


제 14장 포인터 활용

이번 장에서 학습할 내용

- 
- 포인터의 포인터, 포인터 배열, 함수 포인터 등을 학습한다.
 - 다차원 배열과 포인터의 관계를 살펴본다.
 - `main()` 함수의 인수에 대하여 살펴본다.

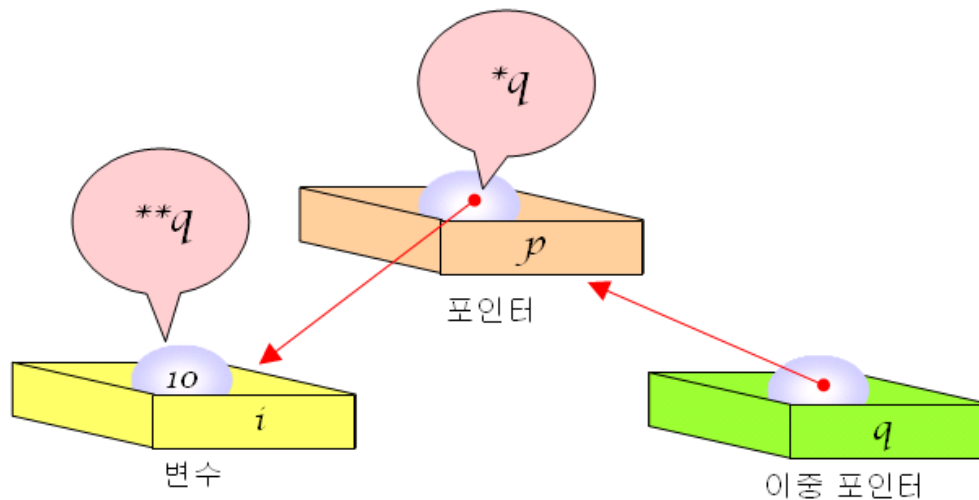
포인터는 상당히 많은 분야에 응용됩니다. 이 장에서는 필요한 것만 골라서 학습하면 됩니다.



이중 포인터

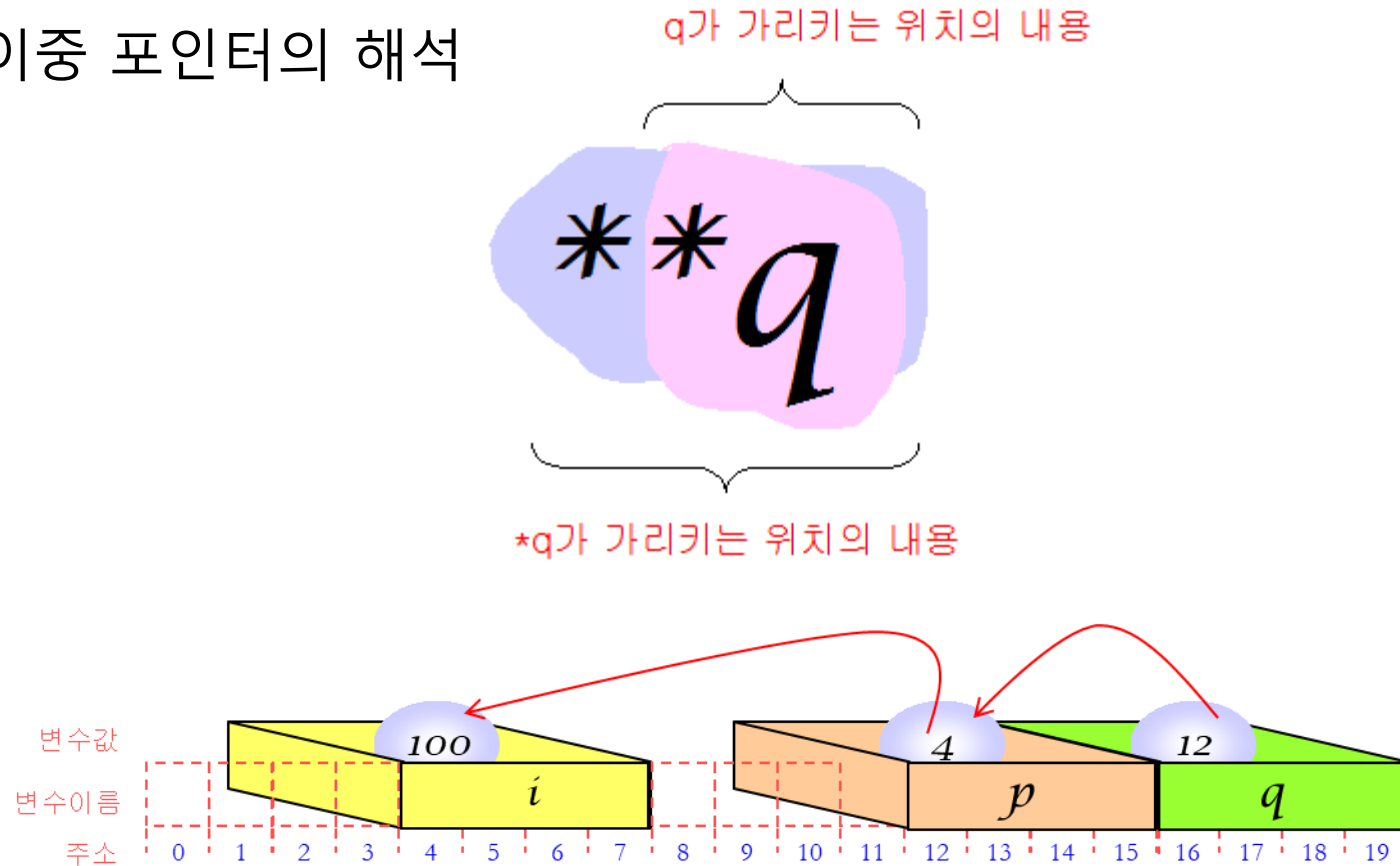
- 이중 포인터(double pointer) : 포인터를 가리키는 포인터

```
int i = 10;           // i는 int형 변수  
int *p = &i;         // p는 i를 가리키는 포인터  
int **q = &p;        // q는 포인터 p를 가리키는 이중 포인터
```



