

C언어 강의자료

문정욱

A decorative graphic consisting of several light blue squares of varying sizes arranged in a stepped pattern on the left side of the slide.

C언어 더 알아보기 5

문자열(string) 입출력

■ printf 함수의 주소 FSF

- %p
 - 정수(16진수 8자리) 출력
 - **X* 타입**(임의의 주소 타입)의 주소 값 인자와 대응
- %s
 - 문자열 출력
 - **char* 타입**의 주소 값 인자와 대응

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
char s[] = "abc";
```

```
char* p = "abc";
```

```
printf("%p (%s)\n", s, s);
```

```
printf("%p (%s)\n", p, p);
```

```
return 0;
```

```
}
```

배열 s의
첫 번째 요소의 주소
char[4] --> char*

문자열 상수 "abc"의
첫 번째 요소의 주소

X*

char*

입출력 결과

```
0041FB08 (abc)
```

```
0123573C (abc)
```

```
계속하려면 아무 키나 누르십시오 . . .
```

문자열(string) 입출력

■ scanf 함수의 문자열 FSF

- %s
 - 문자열 입력
 - **char* 타입**의 주소 값 인자와 대응
 - 반드시 할당된 메모리 영역의 주소를 전달해야 한다.
 - 할당되지 않은 메모리 영역에 자료를 저장하면 **실행오류** 발생

```
#include <stdio.h>

int main(void)
{
    char s[512];
    char* p;           // garbage value

    scanf("%s", s );
    scanf("%s", p );   // run-time error
    return 0;
}
```

char*

p에 쓰레기 값(garbage value)이
저장되어 있다.
이는 p가 할당된 메모리 영역을
가리키고 있지 않다는 뜻이다.

문자열 상수(string constant)

- 문자열 요소 포인터(char pointer)
 - 이름 없는 문자열의 첫 번째 요소를 가리킴.
 - 배열 요소 포인터
- 문자열(char array)
 - 이름 있는 문자열

입출력 결과

```

00FD5744 4
00FD5744 4
00FD5744 4
00FD5744 4
003CFA70 6
003CFA60 6
계속하려면 아무 키나 누르십시오 . . .
  
```

동일한 주소

```
#include <stdio.h>
```

```
int main(void)
{
```

같은 내용의 문자열 상수는
동일한 문자열 상수로 간주되고
동일한 메모리공간을 차지한다.

```
    char* s1="abcde";    // pointer
```

```
    char* s2="abcde";    // pointer
```

```
    char* s3="abc" "de"; // pointer
```

```
    char* s4="abc\
```

```
de";    // pointer
```

```
    char s5[6]="abcde"; // array
```

```
    char s6[]="abcde";  // array
```

```
    printf("%p %d\n", s1, sizeof s1);
```

```
    printf("%p %d\n", s2, sizeof s2);
```

```
    printf("%p %d\n", s3, sizeof s3);
```

```
    printf("%p %d\n", s4, sizeof s4);
```

```
    printf("%p %d\n", s5, sizeof s5);
```

```
    printf("%p %d\n", s6, sizeof s6);
```

```
    return 0;
```

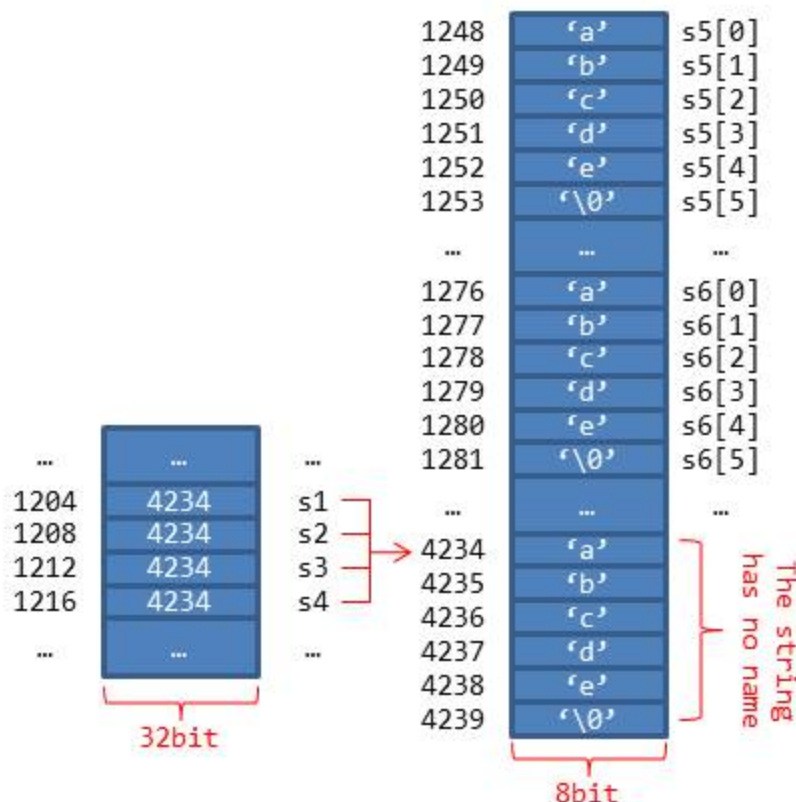
```
}
```

char[6] --> char*

문자열 상수(string constant)

■ 구조

- 문자열의 메모리에서 위치



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    char* s1="abcde";    // pointer
```

```
    char* s2="abcde";    // pointer
```

```
    char* s3="abc" "de"; // pointer
```

```
    char* s4="abc\nde";  // pointer
```

```
    char s5[6]="abcde";  // array
```

```
    char s6[]="abcde";   // array
```

```
    printf("%p %d\n", s1, sizeof s1);
```

```
    printf("%p %d\n", s2, sizeof s2);
```

```
    printf("%p %d\n", s3, sizeof s3);
```

```
    printf("%p %d\n", s4, sizeof s4);
```

```
    printf("%p %d\n", s5, sizeof s5);
```

```
    printf("%p %d\n", s6, sizeof s6);
```

```
    return 0;
```

```
}
```


