // pi는 정수 변수를 가리키는 포인터
float *pf; // pf는 실수 변수를 가리키는 포인터
char *pc; // pc는 문자 변수를 가리키는 포인터
int **pp; // pp는 포인터를 가리키는 포인터
struct test *ps; // ps는 test 타입의 구조체를 가리키는 포인터
void (*f)(int); // f는 함수를 가리키는 포인터
```

- 포인터의 형변환: 필요할 때마다 형변환하는 것이 가능하다.

```
void *p;
pi=(int *) p;
```

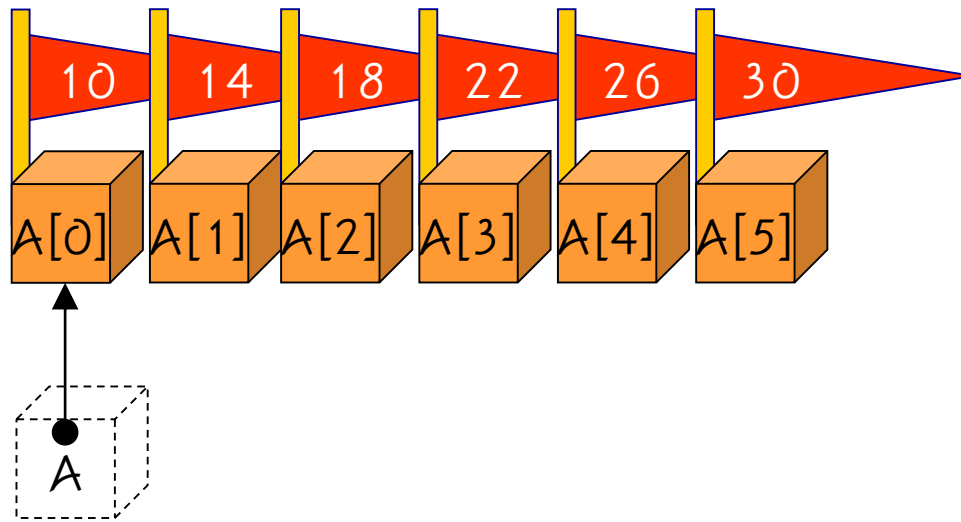
함수의 파라미터로서의 포인터

- 함수안에서 파라미터로 전달된 포인터를 이용하여 외부 변수의 값 변경 가능

```
void swap(int *px, int *py)
{
    int tmp;
    tmp = *px;
    *px = *py;
    *py = tmp;
}
main()
{
    int a=1,b=2;
    printf("swap을 호출하기 전: a=%d, b=%d\n", a,b);
    swap(&a, &b);
    printf("swap을 호출한 다음: a=%d, b=%d\n", a,b);
}
```

배열과 포인터

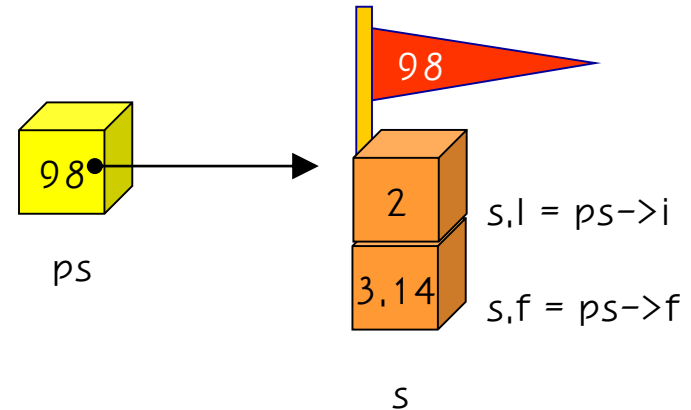
- 배열의 이름: 사실상의 포인터와 같은 역할



- 컴파일러가 배열의 이름을 배열의 첫번째 주소로 대치

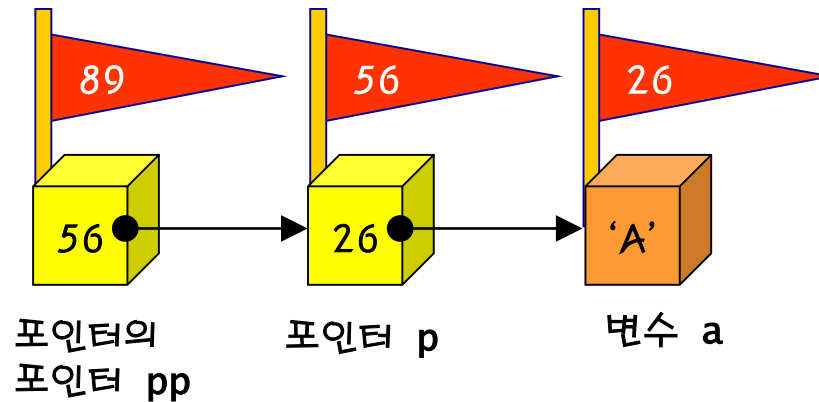
구조체의 포인터

- 구조체의 요소에 접근하는 연산자: ->



