

**Robotics**



대경혁신인재양성프로젝트  
**HuStar**

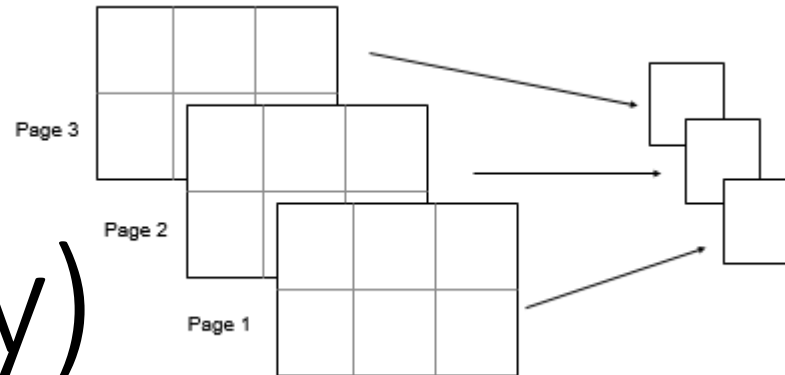
# C프로그래밍3



대경혁신인재양성프로젝트

# HuStar

## 배열(array)

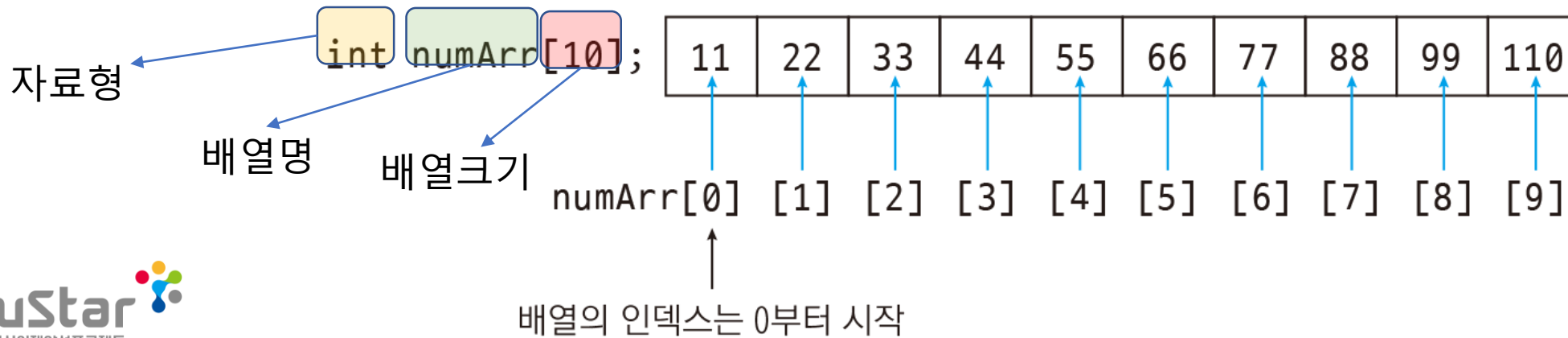


# 배열(array)

- 배열

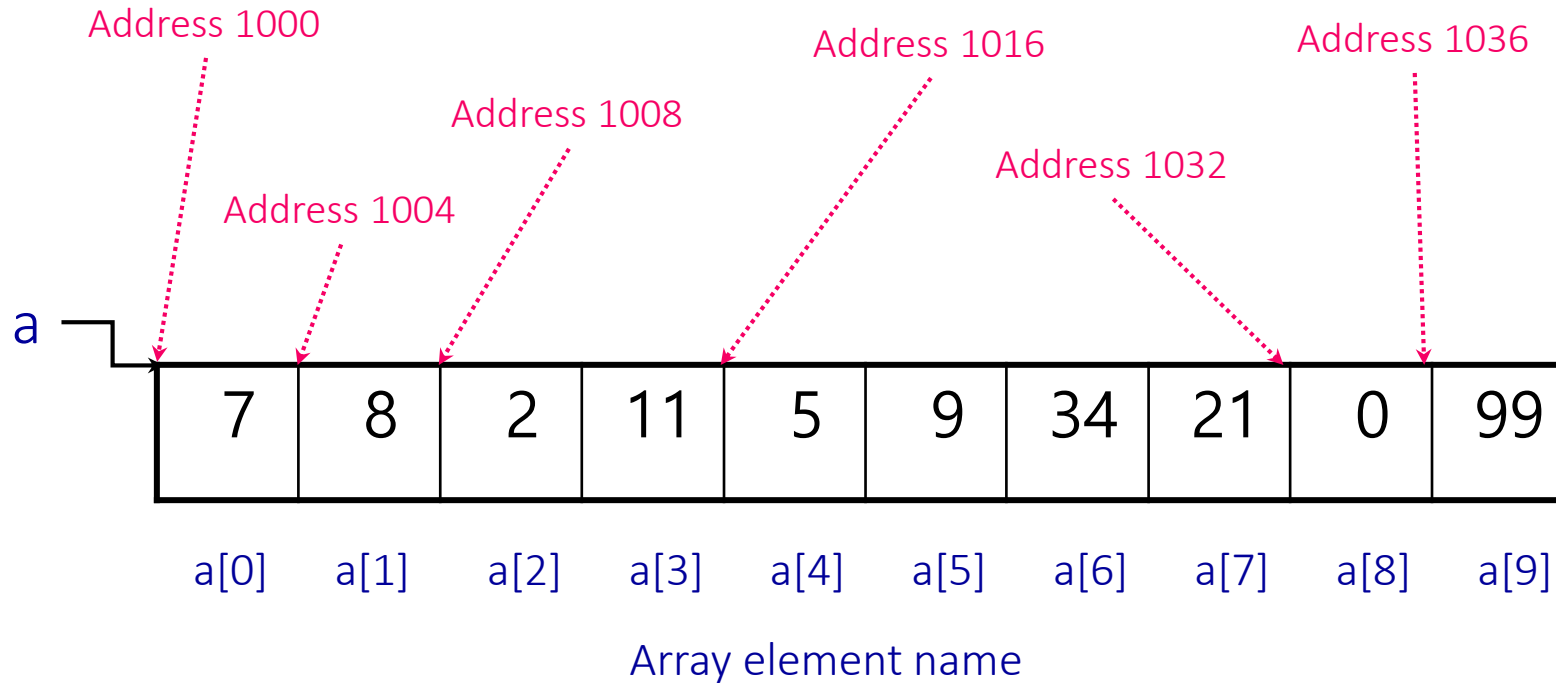
- 같은 자료형을 가진 연속된 변수
- 다수의 데이터를 저장하고 처리에 적합한 구조
- 배열 선언
  - 자료형: 배열에 저장할 수 있는 데이터의 형식
  - 배열명: 배열에 접근할 때 사용되는 이름
  - 배열크기: 저장할 수 있는 데이터의 수
- 데이터에 접근 하기 위한 index 필요
  - Index는 0부터 시작

```
int arr[10];
```



# 배열의 주소

- `int a[10];` 으로 선언된 배열의 구조와 인덱스



# 다양한 배열의 선언

- 상수를 이용한 선언

```
int arr[10];
```

```
//(c99)부터 지원, 많은 컴파일러에서 사용가능  
const int arraySize = 10;  
int a[arraySize];
```

- #define을 이용한 배열 선언

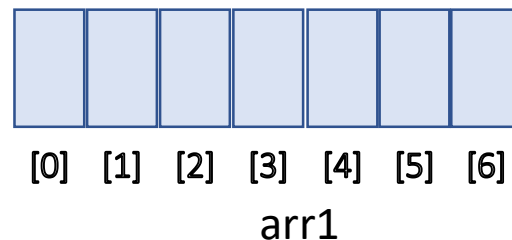
```
#define ArraySize 10  
int a[ArraySize];
```

- 변수를 이용한 선언

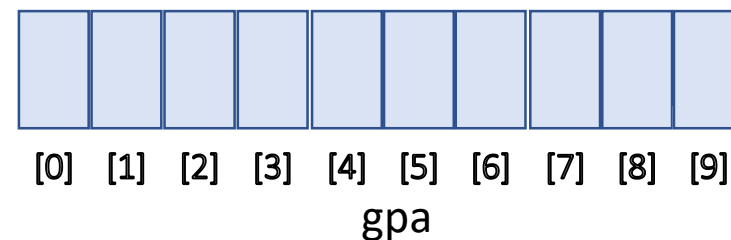
```
//(C99)부터 지원, 일부 컴파일러에서 사용가능  
int arraySize = 10;  
int a[arraySize];
```

# 다양한 배열의 선언

```
int arr1[7]; //크기가 7인 정수형 1차원 배열 arr1
```



```
float gpa[10]; //크기가 10인 float형 1차원 배열 gpa
```



```
char name[40]; //크기가 40인 char형 1차원 배열 name
```

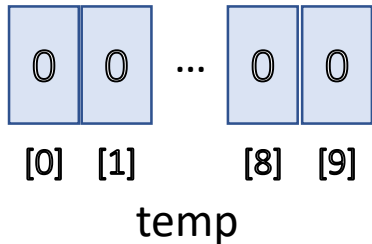


# 배열의 초기화와 데이터 입력

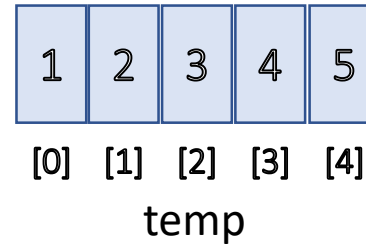
## • 초기화

//모든 요소를 0으로 초기화  
//배열 크기를 const int나 변수로 지정하면  
초기화는 안됨

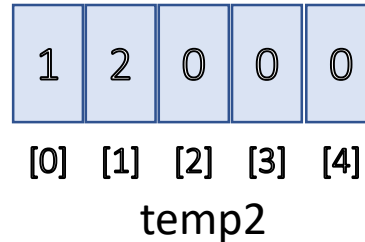
```
int temp[10]={0,};  
int temp1[10]={0};
```



//컴파일러에서 자동으로 배열의 크기가  
입력되고 각 요소들 입력된 값으로 초기화  
`int temp[]={1,2,3,4,5};`



//3,4,5번째 요소는 0으로 채워짐  
`int temp2[5]={1,2}`



# 배열의 초기화와 데이터 입력

- 배열의 각요소는 하나의 변수처럼 사용할 수 있음

```
int a[10] = {0};  
  
a[0] = 1 ;           //배열 a의 첫번째에 1을 입력  
printf(“%d\n”, a[5]) ;  
++a[3] ;             //a[3]의 값을 1 증가
```

- FOR문을 이용한 배열 데이터 입력

```
for (i = 0; i < N ; i++)  
    a[i] = 0 ;           /* clears array a */  
  
for (i = 0; i < N ; i++)  
    scanf(“%d”, &a[i]) ; /* read data into array a */
```



# 배열의 초기화와 데이터 입력

- 배열 index는 변수 또는 수식으로 표현 가능

```
i=1; j=2;
```

```
a[i + j * 3] = 0 ;
```

```
//same as a[7] = 0;
```

```
a[i++] = 0 ;
```

```
//same as a[1] = 0; i=2;
```

# 배열의 초기화와 데이터 입력 - 실습

```
#include <stdio.h>

int main(void)
{
    int arr1[5]={1, 2, 3, 4, 5};
    int arr2[]={1, 2, 3, 4, 5, 6, 7};
    int arr3[5]={1, 2};
    int ar1Len, ar2Len, ar3Len, i;

    printf("배열 arr1의 크기: %d \n", sizeof(arr1));
    printf("배열 arr2의 크기: %d \n", sizeof(arr2));
    printf("배열 arr3의 크기: %d \n", sizeof(arr3));

    ar1Len = sizeof(arr1) / sizeof(int);    // 배열 arr1의 길이 계산
    ar2Len = sizeof(arr2) / sizeof(int);    // 배열 arr2의 길이 계산
    ar3Len = sizeof(arr3) / sizeof(int);    // 배열 arr3의 길이 계산

    for(i=0; i<ar1Len; i++)
        printf("%d ", arr1[i]);
    printf("\n");

    for(i=0; i<ar2Len; i++)
        printf("%d ", arr2[i]);
    printf("\n");

    for(i=0; i<ar3Len; i++)
        printf("%d ", arr3[i]);
    printf("\n");
    return 0;
}
```

```
배열 arr1의 크기: 20
배열 arr2의 크기: 28
배열 arr3의 크기: 20
1 2 3 4 5
1 2 3 4 5 6 7
1 2 0 0 0
```