문자열 및 배열의 길이

- 문자열의 길이
 - 첫 번째 '\0'까지 문자의 개수 ('\0'제외)
- 배열의 길이
 - 배열 요소의 개수
 - 타입을 활용하여 개수를 알아야 한다.

입출력 결과

```
string size == 4
array size == 10
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    char a[10]={'a','b','c','d','\0'};
    int i,s;

    printf("string size == %d\n", i );

    printf("array size == %d\n", s );
    return 0;
}
```

연산자

■ 연산자의 결합 순서

• 왼쪽 → 오른쪽 결합

- 수식에서 동일한 연산자가 나열될 경우 **왼쪽 연산자를 우선**한다.

• 오른쪽 → 왼쪽 결합

- 수식에서 동일한 연산자가 나열될 경우 **오른쪽 연산자를 우선**한다.

```
#include <stdio.h>

int main(void)
{
    int a, b, c;

    a = 1 + 2 + 3;
    c = b = a;

    return 0;
}
```


연산자

연산자 우선 순위

	연산자	결함 순서
배열/구조체 연산자	[] -> .	left → right
단일 연산자	* & (type) sizeof ! ~ ++ -- + -	right → left
산술 연산자	* / %	left → right
	+ -	left → right
관계 연산자	< <= > >=	left → right
	== !=	left → right
논리 연산자	&&	left → right
		left → right
조건 연산자	? :	right → left
대입 연산자	= += -= *= /= %=	right → left
콤마 연산자	, (comma operator)	left → right

2차원 배열

■ 1차원 배열

- 비배열(non-array) 변수들로 구성된 배열

```
X a[N];
```

■ 2차원 배열

- 1차원 배열들로 구성된 배열

```
Y a[R];
```

```
Y == X[C]
```

```
X[C] a[R];
```

```
개념적 타입
```

```
X a[R][C];
```

```
C언어 타입
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int a[2][3]; // T(a) == int[3][2]
```

```
    return 0;
```

```
}
```

2차원 배열

■ 2차원 배열 구조

`int a[2][3];` C언어 타입

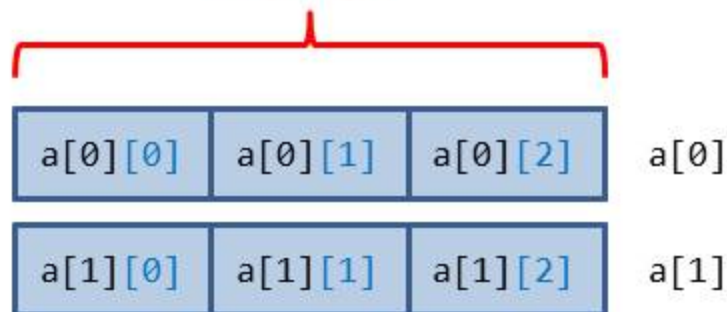
`int[3] a[2];` 개념적 타입

→ `int[3]` 타입의 원소 2개로 구성된 배열

`int[3][2] a;` 개념적 타입

→ `int[3][2]` 타입의 배열 `a`

1차원 배열



```
#include <stdio.h>
```

```
int main(void)
{
    int a[2][3]; // T(a) == int[3][2]
    return 0;
}
```

(2x3 matrix)
`a[0][0]` `a[0][1]` `a[0][2]`
`a[1][0]` `a[1][1]` `a[1][2]`

2차원 배열

■ 2차원 배열의 초기화

`int a[2][3];` C언어 타입

`int[3] a[2];` 개념적 타입

`int[3]`을 초기화하기 위한 내용이 `initializer`에 들어가야 한다.