이중 포인터

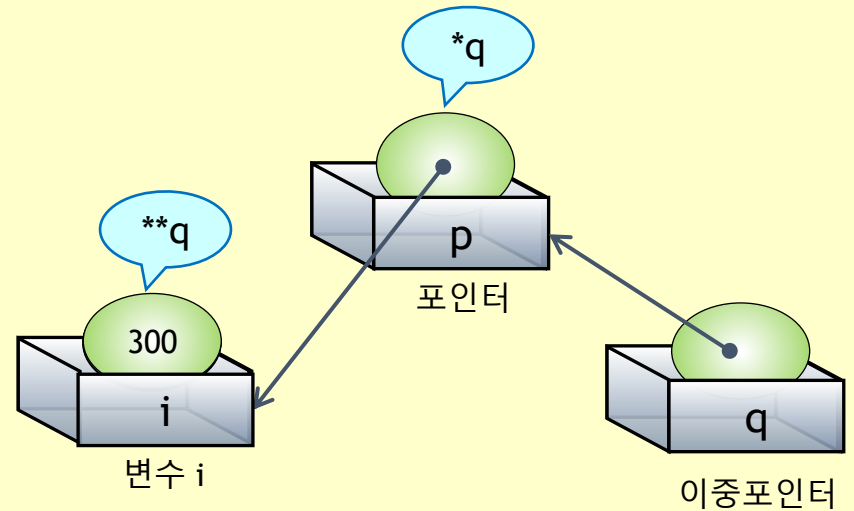
- 이중 포인터의 해석



이중 포인터

```
// 이중 포인터 프로그램  
#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

예제 #2

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

```
void set_pointer(char** q);
```

```
int main(void)
```

```
{
```

```
    char* p;
```

```
    set_pointer(&p);
```

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

```
    return 0;
```

```
}
```

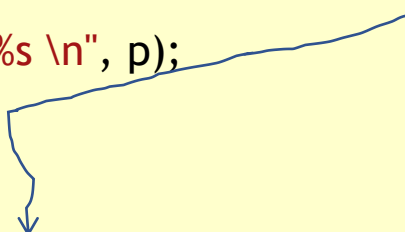
```
void set_pointer(char** q)
```

```
{
```

```
    *q = "All that glisters is not gold.";
```

```
}
```

포인터 p의 값을 함수에서 변경하려면
주소를 보내야 한다.



오늘의 격언: All that glisters is not gold

잘못된 코드

```
int main(void)
```

```
{
```

```
    ...
```

```
    char *p;
```

```
    set_pointer(p);
```

```
    ...
```

```
}
```

```
void set_pointer(char *q)
```

```
{
```

```
    q = "All that glisters is not gold.";
```

```
}
```

매개 변수 q만 변경된다.



중간 점검

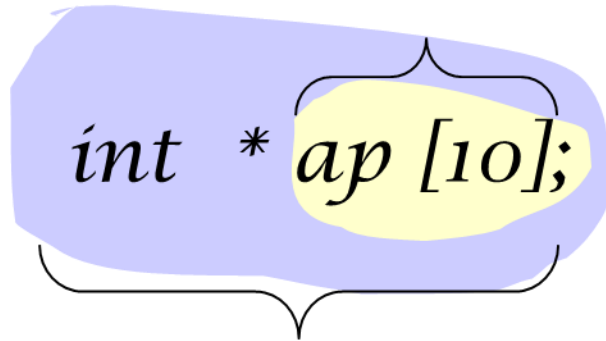
1. double형 포인터를 가리키는 이중 포인터 dp를 선언하여 보자.
2. char c; char *p; char **dp; p = &c; dp = &p;와 같이 정의되었을 때 **dp은 무엇을 가리키는가?



포인터 배열

- *포인터 배열(array of pointers)*: 포인터를 모아서 배열로 만든것

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

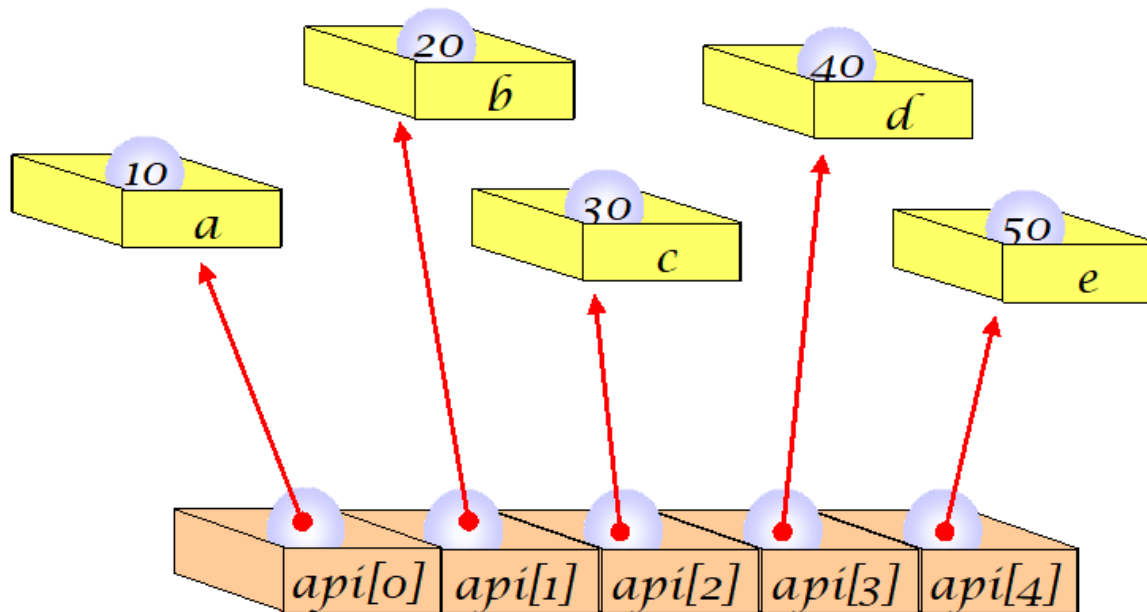


*int *ap[10];*

② 어떤 배열이냐 하면 int *(포인터)들의 배열이 된다.

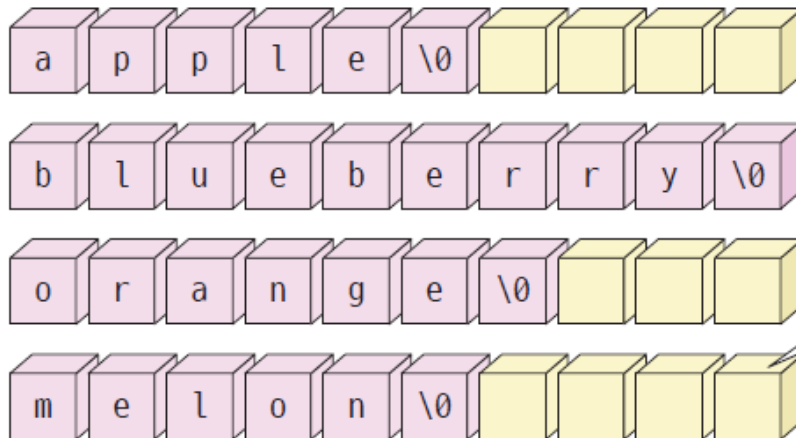
정수형 포인터 배열

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



2차원 배열에 문자열을 저장

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



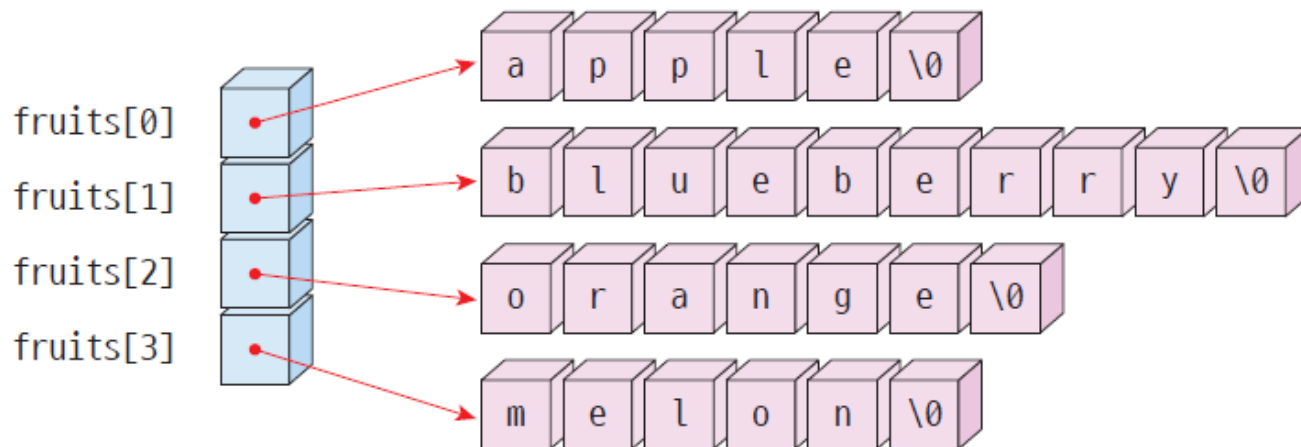
낭비되는
공간!

2차원 배열을 사용하면
낭비되는 공간이 생성되죠.



문자형 포인터 배열

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



문자열 배열

