

자료구조(Data Structures)

3장. 배열, 구조체, 포인터

담당 교수 : 조 미경

이번 장에서 학습할 내용



- 1) 배열이란?
- 2) 1차원 배열과 2차원 배열
- 3) 배열의 응용: 다항식, 희소행렬
- 4) 구조체란?
- 5) 구조체 사용 방법
- 6) 포인터란?
- 7) 포인터 연산
- 8) 구조체와 포인터
- 9) 배열과 포인터
- 10) 동적 메모리 할당 및 반납에 대한 개념

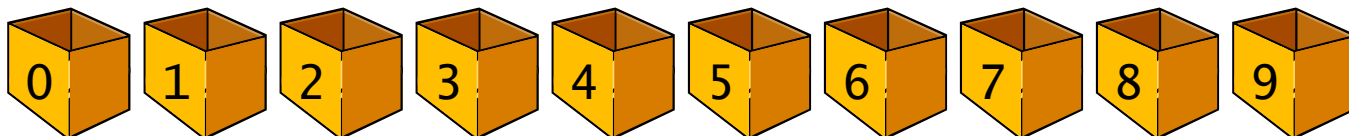
배열이란?

- 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

배열이란?

- ▶ 예) 최대값을 구하는 프로그램: 배열 사용

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

- ▶ 예) 최대값을 구하는 프로그램: 만약 배열이 없었다면?

배열 ADT

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

배열 ADT

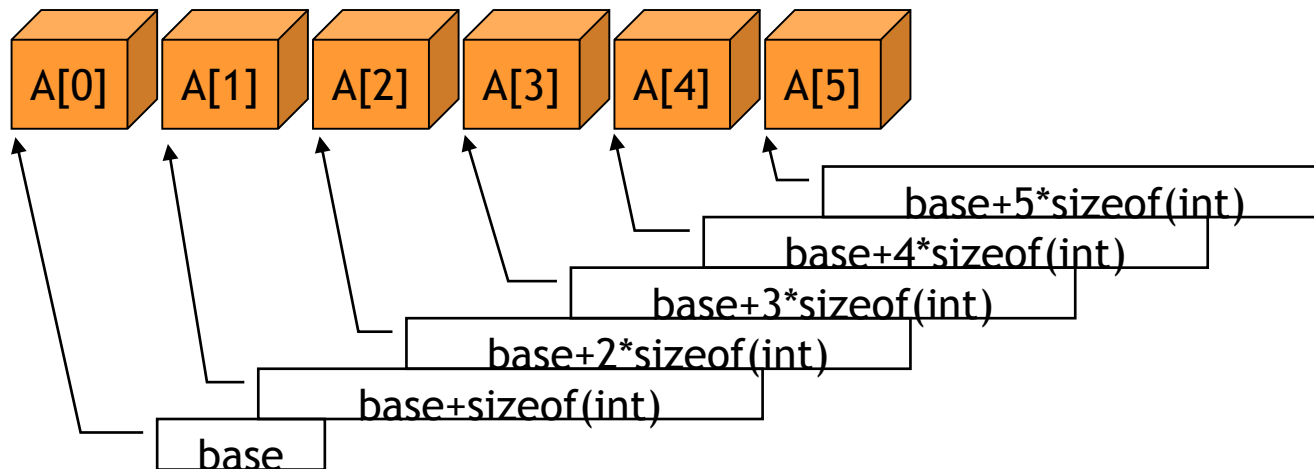
객체: <인덱스, 요소> 쌍의 집합

연산:

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

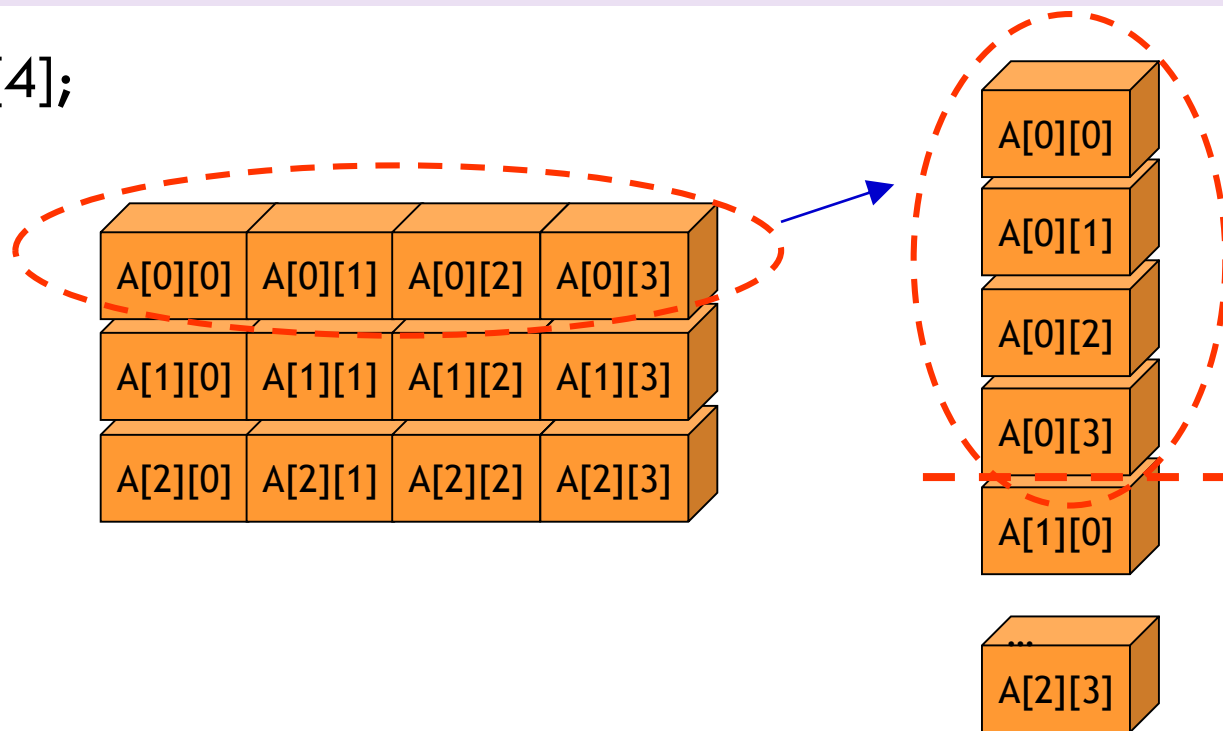
1차원 배열

- `int A[6];`



2차원 배열

● `int A[3][4];`



실제 메모리 안에서의 위치

1차원과 2차원 배열 사용

- 1차원 배열이 적당한 데이터
 - ▶ 40명 학생의 신장
 - ▶ 40명 학생의 체중
 - ▶ 40명 학생의 수강 과목 수
 - ▶ 40명 학생의 평점
 - ▶ A지점의 1년(12개월) 물건 판매량
- 2차원 배열이 적당한 데이터
 - ▶ 40명 학생의 신장과 체중
 - ▶ 40명 학생의 수강 과목수와 평점
 - ▶ 부산지역 모든 지점(A,BC,D)의 분기별 물건 판매량

함수의 매개 변수로서의 배열

```
#include <stdio.h>

#define MAX_SIZE 10

void sub( int var, int listA[], int listB[][MAX_SIZE] )
{
    var = 10; list[0] = 10; list[2][2] = 200;
}

void main()
{
    int var, list1[MAX_SIZE], list2[MAX_SIZE][MAX_SIZE];
    var = 0; list1[0]=0;
    list[2][2] = 0;
    sub( var, list1, list2 );
    printf("var=%d, list1[0] = %d, list2[2][2] = %d\n", var, list[0], list[2][2] );
}
```

배열의 응용: 다항식

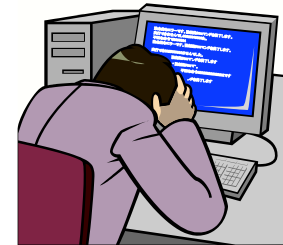
- 다항식의 일반적인 형태

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

- 프로그램에서 다항식을 처리하려면 다항식을 위한 자료구조가 필요-> 어떤 자료구조를 사용해야 다항식의 덧셈, 뺄셈, 곱셈, 나눗셈 연산을 할 때 편리하고 효율적일까?

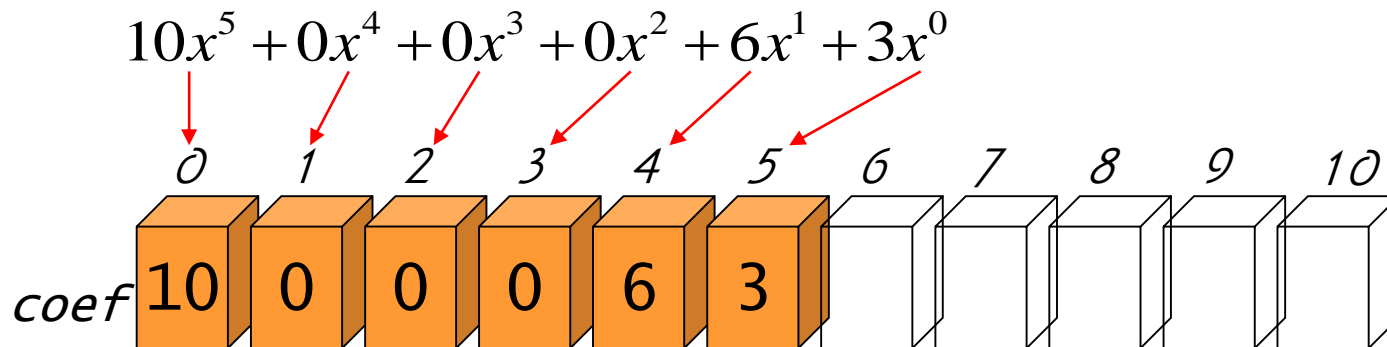
- 배열을 사용한 2가지 방법

- 1) 다항식의 모든 항을 배열에 저장
- 2) 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 $10x^{50} + 3$
- 예) 다항식의 덧셈 연산

```
#include <stdio.h>

#define MAX(a,b) (((a)>(b))?(a):(b))

#define MAX_DEGREE 101

typedef struct {                                // 다항식 구조체 타입 선언
    int degree;                                // 다항식의 차수
    float coef[MAX_DEGREE];                   // 다항식의 계수
} polynomial;
```