입출력 결과

1 2 3

4 5 6

계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int a[2][3] = {
```

```
        {1, 2, 3},
```

```
        {4, 5, 6}
```

`a[0]` 초기화

`a[1]` 초기화

```
    };
```

```
    int row, col;    // row, column
```

```
    for(row=0; row<2; ++row) {
```

```
        for(col=0; col<3; ++col)
```

```
            printf("%d ", a[row][col] );
```

```
        printf("\n");
```

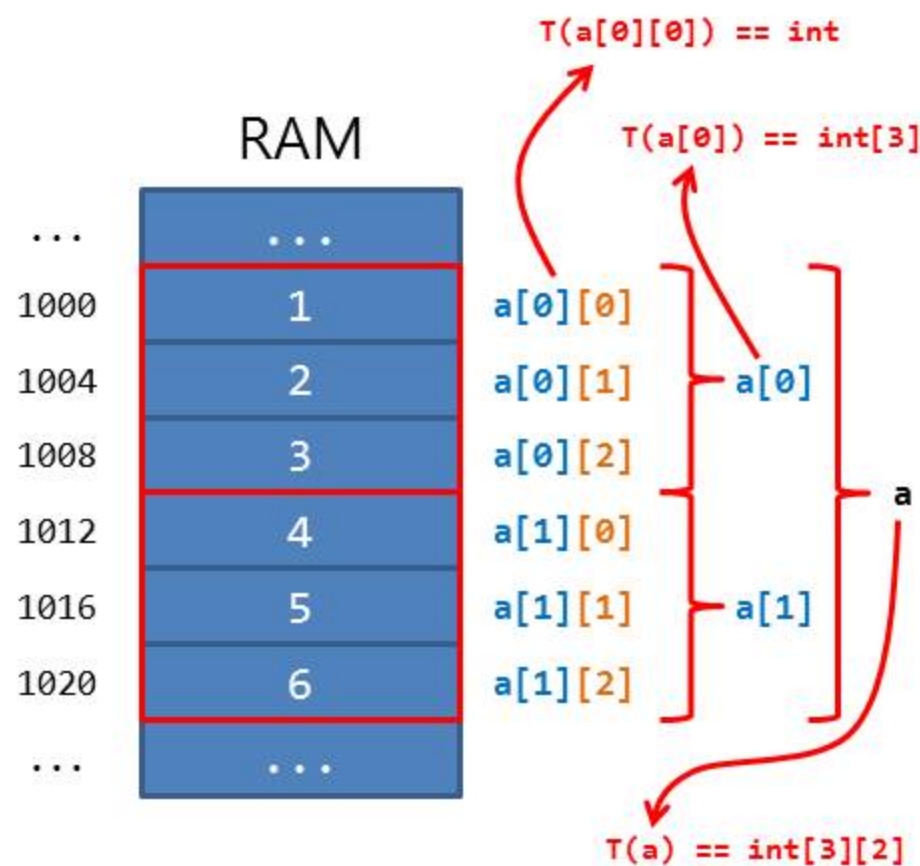
```
    }
```

```
    return 0;
```

```
}
```

2차원 배열

■ 2차원 배열의 메모리 구조



```
#include <stdio.h>

int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int row, col;    // row, column

    for(row=0; row<2; ++row) {
        for(col=0; col<3; ++col)
            printf("%d ", a[row][col] );
        printf("\n");
    }
    return 0;
}
```

2차원 배열

■ 2차원 배열 각 요소의 타입

`int a[2][3];` C언어 타입

`int[3] a[2];` 개념적 타입

`int[3][2] a;` 개념적 타입

입출력 결과

```
4
12
24
row == 2
col == 3
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int row, col;

    printf("%d\n", sizeof(a[0][0]) );
    printf("%d\n", sizeof(a[0]) );
    printf("%d\n", sizeof(a) );

    row = sizeof(a)/sizeof(a[0]);
    col = sizeof(a[0])/sizeof(a[0][0]);
    printf("row == %d\n", row );
    printf("col == %d\n", col );
    return 0;
}
```

$T(a[0][0]) == \text{int}$
 $T(a[0]) == \text{int}[3]$
 $T(a) == \text{int}[3][2]$
 $ET(a[0]) == \text{int}$
 $ET(a) == \text{int}[3]$

2차원 배열

■ 2차원 배열 포인터

```
X a[R][C];
T(a) == X[C][R];
```

```
PT(a) == X[C][R]*
T(&a) == X[C][R]*
```

```
X[C][R]* p;
```

개념적 타입



```
X[C] (*p)[R];
```

개념적 타입



```
X (*p)[R][C];
```

C언어 타입

```
#include <stdio.h>

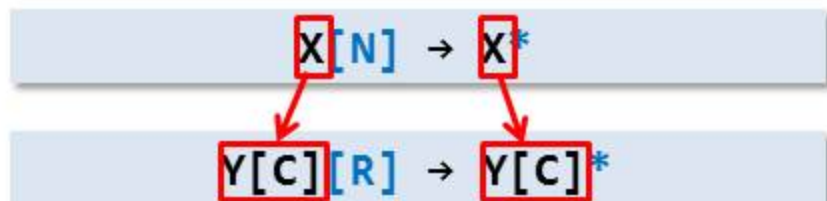
int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int (*pa)[2][3];

    pa = &a;    // T(&a) == int[3][2]*

    (*pa)[0][0] = 11;
    (*pa)[0][1] = 22;
    (*pa)[0][2] = 33;
    (*pa)[1][0] = 44;
    (*pa)[1][1] = 55;
    (*pa)[1][2] = 66;
    return 0;
}
```

2차원 배열

- 2차원 배열의 묵시적 타입 변환



