Ch.14 Using Pointers

What you will learn in this chapter

•Learn about pointers to pointers, arrays of pointers, and function pointers.

•Let's look at the relationship between multidimensional arrays and pointers.

•Let's look at the arguments of the main() function .
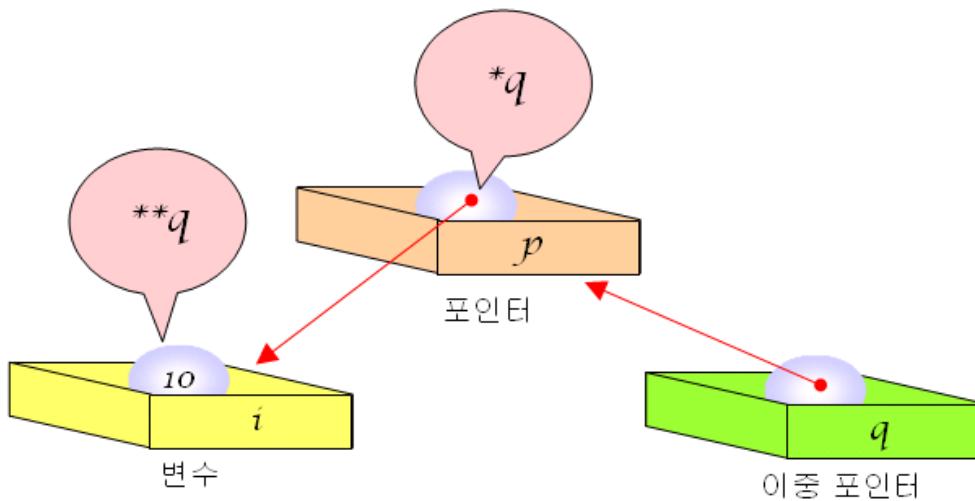
Pointers have many applications . In this chapter, you will learn only the ones you need .

double pointer
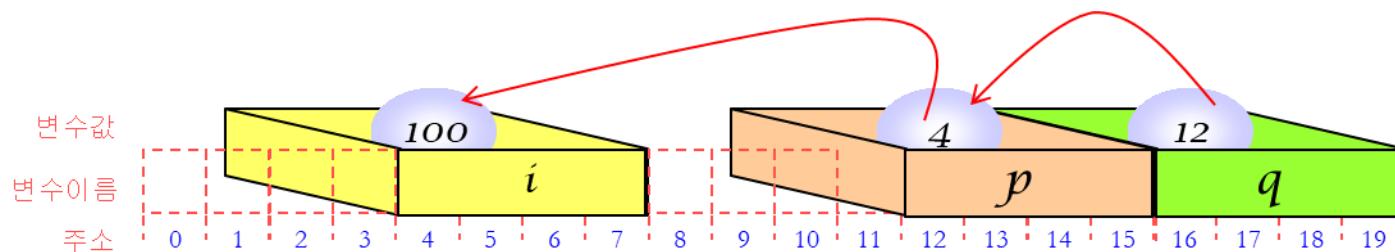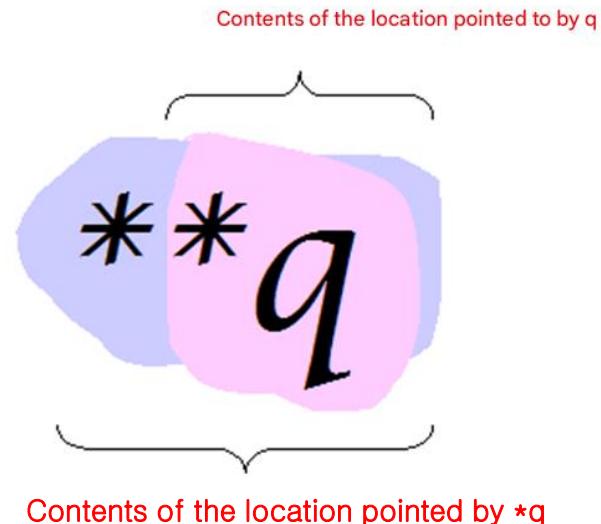
- Double pointer : a pointer that points to a pointer

int i = 10; // i is int type Variable
int *p = & i ; // p is i indicated Pointer
int **q = &p; // q is Pointer p indicated duplication Pointer

double pointer

- Interpretation of double pointers

Contents of the location pointed to by q

$**q$

Contents of the location pointed by $*$q

변수값   100   4   12

변수이름   i   p   q

주소   0   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16   17   18   19

```
int i = 10;
int *p = &i;      // single pointer
int **pp = &p;    // double pointer
int ***ppp = &pp; // triple pointer
```

double pointer
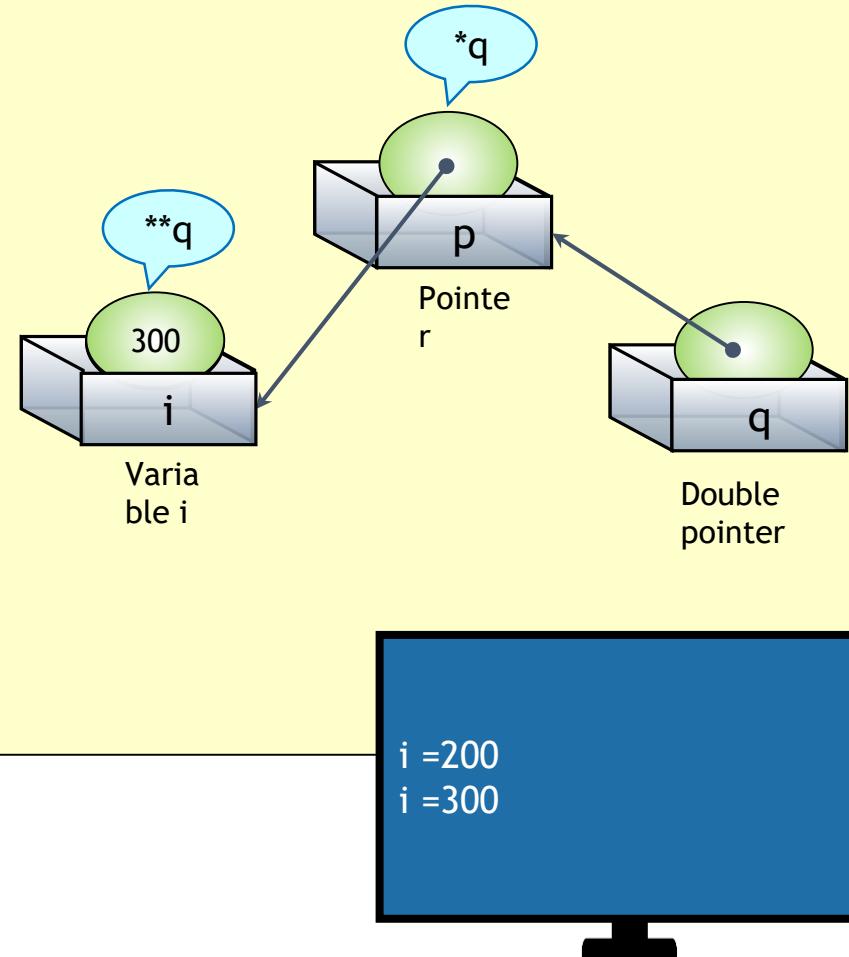
```c
// Double pointer program
#include < stdio.h >

int main( void )
{
    int i = 100;
    int *p = &i ;
    int **q = &p;

    *p = 200;
    printf("i =%d\n" , i );

    **q = 300;
    printf("i =%d\n" , i );

    return 0;
}
```


i =200
i =300

Example #2

```c
#include < stdio.h >

void set_pointer ( char ** q );
int main( void )
{
    char * p;
    set_pointer (&p);

    printf ( " Quote of the day : %s \n" , p);
    return 0;
}

void set_pointer ( char ** q )
{
    *q = "All that glisters are not gold." ;
}
```

