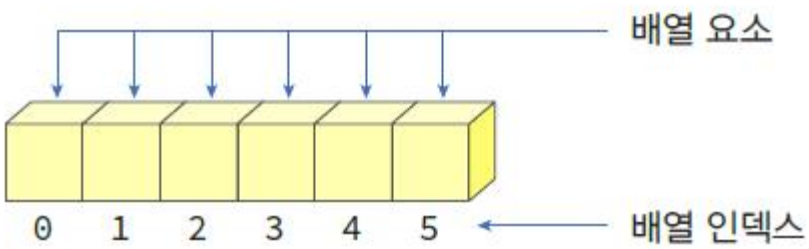
3장 배열, 구조체, 포인터

배열이란?

- 같은 형의 변수를 여러 개 만드는 경우에 사용
  - int list1, list2, list3, list4, list5, list6; → int list[6];



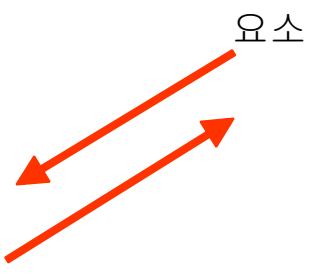
배열 ADT

ADT 3.1 Array

객체: <인덱스, 값> 쌍의 집합

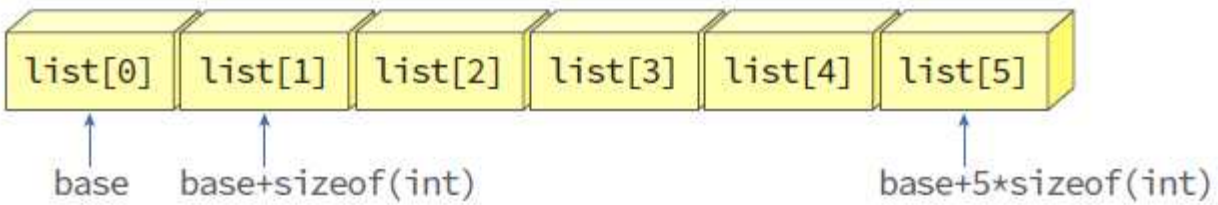
연산:

- create(size) ::= size개의 요소를 저장할 수 있는 배열 생성
- get(A, i) ::= 배열 A의 i번째 요소 반환.
- set(A, i, v) ::= 배열 A의 i번째 위치에 값 v 저장.



1차원 배열

```
int list[6];
list[0] = 100;   // set 연산에 해당된다.
value = list[0];          // get 연산에 해당된다.
```



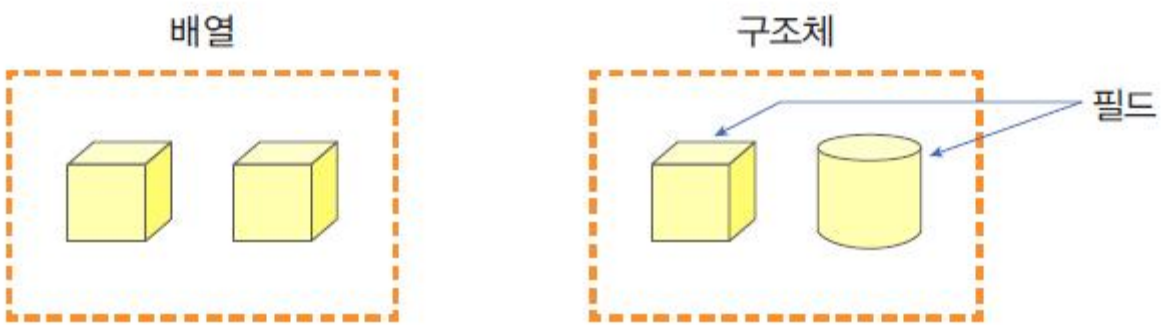
2차원 배열

- int list[3][5];

	0열	1열	2열	3열	4열
0행	list[0][0]	list[0][1]	list[0][2]	list[0][3]	list[0][4]
1행	list[1][0]	list[1][1]	list[1][2]	list[1][3]	list[1][4]
2행	list[2][0]	list[2][1]	list[2][2]	list[2][3]	list[2][4]