```
#include <stdio.h>

int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int (*p1)[3];
    int (*p2)[3];

    p1 = &a[0]; // T(a[0]) == int[3]
    p2 = a;     // T(&a[0]) == int[3]*
               // int[3][2] → int[3]*

    return 0;
}
```

2차원 배열

■ 2차원 배열 요소 포인터

EPT(X[N]) == X*
EPT(X[C][R]) == X[C]*

X a[R][C];
X[C] a[R];
PT(ET(a)) == EPT(a) == X[C]*

X[C]* p;

개념적 타입



X (*p)[C];

C언어 타입

```
#include <stdio.h>

int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int (*pe)[3];

    pe = a; // int[3][2] → int[3]*

    pe[0][0] = 11; // == a[0][0]
    pe[0][1] = 22; // == a[0][1]
    pe[0][2] = 33; // == a[0][2]
    pe[1][0] = 44; // == a[1][0]
    pe[1][1] = 55; // == a[1][1]
    pe[1][2] = 66; // == a[1][2]

    return 0;
}
```

2차원 배열

배열형 인자 전달

```
#include <stdio.h>

void add(int d[2][3], int s1[2][3], int s2[2][3])
{
    int r, c;

    for(r=0; r<2; ++r)
        for(c=0; c<3; ++c)
            d[r][c] = s1[r][c] + s2[r][c];
}

void print(int m[2][3])
{
    int r, c;

    for(r=0; r<2; ++r) {
        for(c=0; c<3; ++c)
            printf("%d\t", m[r][c]);
        printf("\n");
    }
}
```

```
int main(void)
{
    int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
    int b[2][3] = {{2, 3, 4}, {5, 6, 7}};
    int c[2][3];

    add(c, a, b); // c=a+b
    print(c);
    return 0;
}
```

2차원 배열

생략형 인자

```
#include <stdio.h>

void add(int d[][3],int s1[][3],int s2[][3])
{
    int r,c;

    for(r=0;r<2;++r)
        for(c=0;c<3;++c)
            d[r][c]=s1[r][c]+s2[r][c];
}

void print(int m[][3])
{
    int r,c;

    for(r=0;r<2;++r) {
        for(c=0;c<3;++c)
            printf("%d\t",m[r][c]);
        printf("\n");
    }
}
```

```
int main(void)
{
    int a[2][3]={1,2,3},{4,5,6};
    int b[2][3]={2,3,4},{5,6,7};
    int c[2][3];

    add(c,a,b); // c=a+b
    print(c);
    return 0;
}
```

2차원 배열

포인터형 인자 전달

```
#include <stdio.h>

void add(int (*d)[3], int (*s1)[3], int (*s2)[3])
{
    int r, c;

    for(r=0; r<2; ++r)
        for(c=0; c<3; ++c)
            d[r][c] = s1[r][c] + s2[r][c];
}

void print(int (*m)[3])
{
    int r, c;

    for(r=0; r<2; ++r) {
        for(c=0; c<3; ++c)
            printf("%d\t", m[r][c]);
        printf("\n");
    }
}
```

```
int main(void)
{
    int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
    int b[2][3] = {{2, 3, 4}, {5, 6, 7}};
    int c[2][3];

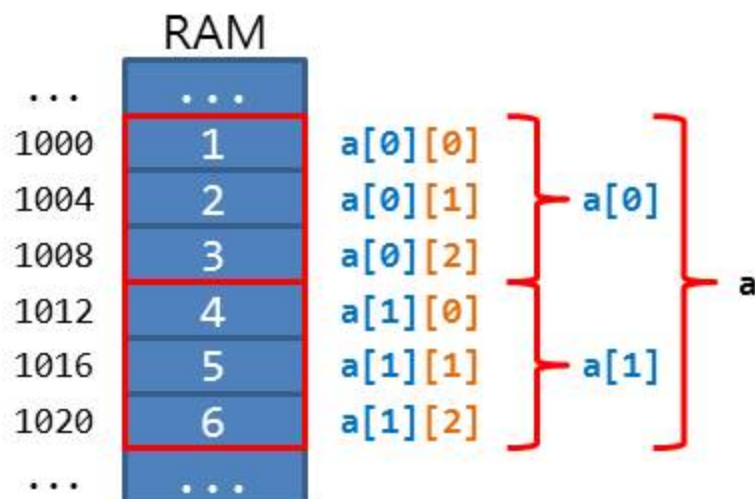
    add(c, a, b); // c=a+b
    print(c);
    return 0;
}
```


2차원 배열

- 2차원 배열의
행과 열, 전체 원소의 개수

`X a[R][C]; // N == R*C`

```
R == sizeof(a)/sizeof(a[0])
C == sizeof(a[0])/sizeof(a[0][0])
N == sizeof(a)/sizeof(a[0][0])
```



```
#include <stdio.h>

int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int row = sizeof(a)/sizeof(a[0]);
    int col = sizeof(a[0])/sizeof(a[0][0]);
    int num = sizeof(a)/sizeof(a[0][0]);

    printf("row == %d\n", row);
    printf("col == %d\n", col);
    printf("num == %d\n", num);
    return 0;
}
```

입출력 결과

```
row == 2
col == 3
num == 6
계속하려면 아무 키나 누르십시오 . . .
```

2차원 배열

- 2차원 배열의
행과 열, 전체 원소의 개수

```
X a[R][C]; // N == R*C
```

```
R == sizeof(a)/sizeof(a[0])
C == sizeof(a[0])/sizeof(a[0][0])
N == sizeof(a)/sizeof(a[0][0])
```