```
#include <stdio.h>

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

    n = sizeof(fruits)/sizeof(fruits[0]); // 배열 원소 개수 계산

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

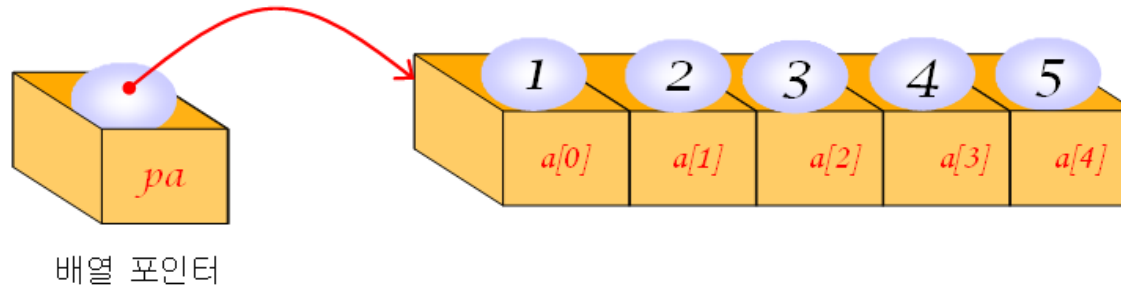
    return 0;
}
```



apple
blueberry
orange
melon

배열 포인터

- 배열 포인터(a pointer to an array)는 배열을 가리키는 포인터



① 괄호가 있으므로 `pa`는
먼저 포인터가 된다.

```
int (*pa)[10];
```

② 어떤 포인터냐 하면 `int [10]`
을 가리키는 포인터가 된다.

예제

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

```
int main(void)
```

```
{
```

```
    int a[5] = { 1, 2, 3, 4, 5 };
```

```
    int (*pa)[5];
```

```
    int i;
```

```
    pa = &a;
```

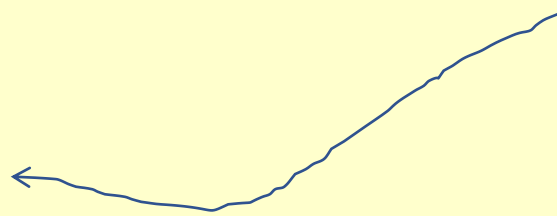
```
    for(i=0 ; i<5 ; i++)
```