다항식 표현 방법 #1(계속)

```
// 주함수
```

```
main()
```

```
{
```

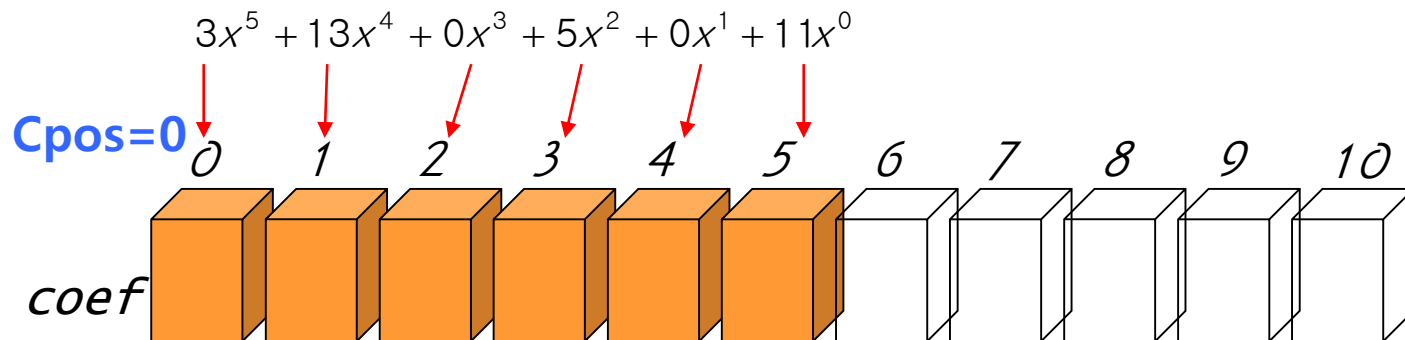
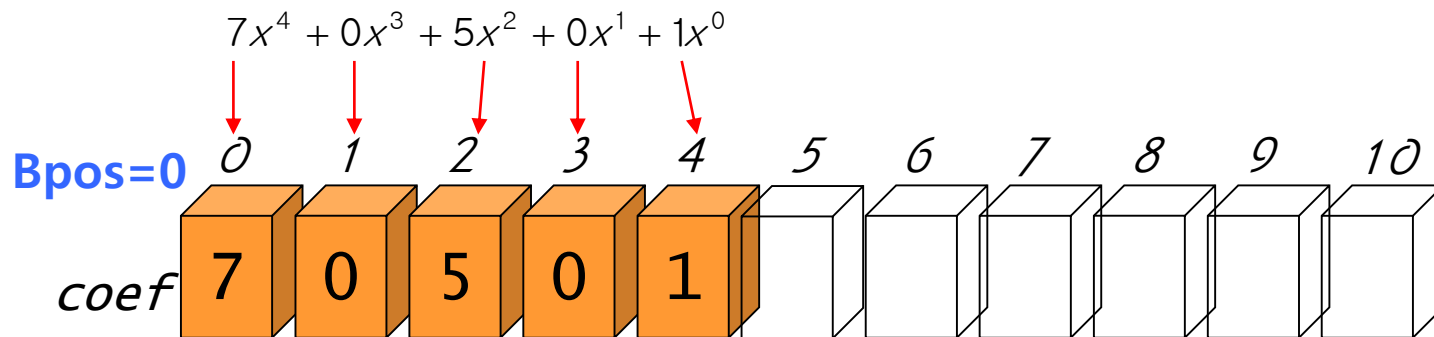
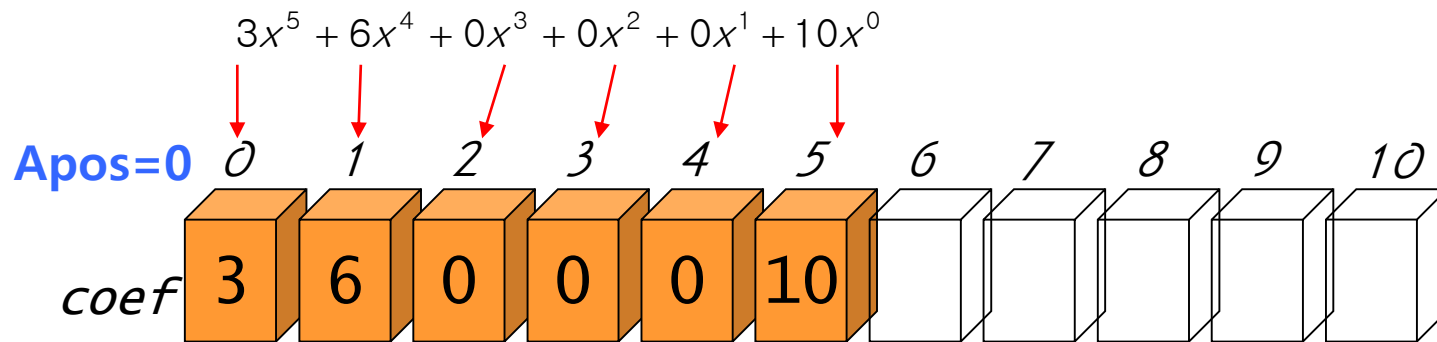
```
    polynomial a = { 5, {3, 6, 0, 0, 0, 10} };
```

```
    polynomial b = { 4, {7, 0, 5, 0, 1} };
```

```
    polynomial c;
```

```
    c = poly_add1(a,b);
```

```
}
```



다항식 표현 방법 #1(계속)

// $C = A+B$ 여기서 A와 B는 다항식이다.

```

polynomial poly_add1(polynomial A, polynomial B) {
    polynomial C;                                // 결과 다항식
    int Apos=0, Bpos=0, Cpos=0; // 배열 인덱스 변수
    int degree_a=A.degree;
    int degree_b=B.degree;
    C.degree = MAX(A.degree, B.degree); // 결과 다항식 차수
    while( Apos<=A.degree && Bpos<=B.degree ){
        if( degree_a > degree_b ){ // A항 > B항
            C.coef[Cpos++]= A.coef[Apos++];
            degree_a--;
        }
    }
}