구조체

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



구조체의 사용예

- 구조체의 선언과 구조체 변수의 생성

```
struct studentTag {
        char name[10];    // 문자배열로 된 이름
        int age;          // 나이를 나타내는 정수값
        double gpa;         // 평균평점을 나타내는 실수값
};
```

```
struct studentTag s1;

strcpy(s.name, "kim");
s.age = 20;
s.gpa = 4.3;
```

typedef

```
typedef studentTag {
        char name[10]; // 문자배열로 된 이름
        int age;    // 나이를 나타내는 정수값
        double gpa;         // 평균평점을 나타내는 실수값
} student;


student s;

student s = { “kim", 20, 4.3 };
```

예 제

```c
#include <stdio.h>

typedef struct studentTag {
        char name[10]; // 문자배열로 된 이름
        int age;   // 나이를 나타내는 정수값
        double gpa;         // 평균평점을 나타내는 실수값
} student;

int main(void)
{
        student a = { "kim", 20, 4.3 };
        student b = { "park", 21, 4.2 };
        return 0;
}
```

배열의 응용: 다항식

- 다항식의 일반적인 형태

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

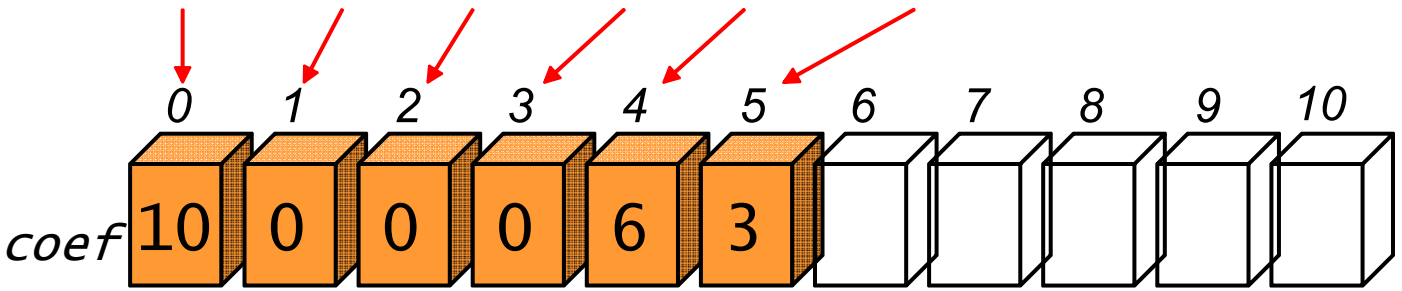
- 프로그램에서 다항식을 처리하려면 다항식을 위한 자료구조가 필요-> 어떤 자료구조를 사용해야 다항식의 덧셈, 뺄셈,곱셈, 나눗셈 연산을 할 때 편리하고 효율적일까?

배열의 응용: 다항식

- 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



다항식 표현 방법 #1(계속)

```
#define MAX_DEGREE 101 // 다항식의 최대차수 + 1
typedef struct {
        int degree;
        float coef[MAX_DEGREE];
} polynomial;
polynomial a = { 5, {10, 0, 0, 0, 6, 3} };
```

다항식 표현 방법 #1(계속)

```
// C = A+B 여기서 A와 B는 다항식이다. 구조체가 반환된다.
polynomial poly_add1(polynomial A, polynomial B)
{
        polynomial C;                                   // 결과 다항
식
        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;
}
void print_poly(polynomial p)
{
    for (int i = p.degree; i>0; i--)
        printf("%3.1fx^%d + ", p.coef[p.degree - i], i);
    printf("%3.1f \n", p.coef[p.degree]);
}
```

다항식 표현 방법 #1(계속)

```
// 주함수
int main(void)
{
        polynomial a = { 5,{ 3, 6, 0, 0, 0, 10 } };
        polynomial b = { 4,{ 7, 0, 5, 0, 1 } };
        polynomial c;
        print_poly(a);
        print_poly(b);
        c = poly_add1(a, b);
        printf("-----------------------------------------------------------------
--------\n");
        print_poly(c);
        return 0;
}
```

실 행 결 과

```
3.0x^5 + 6.0x^4 + 0.0x^3 + 0.0x^2 + 0.0x^1 + 10.0
7.0x^4 + 0.0x^3 + 5.0x^2 + 0.0x^1 + 1.0
--------------------------------------------------------------------------
3.0x^5 + 13.0x^4 + 0.0x^3 + 5.0x^2 + 0.0x^1 + 11.0
```