```
        printf("%d \n", (*pa)[i]);
```

```
    return 0;
```

```
}
```

배열 포인터



1
2
3
4
5

참고

참고사항

포인터 배열과 배열 포인터의 비교

상당히 복잡하다. 그러나 원리를 이해하면 쉽다.

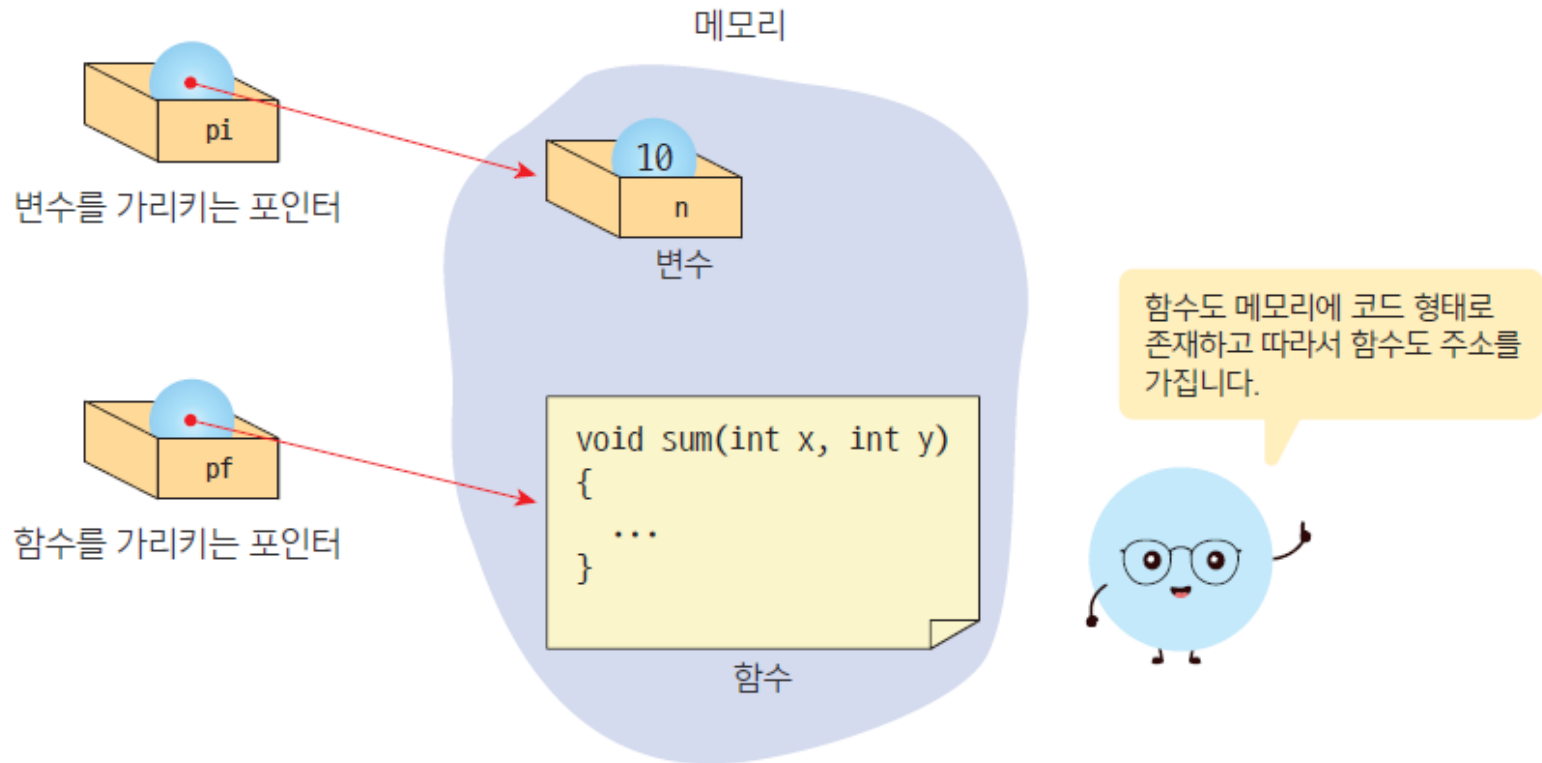
- `int *ap[10];` --- 포인터의 배열이다. 배열을 나타내는 `[]` 연산자는 포인터를 나타내는 `*` 연산자보다 우선순위가 높다. 따라서 `ap`는 먼저 배열이 된다. 그리고 어떤 배열이냐 하면 포인터들의 배열이 된다.

- `int (*pa)[10];` --- 이번에는 괄호에 의하여 우선 순위가 바뀌었다. 괄호 때문에 `*` 연산자가 먼저 적용되어서 `pa`는 먼저 포인터가 된다. 그리고 어떤 포인터냐 하면 `int [10]`을 가리키는 포인터가 된다.

복잡하지만 실전에서는 포인터의 배열이 훨씬 많이 나온다. 따라서 괄호 없이 사용하면 별 문제가 없다. 사실 전문적인 프로그래머도 배열 포인터(즉 배열에 대한 포인터)는 잘 사용하지 않는다.

함수 포인터

- 함수 포인터(function pointer): 함수를 가리키는 포인터



함수 포인터 정의

Syntax

함수 포인터 정의

예

```
int (*pf)(int, int);
```

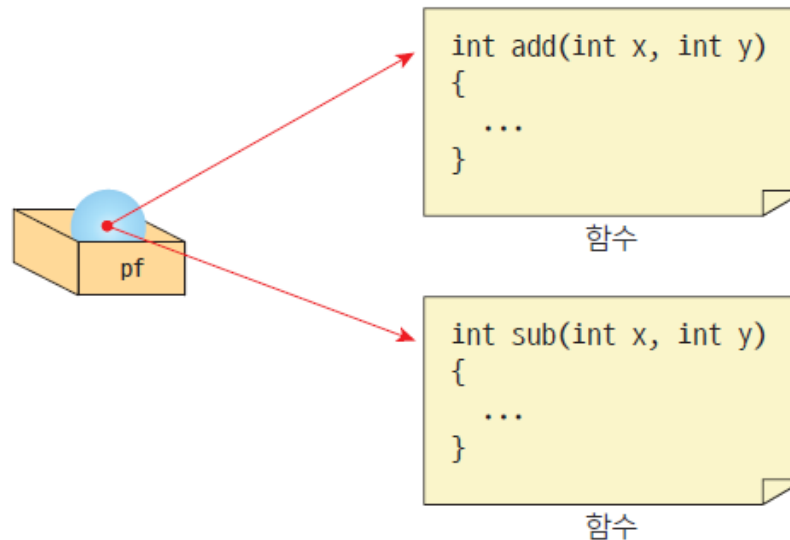
반드시 괄호가 필요하다. 왜냐하면 괄호에 의하여 pf가 먼저 포인터가 되어야 한다.

함수를 가리키는 포인터를 선언한다.

포인터가 가리키는 함수의 매개 변수

함수 포인터의 사용