```

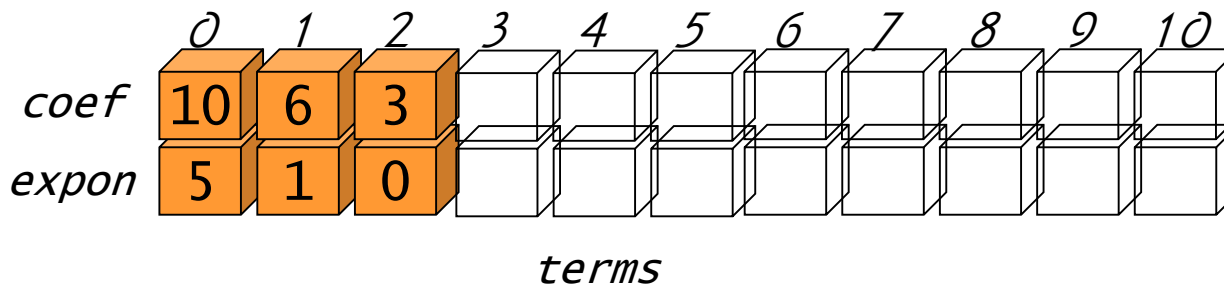
다항식 표현 방법 #1(계속)

```
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--;
}
return C;
}
```


다항식 표현 방법 #2

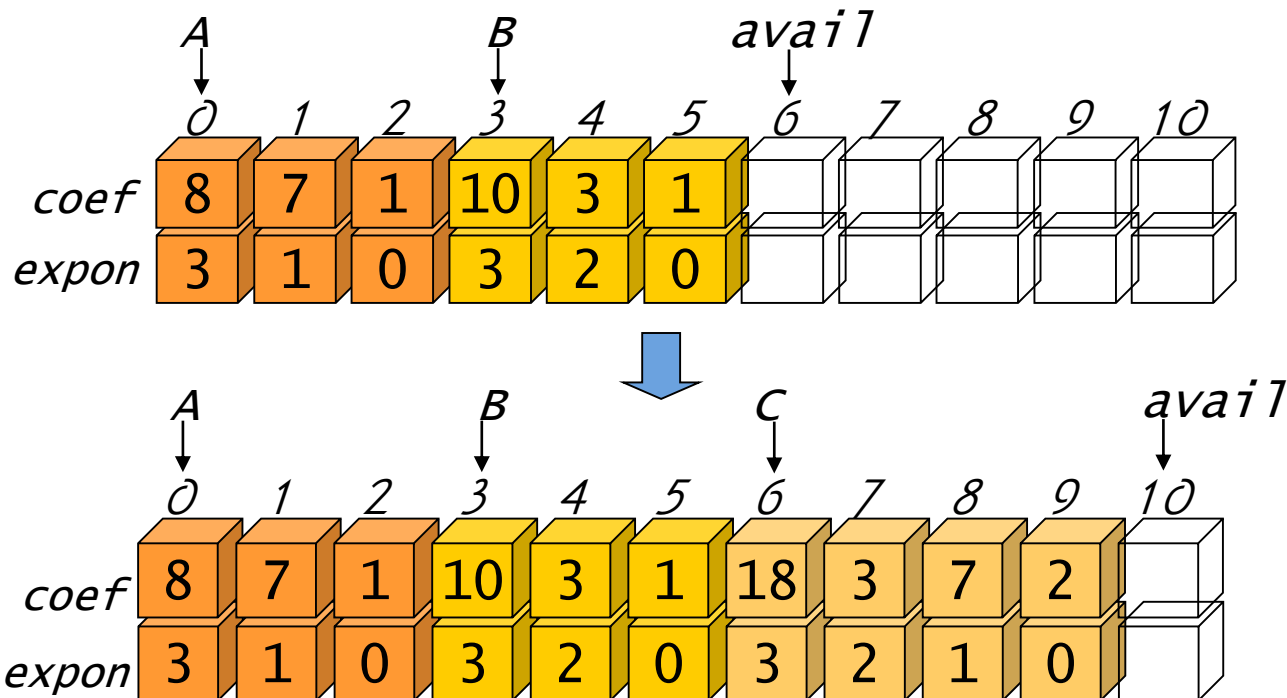
- 다항식에서 0이 아닌 항만을 배열에 저장
- (계수, 차수) 형식으로 배열에 저장
 - ▶ (예) $10x^5 + 6x + 3 \rightarrow ((10,5), (6,1), (3,0))$

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



다항식 표현 방법 #2(계속)

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



다항식 표현 방법 #2(계속)

```
#define MAX_TERMS 101

struct {
    float coef;
    int expon;
} terms[MAX_TERMS]={ {8,3}, {7,1}, {1,0}, {10,3}, {3,2},{1,0} };

int avail=6;

// 두 개의 정수를 비교
char compare(int a, int b) {
    if( a>b ) return '>';
    else if( a==b ) return '=';
    else return '<';
}
```

다항식 표현 방법 #2(계속)

// 새로운 항을 다항식에 추가한다.

```
void attach(float coef, int expon)
{
    if( avail > MAX_TERMS ){
        fprintf(stderr, "항의 개수가 너무 많음\n");
        exit(1);
    }
    terms[avail].coef = coef;
    terms[avail++].expon = expon;
}
```

다항식 표현 방법 #2(계속)

```
// C = A + B
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;
        }
}
```


다항식 표현 방법 #2(계속)

```
// 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;
}

//
void main()
{
    int Cs, Ce;
    poly_add2(0,2,3,5,&Cs,&Ce);
}
```