To change the value of pointer p in a function, you must send its address .
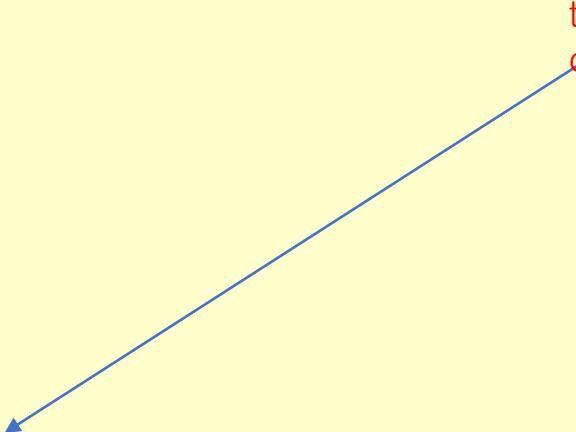
Proverb of the day : All that glisters is not gold

Bad code

```
int main( void )
{
 ...
  char *p;

  set_pointer (p);
 ...
}

void set_pointer ( char *q)
{
   q = "All that glisters are not gold." ;
}
```

the parameter q changes .

Check points
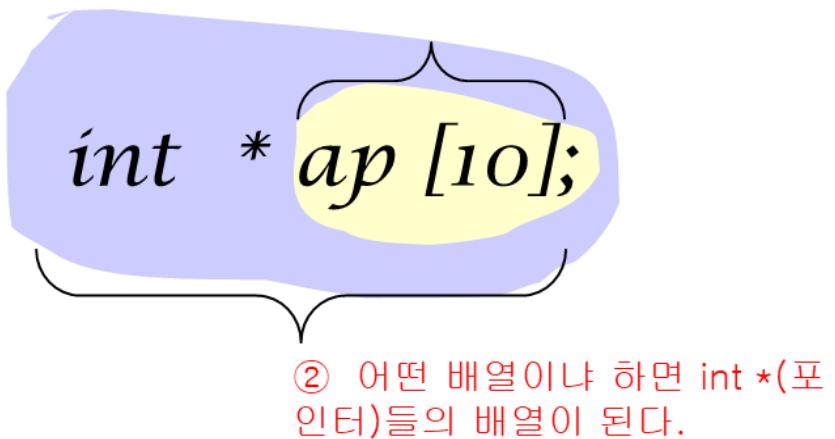
1. Let's declare a double pointer dp that points to a double pointer .
2. When defined as char c; char *p; char **dp ; p = &c; dp =&p; what does **dp refer to ?

# Array of pointers
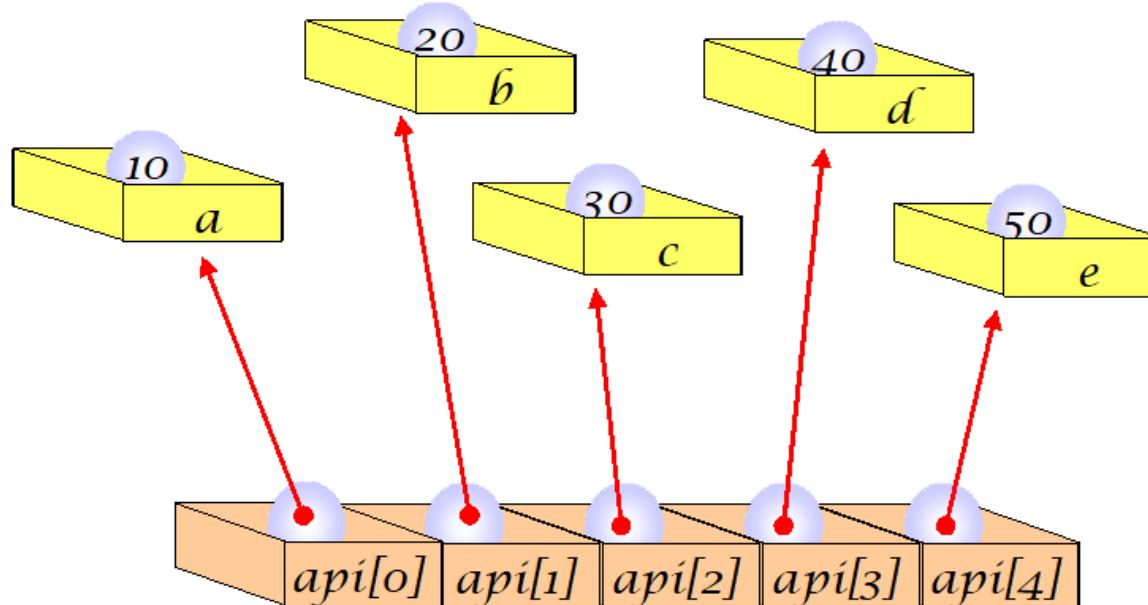
- *Array of pointers* : *An array of* pointers

① [ ] 연산자가 * 연산자보
다 우선 순위가 높으므로
ap는 먼저 배열이 된다.

int * ap [10];

② 어떤 배열이냐 하면 int *(포
인터)들의 배열이 된다.
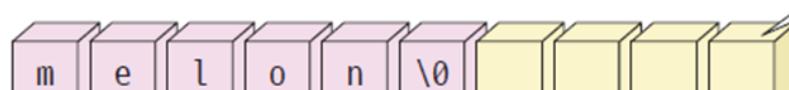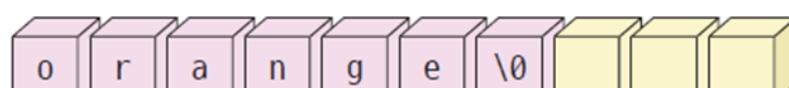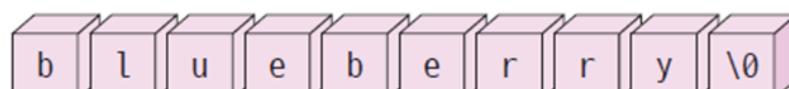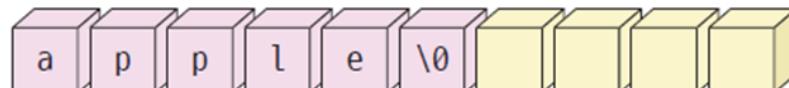
Array of integer pointers

```
int a = 10, b = 20, c = 30, d = 40, e = 50;
int *pa[ 5] = { &a, &b, &c, &d, &e };
```