```
R == sizeof(a)/sizeof(*(a+0))
C == sizeof(*(a+0))/sizeof(*(a+0)+0)
N == sizeof(a)/sizeof(*(a+0)+0)
```

```
R == sizeof(a)/sizeof(*a)
C == sizeof(*a)/sizeof(**a)
N == sizeof(a)/sizeof(**a)
```

```
#include <stdio.h>

int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int row = sizeof(a)/sizeof(*a);
    int col = sizeof(*a)/sizeof(**a);
    int num = sizeof(a)/sizeof(**a);

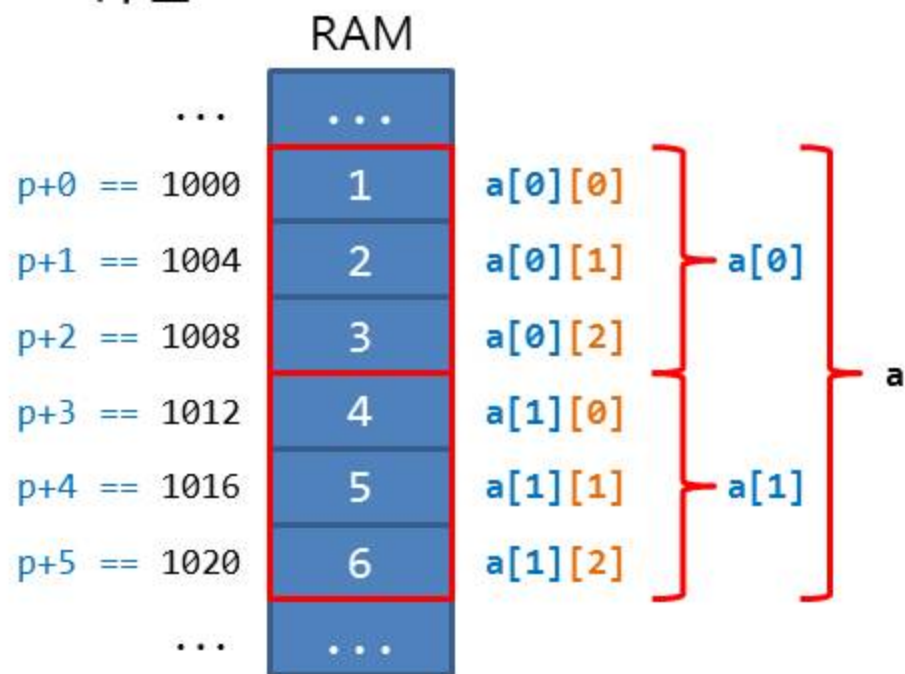
    printf("row == %d\n", row);
    printf("col == %d\n", col);
    printf("num == %d\n", num);
    return 0;
}
```

입출력 결과

```
2
3
6
계속하려면 아무 키나 누르십시오 . . .
```

2차원 배열

■ 2차원 배열의 1차원 배열 취급



입출력 결과

```
1 2 3 4 5 6
계속하려면 아무 키나 누르십시오 . . .
```

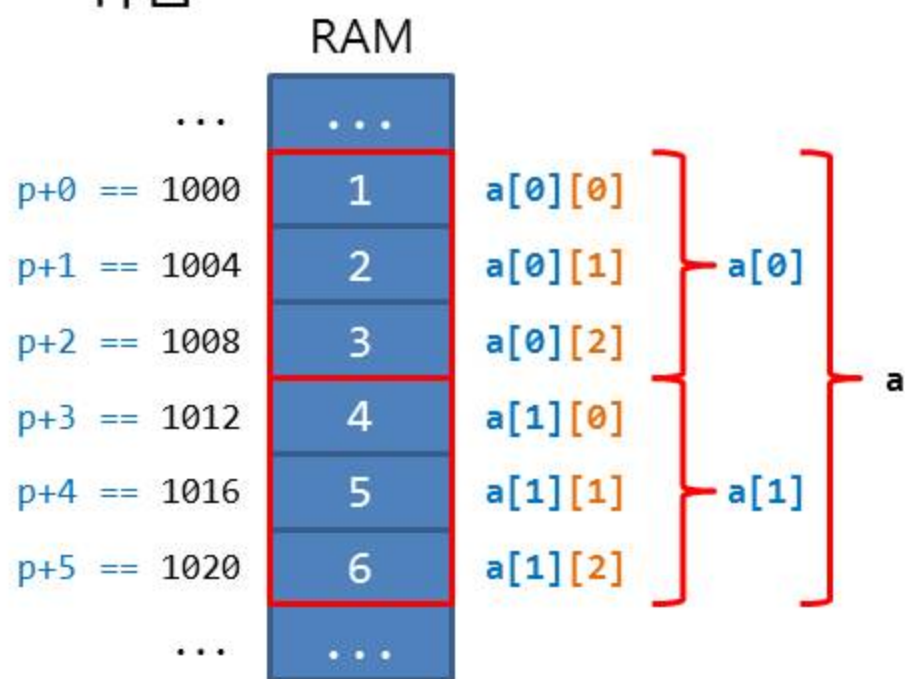
```
#include <stdio.h>

int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int* p = &a[0][0];
    int i;
    int num = sizeof(a)/sizeof(**a);

    for(i=0; i<num; ++i)
        printf("%d ", p[i]); // p[i] == *(p+i)
    printf("\n");
    return 0;
}
```

2차원 배열

■ 2차원 배열의 1차원 배열 취급



입출력 결과