희소행렬

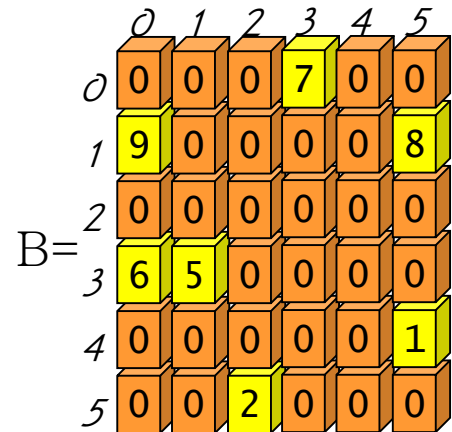
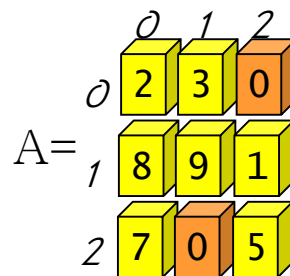
- 배열을 이용하여 행렬(matrix)를 표현하는 2가지 방법
 - (1) 2차원 배열을 이용하여 배열의 전체 요소를 저장하는 방법
 - (2) 0이 아닌 요소들만 저장하는 방법
- 희소행렬: 대부분의 항들이 0인 배열


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

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

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



희소 행렬 #1

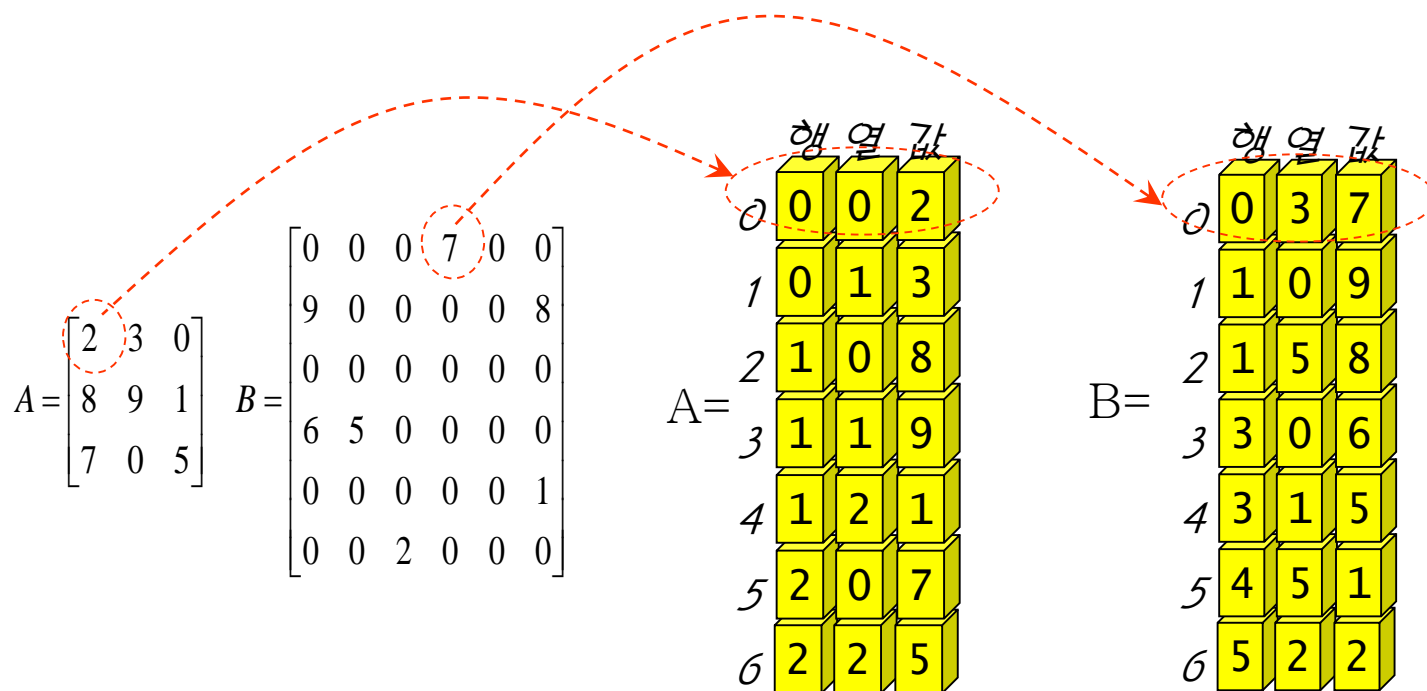
```
#include <stdio.h>

#define ROWS 3
#define COLS 3

// 희소 행렬 덧셈 함수
void sparse_matrix_add1(int A[ROWS][COLS], int B[ROWS][COLS],
    int C[ROWS][COLS] ) // C=A+B
{
    int r,c;
    for(r=0;r<ROWS;r++)
        for(c=0;c<COLS;c++)
            C[r][c] = A[r][c] + B[r][c];
}
```

희소행렬 표현방법 #2

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



희소 행렬 #2

```
#define ROWS 3
#define COLS 3
#define MAX_TERMS 10
typedef struct {
    int row; int col; int value;
} element;
typedef struct SparseMatrix {
    element data[MAX_TERMS];
    int rows; // 행의 개수
    int cols; // 열의 개수
    int terms; // 항의 개수
} SparseMatrix;
```

희소 행렬 #2

```
// 주함수
```

```
main()
```

```
{
```

```
    SparseMatrix m1 = { {{ 1,1,5 },{ 2,2,9 }}, 3,3,2 };
```

```
    SparseMatrix m2 = {{{ 0,0,5 },{ 2,2,9 }}, 3,3,2 };
```

```
    SparseMatrix m3;
```

```
    m3 = sparse_matrix_add2(m1, m2);
```

```
}
```