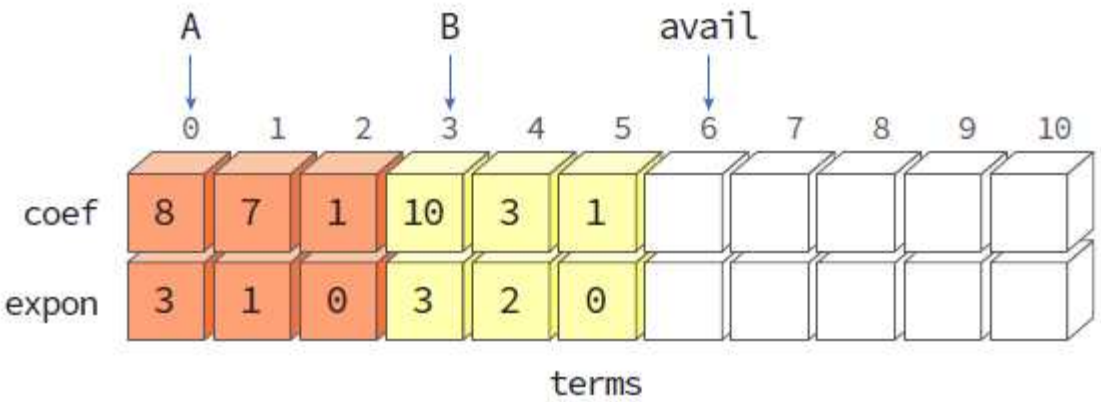
다항식 표현 방법 #2

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

```
#define MAX_TERMS 101
struct {
        float coef;
        int expon;
} terms[MAX_TERMS];
int avail;
```

예 제

$$A = 8x^3 + 7x + 1, \qquad B = 10x^3 + 3x^2 + 1$$



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 print_poly(int s, int e)
{
        for (int i = s; i < e; i++)
                printf("%3.1fx^%d + ", terms[i].coef, terms[i].expon);
        printf("%3.1fx^%d\n", terms[e].coef, terms[e].expon);
}
```

다항식 표현 방법 #2(계속)