```
1 2 3
4 5 6
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int* p = (int*) &a; // T(&a) == int[3][2]*
    int i;
    int num = sizeof(a)/sizeof(a[0][0]);
    int row = sizeof(a)/sizeof(a[0]);
    int col = sizeof(a[0])/sizeof(a[0][0]);

    for(i=0; i<num; ++i) {
        printf("%d ", p[i]);
        if( (i+1)%col==0 ) printf("\n");
    }
    printf("\n");
    return 0;
}
```

다차원 배열

■ 3차원 배열

```
T a[N1][N2][N3];
```

```
T(a) == T[N3][N2][N1]
```

입출력 결과

```
1 2 3 4
2 3 4 5
3 4 5 6
11 22 33 44
22 33 44 55
33 44 55 66
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int a[2][3][4] = {
```

```
        {{ 1, 2, 3, 4},
```

```
         { 2, 3, 4, 5},
```

```
         { 3, 4, 5, 6}},
```

```
        {{11,22,33,44},
```

```
         {22,33,44,55},
```

```
         {33,44,55,66}}
```

```
    }; // T(a) == int[4][3][2]
```

```
    int i1,i2,i3;
```

```
    for(i1=0;i1<2;++i1)
```

```
        for(i2=0;i2<3;++i2) {
```

```
            for(i3=0;i3<4;++i3)
```

```
                printf("%d ",
```

```
                    a[i1][i2][i3]);
```

```
                printf("\n");
```

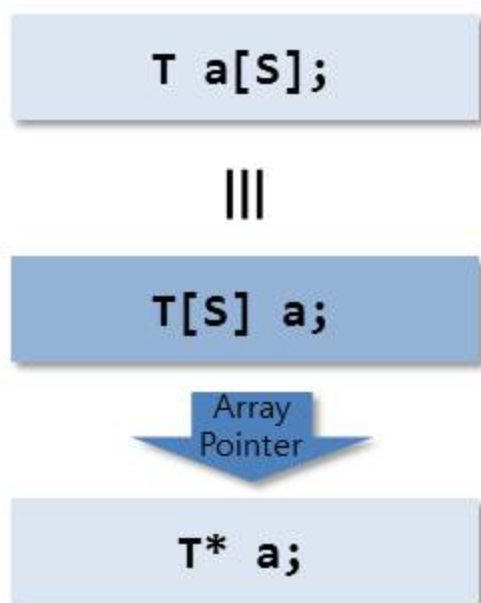
```
            }
```

```
        return 0;
```

```
}
```


다차원 배열

■ 다차원 배열 요소 포인터



```
#include <stdio.h>

int main(void)
{
    double a1[2];           // double[2]
    double a2[3][2];        // double[2][3]
    double a3[4][3][2];     // double[2][3][4]
    double a4[5][4][3][2];  // double[2][3][4][5]

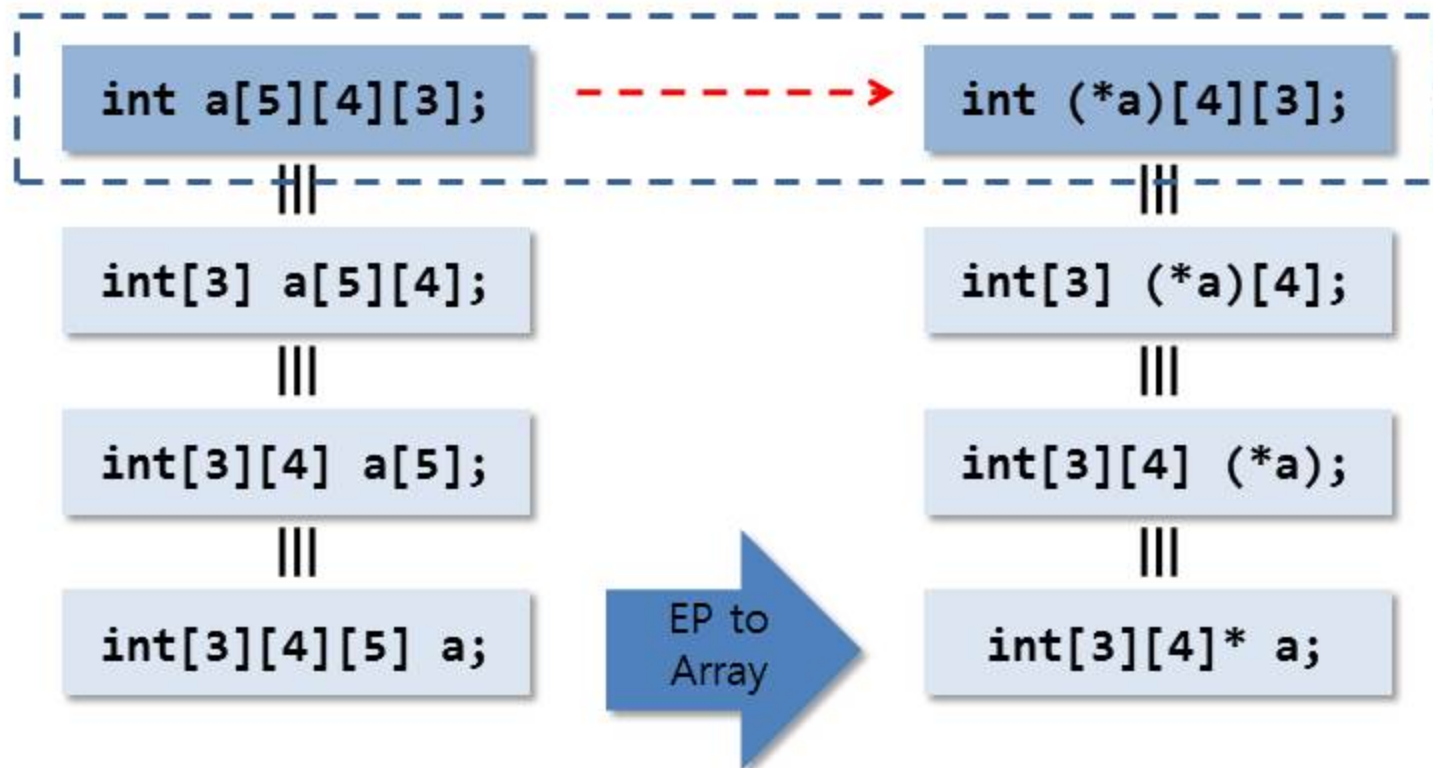
    double* p1;             // double*
    double (*p2)[2];        // double[2]*
    double (*p3)[3][2];     // double[2][3]*
    double (*p4)[4][3][2];  // double[2][3][4]*