희소 행렬 #2

```
// 희소 행렬 덧셈 함수  $c = a + b$ 
```

```
SparseMatrix sparse_matrix_add2(SparseMatrix a, SparseMatrix b) {
```

```
    SparseMatrix c;
```

```
    int ca=0, cb=0, cc=0; // 각 배열의 항목을 가리키는 인덱스
```

```
    // 배열 a와 배열 b의 크기가 같은지를 확인
```

```
    if( a.rows != b.rows || a.cols != b.cols ){
```

```
        fprintf(stderr, "희소행렬 크기에러\n");
```

```
        exit(1);
```

```
    }
```

```
    c.rows = a.rows;
```

```
    c.cols = a.cols;
```

```
    c.terms = 0;
```

희소 행렬 #2

```
while( ca < a.terms && cb < b.terms ){  
    // 각 항목의 순차적인 번호를 계산한다.  
    int inda = a.data[ca].row * a.cols + a.data[ca].col;  
    int indb = b.data[cb].row * b.cols + b.data[cb].col;  
    if( inda < indb ) {  
        // a 배열 항목이 앞에 있으면  
        c.data[cc++] = a.data[ca++];  
    }  
    else if( inda == indb ){  
        // a와 b가 같은 위치  
        c.data[cc].row = a.data[ca].row;  
        c.data[cc].col = a.data[ca].col;  
        c.data[cc++].value = a.data[ca++].value +  
                             b.data[cb++].value;  
    }  
    else  
        // b 배열 항목이 앞에 있음  
        c.data[cc++] = b.data[cb++];  
}
```

희소 행렬 #2

// 배열 a와 b에 남아 있는 항들을 배열 c로 옮긴다.

for(; ca < a.terms; ca++)

c.data[cc++] = a.data[ca++];

for(; cb < b.terms; cb++)

c.data[cc++] = b.data[cb++];

c.terms = cc;

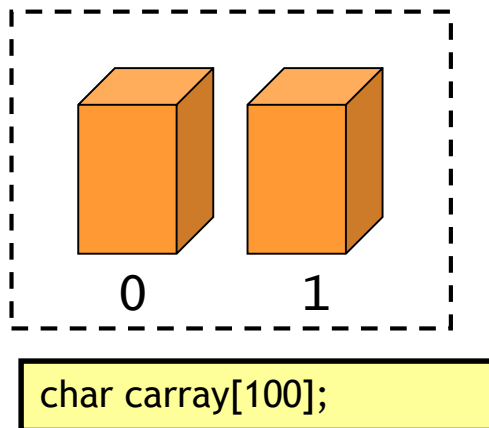
return c;

}

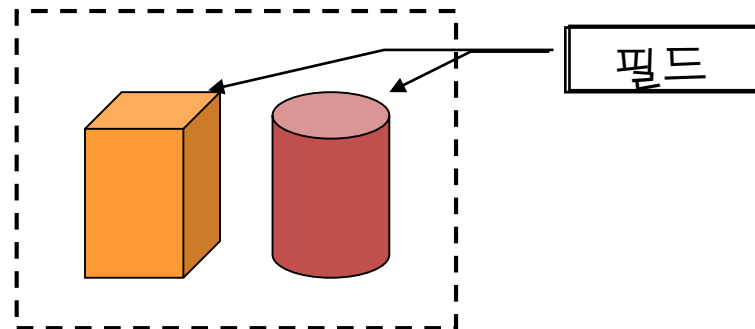
구조체

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

배열



구조체



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


구조체의 사용 예

- 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
struct person korea[100]; //구조체 배열
```

- 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
person korea[100];    //구조체 배열
```

구조체의 대입과 비교 연산

- 구조체 변수의 대입: 가능

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

구조체의 대입과 비교 연산

- 구조체 변수끼리의 비교: 불가능

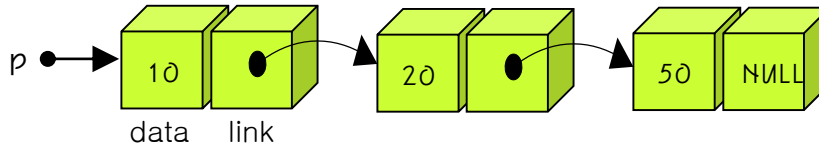
```
struct person {  
    char name[10];    // 문자배열로 된 이름  
    int age;           // 나이를 나타내는 정수값  
    float height;      // 키를 나타내는 실수값  
};  
main()  
{  
    struct person a, b;  
    if( a > b )  
        printf("a가 b보다 나이가 많음");    // 불가능  
}
```

자체참조 구조체

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

```
typedef struct ListNode {
    int      data;
    struct ListNode *link;
} ListNode;

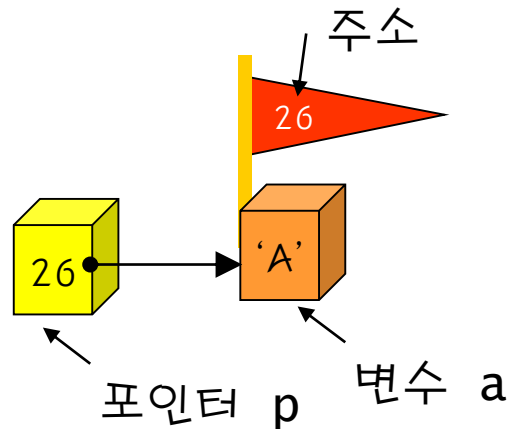
ListNode *p;
```



포인터(pointer)