# 배열 실습1

- 길이가 7인 int형 배열 선언, 7개의 정수 입력
  - 입력된 정수에서 최댓값 찾기
  - 입력된 정수에서 최솟값 찾기
  - 입력된 정수의 합

```
input integer 1:-10
input integer 2:-1
input integer 3:0
input integer 4:-1
input integer 5:10
input integer 6:100
input integer 7:-90
max:100, min:-90, sum:8
```

# 배열 실습2

- 정수형 socre[10]를 선언하고 평균 계산
  - 사용자는 최대 10개 까지의 값을 입력 할 수 있음
  - 사용자가 0을 입력하면 입력을 중단하고 입력한 점수의 평균을 출력
  - 평균은 소수점 3자리까지 출력

```
input socre:77
input socre:88
input socre:99
input socre:45
input socre:0
sum=309 cnt=4 average: 77.250
```

# 배열의 전달

- 함수로 배열을 전달하는 방법
  - Call by value
    - 함수로 배열이 가지고 있는 데이터 중 하나의 데이터를 전달 하는 방법
    - 데이터를 복사하여 전달
    - 함수에서 데이터를 수정하여도 원본 데이터는 변경되지 않음
  - Call by reference
    - 함수로 데이터가 저장된 위치를 전달
    - 값을 전달하는 것이
    - 아니라 값이 저장된 위치를 전달
    - 함수에서 원본 데이터에 직접 접근해서 데이터를 읽어오는 방식이기 때문에 데이터를 변경할 경우 원본 데이터도 변경됨

# 배열의 전달

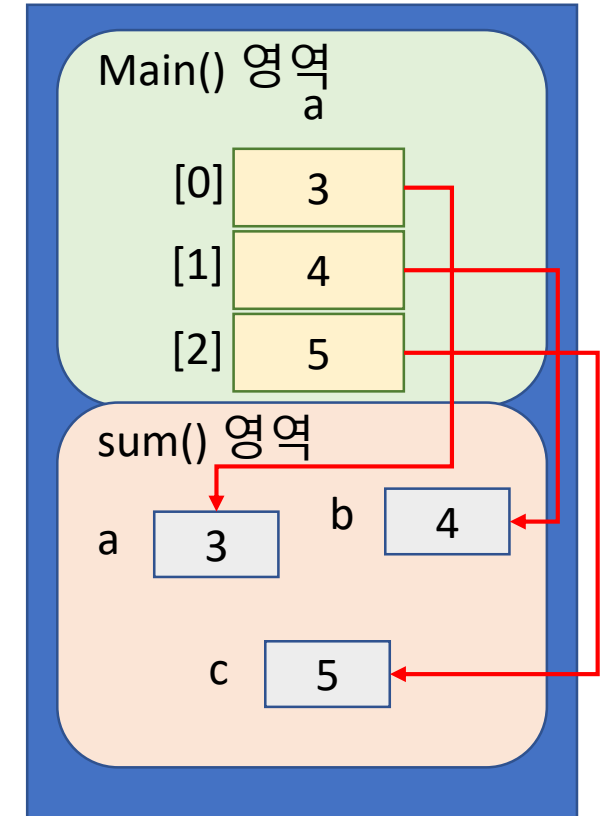
- Call by value

```
#include <stdio.h>
int sum(int a, int b, int c);

int main(){
    int a[]={3,4,5};
    printf("sum: %d", sum(a[0],a[1],a[2]));
    return 0;
}

int sum(int a, int b, int c){
    return a+b+c;
}
```

메모리



# 배열의 전달

- Call by reference

```
#include <stdio.h>
int sum(int a[]);

int main(){

    int a[]={3,4,5};

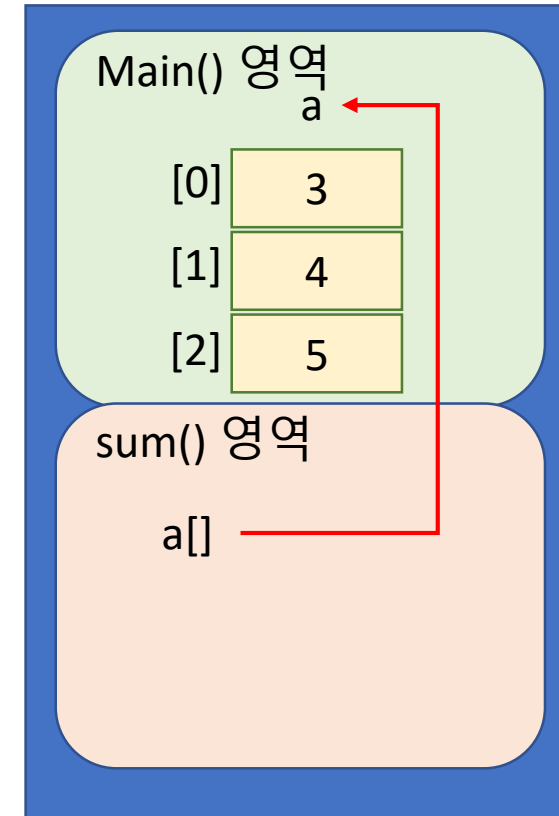
    printf("sum: %d", sum(a));

    return 0;

}

int sum(int a[]){
    return a[0]+a[1]+a[2];
}
```

메모리



# 배열의 전달

## • Call by value vs Call by reference 비교

```
#include <stdio.h>
int sumref(int a[]);
int sumval(int a, int b, int c);

int main(){

    int arr[]={3,4,5};
    int data1=3, data2=4, data3=5;

    printf("sumref: %d // ", sumref(arr));
    printf("main: %d\n",arr[0]+arr[1]+arr[2]);
    printf("sumval: %d // ", sumval(data1,data2,data3));
    printf("main: %d\n",data1+data2+data3);

    return 0;
}
```

```
int sumref(int a[]){
    a[0]=103;
    return a[0]+a[1]+a[2];
}

int sumval(int a, int b, int c){
    a=103;
    return a+b+c;
}
```

```
sumref: 112 // main: 112
sumval: 112 // main: 12
```



# 배열 전달 실습

- 정수형 배열을 선언하고 10개의 값을 입력 받음
- 출력함수에서 10개의 값을 출력

```
input integer:1
input integer:2
input integer:3
input integer:4
input integer:5
input integer:6
input integer:7
input integer:8
input integer:9
input integer:0
numbers in array:1 2 3 4 5 6 7 8 9 0
```

# 다차원 배열(multi array)

- 일차원 배열(one-dimensional array)

- 하나의 행으로 구성된 배열

```
char arrch[7];
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
array	A	B	C	D	E	F	G

- 이차원 배열(Tow-dimensional array)

- 행과 열로 구성된 배열

```
int arrint[3][4];
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

# 다차원 배열(multi array)

- 이차원 배열의 구성

`int table[2][5];`

행위치

열위치

00	01	02	03	04
10	11	12	13	14

사람의 생각에서 2차원 배열

row 0					row 1				
00	01	02	03	04	10	11	12	13	14
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]

메모리에서 2차원 배열

# 이차원 배열 초기화

- 이차원 배열의 초기화
  - 일차원 배열 초기화 방법 사용 가능 ex) `int table[5][4]={0}`

나열 `int table[5][4] = {0, 1, 2, 3, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33, 40, 41, 42, 43} ;`

행 구별 `int table[5][4] =  
{  
 {0, 1, 2, 3},  
 {10, 11, 12, 13},  
 {20, 21, 22, 23},  
 {30, 31, 32, 33},  
 {40, 41, 42, 43}  
} ;`

0	0	1	2	3
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33
4	40	41	42	43
	0	1	2	3

# 이차원 배열 예제

```
#include <stdio.h>
#define M 3      /* number of rows */
#define N 4      /* number of columns */

int main( ) {
    int a[M][N], i, j, sum = 0;

    for ( i = 0; i < M; ++i )
        for ( j = 0; j < N; ++j ) {
            a[i][j] = i + j;
            sum += a[i][j];
        }

    for ( i = 0; i < M; ++i ) {
        for ( j = 0; j < N; ++j )
            printf("a[%d][%d] = %d ", i, j, a[i][j] );
        printf("\n");
    }
    printf("\nsum = %d\n", sum);
}
```

```
a[0][0] = 0 a[0][1] = 1 a[0][2] = 2 a[0][3] = 3
a[1][0] = 1 a[1][1] = 2 a[1][2] = 3 a[1][3] = 4
a[2][0] = 2 a[2][1] = 3 a[2][2] = 4 a[2][3] = 5

sum = 30
```

# 이차원 배열 실습1

- 정수형 이차원 배열을 [3][4]의 크기로 만들고 각 열의 합을 계산하고 결과를 출력 하시오.
  - 각 열의 합은 일차원 배열 sum[4]에 저장 할 것
  - 배열의 숫자는 임의로 초기화에서 입력할 것

1	2	3	4
5	6	7	8
3	7	8	9
-----			
9	15	18	21

# 이차원 배열의 전달

- Call by value

- 배열 하나의 요소만 전달하는 방법
- 배열의 값을 복사하여 전달

`printf(table[0][0]);`

- Call by reference

- 2차원 배열의 한 행만 전달

- 해당 행의 시작 주소를 전달하는 방식
- 함수에서 데이터 수정 시 원본 데이터 변경

`sum(table[1]);`

- 2차원 배열 전체를 전달

- 원본 데이터 변경
- 함수에서 데이터 수정 시 원본 데이터 변경
- 배열 전체를 전달할 때에 배열의 최대 열의 길이를 같이 전달

0	1	2	3
10	11	12	13
20	21	22	23
30	31	32	33
40	41	42	43

`sum(table[][MAX_col]);`

# 이차원 배열 하나의 행 전달

```
#include <stdio.h>
#define MAX_ROWS 3
#define MAX_COLS 4
void print_square (int x[ ] ) ;

int main() {
    int row ;
    int table[MAX_ROWS][MAX_COLS] = { {0, 1, 2, 3}, {10, 11, 12, 13},
                                         {20, 21, 22, 23}} ;

    for (row = 0 ; row < MAX_ROWS ; row++)
        print_square( table[row] ) ;

    return 0;
}

void print_square ( int x[ ] ) {
    int col ;
    for (col = 0 ; col < MAX_COLS ; col++)
        printf("%6d", x[col] * x[col] ) ;
    printf("\n") ;}
```