```
main()
{
    struct {
        int i;
        float f;
    } s, *ps;
    ps = &s;
    ps->i = 2;
    ps->f = 3.14;
}
```


포인터의 포인터

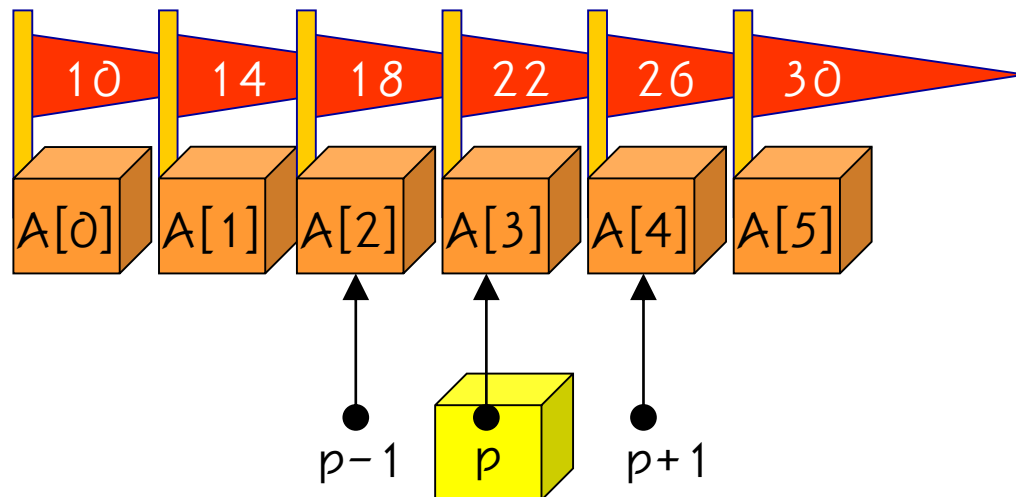


```
int a; // 정수 변수 선언
int *p; // 정수 포인터 선언
int **pp; // 정수 포인터의 포인터 선언
p = &a; // 변수 a와 포인터 p를 연결
pp = &p; // 포인터 p와 포인터의 포인터 pp를 연결
```

포인터 연산

- 포인터에 대한 사칙연산: 포인터가 가리키는 객체단위로 계산된다.

p	// 포인터
p+1	// 포인터 p가 가리키는 객체의 바로 뒤 객체
p-1	// 포인터 p가 가리키는 객체의 바로 앞 객체



포인터 사용시 주의할 점

- 포인터가 아무것도 가리키고 있지 않을 때는 NULL로 설정
 - `int *pi=NULL;`
- 초기화가 안된 상태에서 사용 금지

```
main()
{
    char *pc; // 포인터 pi는 초기화가 안되어 있음
    *pc = 'E'; // 위험한 코드
}
```

- 포인터 타입간의 변환시에는 명시적인 타입 변환 사용

```
int *pi;
float *pf;
pf = (float *)pi;
```

동적 메모리 할당

- 프로그램이 메모리를 할당 받는 방법
 - 정적 메모리
 - 동적 메모리 할당
- 정적 메모리 할당
 - 메모리의 크기는 프로그램이 시작하기 전에 결정
 - 프로그램의 수행 도중에 그 크기가 변경될 수는 없다.
 - 만약 처음에 결정된 크기보다 더 큰 입력이 들어온다면 처리하지 못할 것이고 더 작은 입력이 들어온다면 남은 메모리 공간은 낭비될 것이다.
 - (예) 변수나 배열의 선언
 - `int buffer[100];`
`char name[] = "data structure";`
- 동적 메모리 할당
 - 프로그램의 실행 도중에 메모리를 할당 받는 것
 - 필요한 만큼만 할당을 받고 또 필요한 때에 사용하고 반납
 - 메모리를 매우 효율적으로 사용가능



동적 메모리 할당

- 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 관련 라이브러리 함수
 - malloc(size) // 메모리 할당
 - free(ptr) // 메모리 할당 해제
 - sizeof(var) // 변수나 타입의 크기 반환(바이트 단위)

동적 메모리 할당 라이브러리

- malloc(int size)
 - size 바이트 만큼의 메모리 블록을 할당

```
(char *)malloc(100) ; /* 100 바이트로 50개의 정수를 저장 */  
(int *)malloc(sizeof(int)); /* 정수 1개를 저장할 메모리 확보 */  
(struct Book *)malloc(sizeof(struct Book)); /* 하나의 구조체 생성 */
```

- free(void ptr)
 - ptr이 가리키는 할당된 메모리 블록을 해제
- sizeof 키워드
 - 변수나 타입의 크기 반환(바이트 단위)

```
size_t i = sizeof( int ); // 4  
struct AlignDepends {  
    char c;  
    int i;  
};  
size_t size = sizeof(struct AlignDepends); // 8  
int array[] = { 1, 2, 3, 4, 5 };  
size_t sizearr = sizeof( array ) / sizeof( array[0] ); // 20/4=5
```

동적 메모리 할당 예제

```
struct Example {  
    int number;  
    char name[10];  
};  
void main()  
{  
    struct Example *p;  
  
    p=(struct Example *)malloc(2*sizeof(struct Example));  
    if(p==NULL){  
        fprintf(stderr, "can't allocate memory\n") ;  
        exit(1) ;  
    }  
    p->number=1;  
    strcpy(p->name,"Park");  
    (p+1)->number=2;  
    strcpy((p+1)->name,"Kim");  
    free(p);  
}
```