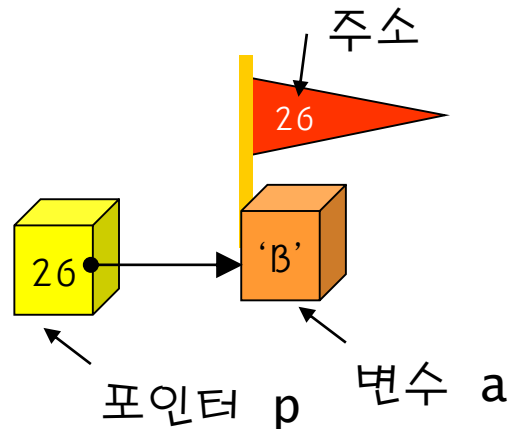
- 포인터: 다른 변수의 주소를 가지고 있는 변수

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



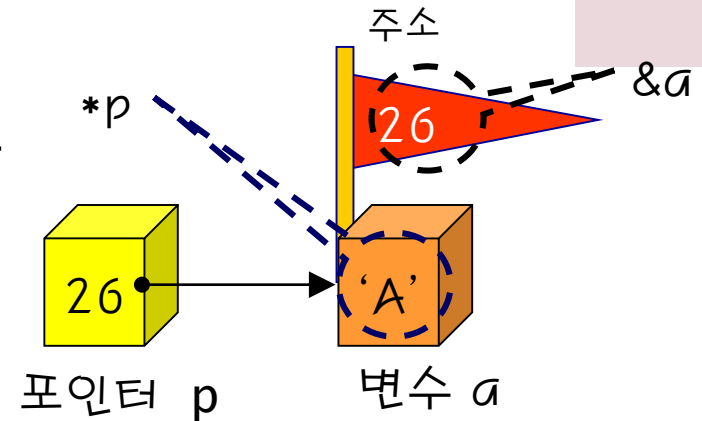
- 포인터가 가리키는 내용의 변경: * 연산자 사용

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



포인터와 관련된 연산자

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

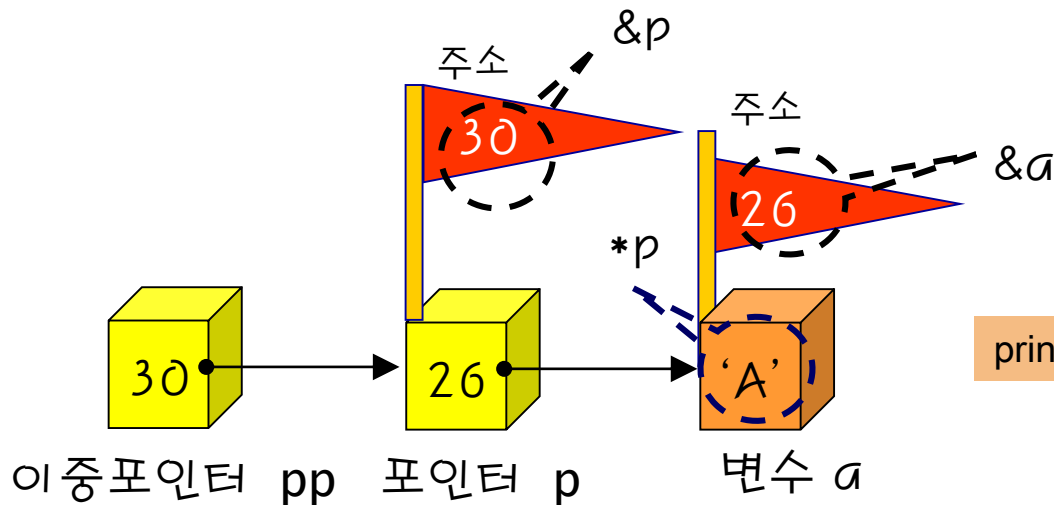


```

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.
  