```
int add(int, int);           // 함수 원형 정의
int (*pf)(int, int);         // 함수 포인터 정의
...
pf = add;                     // 함수의 이름을 함수 포인터에 대입
result = pf( 10, 20);         // 함수 포인터를 통하여 함수 호출
```



fp1.c

```
#include <stdio.h>

// 함수 원형 정의
int add(int, int);
int sub(int, int);

int main(void)
{
    int result;
    int (*pf)(int, int);           // 함수 포인터 정의

    pf = add;                     // 함수 포인터에 함수 add()의 주소 대입
    result = pf(10, 20);          // 함수 포인터를 통한 함수 add() 호출
    printf("10+20은 %d\n", result);


    pf = sub;                     // 함수 포인터에 함수 sub()의 주소 대입
    result = pf(10, 20);          // 함수 포인터를 통한 함수 sub() 호출
    printf("10-20은 %d\n", result);

    return 0;
}
```

fp1.c

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

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



10+20은 30
10-20은 -10

함수 포인터의 배열

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

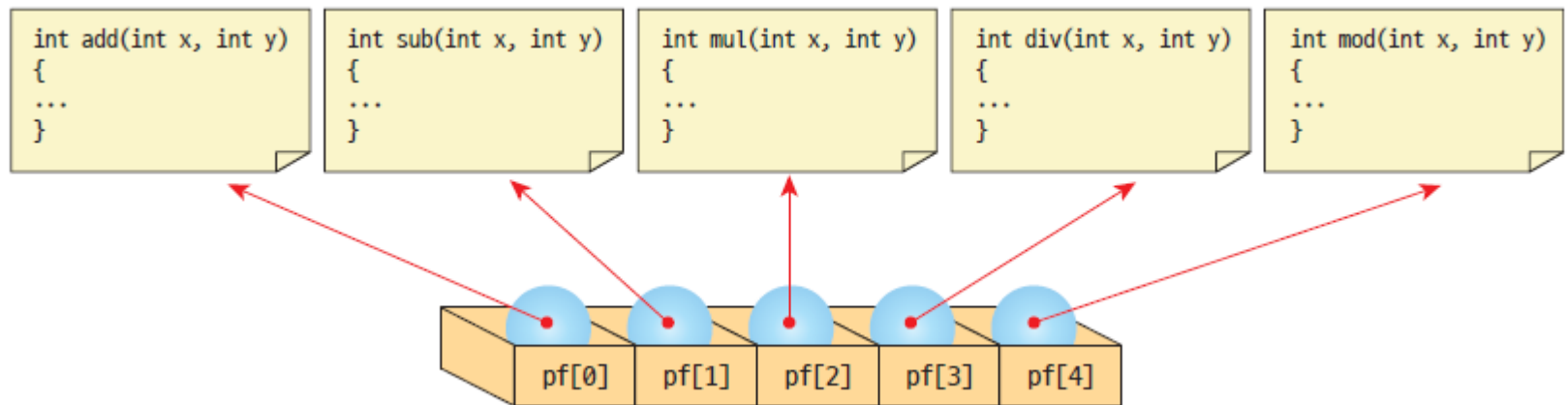
① [] 연산자가 * 연산자보다 우선 순위가 높으므로 pf는 먼저 배열이 된다.

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

② 어떤 배열이냐 하면 i포인터들의 배열이 된다.

③ 어떤 포인터냐 하면 함수를 가리키는 포인터가 된다.

함수 포인터의 배열



함수 포인터를 이용하여 다음 프로그램을 작성해보자.

=====

- 0. 덧셈
- 1. 뺄셈
- 2. 곱셈
- 3. 나눗셈
- 4. 종료

=====

메뉴를 선택하시오:2

2개의 정수를 입력하시오:10 20

연산 결과 = 200

=====

- 0. 덧셈
- 1. 뺄셈
- 2. 곱셈
- 3. 나눗셈
- 4. 종료

=====

메뉴를 선택하시오:

함수 포인터 배열

```
// 함수 포인터 배열
#include <stdio.h>

// 함수 원형 정의
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. 덧셈\n");
    printf("1. 뺄셈\n");
    printf("2. 곱셈\n");
    printf("3. 나눗셈\n");
    printf("4. 종료\n");
    printf("=====\n");
}
```

함수 포인터 배열

```
int main(void)
{
    int choice, result, x, y;
    // 함수 포인터 배열을 선언하고 초기화한다.
    int (*pf[4])(int, int) = { add, sub, mul, div };

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

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

        result = pf[choice](x, y);    // 함수 포인터를 이용한 함수 호출
        printf("연산 결과 = %d\n", result);
    }
    return 0;
}
```

함수 포인터 배열

```
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. 덧셈
- 1. 뺄셈
- 2. 곱셈
- 3. 나눗셈
- 4. 종료

=====

메뉴를 선택하시오:2
2개의 정수를 입력하시오:10 20
연산 결과 = 200

=====

- 0. 덧셈
- 1. 뺄셈
- 2. 곱셈
- 3. 나눗셈
- 4. 종료

=====

메뉴를 선택하시오:

함수 인수로서의 함수 포인터

- 함수 포인터도 인수로 전달이 가능하다.

특정 함수를 호출하게
할 수 있어요.



```
int main(void)
{
    ...
    sub(f);
    ...
}
```

```
int sub( void (*pf)() )
{
    ...
    pf();
    ...
}
```

```
void f()
{
    ...
}
```

예제

- 다음과 같은 수식을 계산하는 프로그램을 작성하여 보자.

$$\sum_1^n (f^2(k) + f(k) + 1)$$

- 여기서 $f(k)$ 는 다음과 같은 함수들이 될 수 있다.