```
//
int main(void)
{
        int As = 0, Ae = 2, Bs = 3, Be = 5, Cs, Ce;
        poly_add2(As, Ae, Bs, Be, &Cs, &Ce);
        print_poly(As, Ae);
        print_poly(Bs, Be);
        printf("-----------------------------------------------------------------------\n");
        print_poly(Cs, Ce);
        return 0;
}
```

희소행렬

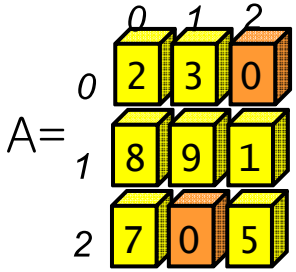
- 배열을 이용하여 행렬(matrix)을 표현하는 2가지 방법
  - (1) 2차원 배열을 이용하여 배열의 전체 요소를 저장하는 방법
  - (2) 0이 아닌 요소들만 저장하는 방법
- 희소행렬: 대부분의 항들이 0인 배열

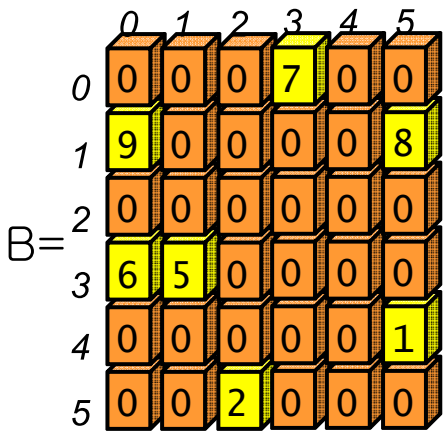
$$A = \begin{bmatrix} 2 & 3 & 0 \\ 8 & 9 & 1 \\ 7 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 & 7 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

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

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





행렬 전치 #1

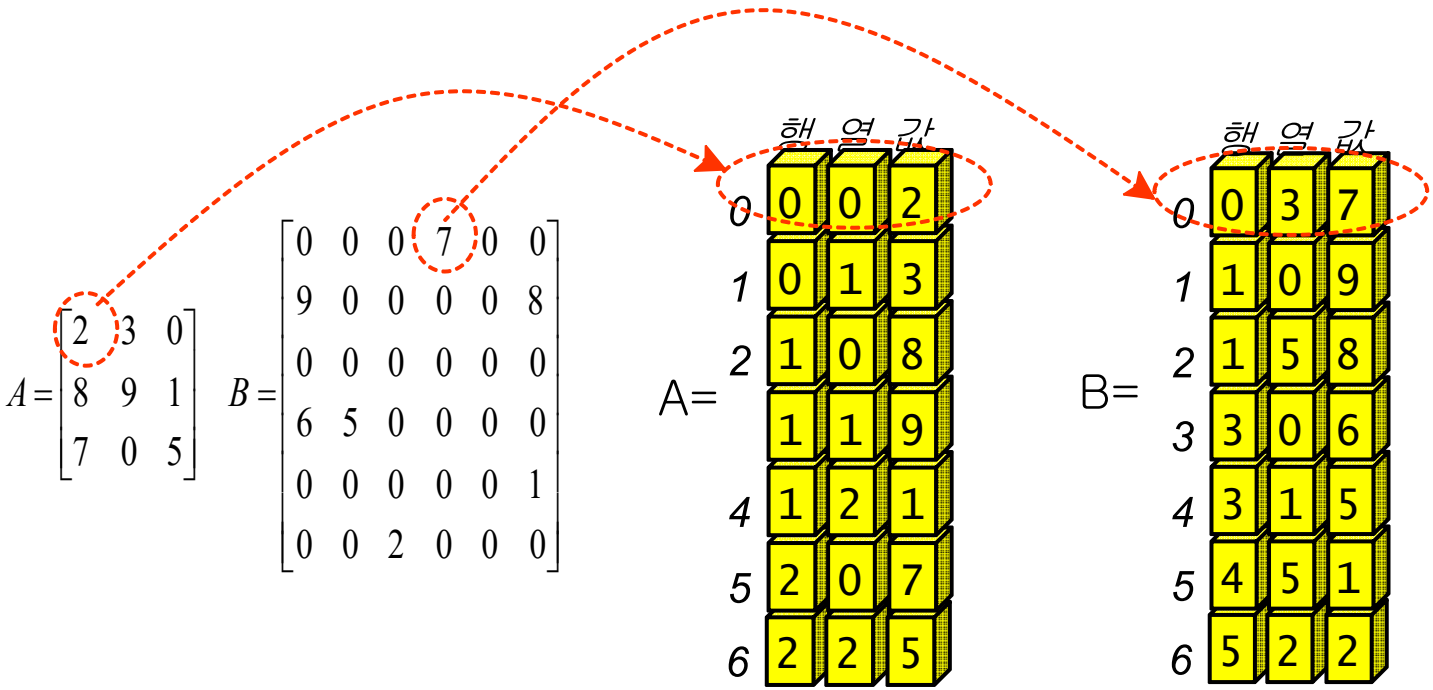
```c
#include <stdio.h>
#define ROWS 3
#define COLS 3
// 행렬 전치 함수
void matrix_transpose(int A[ROWS][COLS], int B[ROWS][COLS])
{
        for (int r = 0; r<ROWS; r++)
                for (int c = 0; c<COLS; c++)
                        B[c][r] = A[r][c];
}
void matrix_print(int A[ROWS][COLS])
{
        printf("===================\n");
        for (int r = 0; r<ROWS; r++) {
                for (int c = 0; c<COLS; c++)
                        printf("%d ", A[r][c]);
                printf("\n");
        }
        printf("===================\n");
}
```

희소 행렬 #1

```
int main(void)
{
        int array1[ROWS][COLS] = { { 2,3,0 },
                                   { 8,9,1 },
                                   { 7,0,5 } };
        int array2[ROWS][COLS];
        matrix_transpose(array1, array2);
        matrix_print(array1);
        matrix_print(array2);
        return 0;
}
```

희소행렬 표현방법 #2

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



희소 행렬 #1

```
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;
```

```
Sparse Matrix matrix_transpose2(SparseMatrix a)
{
   SparseMatrix b;
   int bindex;          // 행렬 b에서 현재 저장 위치
   b.rows = a.rows;
   b.cols = a.cols;
   b.terms = a.terms;
   if (a.terms > 0) {
         bindex = 0;
         for (int c = 0; c < a.cols; c++) {
               for (int i = 0; i < a.terms; i++) {
                     if (a.data[i].col == c) {
                           b.data[bindex].row = a.data[i].col;
                           b.data[bindex].col = a.data[i].row;
                           b.data[bindex].value = a.data[i].value;
                           bindex++;
                     }
               }
         }
   }
   return b;
}
```

희소 행렬 #1

