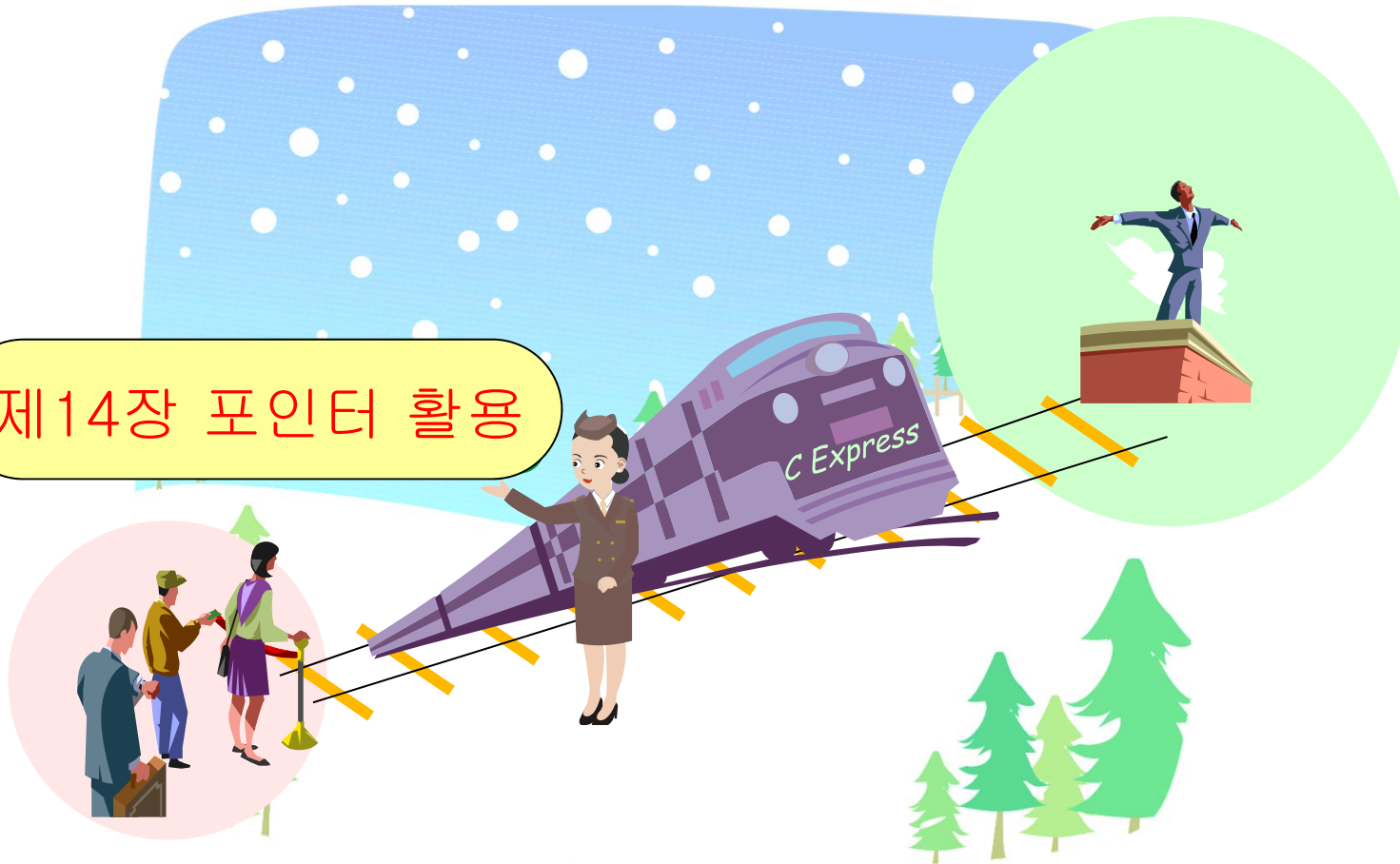




# 쉽게 풀어쓴 C언어 Express

## 제14장 포인터 활용

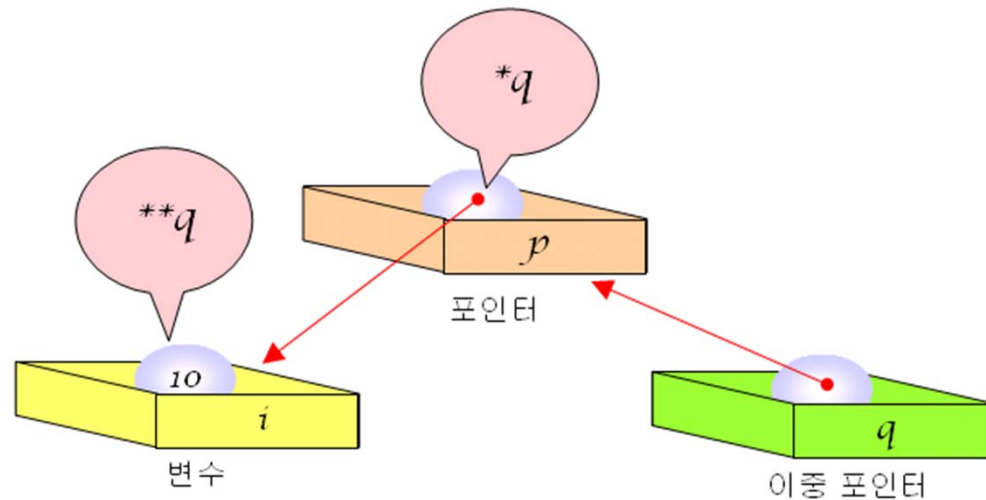




# 이중 포인터

- 이중 포인터(double pointer) : 포인터를 가리키는 포인터

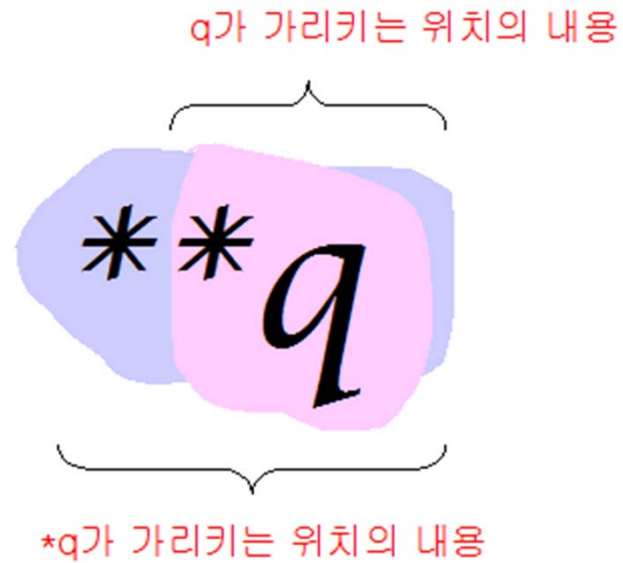
```
int i = 10;           // i는 int형 변수  
int *p = &i;         // p는 i를 가리키는 포인터  
int **q = &p;        // q는 포인터 p를 가리키는 이중 포인터
```