Store strings in a two-dimensional array

```c
char fruits[4 ][10] = {
  "apple",
  "blueberry" ,
  "orange",
  "melon"
};
```
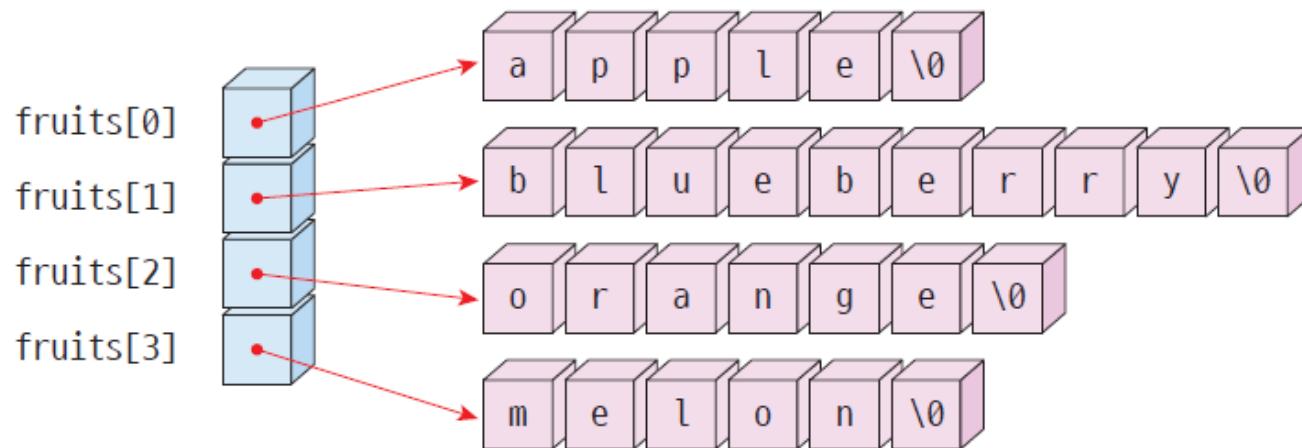

Using a two-dimensional array creates wasted space.

Wasted space!

Array of character pointers

```
char *fruits[ ] = {
  "apple",
  "blueberry" ,
  "orange",
  "melon"
};
```

String array

```c
#include < stdio.h >

int main( void )
{
    int i , n;
    char *fruits[ ] = {
        "apple" ,
        "blueberry" ,
        "orange" ,
        "melon"
    };

    n = sizeof (fruits)/ sizeof (fruits[0]); //calculate the number of array element

    for ( i = 0; i < n; i ++)
        printf ( "%s \n" , fruits[ i ]);

    return 0;
}
```
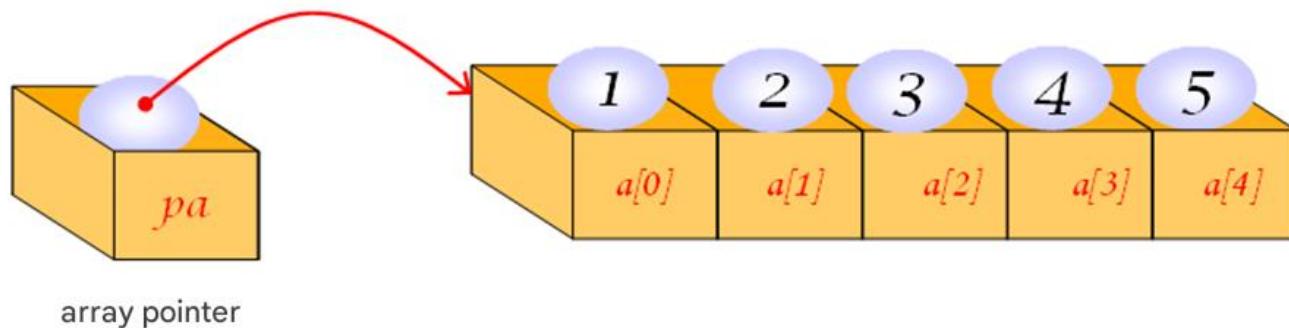
```
apple
blueberry
Orange
Melon
```

Array pointer

- A pointer to an array is a pointer that points to an array .

array pointer

① Since there are parentheses,
pa becomes a pointer first.

$$int \ (* \ pa) \ [10];$$

② It is a pointer that points to int [10].

Example

```c
#include < stdio.h >

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

    pa = &a;
    for ( i =0 ; i <5 ; i ++)
        printf ( "%d \n" , (*pa)[ i ]);
    return 0;
}
```

Array pointer

1
2
3
4
5

reference

Comparison of array of pointers and array of pointers

It's quite complicated, but it's easy once you understand the principle.
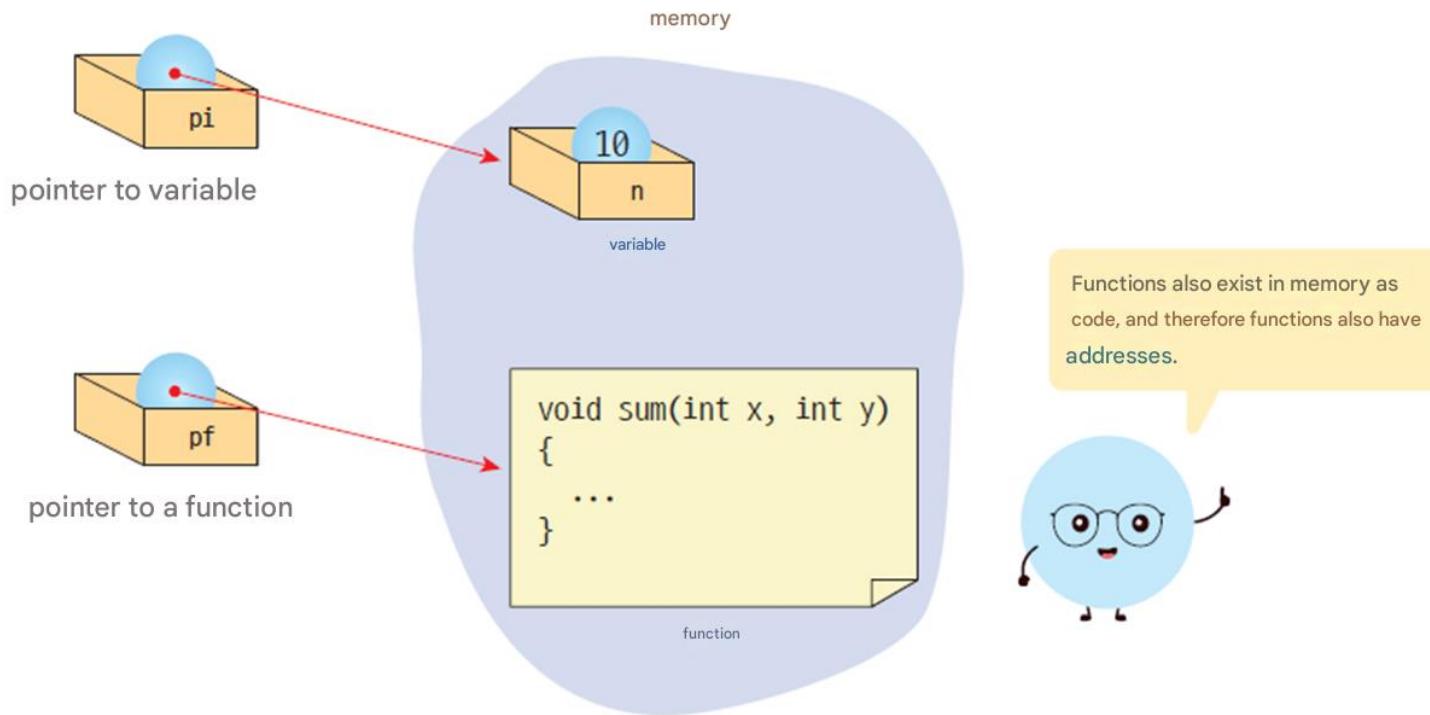
• int *ap[10]; --- It is an array of pointers. The [] operator, which indicates an array, has a higher priority than the * operator, which indicates

a pointer. Therefore, ap becomes an array first. And what kind of array is it? It is an array of pointers.

• int (*pa) [10]; This --- time, the priority has changed due to the parentheses. Because of the parentheses, the * operator is applied first, so

pa becomes a pointer first. And what kind of pointer is it? It becomes a pointer that points to int [10].