0	1	4	9
100	121	144	169
400	441	484	529

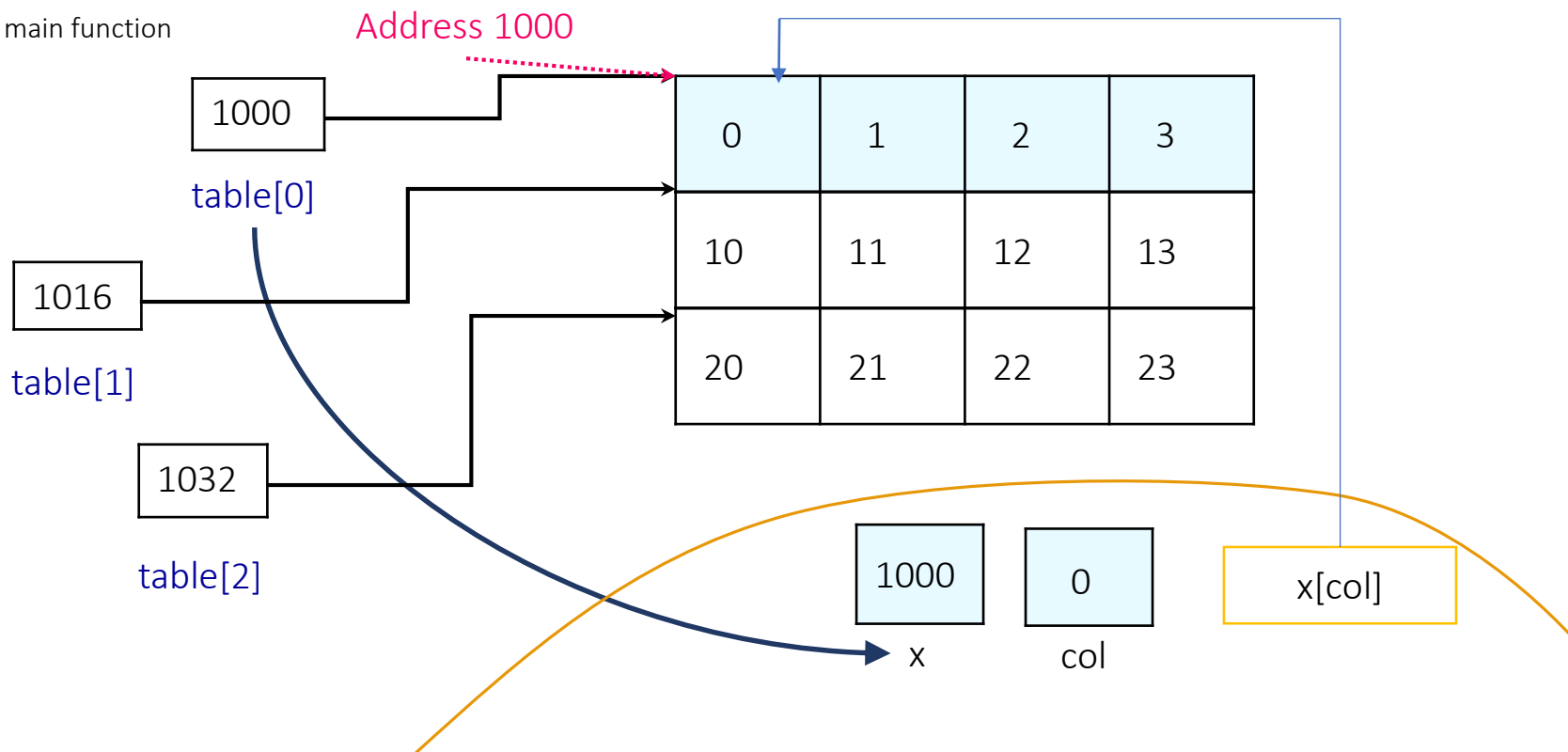


# 이차원 배열 하나의 행 전달

```
print_square( table[row] );
```

```
void print_square ( int x[ ] )
```

In main function



# 이차원 배열 전체 전달

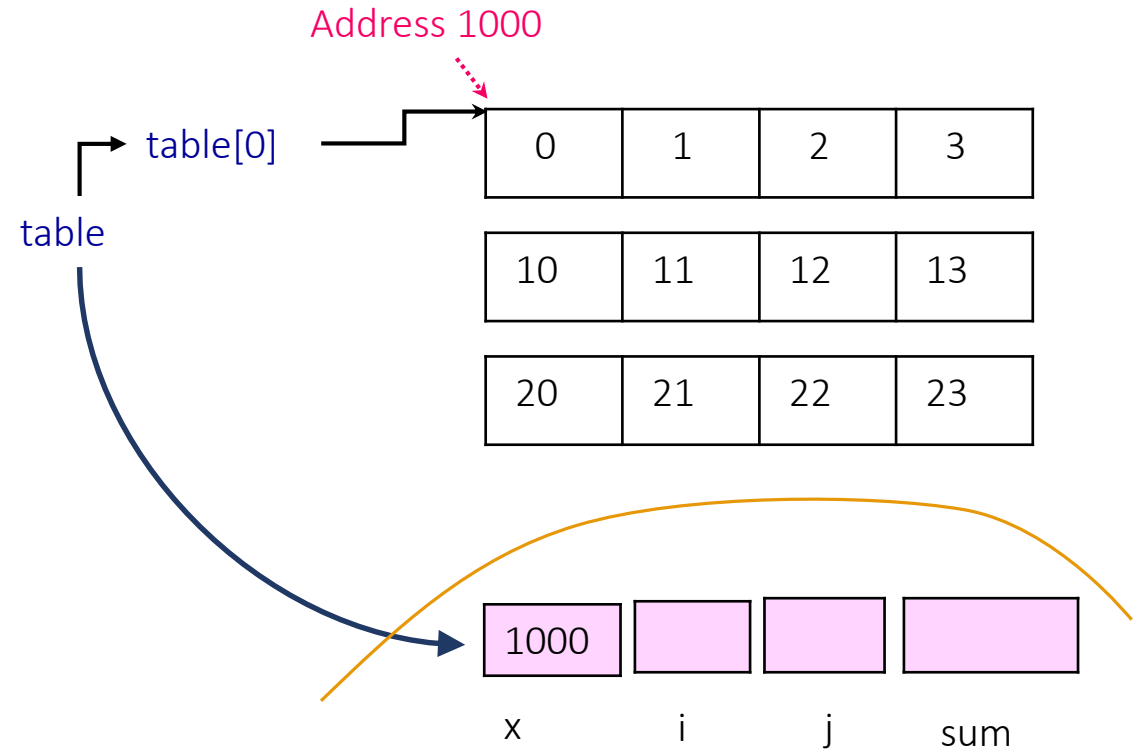
```
#include <stdio.h>
#define MAX_ROWS 3
#define MAX_COLS 4

double average (int x[ ][MAX_COLS] );

int main() {
    double avg ;
    int table[MAX_ROWS][MAX_COLS]
        = { {0, 1, 2, 3}, {10, 11, 12, 13},
            {20, 21, 22, 23} } ;

    avg = average( table );
    printf("%1f", avg);

    return 0;
}
```



```
double average( int x[ ][MAX_COLS] ) {
    int i, j ;
    double sum = 0 ;

    for ( i = 0 ; i < MAX_ROWS ; i++)
        for ( j = 0 ; j < MAX_COLS ; j++)
            sum += x[i][j] ;

    return (sum / (MAX_ROWS * MAX_COLS));
}
```

11.500000

# 이차원 배열 전체 전달 예제

```
#include <stdio.h>
void print_a(int[][3]);
```

```
int main() {
    int array1[][3]={1,2,3,4,5,6};
    int array2[2][3] = {7,8};
    int array3[2][3]={9},{0}};
```

```
    print_a(array1);
    print_a(array2);
    print_a(array3);
```

```
    return 0;
```

```
}
```

```
void print_a(int a[][3])
{
    int i, j;
    for( i=0; i<=1; i++){
        for(j=0;j<=2; j++)
            printf("%5d ", a[i][j]);
        printf("\n");
    }
}
```

배열을 전달받는 함수에서 열의 길이가  
정해져 있어야 함  
배열을 전달하는 경우에는 배열의  
이름만 전달하여도 오류가 없음

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

# 이차원 배열 실습

- 정수형 이차원 배열 [6][6]을 생성하고 아래와 같이 값을 입력하는 프로그램
  - 행과 열이 같은 경우 0
  - 행의 값이 열보다 큰 경우 -1
  - 행의 값이 열보다 작은 경우 1

0	1	1	1	1	1
-1	0	1	1	1	1
-1	-1	0	1	1	1
-1	-1	-1	0	1	1
-1	-1	-1	-1	0	1
-1	-1	-1	-1	-1	0

# 입력 받은 값으로 배열 생성

```
#include <stdio.h>

int main() {
    int array_s, row, col ;

    printf("Enter size of 1-D array: ");
    scanf("%d", &array_s);

    printf("Enter row and column sizes of 2-D array: ");
    scanf("%d %d", &row, &col);

    int array_1[array_s];
    int array_2[row][col];

    printf("1D array size : %d \n", sizeof(array_1));
    printf("2D array size : %d \n", sizeof(array_2));
    printf("2D row size : %d \n", sizeof(array_2[0]));
}
```

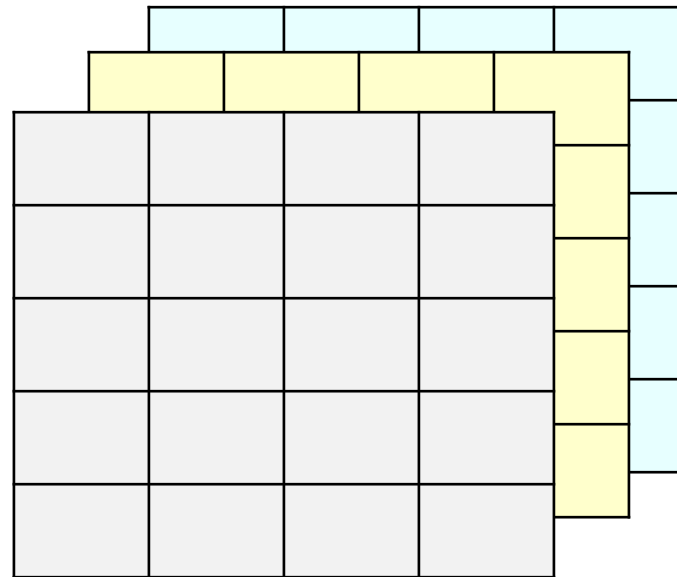
```
Enter size of 1-D array: 3
Enter row and column sizes of 2-D array: 3 3
1D array size : 12
2D array size : 36
2D row size : 12
```

# 3차원 배열

- 3차원으로 구성된 배열
  - `int array[1][2][3]`
  - 2차원 배열을 배열로 만든 것

```
int table(3)(5)(4) = {0};
```

second-D (rows) - 5



first-D (planes) - 3

third-D (columns) - 4

# 3차원 배열 초기화와 전달

- 3차원 배열의 전달

```
int a[ 2 ][ 2 ][ 3 ] = { { {1,1,0}, {2,0,0} }, { {3,0,0}, {4,4,0} } };
```

```
int a[ ][ 2 ][ 3 ] = { { {1, 1}, {2} }, { {3}, {4, 4} } };
```

```
printf("%d\n", sum(a));
```

```
int sum( int a[ ][ 5 ][ 4 ] )  
{  
    int i, j, k, sum = 0;  
  
    for ( i = 0; i < 7; ++i )  
        for ( j = 0; j < 5; ++j )  
            for ( k = 0; k < 4; ++k )  
                sum += a[ i ][ j ][ k ];  
  
    return sum;  
}
```

3차원 배열을 전달하기  
위해서는 행과 열을 전달  
받는 함수가 알고 있어야 함

**Robotics**



대경혁신인재양성프로젝트



**HuStar**

문자 다루기



# character, string 그리고 NULL

- 문자: 'a'
  - 문자형 변수에는 하나의 문자만 저장할 수 있음
  - 문자는 "(작은따옴표)로 구분
  - 그렇다면, hello와 같은 단어를 저장은?

```
char a='k';
```

- 문자열: "hello"
  - 문자 여러 개가 합쳐진 데이터 (단어, 문장)
  - 문자열을 저장하기 위해서는 char 배열을 사용하여 저장
  - 문자는 ""(큰따옴표)로 구분

```
char hu[]="hustar robot!";
```

# character, string 그리고 NULL

```
char hu[]="hustar robot!";
```

 배열의 크기를 출력 해보자!

- 문자열이 저장된 배열의 크기는?

```
#include <stdio.h>

int main(){
    char str[]="hustar robot!";
    char a='a';

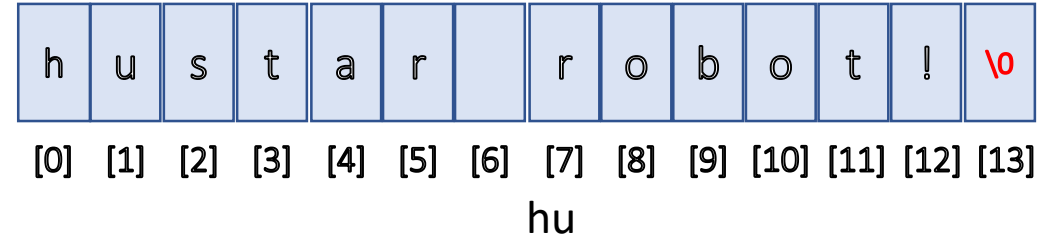
    printf("size of str:%d %d",sizeof(str),sizeof(a));

    return 0;
}
```

size of str:14 1

# character, string 그리고 NULL

```
char hu[]="hustar robot!";
```



- 문자열의 경우 마지막에 '\0'문자를 추가하여 string이 끝났음을 알림
  - '\0'=> NULL 이라고 읽음
  - 문자열의 경우 끝을 명시하기 위해 NULL을 이용
  - 문자형 배열 마지막
    - '\0'문자가 있으면 문자열
    - '\0'문자가 없으면 문자 배열

# NULL을 이용한 예시

```
#include <stdio.h>

int main(void)
{
    char str[50]="I like C programming";
    printf("string: %s \n", str);

    str[8]='\0';    // 9번째 요소에 널 문자 저장
    printf("string: %s \n", str);

    str[6]='\0';    // 7번째 요소에 널 문자 저장
    printf("string: %s \n", str);

    str[1]='\0';    // 2번째 요소에 널 문자 저장
    printf("string: %s \n", str);
    return 0;
}
```

```
string: I like C programming
string: I like C
string: I like
string: I
```

# 문자 다루기

- 문자입/출력을 위한 새로운 입출력 함수
  - getchar()
    - 키보드 입력(표준입력)에서 문자 하나를 읽어 들이는 함수
  - putchar()
    - 모니터 출력(표준출력)에서 문자 하나를 출력 하는 함수
- 두 함수는 모두 서식지정자(%c)가 필요 없음

```
char ch ;  
  
scanf("%c", &ch) ;  
printf("%c", ch) ;  
  
ch = getchar() ;  
putchar(ch) ;
```

# getchar(), putchar()를 이용한 입출력

- 문자 입/출력 예제

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

```
int main(){  
    int c ;
```

```
    while ( ( c = getchar() ) != EOF ) {  
        putchar(c) ;  
        putchar(c) ;  
    }  
    return 0;  
}
```

```
hustar!  
hhuussttaarr!!  
  
popstar  
ppooppssttaarr
```

^Z

EOF(End of File)의 의미이며  
위의 코드는 ctrl+z(^z)가 입력  
되면 프로그램이 종료된다.