# 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 *p=%d **q=%d \n", i, *p, **q);
```

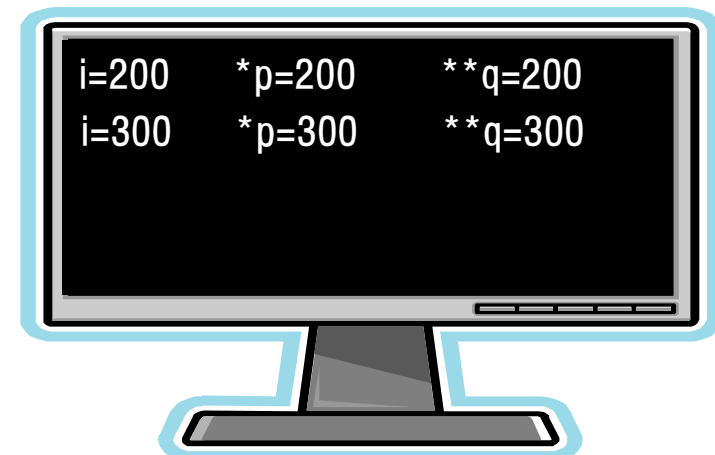
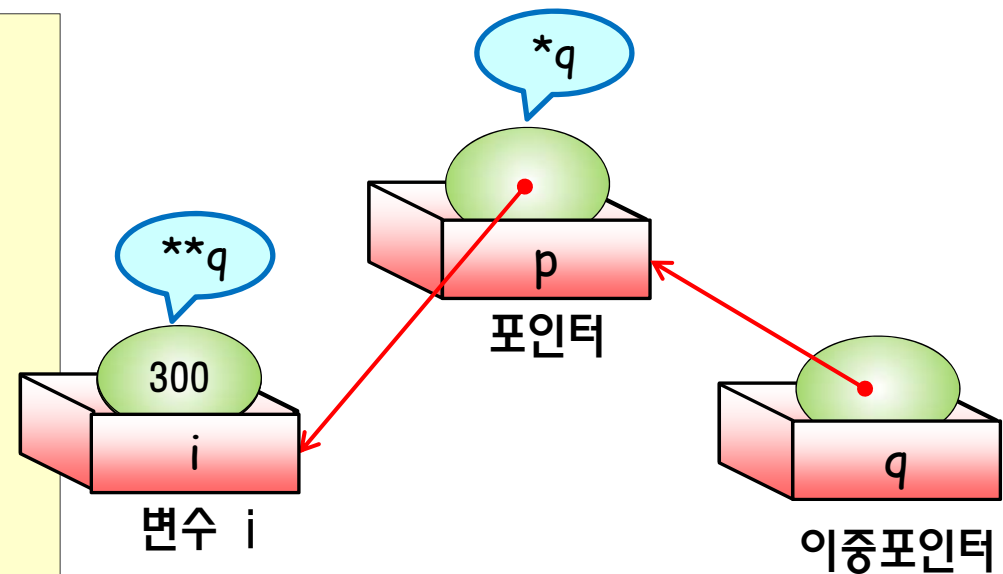
```
**q = 300;
```

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

```
return 0;
```

```
}
```

**\*\*q == \*(\*q)**





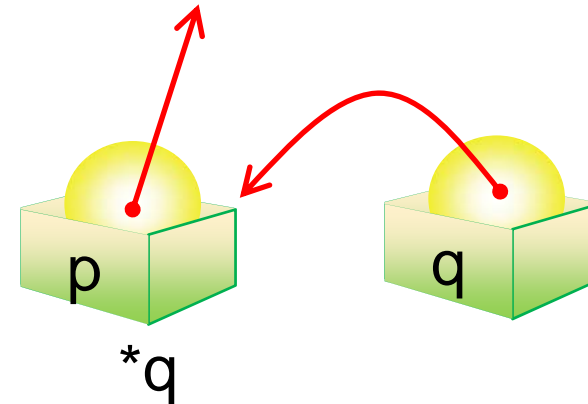
## 예제 #2