It's complicated, but in practice, arrays of pointers appear much more often. Therefore, there is no problem if you use them without parentheses. In

fact, even professional programmers do not use array pointers (i.e. pointers to arrays) very often.

Function pointer

- Function pointer : A pointer that points to a function .

Function pointer definition
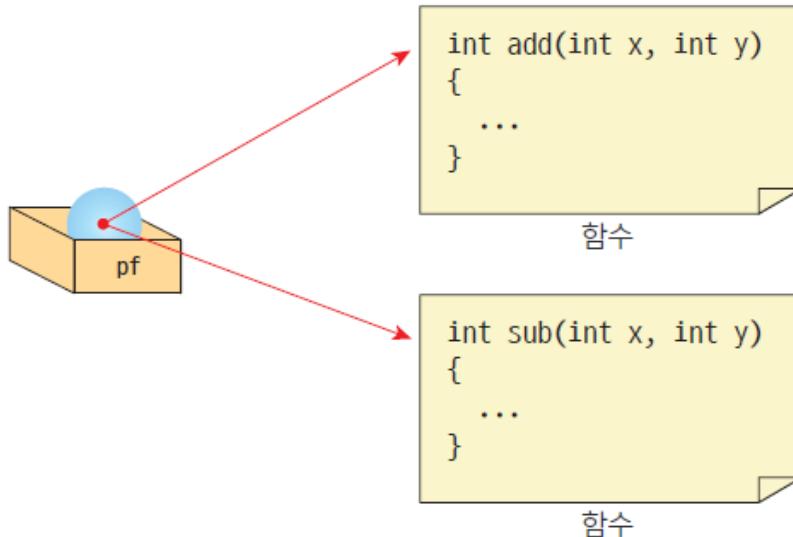
Using function pointers

```
int add( int , int );     // Function prototype definition
int (*pf )( int , int );  // Function pointer definition
...
pf = add;                 // Assign the function name to the function pointer
result = pf(10, 20);      // function pointer Calling a function through
```



```
int add(int x, int y)
{
    ...
}
```
함수

```
int sub(int x, int y)
{
    ...
}
```
함수

pf

fp1.c

```c
#include < stdio.h >

// function circle definition
int add( int , int );
int sub( int , int );

int main( void )
{
    int result;
    int (*pf)( int , int ); // function Pointer definition

    pf = add; // function On the pointer The function add () address Substitution
    result = pf (10, 20); // function Pointer Through Call
    printf ( "10+20 is %d\n" , result);

    pf = sub; // function On the pointer The function sub () address Substitution
    result = pf (10, 20); // function Pointer Through Calling
    printf ( "10-20 is %d\n" , result);

    return 0;
}
```

fp1.c
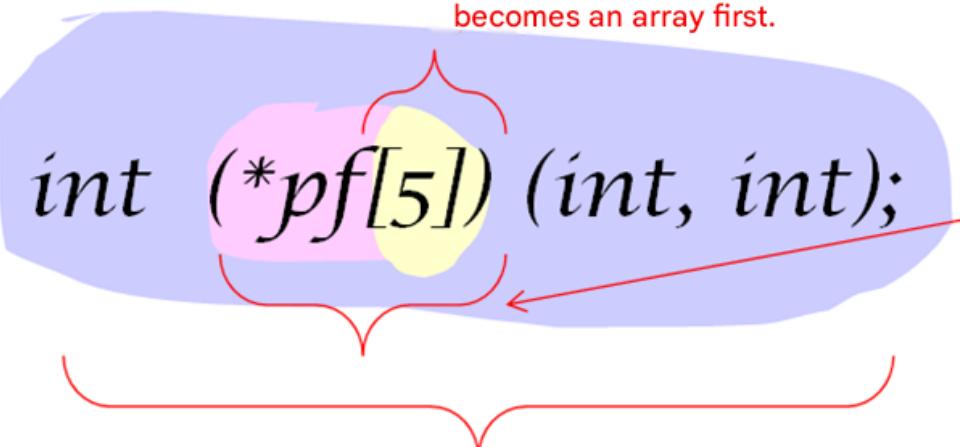
```c
int add( int x, int y)
{
    return x+y ;
}

int sub( int x, int y)
{
    return xy;
}
```

10+20 is 30
10-20 is -10

Array of function pointers

```
int (*pf[5]) ( int, int );
```


① Since the [] operator has a higher priority than other operators, pf becomes an array first.
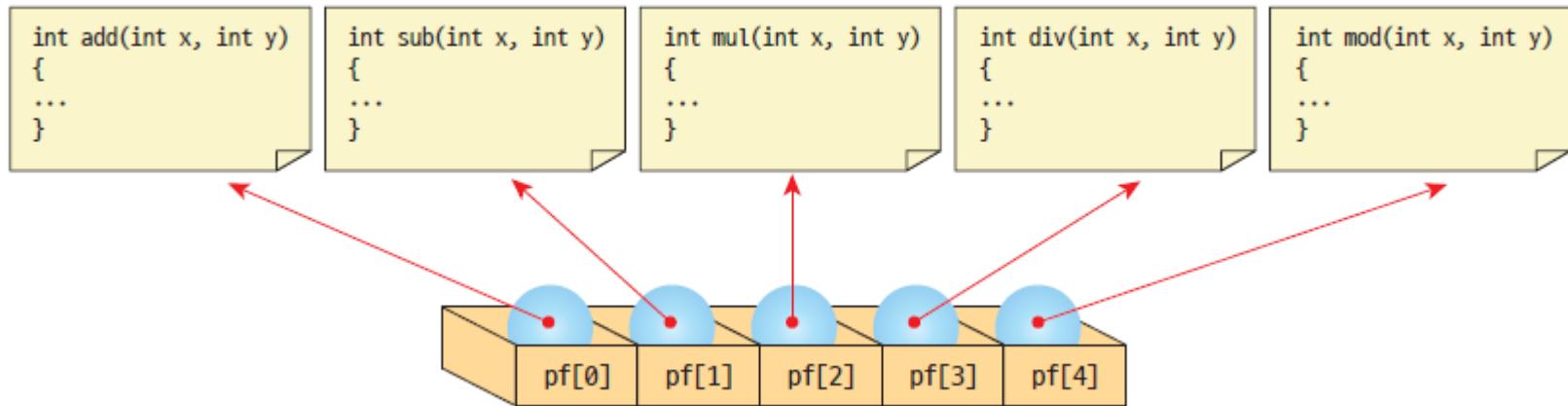
② The array is an array of pointers.

③ It is a pointer that points to a function.

