일차원 배열과 포인터
(배열이름을 이용한 참조)

◆ 배열 score에서 배열이름 score 자체가 배열의 첫 원소의 주소값

score == &score[0]

*score == score[0]

◆ 배열이름 score를 이용하여 모든 배열원소와 각 원소의 주소값을 참조 가능

❖ 간접연산자 이용
score, score+i : 원소의 주소
*score, *(score+i) : 원소
score[0], score[i] : 원소

배열 이름 이용한 원소값과 주소값 참조 (array.c)

```
#include <stdio.h>
#define SIZE 3
int main(void) {
    int score[] = { 89, 98, 76 };

    printf("score: %p, &score[0]: %p₩n",
        score, &score[0]);

    printf("*score: %d, score[0]: %d₩n₩n", *score, score[0]);

    printf("첨자    주소       저장값₩n");
    for (int i = 0; i < SIZE; i++)
        printf("%2d %10p %6d₩n", i, (score + i), *(score + i));

    return 0;
}
```

배열이름 score는 첫 번째 원소의 주소

[결과]

```
score: 001BFCB8, &score[0]: 001BFCB8
*score: 89, score[0]: 89

첨자    주소       저장값
 0   001BFCB8     89
 1   001BFCBC     98
 2   001BFCC0     76
```

Score를 간접참조하면 첫 번째 원소의 값

배열이름 score를 사용한 주소와 원소 값 참조

포인터 변수를 이용한 배열의 원소 참조 (ptrtoary.c)

```c
#include <stdio.h>
int main(void) {
    int a[4] = { 1, 3, 6, 8 };
    int *pa = a;

    //새로 선언한 포인터 변수를 사용
    printf("%d  %d  %d  %d₩n", *(pa), *(pa + 1), *(pa + 2), *(pa + 3));
    //새로 선언한 포인터 변수를 배열과 같이 사용 가능
    printf("%d  %d  %d  %d₩n", pa[0], pa[1], pa[2], pa[3]);

    printf("%d  %d  %d  %d₩n", a[0], a[1], a[2], a[3]);
    printf("%d  %d  %d  %d₩n₩n", *a, *(a + 1), *(a + 2), *(a + 3));
    return 0;
}
```

포인터 변수의 연산을 이용하여 배열 원소에 접근

원래 배열 이름 이용한 참조도 가능

[결과]

```
1  3  6  8
1  3  6  8
1  3  6  8
1  3  6  8
```

포인터 변수의 명시적 형변환
((char *)를 (int *)로 변환. ptypecast.c)

```
#include <stdio.h>

int main(void) {
    char c[4] = { 'A', '₩0', '₩0', '₩0' }; //문자'A' 코드값: 65
    int *pi = (int *)&c[0];

    printf("%d %c₩n", (int)c[0], c[0]);
    printf("%d %c₩n", *pi, (char)*pi);

    return 0;
}
```

[결과]
65 A
65 A

명시적 형변환 해주지 않으면 경고 발생

이차원 배열에서 배열이름

- 이차원 배열에서 배열이름인 td
  - 배열이름인 td는 포인터의 포인터인 이중 포인터
- int td[][3] = {{8, 5, 4}, {2, 7, 6}};
  - 배열이름 td는 이차원 배열을 대표하는 이중 포인터
  - sizeof(td)는 배열전체의 바이트 크기를 반환
  - 배열이름 td를 이용하여 변수 td[0][0]의 값을 20으로 수정
    - **td = 20; // td가 이중 포인터이므로 간접연산자 *이 2개 필요
  - td[i]는 (i+1) 번째 행을 대표
    - (i+1) 번째 행의 처음을 가리키는 포인터 상수
  - sizeof(td[0])와 sizeof(td[1])
    - 각각 첫 번째 행의 바이트 크기와 두 번째 행의 바이트 크기를 반환
  - *td[1] == td[1][0]

2차원 배열의 이름을 이용한 원소 참조 (tdaryptr.c)

```
#include <stdio.h>
#define ROW 2
#define COL 3
int main(void) {
    int td[][COL] = { { 8, 5, 4 },{ 2, 7, 6 } };
    **td = 10;              //td[0][0] = 10;
    *td[1] = 20;            //td[1][0] = 20;
    for (int i = 0, cnt = 0; i < ROW; i++) {
        for (int j = 0; j < COL; j++, cnt++) {
            printf("%d %d %d, ", *(*td + cnt), *(td[i] + j), *(*(td + i) + j));
        }
        printf("₩n");
    }
    printf("%d, %d, %d₩n", sizeof(td), sizeof(td[0]), sizeof(td[1]));
    printf("%p, %p, %p₩n", td, td[0], td[1]);
    printf("%p, %p₩n", &td[0][0], &td[1][0]);
    return 0;
}
```

[결과]
10 10 10, 5 5 5, 4 4 4,
20 20 20, 7 7 7, 6 6 6,
24, 12, 12
00CFFC88, 00CFFC88, 00CFFC94
00CFFC88, 00CFFC94

td가 2중포인터이므로
2회의 간접참조를 해야
원 데이터가 나옴.