```
#include <stdio.h>
void set_pointer(char **q);
char *proverb="All that glisters is not gold.";

int main(void)
{
    char *p="zzz";
    set_pointer(&p);
    printf("%s \n", p);
    return 0;
}

void set_pointer(char **q)
{
    *q = proverb;
}
```

"All that glisters is not gold."



All that glisters is not gold.



## 중간 점검

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





# 포인터 배열

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

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

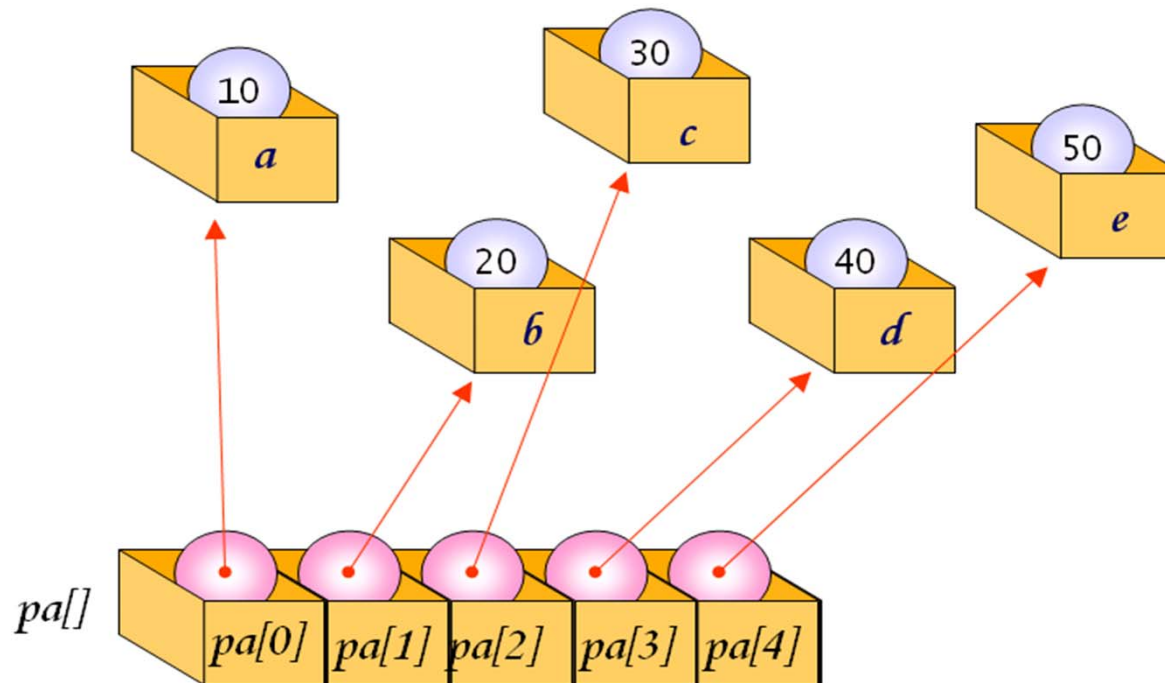
`int * ap [10];`

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"  
};
```

a p p l e \0

b l u e b e r r y \0

o r a n g e \0

m e l o n \0

2차원 배열을 사  
용하면 낭비되는  
공간이 생성되죠.

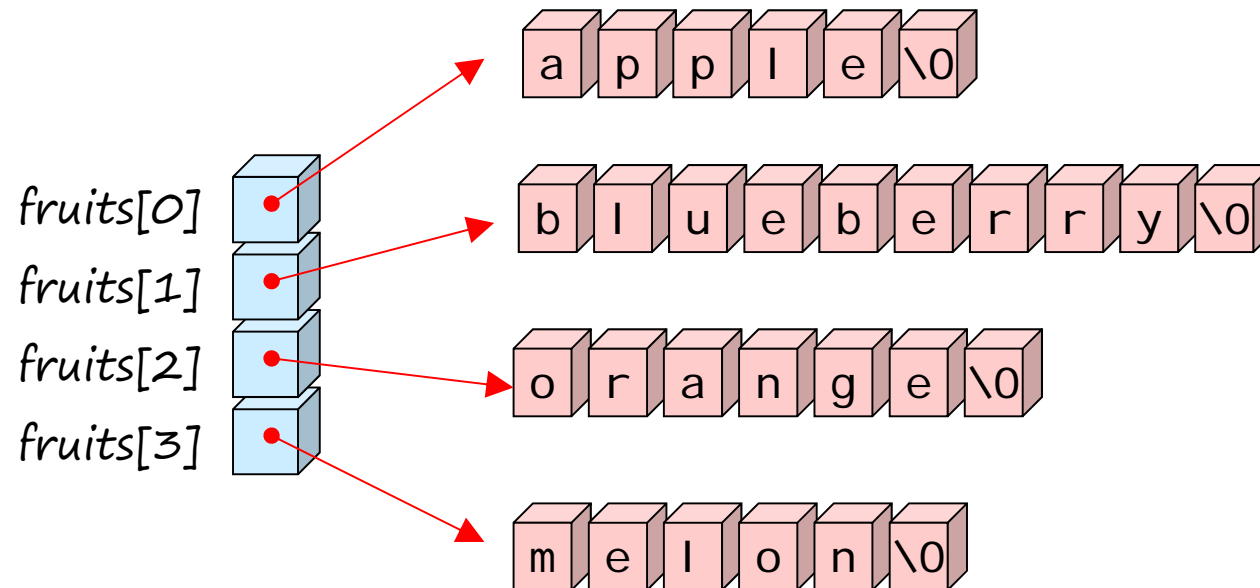
낭비되는  
공간!