Array of function pointers

Let's write the following program using function pointers .

```
====================
0. Addition
1. Subtraction
2. Multiplication
3. Division
4. End
====================
Select a menu :2
Enter two integers : 10 20
Operation result = 200

====================
0. Addition
1. Subtraction
2. Multiplication
3. Division
4. End
====================
Select a menu :
```

Array of function pointers

```c
// function Pointer arrangement
#include < stdio.h >

// function circle definition
void menu( void );
int add( int x, int y);
int sub( int x, int y);
int mul ( int x, int y);
int div( int x, int y);

void menu( void )
{
    printf ( "================================\n" );
    printf ( "0. Addition \n" );
    printf ( "1. Subtraction \n" );
    printf ( "2. Multiplication \n" );
    printf ( "3. Division \n" );
    printf ( "4. End \n" );
    printf ( "================================\n" );
}
```

Array of function pointers

```c
int main( void )
{
    int choice, result, x, y;
    // function Pointer Array Declare and Initialize .
    int (* pf [4])( int , int ) = { add, sub, mul , div };

    while (1)
    {
        menu();
        printf ( " menu Select :" );
        scanf ( "%d" , &choice);

        if ( choice < 0 || choice >=4 )
            break ;
        printf ( "2 integers Enter :" );
        scanf ( "%d %d" , &x, &y);

        result = pf [choice](x, y); // function Pointer Used Function Call
        printf ( " operation result = %d\ n" ,result );
    }
     return 0;
}
```

Array of function pointers

```c
int add( int x, int y)
{
    return x + y;
}

int sub( int x, int y)
{
    return x - y;
}

int mul ( int x, int y)
{
    return x * y;
}

int div( int x, int y)
{
    return x / y;
}
```
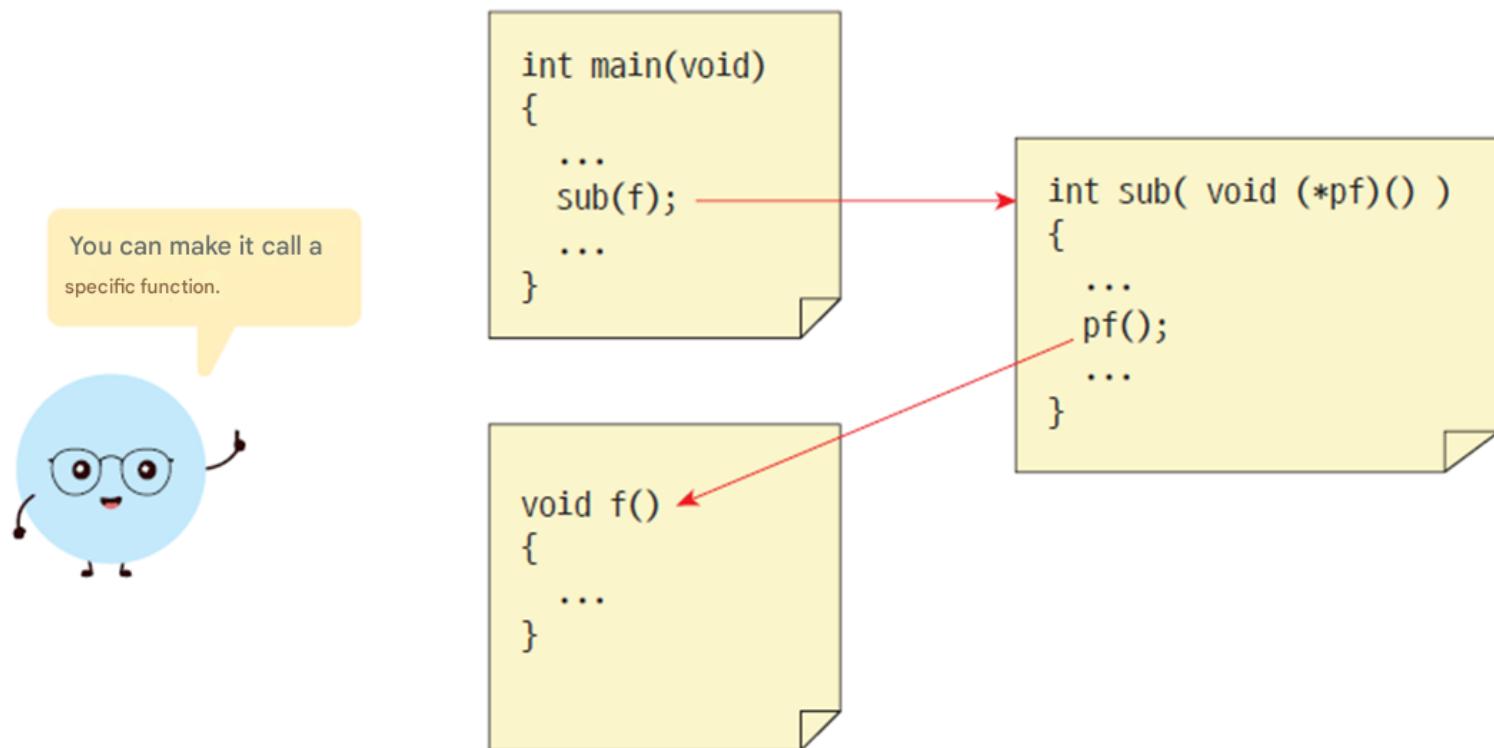
```
====================
0. Addition
1. Subtraction
2. Multiplication
3. Division
4. End
====================
Select a menu :2
Enter two integers : 10 20
Operation result = 200
====================
0. Addition
1. Subtraction
2. Multiplication
3. Division
4. End
====================
Select a menu :
```

Function pointers as function arguments

- Function pointers can also be passed as arguments .

Example

- Let's write a program that calculates the following formula :

$$\sum_{1}^{n}(f^2(k) + f(k) + 1)$$

- Here, f(k) can be the following functions :

$$f(k) = \frac{1}{k} \quad \text{또는} \quad f(k) = \cos(k)$$

# Example

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

double f1( double k);
double f2( double k);
double formula( double (* pf )( double ), int n);

int main( void )
{
    printf ( "%f\n" , formula(f1, 10));
    printf ( "%f\n" , formula(f2, 10));
}

double formula( double (* pf )( double ), int n)
{
    int i ;
    double sum = 0.0;

    for ( i = 1; i < n; i ++)
        sum += pf ( i ) * pf ( i ) + pf ( i ) + 1;
    return sum;
}
```

$$\sum_{1}^{n}(f^2(k) + f(k) + 1)$$

Example
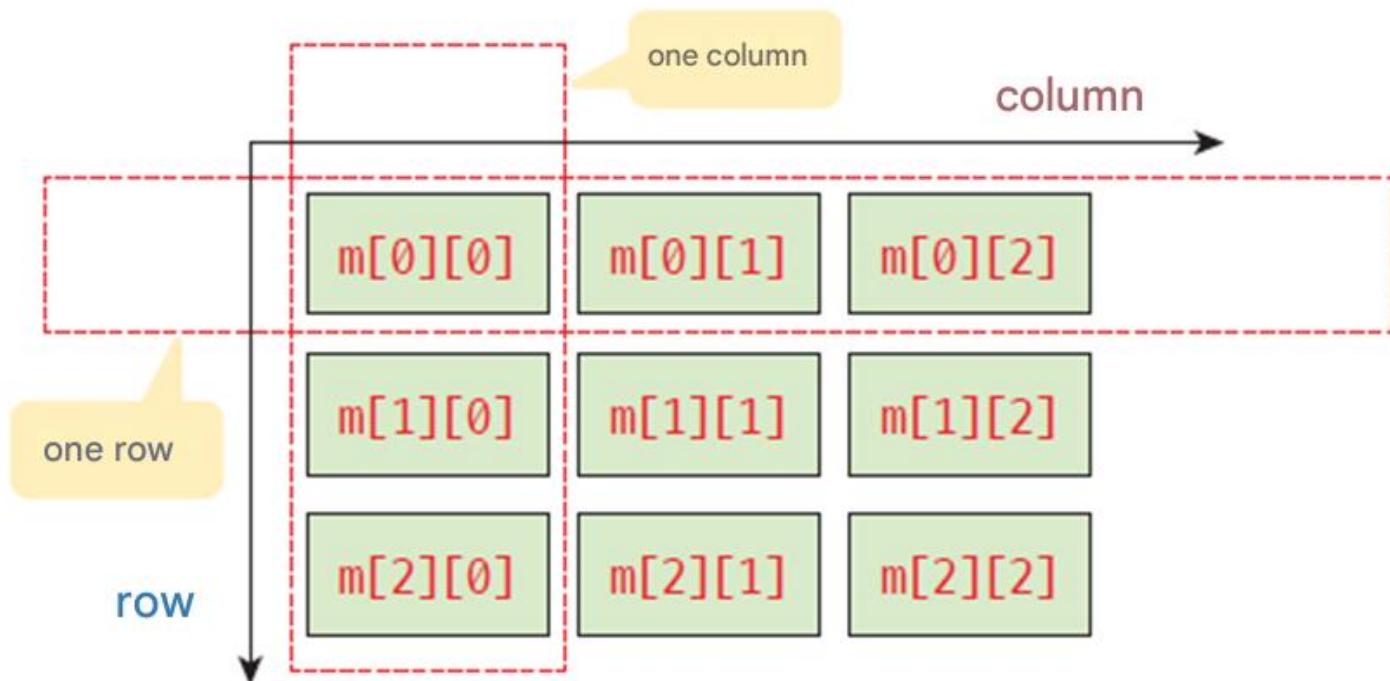
```
double f1( double k)
{
    return 1.0 / k;
}