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

예제

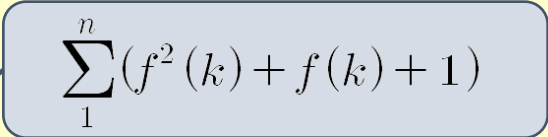
```
#include <stdio.h>
#include <math.h>

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;
}
```


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

예제

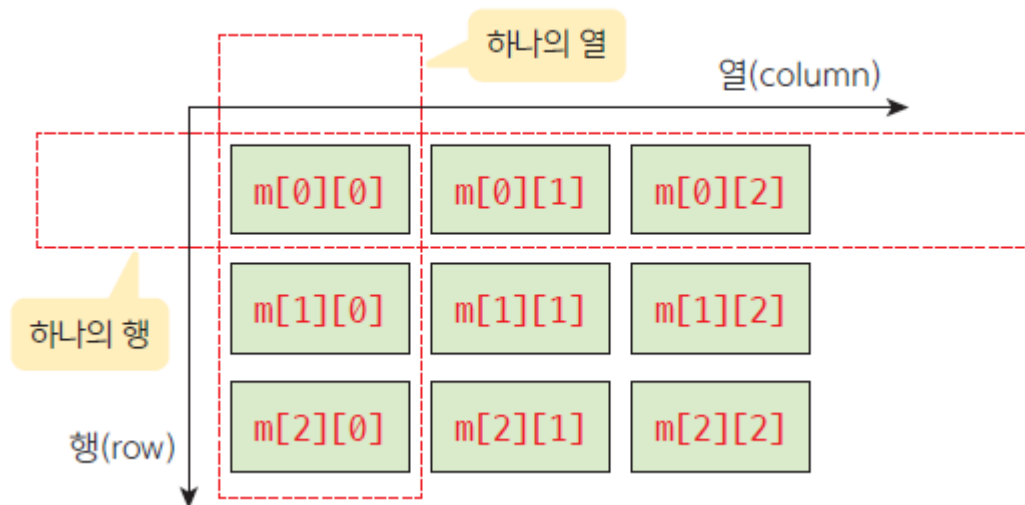
```
double f1(double k)
{
    return 1.0 / k;
}

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

13.368736
12.716152

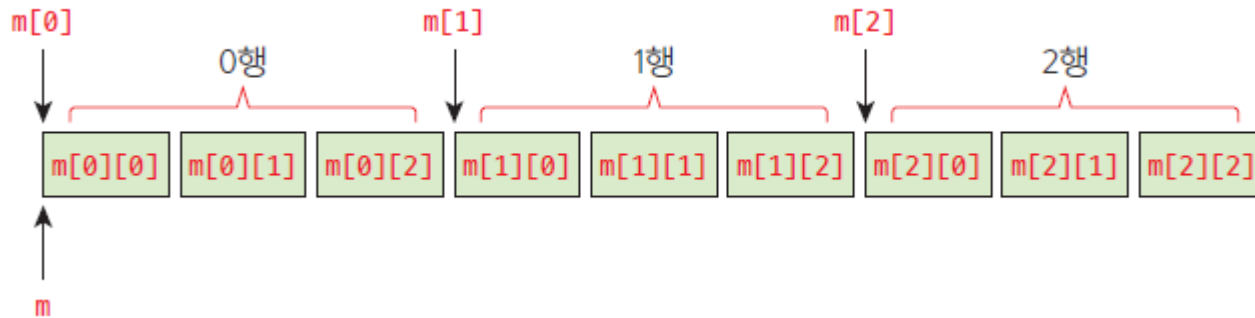
다차원 배열과 포인터

- 2차원 배열 `int m[3][3]`
- 1행->2행->3행->...순으로 메모리에 저장(행우선 방법)



행우선 저장 방법

- 첫 번째 방법은 행우선 방법(row-major)으로 행을 기준으로 하여서 2차원 배열을 메모리에 저장하는 방법이다. 즉 0번째 행을 먼저 저장한 후에 1번째 행에 속하는 요소들을 저장하는 방법이다

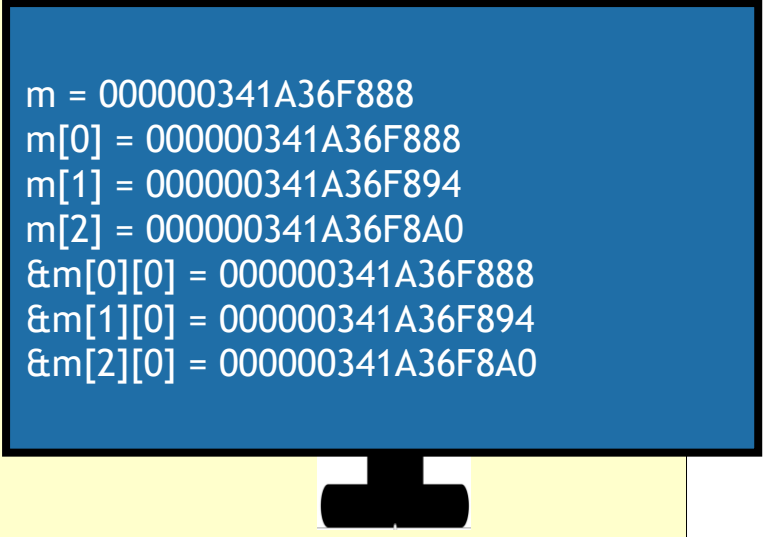


multi_array.c

```
#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]);