# 문자형 포인터 배열

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





# 예제

// 문자열 배열

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

```
int main(void)
```

```
{
```

```
    int i, n;
```

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

```
    // 배열 원소 개수 계산
```

```
    n = sizeof(fruits)/sizeof(fruits[0]);
```

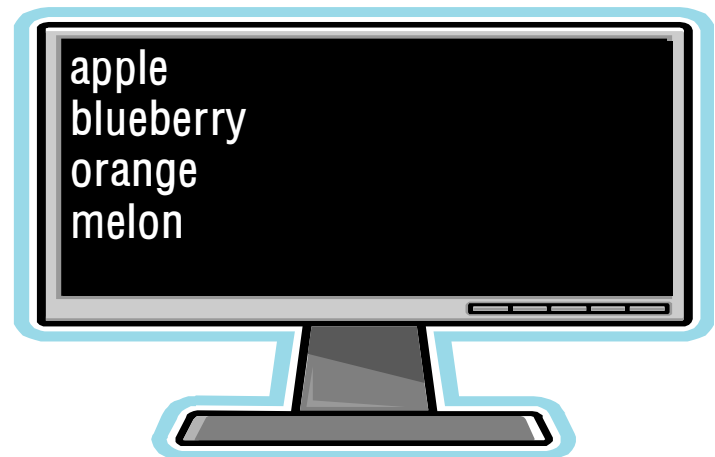
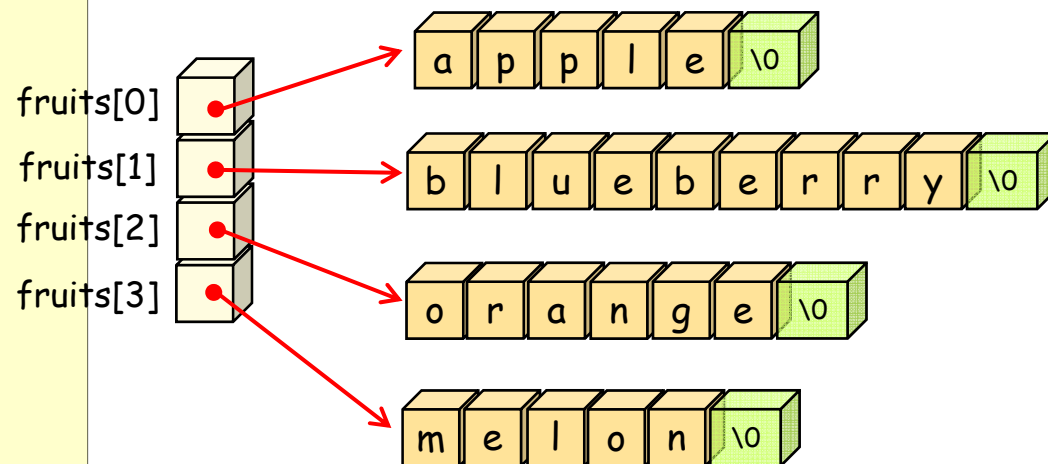
```
    for(i = 0; i < n; i++)
```

```
        printf("%s \n", fruits[i]);
```

```
    return 0;
```

```
}
```

각각의 문자열의 길이가 달라도 메모리의 낭비가 발생하지 않는다.





## 중간 점검

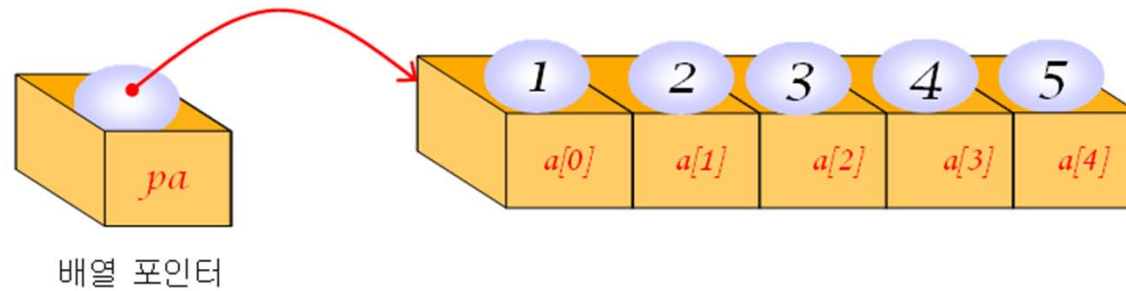
- double형의 포인터 10개를 가지는 배열을 정의하여 보자.
- 래그드 배열이 일반적인 2차원 배열보다 좋은 점은 무엇인가?





# 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.



# 배열 포인터 (의미)

```
#include <stdio.h>

void main(void)
{
    int i_array[10];
    int *p_i;
    int (*p_i10)[10];

    p_i=i_array;
    p_i10=&i_array;

    printf("%d\n",sizeof(i_array));
    printf("%d\n",sizeof(p_i));
    printf("%d\n",sizeof(*p_i));
    printf("%d\n",sizeof(p_i10));
    printf("%d\n",sizeof(*p_i10));

    i_array[3]=33;
    printf("%d\n",(*p_i10)[3]);
    printf("%d\n",p_i[3]);
}
```

```
C:\Windows\system32\cmd.exe

40
4
4
4
40
33
33
계속하려면 아무 키나 누르십시오 . . .
```