double f2( double k)
{
    return cos (k);
}
```

```
13.368736
12.716152
```
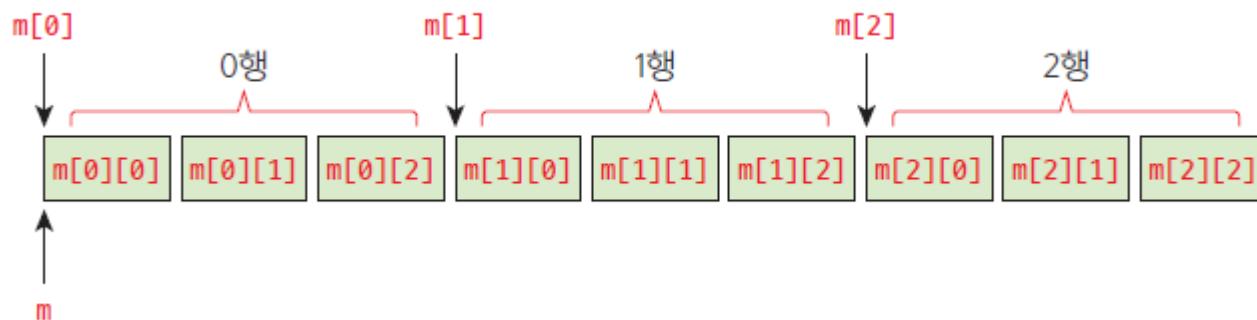
Multidimensional arrays and pointers

- 2-dimensional array int m[3][3]
- Store in memory in the order of
  1st row -> 2nd row -> 3rd row ->... (row-first method)

How to save

- The first method is the row-major method, which stores
  a two- dimensional array in memory based on rows. That is,
  the 0th row is stored first, and then the elements belonging
 to the 1st row are stored.

multi_array.c

```c
#include < stdio.h >
int main( void )
{
    int m[3][3] = { 10, 20, 30, 40, 50, 60, 70, 80, 90 };

    printf ( "m = %p\n" , m);
    printf ( "m[0] = %p\n" , m[0]);
    printf ( "m[1] = %p\n" , m[1]);
    printf ( "m[2] = %p\n" , m[2]);
    printf ( "&m[0][0] = %p\n" , &m[0][0]);
    printf ( "&m[1][0] = %p\n" , &m[1][0]);
    printf ( "&m[2][0] = %p\n" , &m[2][0]);