# 특수 문자를 이용한 입력 종료

```
#include <stdio.h>
int main(){
    char ch;
    int len = 0;

    printf("Enter a message: ");
    ch = getchar() ;

    while( ch != '\n' ) {
        len++;
        ch = getchar() ;
    }
    printf("Your message was %d character(s) long \n", len);

    return 0;
}
```

\n(줄바꿈) 개행문자를 이용하여  
사용자가 엔터를 눌렀을 때  
프로그램이 종료되게 설정

```
Enter a message: hello hustar! good bye
Your message was 22 character(s) long
```

# 문자를 다루는 함수

- 문자를 다루기 위한 추가 함수 ctype.h에 정의 되어 있어 사용하기 위해서는 `#include <ctype.h>` 가 추가되어야 함

`#include <ctype.h>`

함수	설명(True, false를 반환)
isspace()	공백( )인지 확인하여 맞다면 true 아니면 false를 반환
isalnum()	알파벳이나 숫자인지 확인하는 함수
ispunct()	구획문자인지 확인하는 함수( 구획문자: 알파벳, 숫자가 아닌 문자)
isalpha()	알파벳인지 확인하는 함수
isupper()	대문자 알파벳인지 확인하는 함수
islower()	소문자 알파벳인지 확인하는 함수
isdigit()	'0'~'9'사이의 문자인지 확인하는 함수
toupper()	대문자로 변환하는 함수
tolower()	소문자로 변환하는 함수



# 문자 다루기- 실습1

- 입력문자에서 알파벳 문자가 나온 횟수를 확인하는 프로그램

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

int main() {
    int c, i, letter[26] = {0} ;

    printf("input:");
    while(( c = getchar() ) != EOF) {
        c=toupper(c);
        if ( isalpha(c) ) ++letter[c - 'A'];
    }

    for ( i = 0; i < 26; ++i) {
        if ( i % 6 == 0 ) printf("\n");
        printf("%4c:%3d", 'A' + i, letter[i]);
    }

    printf("\n\n");

    return 0;
}
```

input:asdf123ADF  
^Z

A:	2	B:	0	C:	0	D:	2	E:	0	F:	2
G:	0	H:	0	I:	0	J:	0	K:	0	L:	0
M:	0	N:	0	O:	0	P:	0	Q:	0	R:	0
S:	1	T:	0	U:	0	V:	0	W:	0	X:	0
Y:	0	Z:	0								

# 문자 다루기- 실습2

- 문자 다루기- 실습1에서는 모든 알파벳을 대문자로 변경하여 입력된 문자의 횟수를 확인. 이를 소문자로 변경하여 확인하도록 수정하는 프로그램 작성

```
input:asdfZXCV  
^Z
```

a:	1	b:	0	c:	1	d:	1	e:	0	f:	1
g:	0	h:	0	i:	0	j:	0	k:	0	l:	0
m:	0	n:	0	o:	0	p:	0	q:	0	r:	0
s:	1	t:	0	u:	0	v:	1	w:	0	x:	1
y:	0	z:	1								

**Robotics**



대경혁신인재양성프로젝트



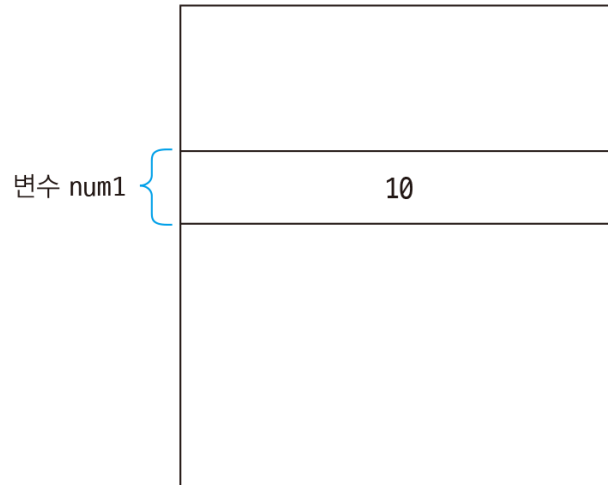
**HuStar**

포인터(pointer)

# 포인터(pointer)

```
int num1 = 10;
```

메모리



변수가 메모리에서  
생성되는 예제

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

```
int main()  
{
```

```
    int num1 = 10;
```

```
    printf("%p\n", &num1);    // 008AF7FC: num1의 메모리 주소를 출력  
                                // 컴퓨터마다, 실행할 때마다 달라짐
```

```
    return 0;
```

```
}
```

실행 결과

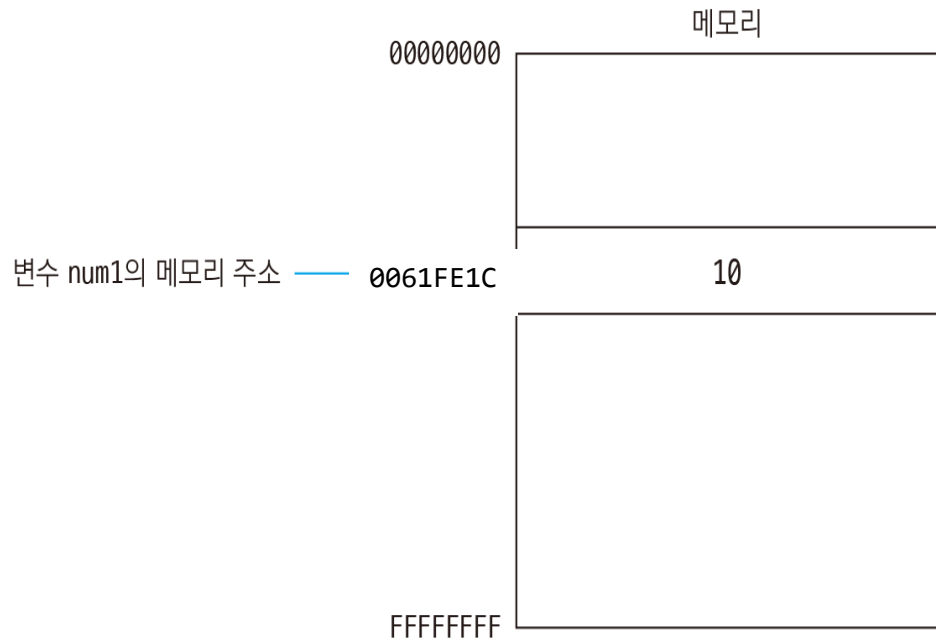
**0061FE1C** (메모리 주소. 컴퓨터마다, 실행할 때마다 달라짐)

- 메모리 주소는 008AF7FC와 같이 16진수 형태이며 printf에서 서식 지정자 %p를 사용하여 출력 (pointer의 약어로 p를 사용)

- 변수의 메모리 주소를 구할 때는 변수 앞에 & (주소 연산자)를 붙여서 나타냄

# 포인터(pointer)

## • 메모리 주소로 표현



시스템이 32비트인지 64비트인지에 따라 메모리 주소의 범위 변경됨

### • 32비트: 16진수 8자리

- 0x00000000 ~ 0xFFFFFFFF
- 예) 0x008AF7FC

### • 64비트: 16진수 16자리

- 0x0000000000000000 ~ 0xFFFFFFFFFFFFFFFF
- 예) 0x000000000008AF7FC
- 64비트 메모리 주소는 0x00000000`00000000처럼 8자리 씩 끊어서 `를 붙이는 경우도 있음

## 리눅스, OS X에서 메모리 주소

리눅스, OS X에서 서식 지정자 %p를 사용하면 메모리 주소 008AF7FC는 0x8af7fc와 같이 앞에 0x가 붙고, A~F는 소문자로 출력

# 포인터(pointer)

- 포인터를 사용하는 이유
  - 효율적인 대형 자료 구조 관리
    - 배열 or 구조체
  - 함수 사이의 데이터 공유
    - 포인터를 파라미터로 사용
  - 동적 메모리 할당 지원
    - 프로그램이 실행되는 동안에 메모리 할당이 가능함
  - Linked list 지원
    - Linked list는 데이터가 담긴 노드(메모리 공간)를 일렬로 연결되어 있는 자료구조
    - 노드 사이를 포인터를 이용하여 연결 할 수 있음



예제 단일 연결 리스트

# 포인터 변수 선언

## • 포인터 변수 선언 방법

`int *numPtr;`  
자료형      \*포인터이름

## • 포인터 변수 사용법

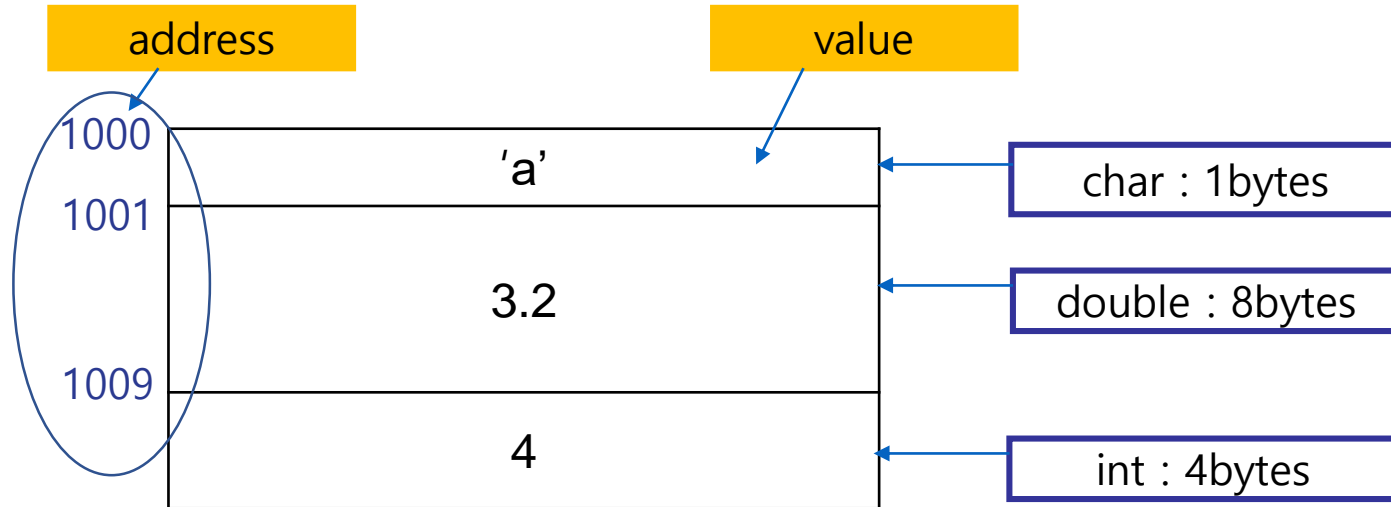
`numPtr = &num1;`  
포인터 변수명:      (&주소 연산)  
메모리 주소를 저장      &로 변수의 주소를 가져옴