```

포인터와 관련된 연산자

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



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

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
```

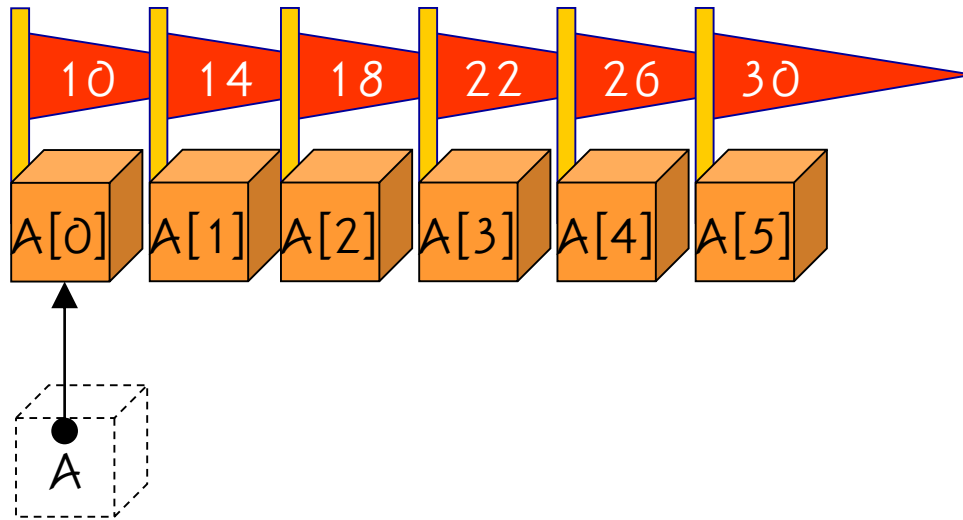

함수의 파라미터로서의 포인터

- 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

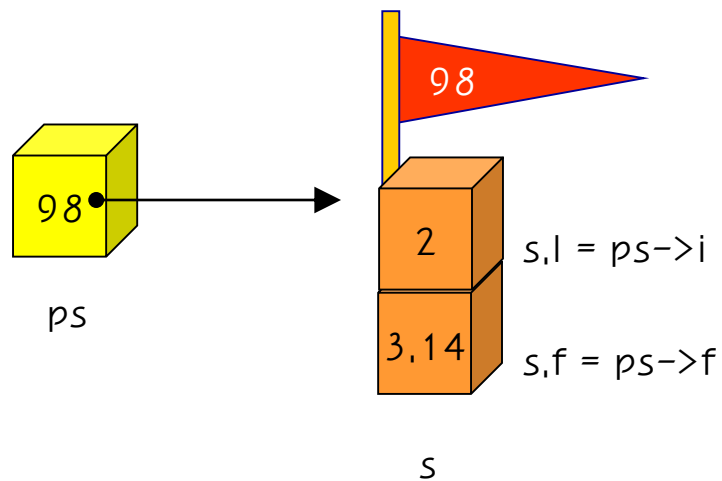
- 배열의 이름: 사실상의 포인터와 같은 역할



- 컴파일러가 배열의 이름을 배열의 첫 번째 주소로 대치

구조체의 포인터

- 구조체의 요소에 접근하는 연산자: ->



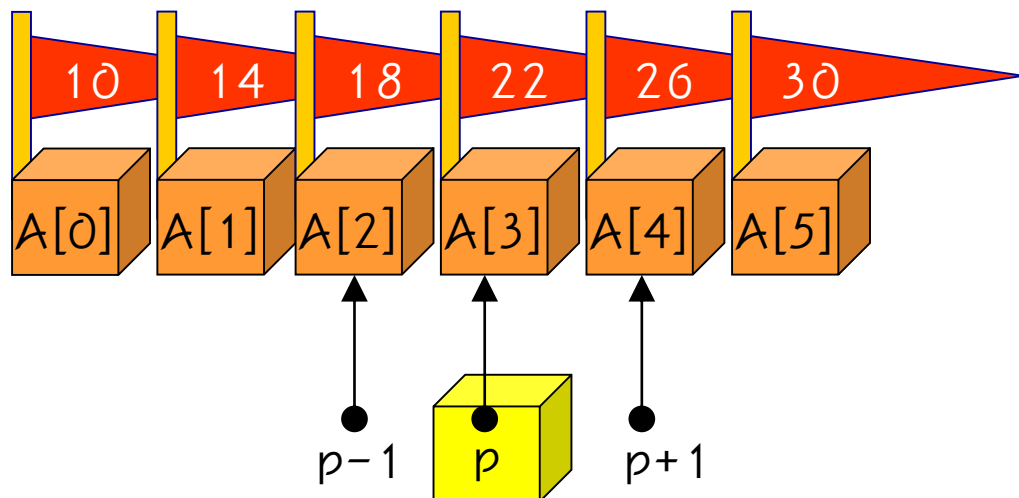
```
main()
{
    struct {
        int i;
        float f;
    } s, *ps;

    ps = &s;
    ps->i = 2;
    ps->f = 3.14;
}
```

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 변환 시에는 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

● 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 100개의 문자를 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)

동적 메모리 할당 라이브러리

```
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 length = sizeof( array ) / sizeof( array[0] ); // 20/4=5
```

```
typedef unsigned int size_t;
```

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);  
}
```

연습문제 1

- 1) `int i=10; int *p; p=&i; *p=8;`의 문장이 수행되면 i값은 얼마인가?
- 2) `int i=10; int *p; p=&i; (*p)--;`의 문장이 수행되면 i값은 얼마인가?
- 3) `int a[10]; int *p; p=a; *p++=5;`의 문장이 수행되면 변경되는 배열의 요소는?
- 4) `int a[10]; int *p; p=a; *++p=5;`의 문장이 수행되면 변경되는 배열의 요소는?
- 5) `int a[10]; int *p; p=a; (*p)++;`의 문장이 수행되면 변경되는 배열의 요소는?

연습문제 2

- 배열 x 를 $\{10,20,30,40,50,60\}$ 으로 초기화한 후 포인터 p 를 선언하고 $x[2]$ 의 주소를 저장한다.
 - 1) $*(p+3)$ 와 $*(p-2)$ 의 값을 무엇인가?
 - 2) 위의 값을 구하는 프로그램을 작성해 보자.

연습문제 3

- 다음과 같은 문장을 수행하고 난 뒤의 a[0]의 값은?

```
void sub(int b[])
```

```
{ b[0] = 0; }
```

```
void main()
```

```
{ int a[]={1,2,3,4,5,6}; sub(a); }
```

연습문제 4

- person이라는 구조체를 만들어보자. 이 구조체에는 문자배열로 된 이름, 사람의 나이를 나타내는 정수값, 각 개인의 월급을 나타내는 float값 등이 변수로 들어가야 한다.
- 앞의 구조체에 생년월일을 추가하고자 한다. 다음과 같은 구조체를 25번 구조체 내부에 포함시켜보라.

```
struct {  
    int month;  
    int day;  
    int year;  
};
```

연습문제 5

- 구조체에 정의한 `struct person`을 사용하여 다음과 같은 프로그램을 작성해 보시오.
 - ▶ 몇 명의 사람 데이터를 입력 받을지 `n`의 값을 입력 받는다.
 - ▶ `n`명의 데이터를 저장할 공간을 동적으로 할당 받고 `n`명의 데이터를 입력 받아 저장한다.
 - ▶ 그 중 가장 나이가 많은 사람을 출력한다.
 - ▶ 평균 나이를 출력한다.