    return 0;
}
```
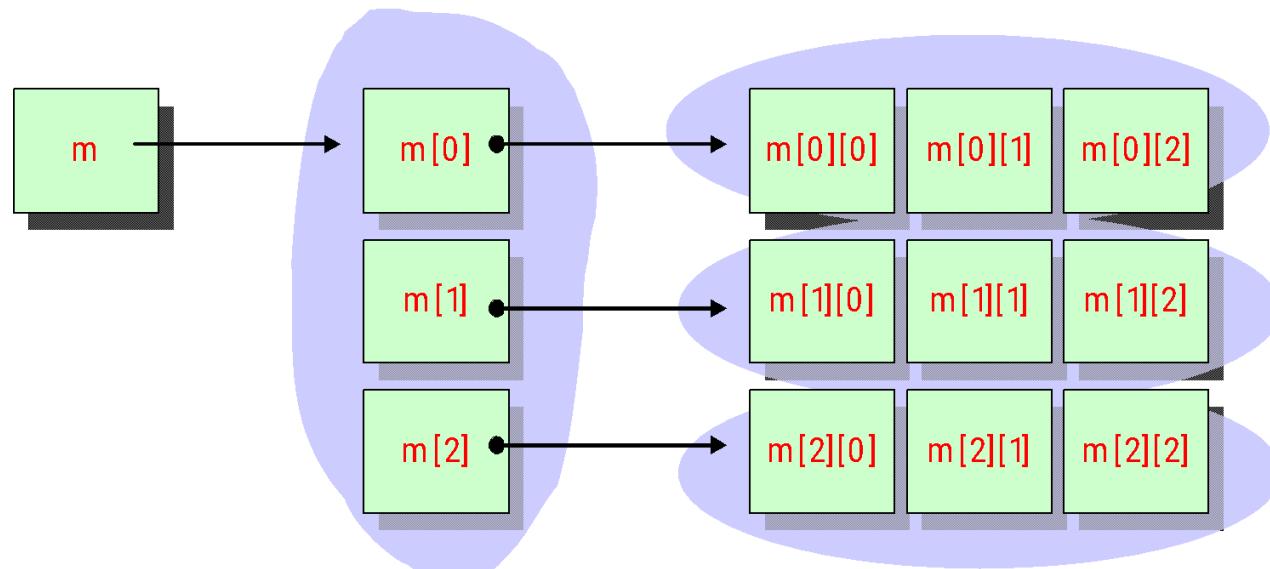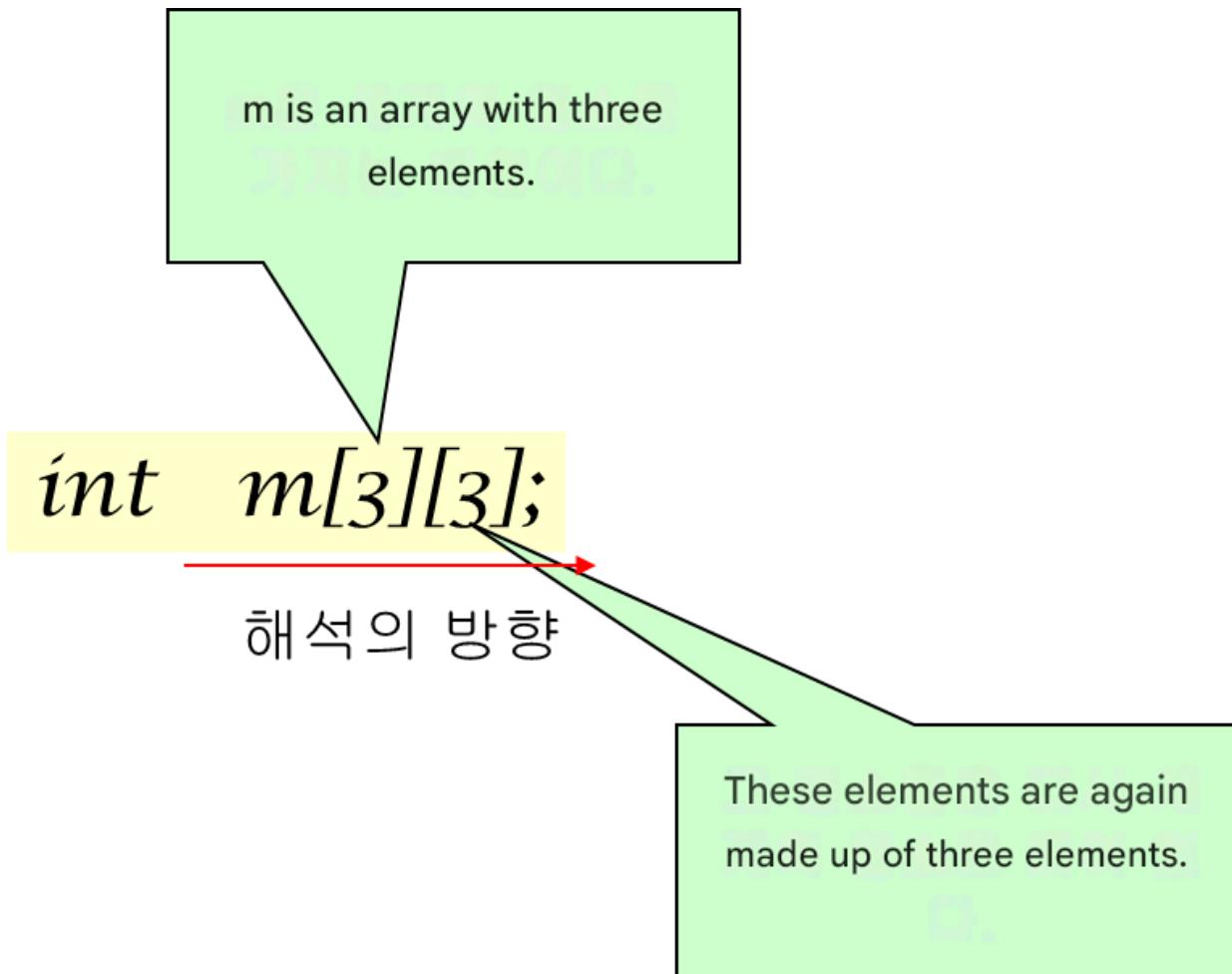
```
m = 000000341A36F888
m[0] = 000000341A36F888
m[1] = 000000341A36F894
m[2] = 000000341A36F8A0
&m[0][0] = 000000341A36F888
&m[1][0] = 000000341A36F894
&m[2][0] = 000000341A36F8A0
```

Two-dimensional arrays and pointers

- The array name m is &m[0][0]
- m[0] is the starting address of row 1
- m[1] is the starting address of row 2
- …
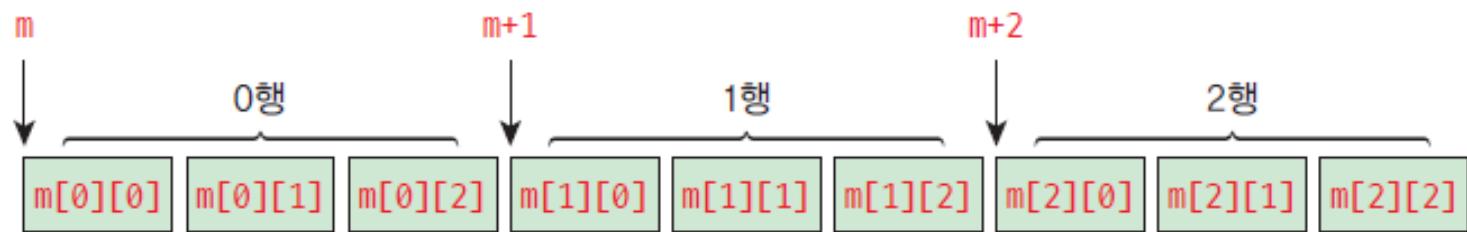
Interpreting a two-dimensional array

m is an array with three elements.

$int \quad m[3][3];$

해석의 방향

These elements are again made up of three elements.

Two-dimensional arrays & pointer operations

• What does it mean to add or subtract 1 from m in a two-dimensional array m[][] ?

Visiting array elements using pointers
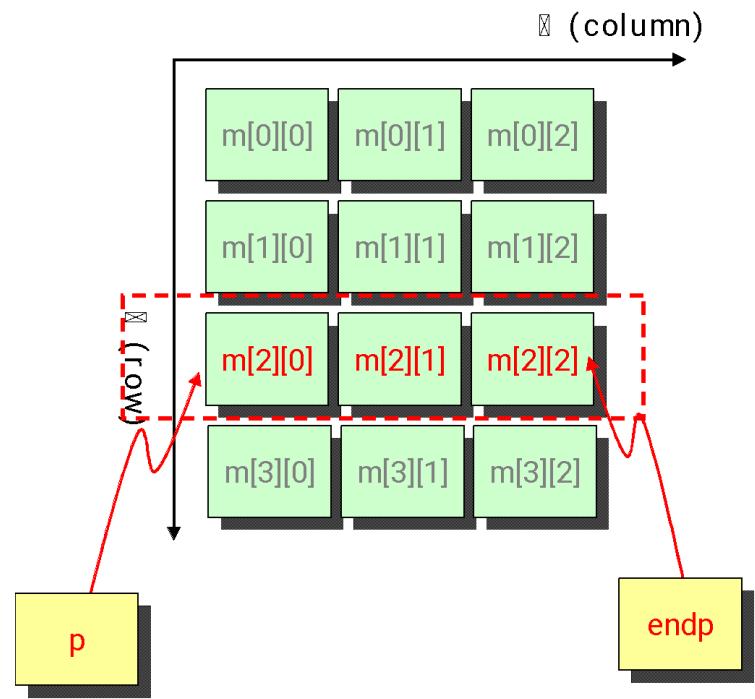
- When calculating the average of rows

```
double get_row_avg ( int m[][COLS], int r)
{
    int *p, * endp ;
    double sum = 0.0;

    p = &m[r][0];
    endp = &m[r][COLS];

    while ( p < endp )
        sum += *p++;

    sum /= COLS;
    return sum;
}
```

(column)

m[0][0]	m[0][1]	m[0][2]
m[1][0]	m[1][1]	m[1][2]
m[2][0]	m[2][1]	m[2][2]
m[3][0]	m[3][1]	m[3][2]

(row)

p

endp

Visiting array elements using pointers
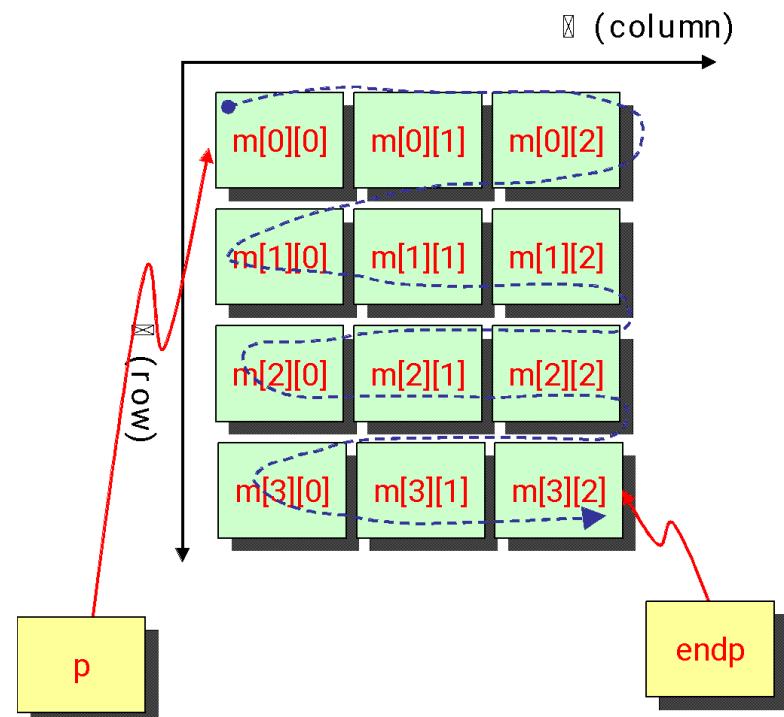
- When calculating the average of all elements