```
#include <stdio.h>

int main()
{
    int *numPtr;    // 포인터 변수 선언
    int num1 = 10;  // int형 변수를 선언하고 10 저장

    numPtr = &num1; // num1의 메모리 주소를 포인터 변수에 저장

    return 0;
}
```



# 포인터 변수 선언

- 포인터 연산자

- 주소 연산자(Address Operator): “ & ”

- 변수의 주소를 되돌려주는 역할

- 역참조 연산자(Indirection Operator): “ \* ”

- 해당 주소에 있는 변수의 값을 되돌려 주는 역할

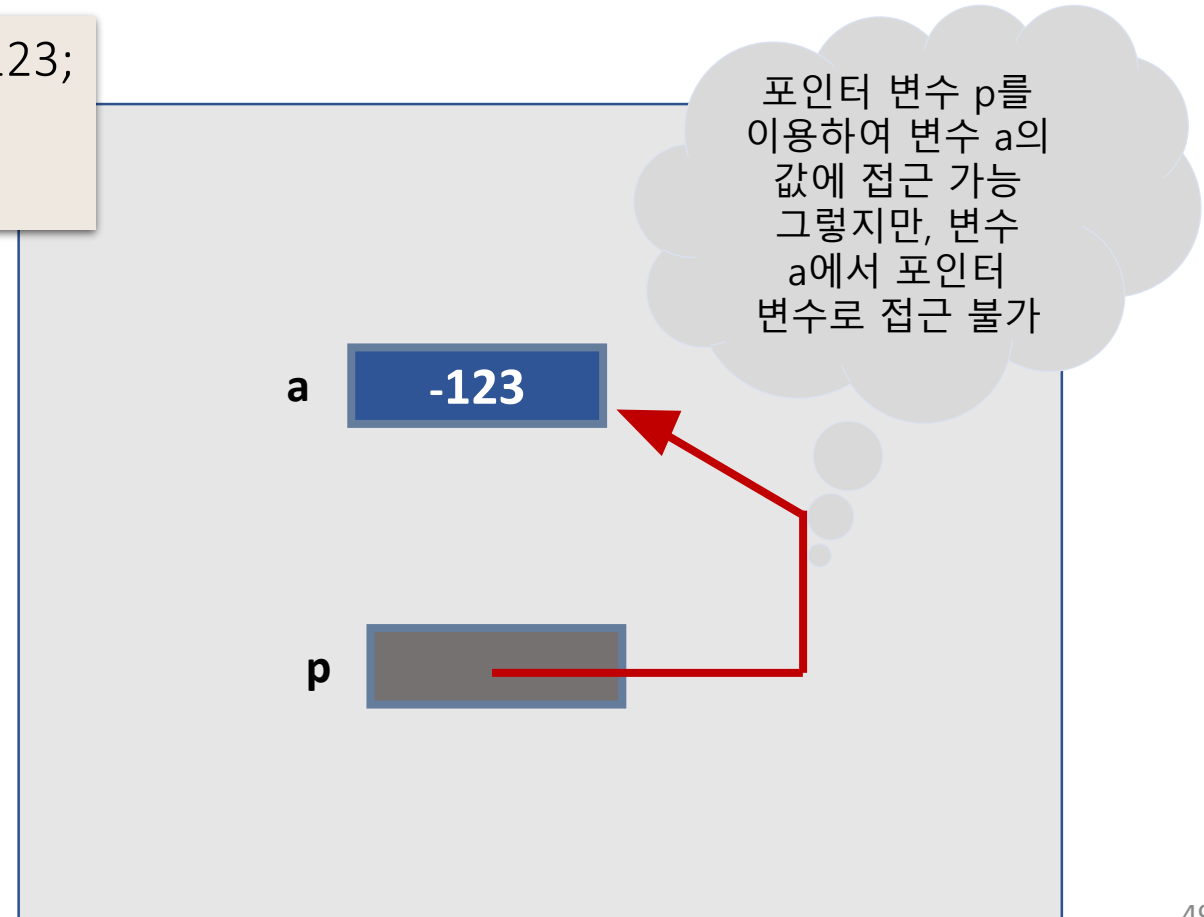
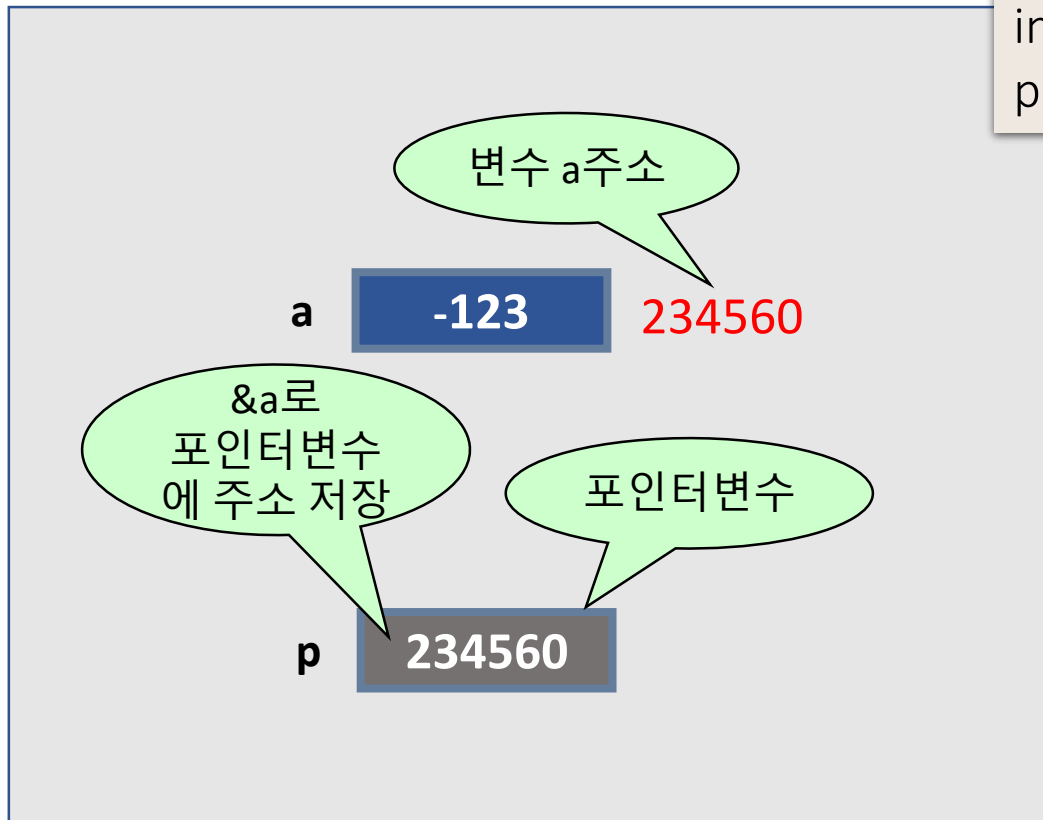
(Asterisk,  
애스터리스크)

```
int *p;      //선언 pointer variable 'p'
int i=3;
p = &i;      //i변수의 주소를 p에 할당: p 포인터 변수를 i를 가르킴
printf("%d", *p); //역참조(*) 연산에 의하여 포인터 변수 p가 가지고 있는 주소에 있는 변수 a의
값 3을 되돌려 받음
```



# 포인터 변수 선언

```
int a = -123;  
int* p;  
p = &a;
```



# 포인터 변수 선언

- 포인터 변수 선언 예제

```
int *p;           //정수형 포인터 변수 p를 선언
char *cp;         //Points only to char
float *fp;        //Points only to float
double *dp;       //Points only to double
```

```
int* p, q, r;     //int *p, q, r; 와 같은 의미를 가짐
                  //p:포인터 변수 q, r: 정수형 변수

int *p, *q, *r;   //p, q, r: 포인터 변수
```

```
int* numPtr;      // 자료형 쪽에 *을 붙임
int * numPtr;     // 자료형과 변수 가운데 *를 넣음
int *numPtr;      // 변수 쪽에 *을 붙임
//같은 의미를 가짐
```

# 포인터 예제

```
#include <stdio.h>

int main()
{
    int *numPtr;      // 포인터 변수 선언
    int num1 = 10;    // int형 변수를 선언하고 10 저장

    numPtr = &num1;   // num1의 메모리 주소를 포인터 변수에 저장

    printf("%p\n", numPtr);    // 포인터 변수 numPtr의 값 출력
                                // 컴퓨터마다, 실행할 때마다 달라짐
    printf("%p\n", &num1);    // 변수 num1의 메모리 주소 출력
                                // 컴퓨터마다, 실행할 때마다 달라짐

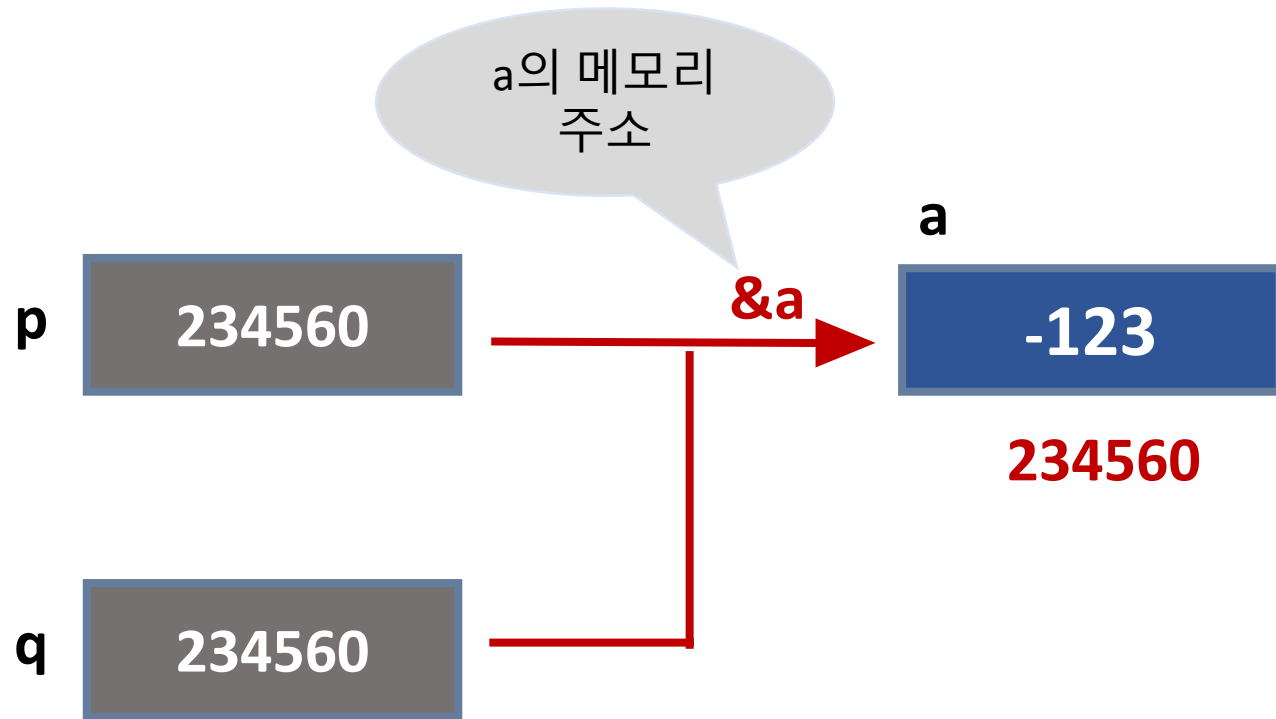
    return 0;
}
```

```
00000000000061FE14
00000000000061FE14
```

# 다중 포인터

- 하나의 변수에 여러 포인터 변수를 연결하여 사용할 수 있음

```
int a = -123;  
int* p = &a;  
int* q;  
q = p;
```



# 다중포인터 예제

```
#include <stdio.h>

int main()
{
    int *numPtr, *ptr2;
    int num1 = 10;

    numPtr = &num1;
    ptr2 = &num1;

    printf("%p : ", numPtr);
    printf("%d\n\n", *numPtr);