    p1=a1; // double[2] --> double*
    p2=a2; // double[2][3] --> double[2]*
    p3=a3; // double[2][3][4] --> double[2][3]*
    p4=a4; // double[2][3][4][5] --> double[2][3][4]*

    return 0;
}
```


다차원 배열

3차원 배열의 배열 요소 포인터 전환 과정

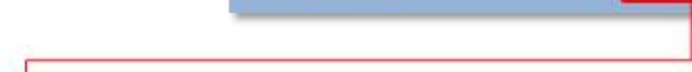


다차원 배열

- 배열 참조 vs. 포인터 참조
 - 배열 참조의 원리
 - 배열 참조는 실제 포인터 참조로 전환된다.
 - 곱셈 연산 포함

```
T a[S];
```

```
a[i] == *(a+i)
```



```
a+i ≡ address(a) + sizeof(T) * i
```

다차원 배열

- 배열 참조 vs. 포인터 참조
 - 포인터 참조에서의 타입 변화

a1	: double[2]
a1+i	: double*
*(a1+i)	: double

a2	: double[2][3]
a2+j	: double[2]*
*(a2+j)	: double[2]
(a2+j)+i	: double
((a2+j)+i)	: double

```
#include <stdio.h>

int main(void)
{
    double a1[2];
    double a2[3][2];
    double a3[4][3][2];
    int i=1,j=2,k=3;

    if(a1[i]==*(a1+i))
        printf("equal\n");
    if(a2[j][i]==*(*(a2+j)+i))
        printf("equal\n");
    if(a3[k][j][i]==*(*(*(a3+k)+j)+i))
        printf("equal\n");
    return 0;
}
```

다차원 배열

- 배열 참조 vs. 포인터 참조
 - 포인터 참조에서의 타입 변화

a3	: double[2][3][4]
a3+k	: double[2][3]*
*(a3+k)	: double[2][3]
(a3+k)+j	: double[2]
((a3+k)+j)	: double[2]
((a3+k)+j)+i	: double*
((*(a3+k)+j)+i)	: double

```
#include <stdio.h>

int main(void)
{
    double a1[2];
    double a2[3][2];
    double a3[4][3][2];
    int i=1,j=2,k=3;

    if(a1[i]==*(a1+i))
        printf("equal\n");
    if(a2[j][i]==*(*(a2+j)+i))
        printf("equal\n");
    if(a3[k][j][i]==*(*(*(a3+k)+j)+i))
        printf("equal\n");
    return 0;
}
```

Macro Function


■ 특징

- 함수를 호출한 부분에 코드를 생성한다.

```
#include <stdio.h>

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

int main(void)
{
    printf("%d\n", MAX(1,2) );
    return 0;
}
```



```
printf("%d\n", ( 1>2 ? 1 : 2 ) );
```

Macro Function

■ 주의 점

- 계산된 값이 인자로 전달되는 것이 아니라 식 자체가 전달된다.