# 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;
}
```

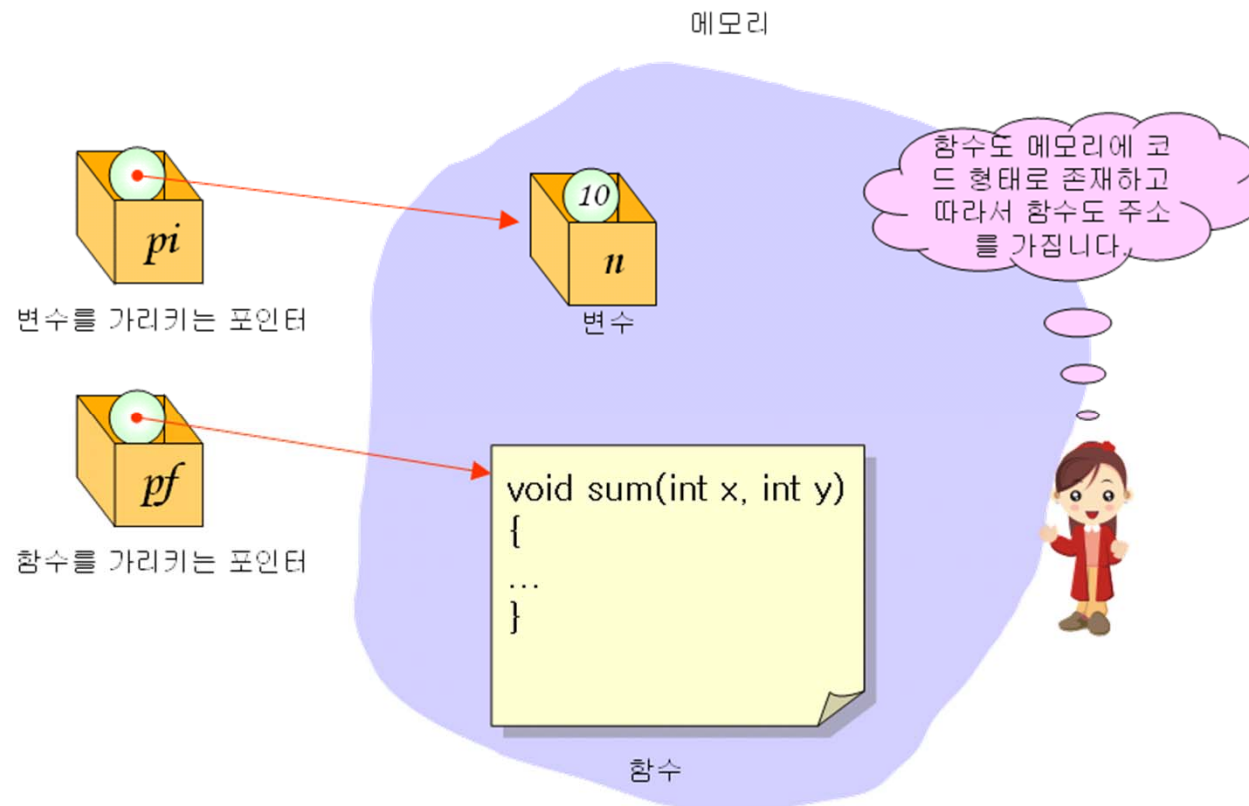




# 함수 포인터

- 함수 포인터(function pointer): 함수를 가리키는 포인터

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







# 함수 포인터의 해석

① 괄호에 의하여 () 연산자보다 \* 연산자가 먼저 적용되어서 pf는 포인터가 된다.

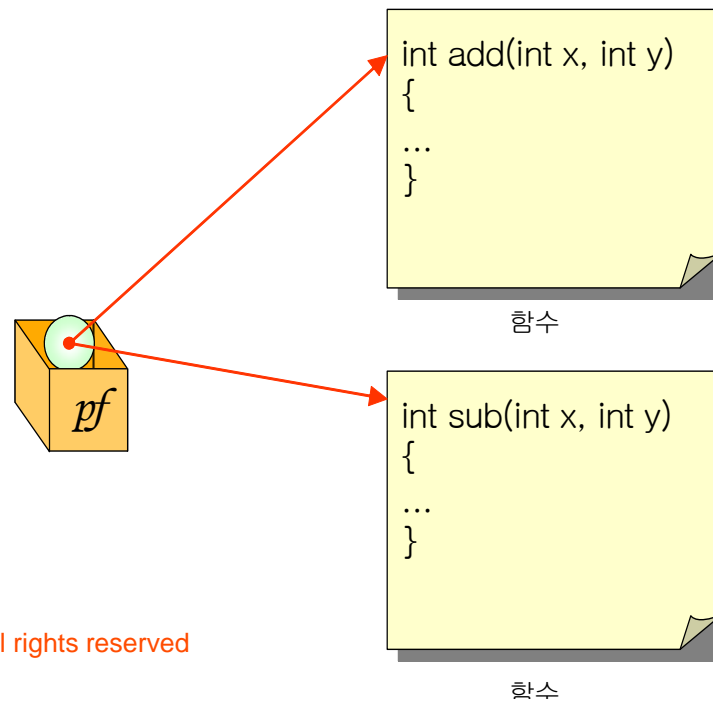
*int* (\*pf) (*int*, *int*);

② 어떤 포인터냐 하면 int f(int, int) 함수를 가리키는 포인터가 된다.



# 함수 포인터의 사용

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





# 예제

```
// 함수 포인터
#include <stdio.h>

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

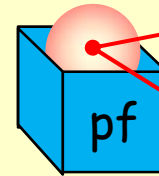
int main(void)
{
    int result;
    int (*pf)(int, int); // 함수 포인터 정의

    pf = add;
    result = pf(10, 20);
    printf("10+20은 %d\n", result);

    pf = sub;
    result = pf(10, 20);
    printf("10-20은 %d\n", result);

    return 0;
}
```

함수의 이름은 배열의 이름과 마찬가지로 함수의 시작주소를 나타내는 포인터 상수로 간주된다.



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

함수

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

함수



## 예제

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

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





# 함수 포인터 (의미)