```
void matrix_print(SparseMatrix a)
{
	printf("====================\n");
	for (int i = 0; i<a.terms; i++) {
		printf("(%d, %d, %d) \n", a.data[i].row, a.data[i].col, a.data[i].value);
	}
	printf("====================\n");
}

int main(void)
{
	SparseMatrix m = {
		{ { 0, 3, 7 },{ 1, 0, 9 },{ 1, 5, 8 },{ 3, 0, 6 },{ 3, 1, 5 },{ 4, 5, 1 },{ 5, 2, 2 } },
		6,
		6,
		7
	};
	SparseMatrix result;
	result = matrix_transpose2(m);
	matrix_print(result);
	return 0;
}
```

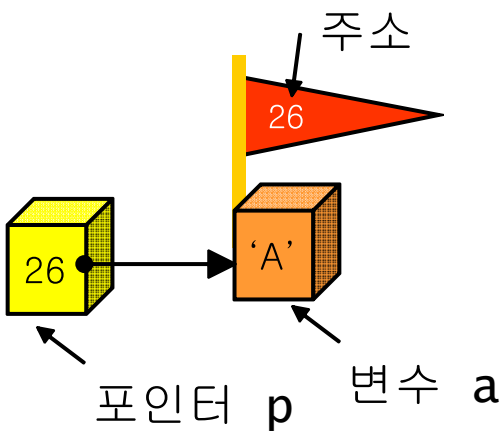
실행결과

```
====================
(0, 1, 9)
(0, 3, 6)
(1, 3, 5)
(2, 5, 2)
(3, 0, 7)
(5, 1, 8)
(5, 4, 1)
====================
```

포인터(pointer)

- 포인터: 다른 변수의 주소를 가지고 있는 변수

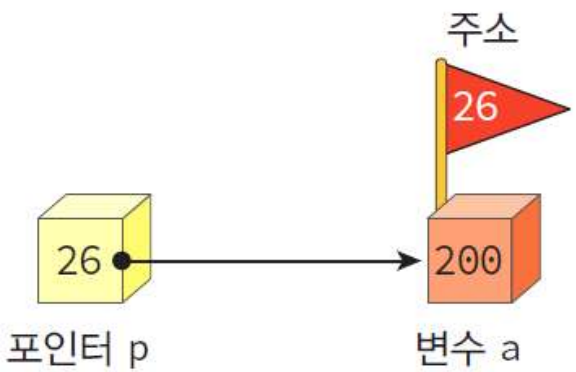
```
char a='A';
char *p;
p = &a;
```



포인터(pointer)

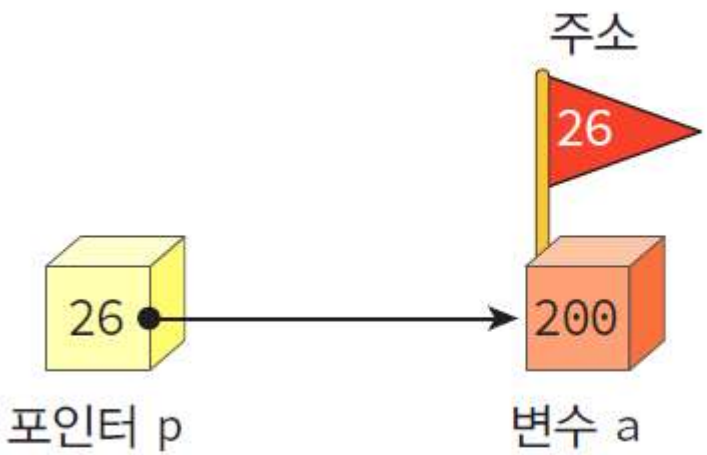
- 포인터가 가리키는 내용의 변경: * 연산자 사용

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



포인터와 관련된 연산자

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



다양한 포인터

- 포인터의 종류