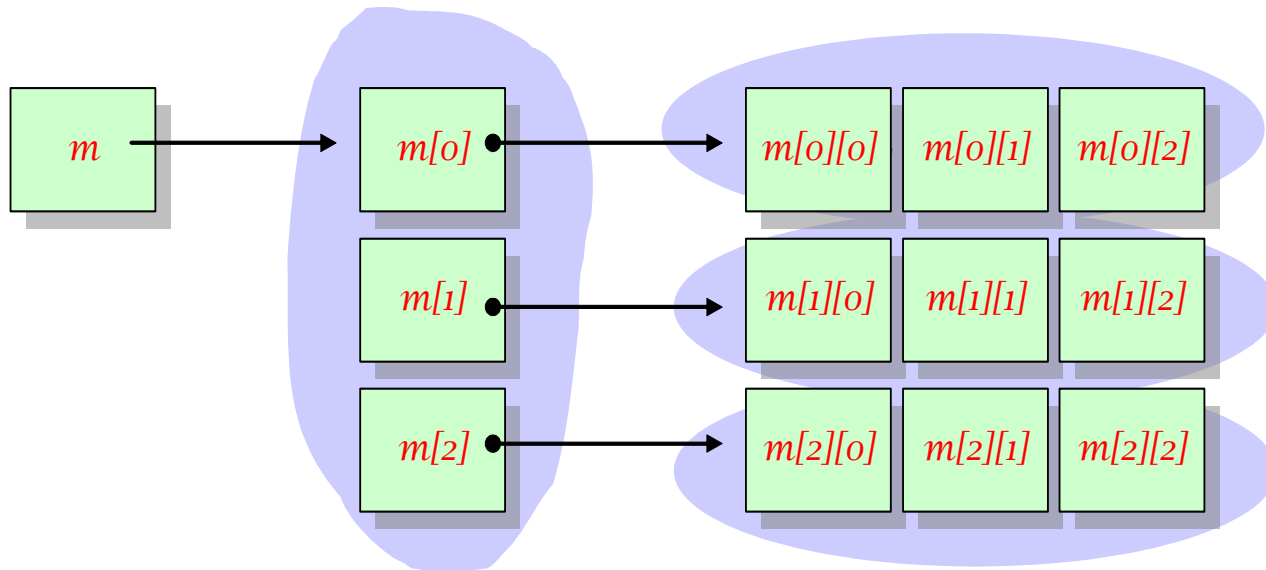
    return 0;
}
```



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

2차원 배열과 포인터

- 배열 이름 m 은 $\&m[0][0]$
- $m[0]$ 는 1행의 시작 주소
- $m[1]$ 은 2행의 시작 주소
- ...



2차원 배열의 해석

m은 세개의 원소를
가지는 배열이다.

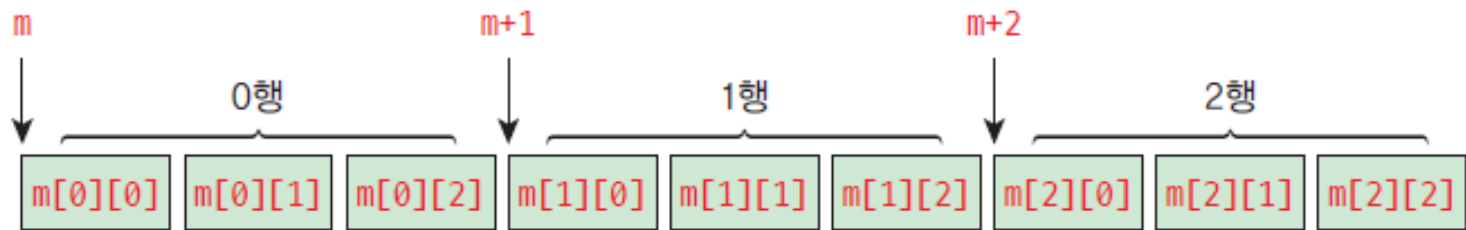
```
int m[3][3];
```

해석의 방향

그 원소들은 다시 세
개의 원소로 되어 있
다.

2차원 배열과 포인터 연산

- 2차원 배열 `m[][]`에서 `m`에 1을 더하거나 빼면 어떤 의미일까?



포인터를 이용한 배열 원소 방문

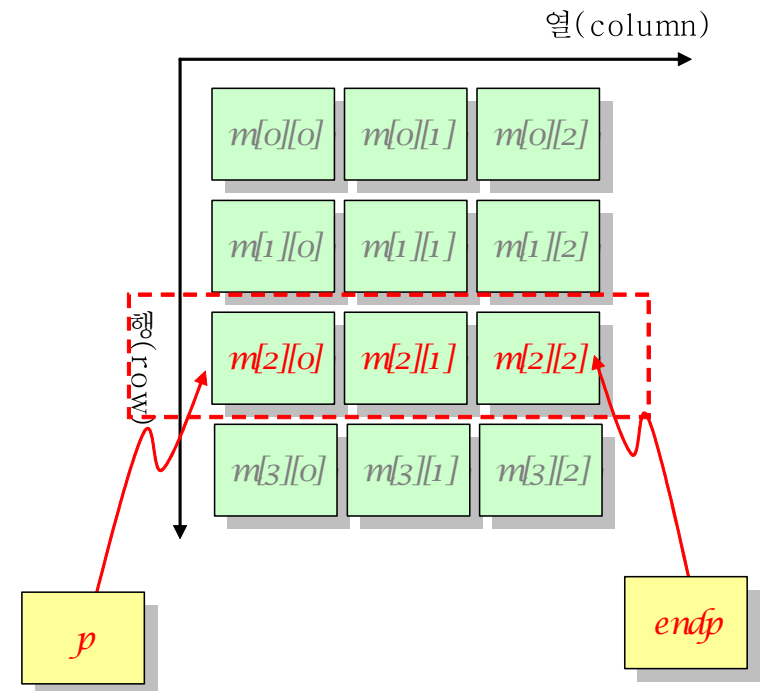
- 행의 평균을 구하는 경우

```
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;
}
```



포인터를 이용한 배열 원소 방문

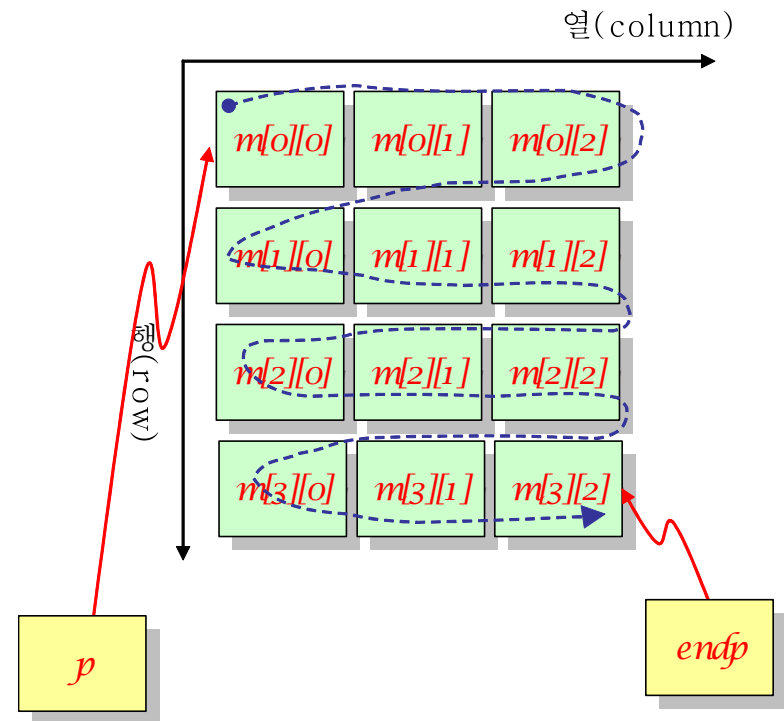
- 전체 원소의 평균을 구하는 경우