```
double get_total_avg ( int m[][COLS])
{
        nt *p, * endp ;
        double sum = 0.0;


        p = &m[0][0];
        endp = &m[ROWS-1][COLS];

        while ( p < endp )
            sum += *p++;

        sum /= ROWS * COLS;
        return sum;
}
```

Check points
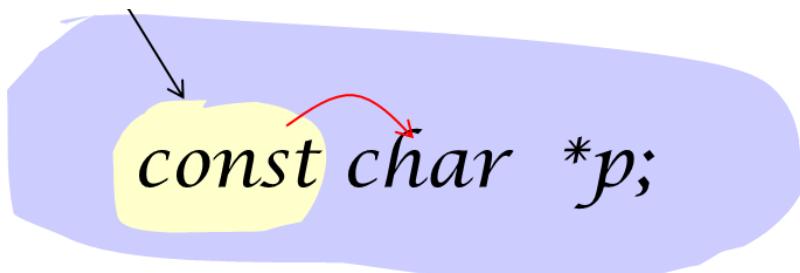
1. m[0] mean in m[10][10] ?
2. (m+1) mean in m[10][10] ?
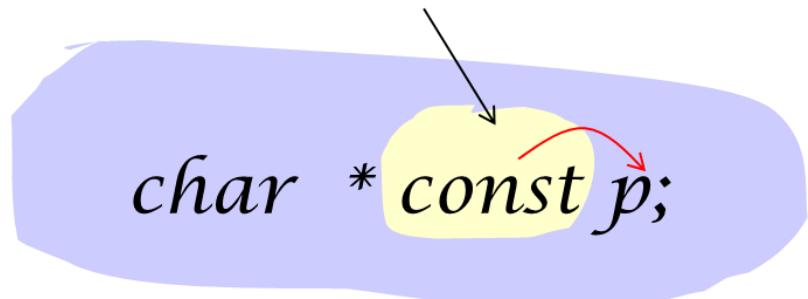
const pointer

- const The meaning changes depending on where it is attached.

const char *p;

char * const p;

Example

```c
#include < stdio.h >
int main( void )
{
    char s[] = "Barking dogs rarely bite.";
    char t[] = "A bad workman blames his tools";
     const char * p=s;
    char * const q=s;


    //p[3] = 'a';              p cannot be changed .


    p = t;                     However, p can be changed .


    q[3] = 'a';                The content pointed to by q
                               can be changed .


    //q = t;                   However, q cannot be changed
                               .


    return 0;
}
```

volatile pointer

- volatile means that the value can always be changed by another process or thread, so it must be read from memory again every time the value is used.

This means that the content pointed to by p changes frequently, so reload it every time you use it.

volatile char *p;

void pointer

- A pointer that purely contains the address of a memory (generic pointer)
- The target being pointed to has not yet been determined
  (Example) void * vp ;

- All of the following operations are errors :

```
* vp ; // error
* ( int *) vp ; // Convert void pointer to int pointer .
vp ++; // error
vp --; // error
```

void pointer is where is it used ?

- Using void pointers, you can write functions that can receive pointers of any type. For example, let's write a function that fills the passed memory with 0 as follows :

```c
void memzero ( void * ptr , size_t len )
{
    for (; len > 0; len --) {
        *( char *) ptr = 0;
    }
}
```

vp.c

```c
#include < stdio.h >
void memzero ( void * ptr , size_t len )
{
    for (; len > 0; len --) {
        *( char *) ptr = 0;
    }
}

int main( void )
{
    char a[10];
    memzero (a, sizeof (a));

    int b[10];
    memzero (b, sizeof (b));

    double c[10];
    memzero (c, sizeof (c));

    return 0;
}
```

main() function

- The main() function format so far
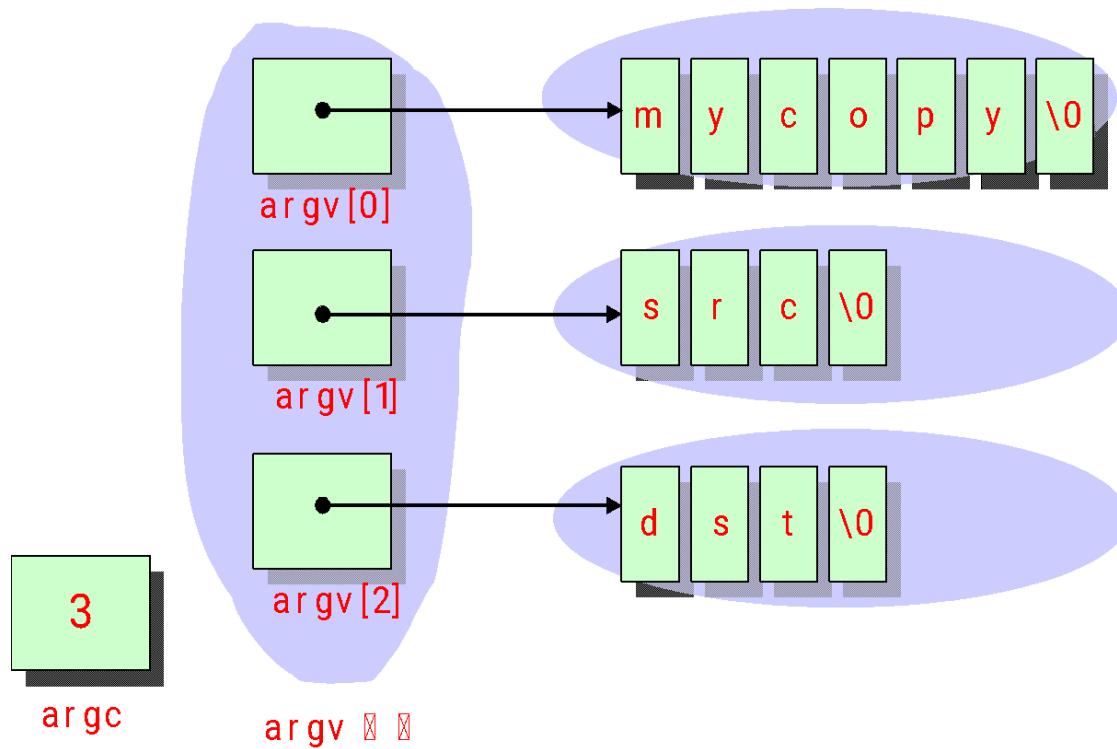
```
int main (void)
{
..
}
```

- the main() function that receives input from outside

```
int main( int argc, char *argv[])
{
..
}
```

How to transfer acquisition

C: \cprogram> mycopy src dst

main_arg.c

```c
#include < stdio.h >

int main( int argc , char * argv [])
{
    int i = 0;

    for ( i = 0; i < argc ; i ++)
        printf ( " command % dth in line string = %s\n" , i , argv [ i ]);

    return 0;
}
```

```
c:\cprogram\mainarg\Debug>mainarg src dst
0th string on command line = mainarg
1st string on command line = src
2nd string on command line = dst
c:\cprogram\mainarg\Debug>
```

Check points

1. When executing like C>main arg1 arg2 arg3, what does argv [0] point to ?
2. When executing like C>main arg1 arg2 arg3, what is the value of argc ?

Q & A