```
#include <stdio.h>

int add(int a, int b)
{
    return a+b;
}

void main(void)
{
    int (*pf)(int,int);

    pf=add;

    printf("%d %d\n",pf(1,3),(*pf)(1,3));
}
```

C:\Windows\system32\cmd.exe

```
4 4
계속하려면 아무 키나 누르십시오 . . .
```



# 함수 포인터의 배열

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

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

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

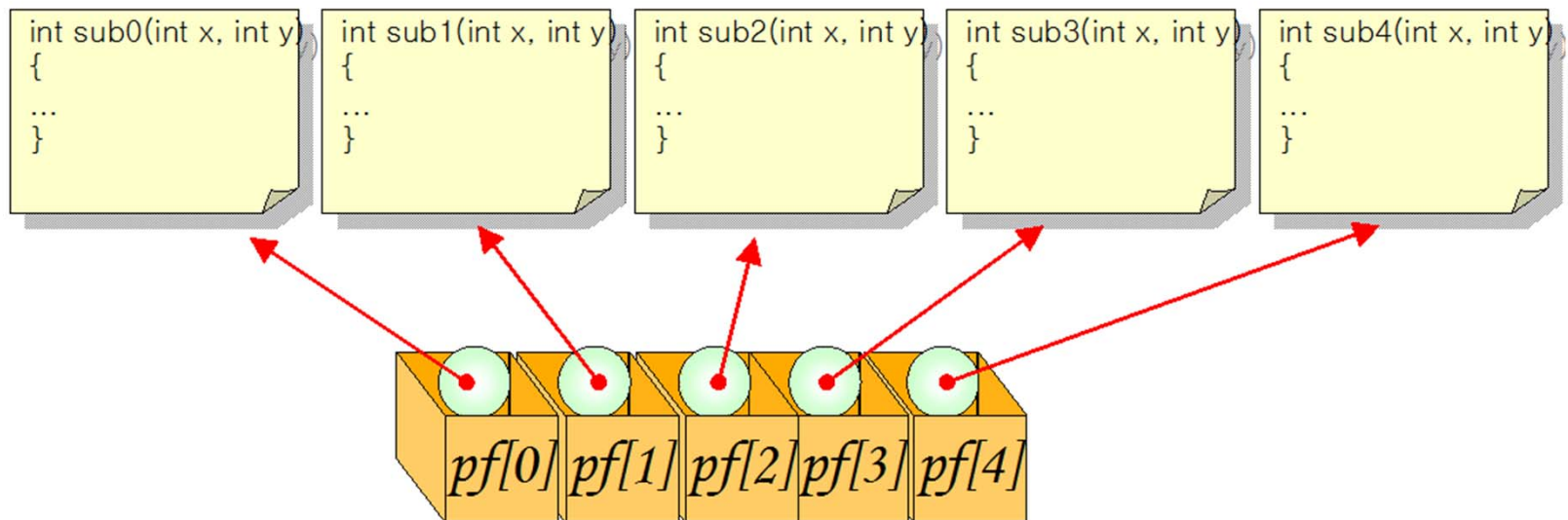
② 어떤 배열이냐 하면 i포인터들의 배열이 된다.

③ 어떤 포인터냐 하면 함수를 가리키는 포인터가 된다.



# 함수 포인터의 배열

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





## 예제: 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");
}
```





# fp2.c

```
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);

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

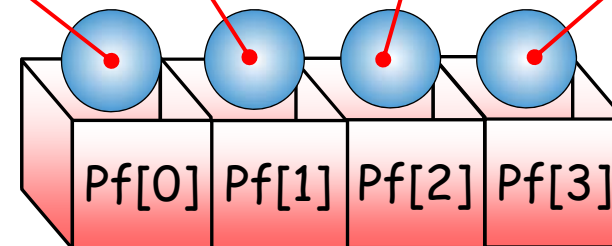
함수 포인터 배열 선언

```
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;
}
```





# fp2.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;
}

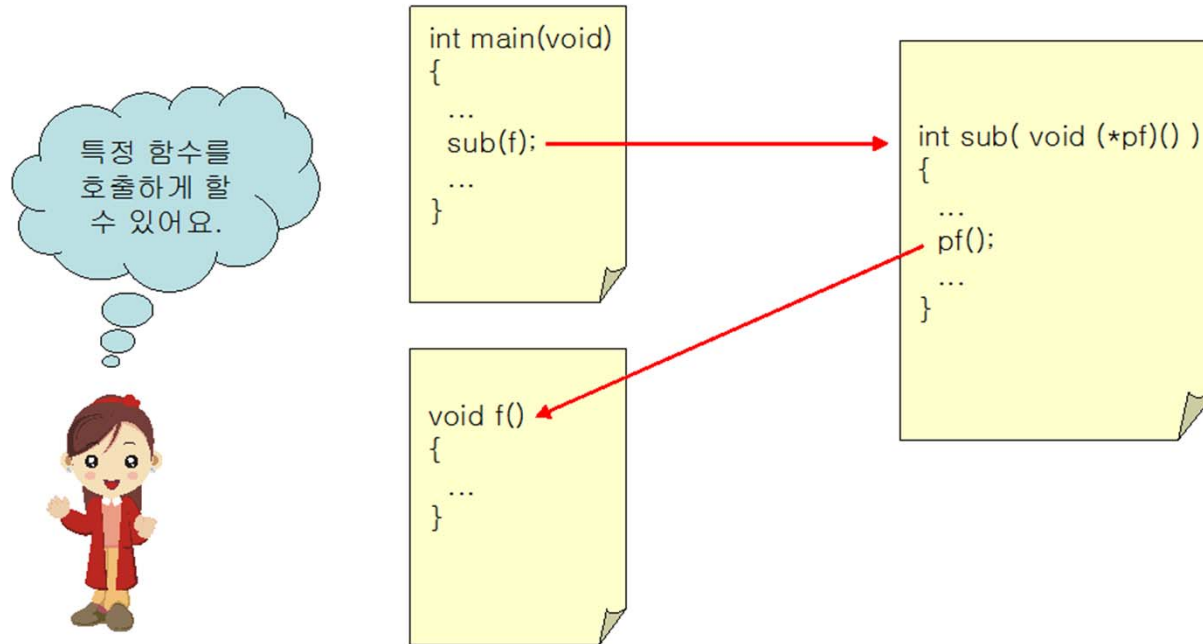
int div(int x, int y)
{
    return x / y;
}
```

```
=====
0. 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.





## 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);
}
```