```
double get_total_avg(int m[][COLS])
{
    int *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;
}
```



중간 점검

1. $m[10][10]$ 에서 $m[0]$ 의 의미는 무엇인가?
2. $m[10][10]$ 에서 $(m+1)$ 의 의미는 무엇인가?



const 포인터

- const를 붙이는 위치에 따라서 의미가 달라진다.

p가 가리키는 내용이 변경되지 않음을 나타낸다.

The diagram shows the C code `const char *p;` inside a light blue rounded rectangle. The word `const` is highlighted in a yellow rounded rectangle. A black arrow points from the red text above to this yellow box. A red curved arrow points from the yellow box to the `*` before `p`, indicating that the content pointed to by `p` is constant.

포인터 p가 변경되지 않음을 나타낸다.

The diagram shows the C code `char *const p;` inside a light blue rounded rectangle. The word `const` is highlighted in a yellow rounded rectangle. A black arrow points from the red text above to this yellow box. A red curved arrow points from the yellow box to the `*` before `p`, indicating that the pointer variable `p` itself is constant.

예제

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

    //p[3] = 'a';
    p = t;
    q[3] = 'a';
    //q = t;

    return 0;
}
```

p가 가리키는 곳의 내용을 변경할 수 없다.

하지만 p는 변경이 가능하다.

q가 가리키는 곳의 내용은 변경할 수 있다.

하지만 q는 변경이 불가능하다.

volatile 포인터

- volatile은 다른 프로세스나 스레드가 값을 항상 변경할 수 있으니 값을 사용할 때마다 다시 메모리에서 읽으라는 것을 의미

p가 가리키는 내용이 수시로 변경되니 사용할 때마다 다시 로드하라는 의미이다.



```
volatile char *p;
```

void 포인터

- 순수하게 메모리의 주소만 가지고 있는 포인터(generic pointer)
- 가리키는 대상물은 아직 정해지지 않음
(예) `void *vp;`
- 다음과 같은 연산은 모두 오류이다.

```
*vp;           // 오류
*(int *)vp;    // void형 포인터를 int형 포인터로 변환한다.
vp++;          // 오류
vp--;          // 오류
```

void 포인터는 어디에 사용하는가?

- void 포인터를 이용하면 어떤 타입의 포인터도 받을 수 있는 함수를 작성할 수 있다. 예를 들어서 전달받은 메모리를 0으로 채우는 함수를 작성해보면 다음과 같다.

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

vp.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() 함수의 인수

- 지금까지의 main() 함수 형태

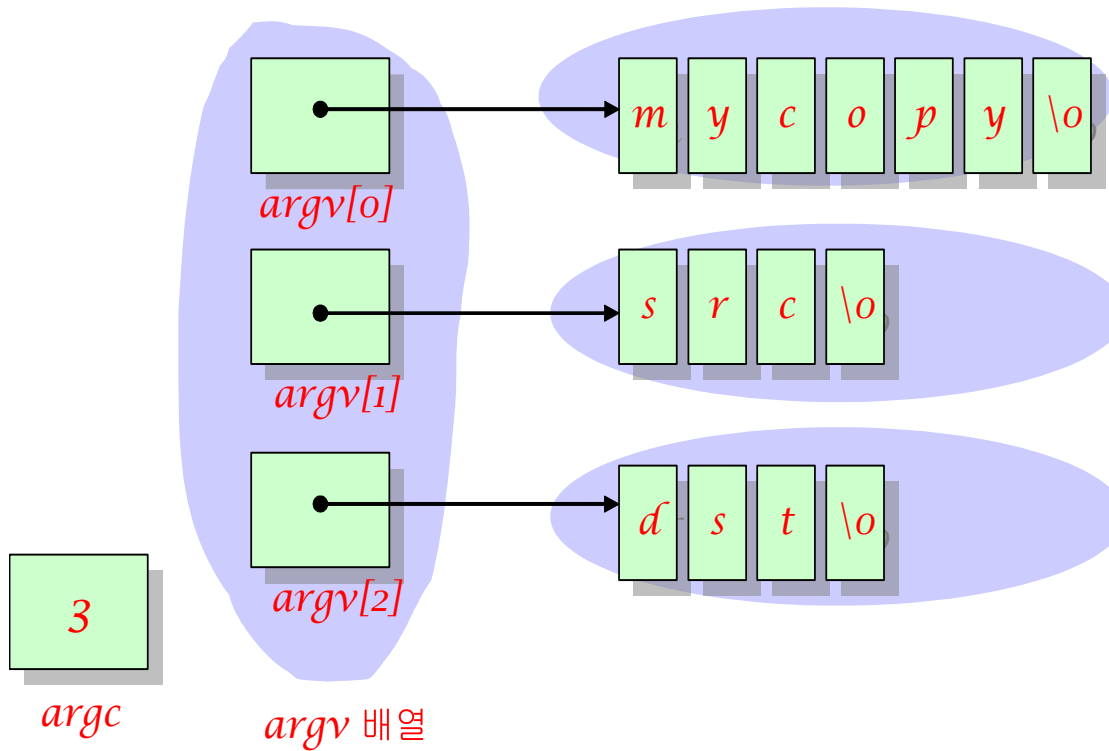
```
int main(void)
{
  ..
}
```

- 외부로부터 입력을 받는 main() 함수 형태

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

인수 전달 방법

```
C: \cprogram> mycopy src dst
```



main_arg.c

```
#include <stdio.h>

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

    for(i = 0; i < argc; i++)
        printf("명령어 라인에서 %d번째 문자열 = %s\n", i, argv[i]);

    return 0;
}
```

```
c:\cprogram\mainarg\Debug>mainarg src dst
명령어 라인에서 0번째 문자열 = mainarg
명령어 라인에서 1번째 문자열 = src
명령어 라인에서 2번째 문자열 = dst
c:\cprogram\mainarg\Debug>
```

중간 점검

1. C>main arg1 arg2 arg3와 같이 실행시킬 때 argv[0]가 가리키는 것은?
2. C>main arg1 arg2 arg3와 같이 실행시킬 때 argc의 값은?



Q & A