포인터 배열(array of pointer)

◆ 포인터를 배열 원소로 하는 배열

```
int a = 5, b = 6, c = 7;
int *pa[3];
pa[0] = &a; pa[1] = &b pa[2]=&c;

double *pda[5] = { NULL };
// 모든 배열원소에 NULL 주소가 저장
```

[x86(32bit)으로 수행한 결과]

```
a[0]=6028520 a[1]=6028528 a[2]=602
p1=6028520 p1+1=6028528
p2=6028492 p2+1=6028496
*p2=6028520 *(p2+1)=6028528
```

```c
#include <stdio.h>
int main(void) {
    int a[3][2] = { {1,2}, {3,4}, {5,6} };
    int (*p1)[2] = a;
    int *p2[2] = { a[0], a[1] };

    printf("a[0]=%u a[1]=%u a[2]=%u₩n", a[0], a[1], a[2]);
    printf("p1=%u p1+1=%u₩n", p1, p1 + 1);
    printf("p2=%u p2+1=%u₩n", p2, p2 + 1);
    printf("*p2=%u *(p2+1)=%u₩n", *p2, *(p2 + 1));
    return 0;
}
```

p1 : 8씩 움직임
p2 : 4씩 움직임
p2 : 간접 참조 해야 a의 한 행이 나옴.

포인터 배열 활용
(pointerarray.c)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define SIZE 3

int main(void) {
    int  *pary[SIZE] = { NULL };
    int a = 10, b = 20, c = 30;

    pary[0] = &a; pary[1] = &b; pary[2] = &c;
    for (int i = 0; i < SIZE; i++)
        printf("*pary[%d] = %d₩n", i, *pary[i]);
    for (int i = 0; i < SIZE; i++) {
        scanf("%d", pary[i]);
        printf("%d, %d, %d₩n", a, b, c);
    }
    return 0;
}
```

[결과]
*pary[0] = 10
*pary[1] = 20
*pary[2] = 30
1
1, 20, 30
2
1, 2, 30
3
1, 2, 3

배열 포인터(pointer to array)

```
int a[] = { 8, 2, 8, 1, 3 };
int *p = a;

printf( "%02d, %02d₩n", *(p+1), *(p+4) );
printf( "%02d, %02d₩n", p[0], p[4] );
printf( "%02d, %02d₩n", sizeof(a), sizeof(p) );
```

- **배열이름 a**
  - **포인터 상수 (다른 주소값을 가질 수 없음.) (a++ 또는 ++a 연산을 수행 불가능)**
  - **sizeof(a)는 배열의 총 바이트 크기인 20 = (5*4)**
- **변수 p**
  - **포인터 변수 (다른 주소값을 가질 수 있음.) (p++ 또는 ++p 연산이 가능)**
  - **sizeof(p)은 단순히 포인터의 크기인 4**
- **공통점**
  - **p와 a 모두 배열 a의 첫 원소의 주소값을 가짐**
  - **a[i]와 같이 p[i]로 배열 a의 모든 원소를 참조**

배열 원소 수 구하기

- 일차원 배열의 원소 수 구하기
  - sizeof(배열이름) / sizeof(배열원소)
  - int arr[] = { 1, 2, 3};
    int count = sizeof(arr) / sizeof(arr[0]);

- 이차원 배열의 행의 수 구하기
  - sizeof(a) / sizeof(a[0])
- 이차원 배열의 열의 수 구하기
  - sizeof(a[0]) / sizeof(a[0][0])

1차원,2차원 배열의 배열크기 활용 (arraysize.c)

```
#include <stdio.h>
int main(void) {
    int data[] = { 3, 4, 5, 7, 9 };
    printf("%d %d₩n", sizeof(data), sizeof(data[0]));
    printf("일차원 배열: 배열 크기 == %d₩n",
        sizeof(data) / sizeof(data[0]));
    double x[][3] = { { 1, 2, 3 },{ 7, 8, 9 },{ 4, 5, 6 },{ 10, 11, 12 } };
    printf("%d %d %d₩n", sizeof(x), sizeof(x[0]), sizeof(x[0][0]));
    int rowcount = sizeof(x) / sizeof(x[0]);
    int colcount = sizeof(x[0]) / sizeof(x[0][0]);
    printf("이차원 배열: 행수 == %d  열수 == %d₩n", rowcount, colcount);
    printf("이차원 배열: 전체 원소 수 == %d₩n", sizeof(x) / sizeof(x[0][0]));
    return 0;
}
```

[결과]

```
20 4
일차원 배열: 배열 크기 == 5
96 24 8
이차원 배열: 행수 == 4  열수 == 3
이차원 배열: 전체 원소 수 == 12
```