## 중간 점검

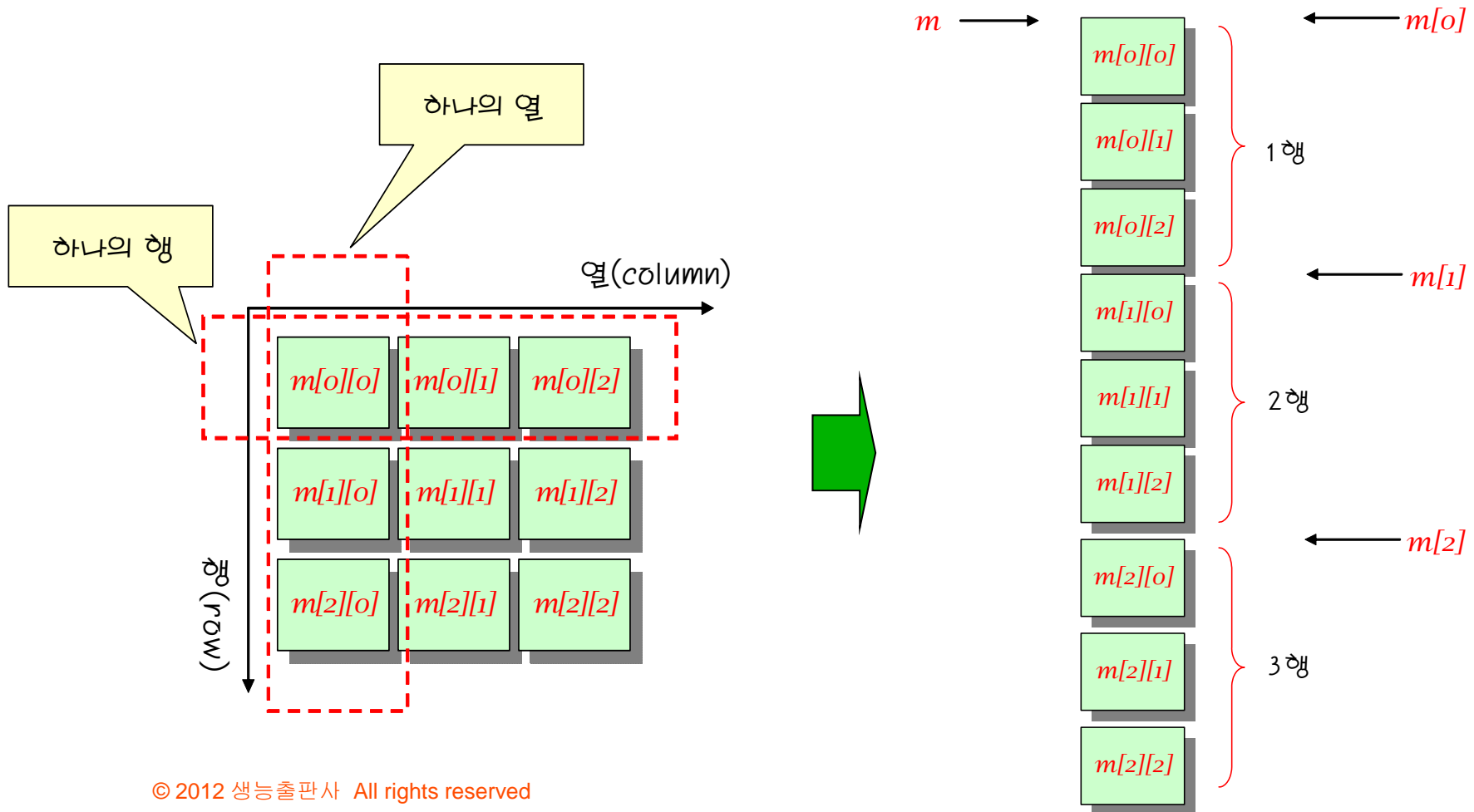
- int 값을 반환하고 double 값을 인수로 받는 함수의 포인터 pf를 선언하여 보자.
- 1번의 함수 포인터를 통하여 3.0을 인수로 하여 함수를 호출하는 문장을 작성하라.





# 다차원 배열과 포인터

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

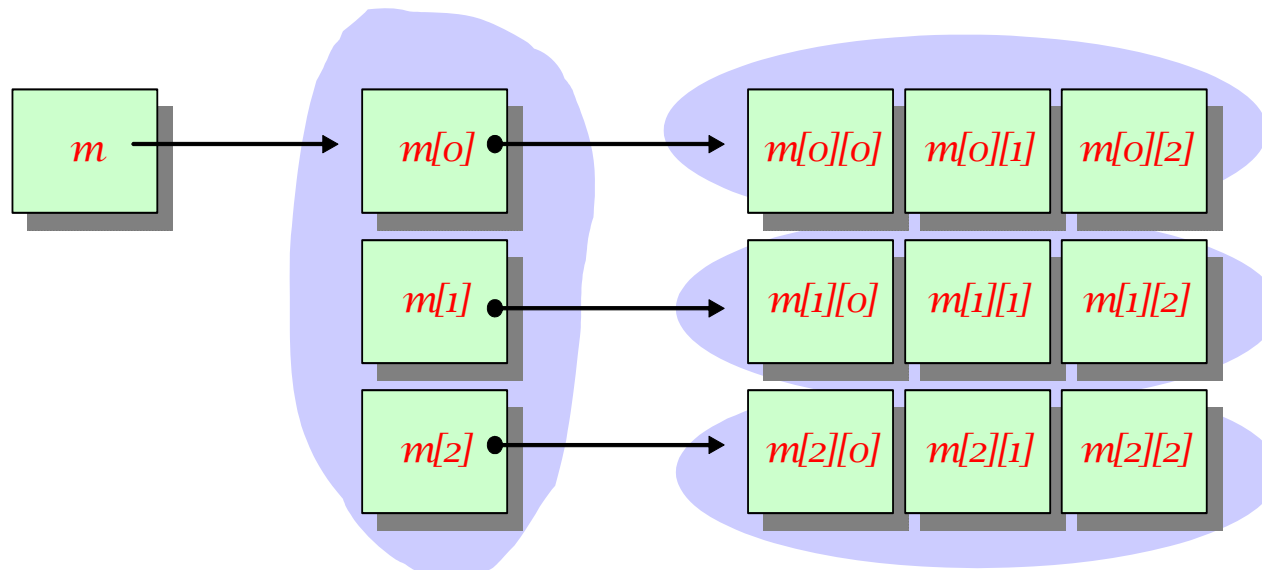






## 2차원 배열과 포인터

- 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  
세개의 원소로 되  
어 있다.