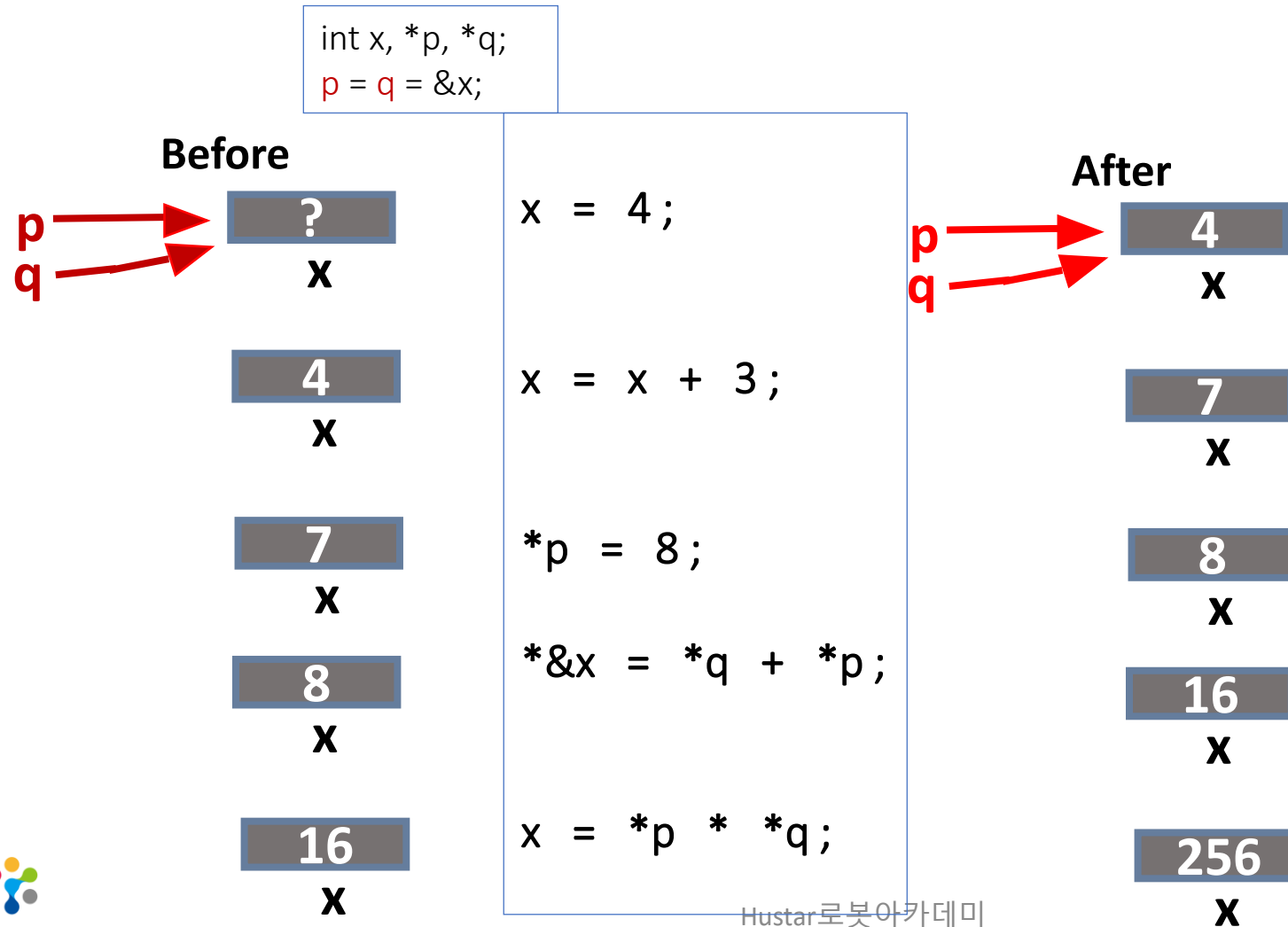
    printf("%p : ", ptr2);
    printf("%d\n\n", *ptr2);

    return 0;
}
```

000000000061FE0C : 10

000000000061FE0C : 10

# 포인터 연산자 연습



# 포인터 연산자 연습

- 포인터 연산자의 활용

```
int x, *p;  
x = 10;  
p = &x;
```

같은 의미

```
int x, *p = &x;  
*p = 10;
```

```
printf("%d", *p);
```

같은 의미

```
printf("%d", x);  
printf("%d", *&x);
```

# 포인터 연습

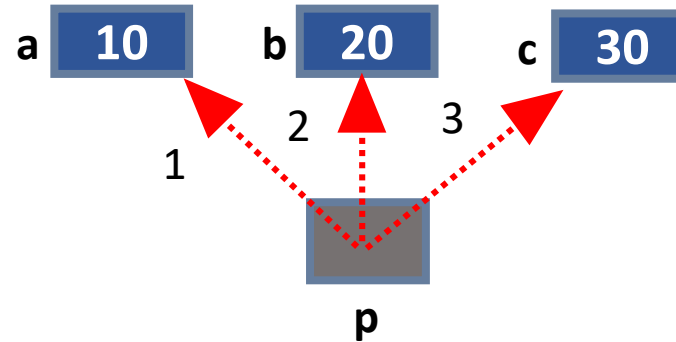
- 여러 변수와 포인터

```
int a = 10, b = 20, c = 30;  
int *p;
```

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

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

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



포인터 변수 p는 순차적으로 a, b, c  
변수의 메모리주소를 가짐  
(동시에 가지는 것이 아님)



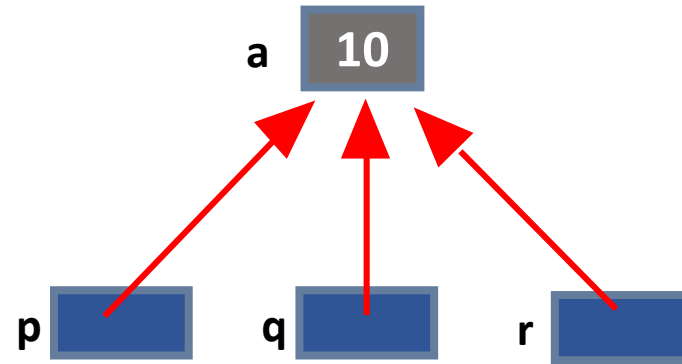
# 포인터 연습

- 여러 포인터 변수와 하나의 정수 변수

```
int a = 10;  
int *p, *q, *r;
```

```
p = q = r = &a;
```

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



# 포인터 연습

- 포인터 연산의 오류 예시

```
int x, *p;  
x = 10;  
*p = x;
```

**Error!!**

포인터 변수 p는 초기화 되지 않아서 값을 저장할 수 없음.

```
int x, *p;  
x = 10;  
p = x;
```

**Error!!**

포인터 변수에는  
일반 정수형 변수가 가지는 값을 저장할 수 없음

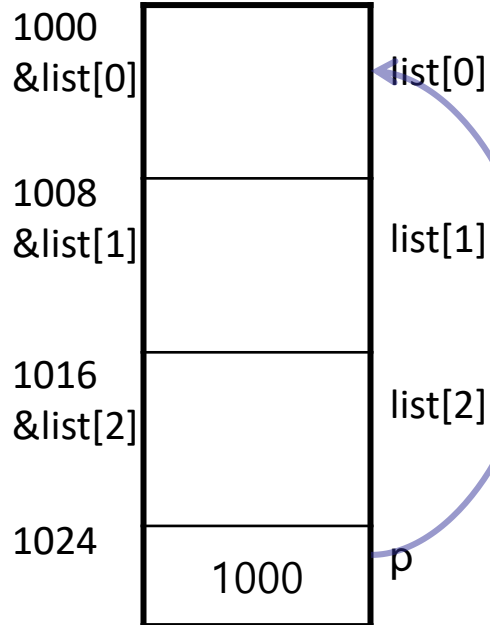
# 배열과 포인터

- 배열의 이름은 포인터 변수와 같은 역할

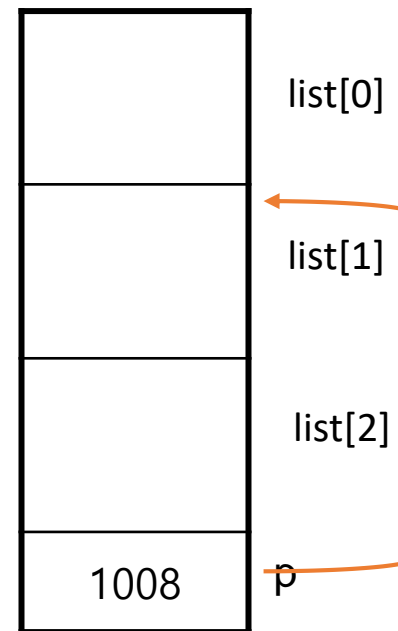
[Ex]  
double list[3], \*p;

p = &list[0];  
와 같다

p = list;

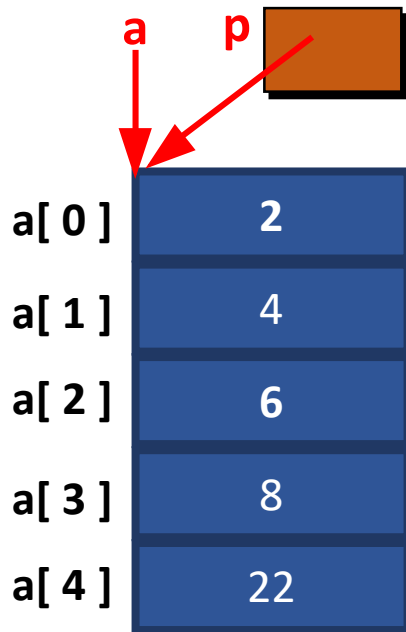


p = &list[1];



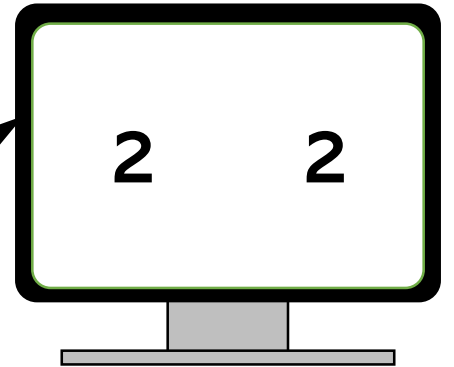
# 배열과 포인터

- 포인터 변수에 배열을 지정



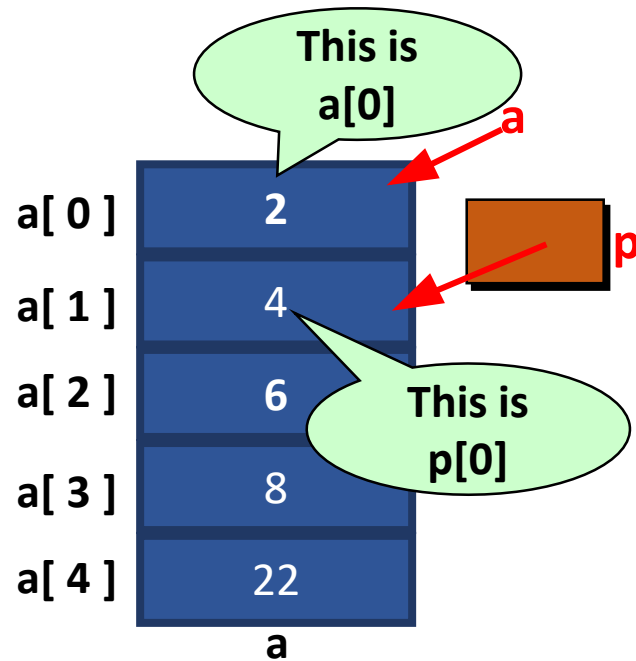
```
# include <stdio.h>
int main ( void )
{
    int    a[5]    = { 2, 4, 6, 8, 22 };
    int*    p      = a ;
    ...
    printf ( " %d %d\n" , a[0], *p );
    ...
    return 0 ;
} // main
```

Same as  
p[0]

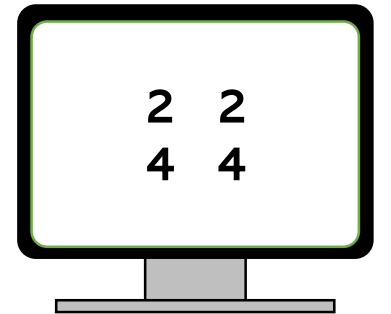


# 배열과 포인터

- 포인터변수에 배열의 두번째 항목 연결

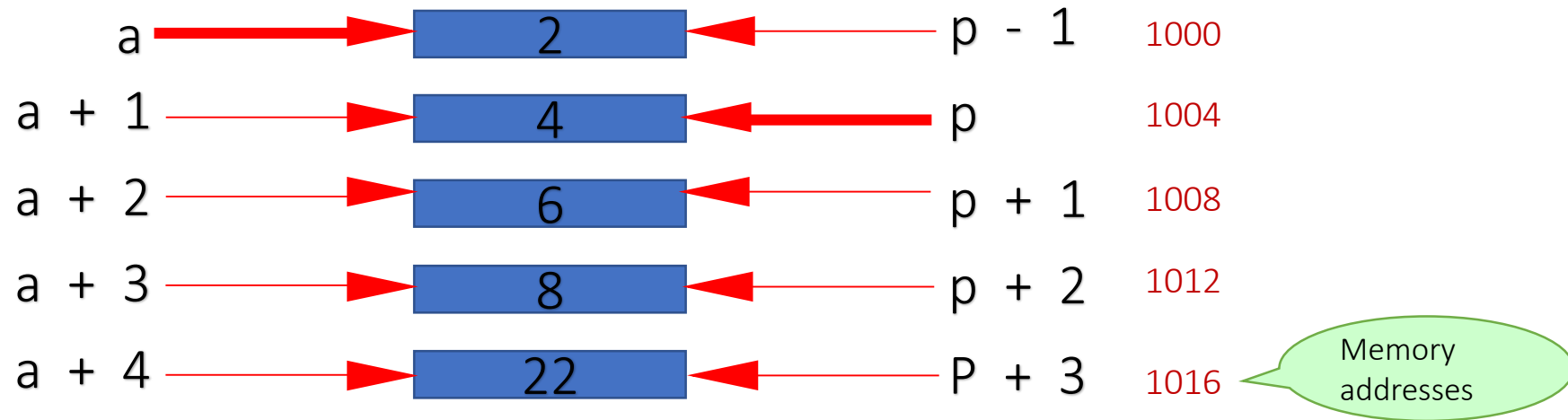