```
int *p;   // p는 int형 변수를 가리키는 포인터
float *pf; // pf는 double형 변수를 가리키는 포인터
char *pc; // pc는 char형 변수를 가리키는 포인터
```

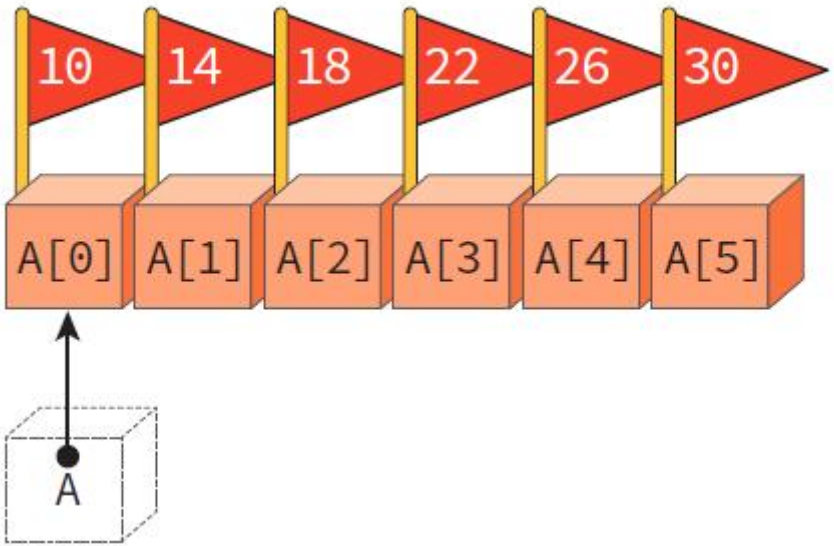
함수의 매개변수로 포인터 사용하기

- 함수안에서 매개변수로 전달된 포인터를 이용하여 외부 변수의 값 변경 가능

```
#include <stdio.h>
void swap(int *px, int *py)
{
        int tmp;
        tmp = *px;
        *px = *py;
        *py = tmp;
}
int main(void)
{
        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);
        return 0;
}
```

배열과 포인터

- 배열의 이름: 사실상의 포인터와 같은 역할

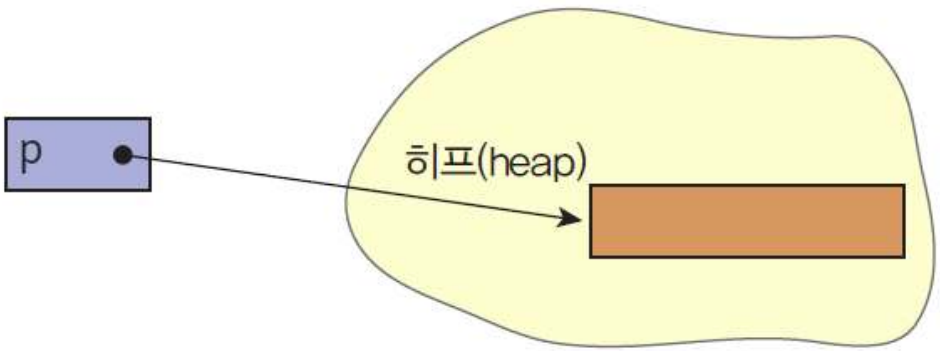


예제

```c
#include <stdio.h>
#define SIZE 6
void get_integers(int list[])
{
        printf("6개의 정수를 입력하시오: ");
        for (int i = 0; i < SIZE; ++i) {
                scanf("%d", &list[i]);
        }
}
int cal_sum(int list[])
{
        int sum = 0;
        for (int i = 0; i < SIZE; ++i) {
                sum += *(list + i);
        }
        return sum;
}
int main(void)
{
        int list[SIZE];
        get_integers(list);
        printf("합 = %d \n", cal_sum(list));
        return 0;
}
```

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 예제

```c
// 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;
}
```

실 행 결 과

```
0 1 2 3 4 5 6 7 8 9
```

구조체와 포인터

- (*ps).i보다 ps->i

예 제

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct studentTag {
        char name[10]; // 문자배열로 된 이름
        int age;    // 나이를 나타내는 정수값
        double gpa;          // 평균평점을 나타내는 실수값
} student;

int main(void)
{
        student *p;
        p = (student *)malloc(sizeof(student));
        if (p == NULL) {
                fprintf(stderr, "메모리가 부족해서 할당할 수 없습니다.\n");
                exit(1);
        }
```

예 제

```
        strcpy(p->name, "Park");
        p->age = 20;
        free(s);
        return 0;
}
```