```
#include <stdio.h>

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

int main(void)
{
    int a=1,b=0;

    printf("%d\n", MAX(++a,b) );
    return 0;
}
```



```
printf("%d\n", ( ++a>b ? ++a : b ) );
```



Macro Function

주의 점

```
#include <stdio.h>

#define ABS(x)    x>0 ? x : -x

int main(void)
{
    printf("%d\n", ABS(-3) );
    return 0;
}
```



```
printf("%d\n", -3>0 ? -3 : --3 );
```

해결 방법

```
#include <stdio.h>

#define ABS(x)    x>0 ? x : - x

int main(void)
{
    printf("%d\n", ABS(-3) );
    return 0;
}
```

```
#include <stdio.h>

#define ABS(x)    (x)>0 ? (x) : -(x)

int main(void)
{
    printf("%d\n", ABS(-3) );
    return 0;
}
```


Macro Function

주의 점

```
#include <stdio.h>

#define ABS(x)  (x)>0 ? (x) : -(x)

int main(void)
{
    printf("%d\n", ABS(3)+5 );
    return 0;
}
```



```
printf("%d\n", (3)>0 ? (3) : -(3)+5 );
```

해결 방법

```
#include <stdio.h>

#define ABS(x)  ((x)>0 ? (x) : -(x))

int main(void)
{
    printf("%d\n", ABS(3)+5 );
    return 0;
}
```

Switch

■ switch-statement

switch statement:

switch (*expression*)
labeled-statement

labeled-statement:

case *constant-expression* :
statement
default : *statement*

```
#include <stdio.h>

int main(void)
{
    int a;

    scanf(" %d",&a);
    switch( a ) {
        case 1:
            printf("a==one\n");
            break;
        case 2:
            printf("a==two\n");
            break;
        case 3:
            printf("a==three\n");
            break;
        case 4:
            printf("a==four\n");
            break;
        case 5:
            printf("a==five\n");
            break;
        default:
            printf("a==other\n");
    }
    return 0;
}
```

Switch

Switch 문

```
#include <stdio.h>

int main(void)
{
    int a;

    scanf(" %d",&a);
    switch( a ) {
        case 1:
            printf("a==one\n");
            break;
        case 2:
            printf("a==two\n");
            break;
        default:
            printf("a==other\n");
    }
    return 0;
}
```

연속 if문

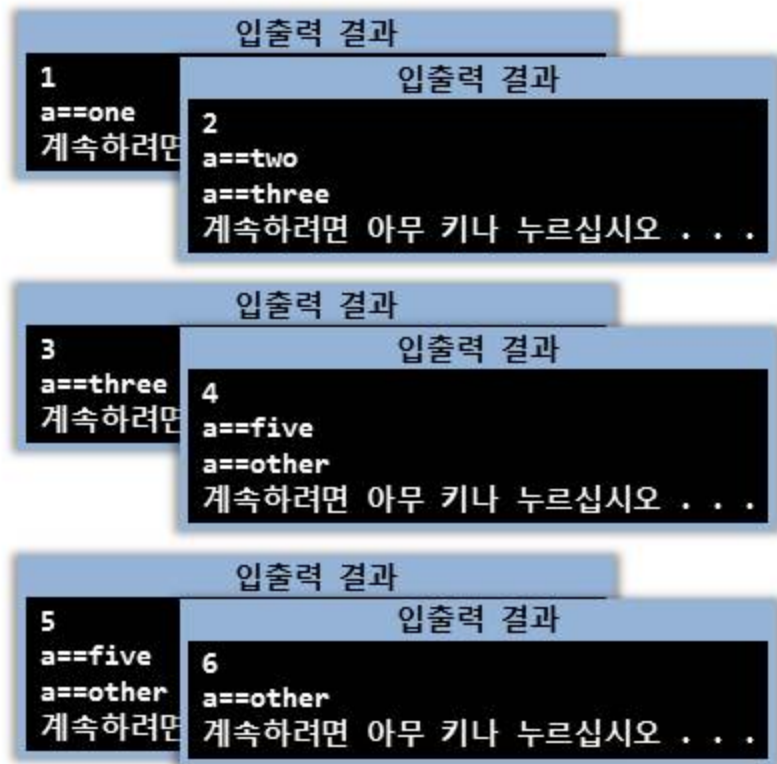
```
#include <stdio.h>

int main(void)
{
    int a;

    scanf(" %d",&a);
    if( a==1 )
        printf("a==one\n");
    else if ( a==2 )
        printf("a==two\n");
    else
        printf("a==other\n");
    return 0;
}
```

Switch

■ Break 문의 역할



```
#include <stdio.h>

int main(void)
{
    int a;

    scanf("%d",&a);
    switch( a ) {
        case 1:
            printf("a==one\n");
            break;
        case 2:
            printf("a==two\n");
        case 3:
            printf("a==three\n");
            break;
        case 4:
        case 5:
            printf("a==five\n");
        default:
            printf("a==other\n");
    }
    return 0;
}
```

Switch

- Constant expression
 - 컴파일 타임에 그 값이 계산될 수 있는 수식
- **case** *constant-expression*:
 - case 뒤의 수식은 반드시 constant expression 이어야 한다.

```
#include <stdio.h>

#define FOUR    4

int main(void)
{
    int a, two=2;

    scanf(" %d", &a);
    switch( a ) {
        case 1: // ok
            printf("a==one\n");
            break;
        case two: // error: not const expression
            printf("a==two\n");
            break;
        case 1+2: // ok: const expression
            printf("a==three\n");
            break;
        case FOUR: // ok
            printf("a==four\n");
            break;
        default:
            printf("a==other\n");
    }

    return 0;
}
```