```
# include <stdio.h>
int main ( void )
{
    int    a[5]    = { 2, 4, 6, 8, 22 };
    int*    p;
    ...
    p = &a [1];
    printf ( " %d %d " , a[0], p[-1] );
    printf ( " \n " );
    printf ( " %d %d " , a[1], p[0] );
    ...
} // main
```



# 배열과 포인터

- 포인터변수에 배열의 두번째 항목 연결 결과



# 포인터의 연산

- 포인터의 연산

[Ex]

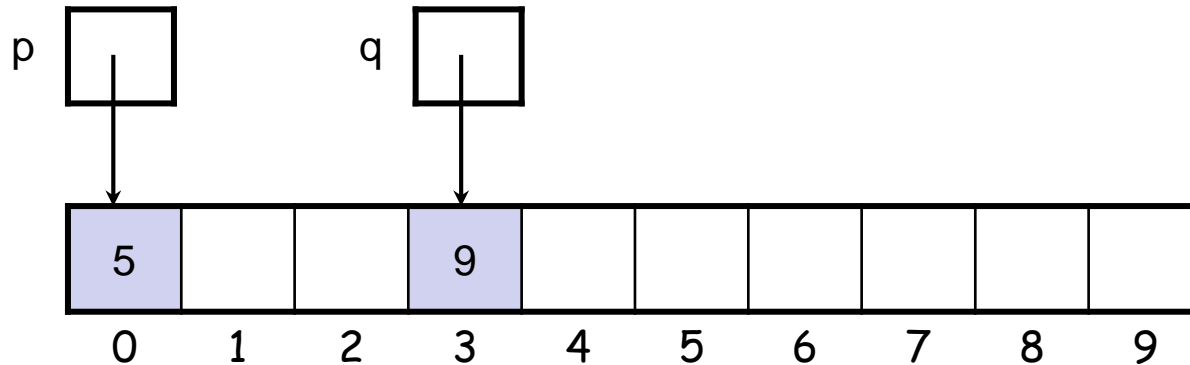
```
int a[10], *p, *q;
```

```
p=a;           //p=&a[0];와 같음
```

```
q=a+3;         //q=&a[3];와 같음
```

```
*p=5;          // a[0]=5; 와 같음
```

```
*q=9;          // a[3]=9;와 같음
```



[Ex]

```
a = a + 1;
```

// 오류

```
p = p + 1;
```

// valid : \*p ≡ a[1]

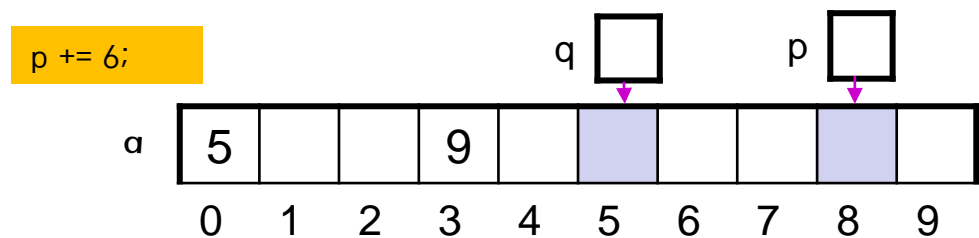
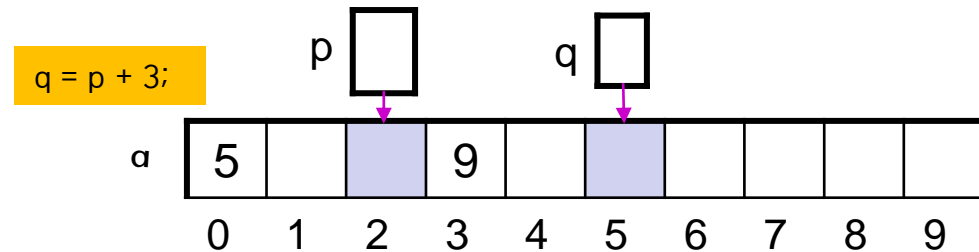
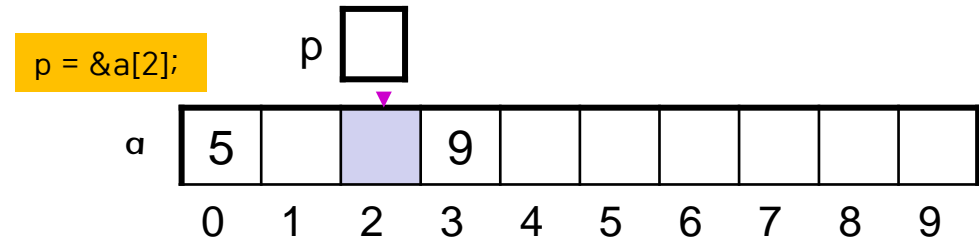
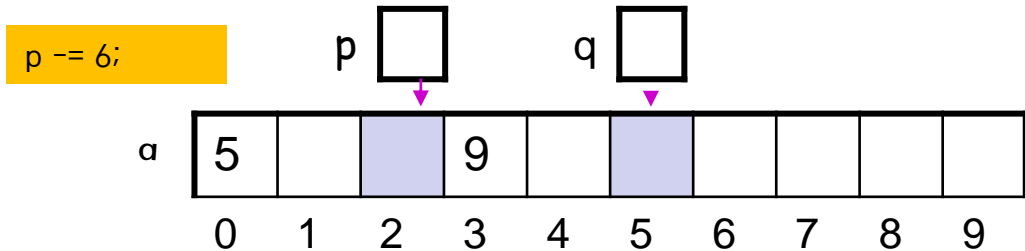
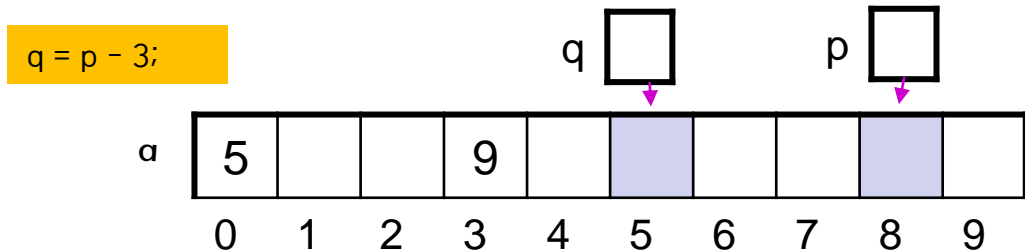
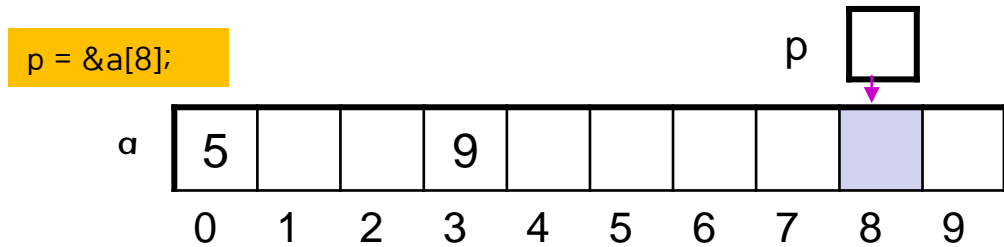
```
p++;
```

// valid : \*p ≡ a[2]

포인터 상수(변수명)은  
데이터를 변경할 수 없음,  
그러나 포인터 변수 p는  
데이터 변경 가능

# 포인터의 연산

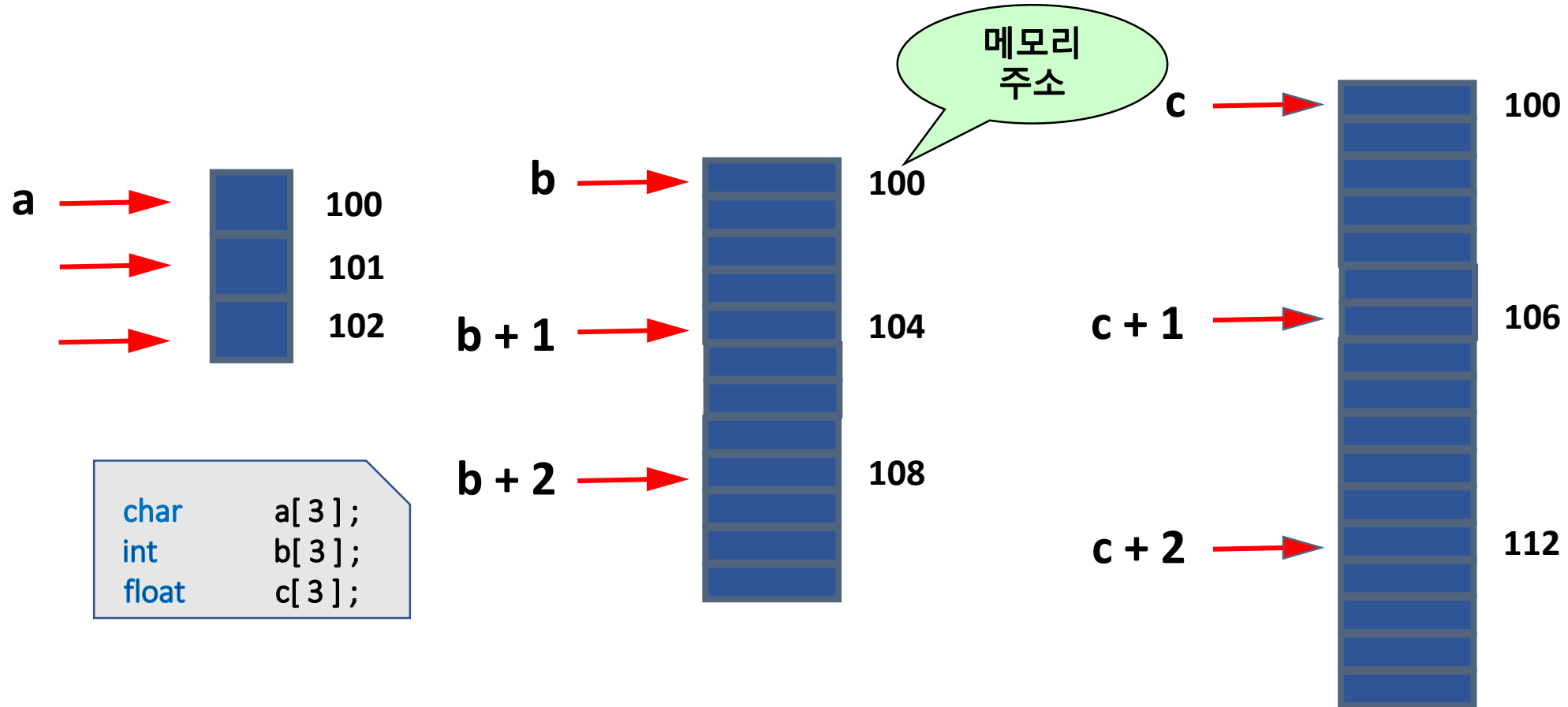
- 포인터에 더하기 빼기
  - 포인터에 더하기와 빼기의 경우 선언한 데이터 형식의 크기만큼 이동





# 포인터와 다양한 데이터 형식

- 데이터 형식에 따른 포인터연산 비교



$$a + n \longrightarrow a + n * (\text{sizeof}(\text{one element}))$$

# 포인터의 연산

- 포인터 변수의 빼기
  - 두 포인터 사이의 빼기 연산 결과

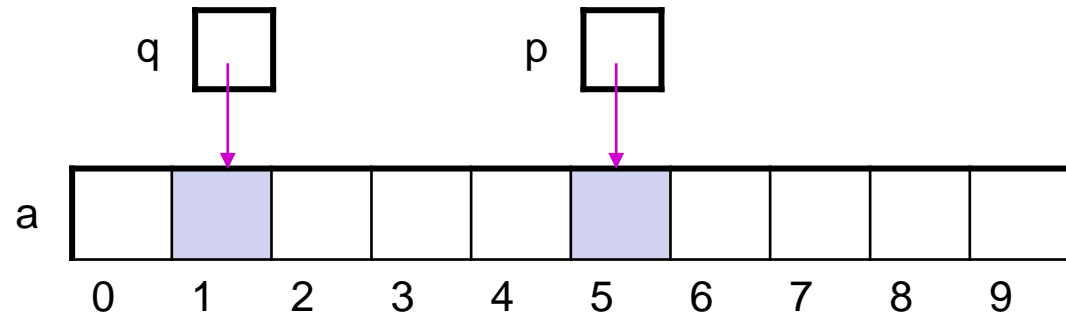
[Ex]

```
p = &a[5];
```

```
q = &a[1];
```

```
i = p - q;           /* i is 4 */
```

```
i = q - p;           /* i is -4 */
```



# 포인터의 연산

- 포인터 변수의 비교
  - 관계연산자: <, <=, >, >=
  - 비교연산자: ==, !=

[Ex]

```
p = &a[5];
```

```
q = &a[1];
```

```
if (p <= q) printf("pointer p has a small value");           /* result is 0 */
```

```
if (p >= q) printf("pointer p has a large value");           /* result is 1 */
```

```
if (p == q) printf("equal value");                           /* result is 0 */
```

# 포인터의 연산 예제

```
int a[ ] = { 5,15,25,43,12,1,7,89,32,11}
```

```
int *p = &a[1], *q = &a[5] ;
```

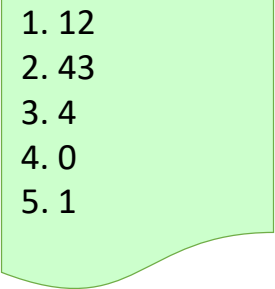
1.  $*(p + 3)$  ?

2.  $*(q - 2)$  ?

3.  $q - p$  ?

4.  $\text{if} ( p > q )$  ?

5.  $\text{if} ( *p > *q )$ ?



- 1. 12
- 2. 43
- 3. 4
- 4. 0
- 5. 1

# 포인터 연산 예제

```
#include <stdio.h>

int main()
{
    double a[2], *p, *q;
    p = &a[0];
    q = p + 1;

    printf("%d\n", q - p );
    printf("%d\n", (int) q - (int) p );
    printf("%d\n", sizeof(double) );

    return 0;
}
```

1  
8  
8

# 배열의 포인터 연산

- 배열 표현과 포인터 표현 비교

```
[Ex]  
#define N 5  
:  
int a[N], *p;  
p=a;  
.
```

배열 표현법	포인터 표현법
a[0], p[0]	*a, *p
a[1], p[1]	*(a+1), *(p+1)
a[2], p[2]	*(a+2), *(p+2)
a[3], p[3]	*(a+3), *(p+3)
a[4], p[4]	*(a+4), *(p+4)

# 배열의 포인터 연산

- 배열명을 이용한 데이터 접근과 포인터 배열을 이용한 데이터 접근 비교

[Ex]

```
#define N 10  
int a[N], sum, *p, i;  
:  
sum = 0;  
for ( p = a ; p < &a[N] ; ++p )  
    sum += *p;
```

// Corresponding to  
for(i=0; i<N; i++) sum += \*(p+i);  
for(i=0; i<N; i++) sum += p[i];

[Ex]

```
sum = 0;  
for ( i = 0 ; i < N ; ++i )  
    sum += *( a + i );
```

//Same as sum += a[i];

# 배열의 포인터 연산

- 역참조연산자(\*)와 증가연산자(++)

표현식	의미
*p++ or *(p++)	*p의 값에 접근 후, p가 가지고 있는 주소 증가
(*p)++	*p의 값에 접근후에, 해당하는 값의 증가
*++p or *(++p)	포인터 변수 p가 가지고 있는 주소의 증가 후 값에 접근
++*p or ++(*p)	포인터 변수 p가 값에 접근 후, 해당하는 값의 증가



# 배열의 포인터 연산 예시

- 역참조연산자(\*)와 증가연산자(++)

[Ex]

```
int a[3]={20, 7, -9}, *p, i ;  
p = &a[1] ;
```

① `i = *p++; printf("%d %d", i, *p) ;`      7 -9

② `i = *++p ; printf ("%d %d", i, *p) ;`      -9 -9

③ `i = ++*p ; printf ("%d %d", i, *p) ;`      8 8

④ `i = (*p)++; printf ("%d %d", i, *p) ;`      7 8

# 배열의 포인터 연산

- 0으로 배열 초기화 예시

- // clear every elements in 1-D array

```
int a[10], *p = &a[0] , i ;  
for( i = 0 ; i < 10 ; i++) {  
    a[i] = 0 ;  
}
```

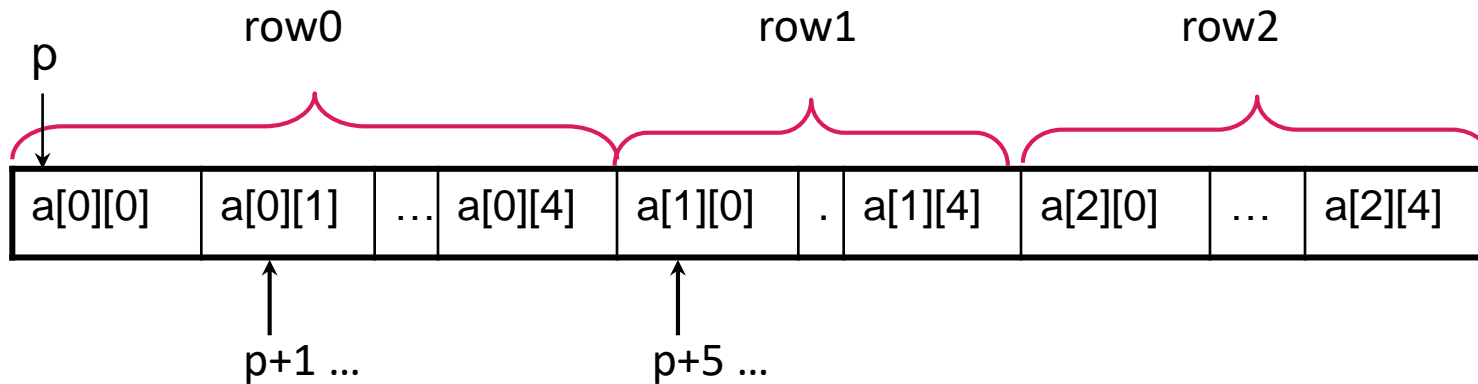
```
int a[10], *p = &a[0] , i ;  
  
for( ; p <= &a[9] ; p++) {  
    *p = 0 ;  
}  
  
for( p=a ; p <= &a[9] ; p++) {  
    printf("%d ", *p);  
}
```

```
int a[10], *p = &a[0] , i ;  
  
for( ; p <= &a[9] ; ) {  
    *p++ = 0 ;  
}  
  
for( p=a ; p <= &a[9] ; ) {  
    printf("%d ", *p++);  
}
```

# 2차원 배열과 포인터

- 2차원 배열은 메모리에서는 순차적으로 나열되어 나타남

[Ex]  
int a[3][5];  
int \*p=&a[0][0];



# 2차원 배열과 포인터

```
#include <stdio.h>
#define M 3      /* number of rows */
#define N 4      /* number of columns */

int main(void)
{
    int a[M][N], i, j, *p, sum = 0;

    for(p=&a[0][0];p<=&a[M-1][N-1]; p++)
        *p = 1;

    for (i = 0; i < M; ++i){
        for (j = 0; j < N; ++j){
            printf("a[%d][%d] = %d ", i, j, a[i][j] );
        }
        printf("\n");
    }

    for ( i = 0; i < M; ++ i )
        for ( j = 0; j < N; ++j )
            sum += a[i][j];

    printf("\nsum = %d\n\n", sum);

    return 0;
}
```

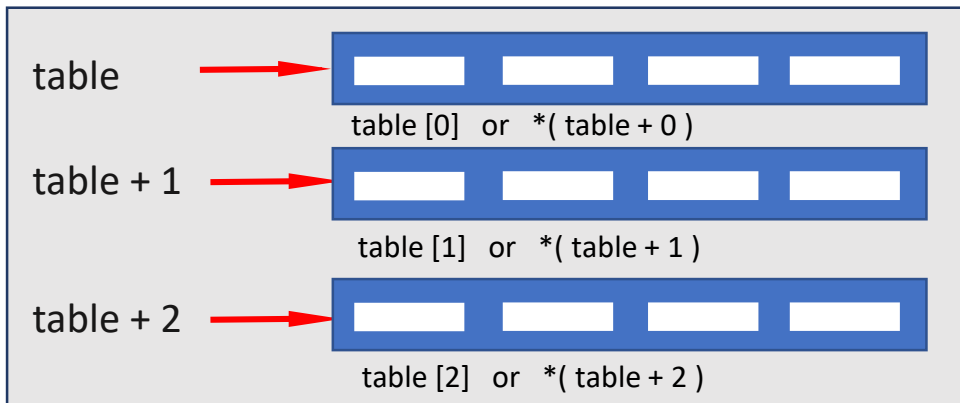
```
a[0][0] = 1 a[0][1] = 1 a[0][2] = 1 a[0][3] = 1
a[1][0] = 1 a[1][1] = 1 a[1][2] = 1 a[1][3] = 1
a[2][0] = 1 a[2][1] = 1 a[2][2] = 1 a[2][3] = 1

sum = 12
```

# 2차원 배열과 포인터

- 2차원 배열에서 행의 증가를 포인터 연산으로 처리
  - `*(table+1)`

```
int table[3][4];
```



```
for ( i = 0 ; i < 3 ; i ++ )  
{  
    for ( j = 0 ; j < 4 ; j ++ )  
        printf ( "%6d", *( * ( table + i ) +  
j ) );  
    printf ( "\n" );  
}
```

# 2차원 배열과 포인터

- 2차원 배열의 초기화

```
// clear every elements in 2-D array
```

```
int a[2][3], *p = &a[0][0], i, j;
```

```
for( i = 0 ; i < 2 ; i++) {  
    for( j=0 ; j < 3 ; j++)  
        a[i][j] = 0 ;  
}
```

```
// clear every elements in 2-D array
```

```
[Ex] for( i = 0 ; i < 2 ; i++) {  
    for( p = a[i] ; p < a[i]+3 ; p++) //3 : 열의 크기  
        *p = 0 ;  
}
```

```
[Ex] for( p = &a[0][0] ; p <= &a[1][2] ; p++)  
    *p = 0 ;
```

# 2차원 배열과 함수

- 함수에서 2차원 배열을 포인터 함수로 넘겨줄 경우
  - 2차원 배열의 열의 값은 반드시 고정된 상수로 지정하여야 함
  - 행의 경우는 없어도 무방

```
int mult[3][4];  
  
:  
  
total = sum(mult);
```

```
//function prototype of sum  
int sum(int m[][4]);    // int sum(int (*m)[4]);
```