# multi\_array.c

```
// 다차원 배열과 포인터
```

```
#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]);
```

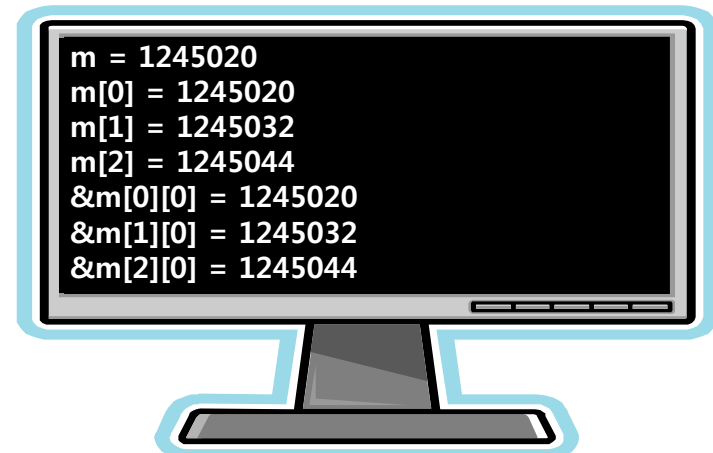
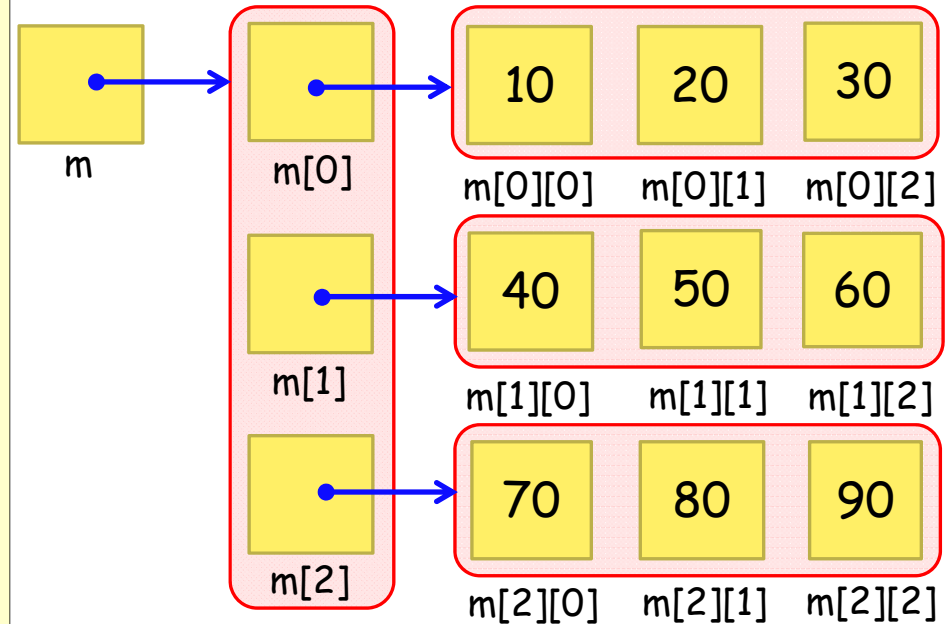
```
    printf("&m[0][0] = %p\n", &m[0][0]);
```

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

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

```
    return 0;
```

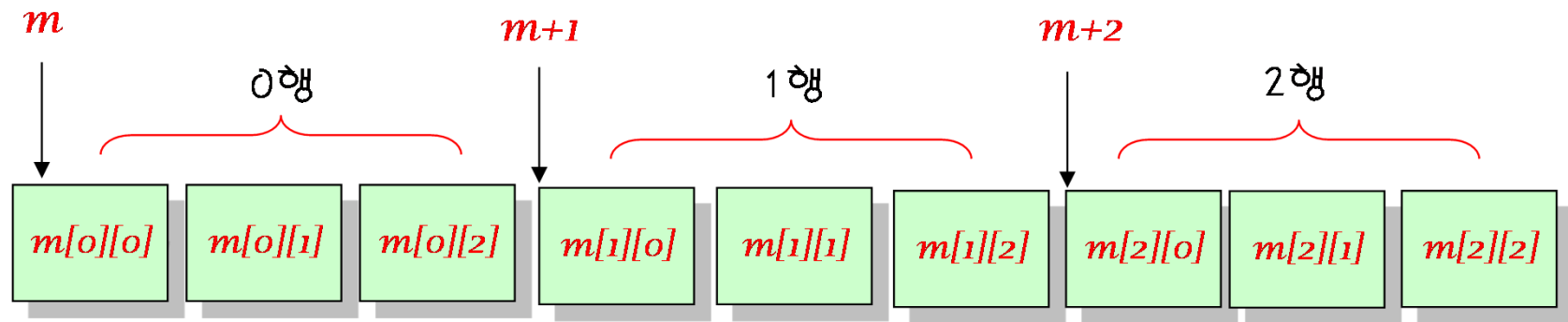
```
}
```





## 2차원 배열과 포인터 연산

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





# 포인터를 이용한 배열 원소 방문

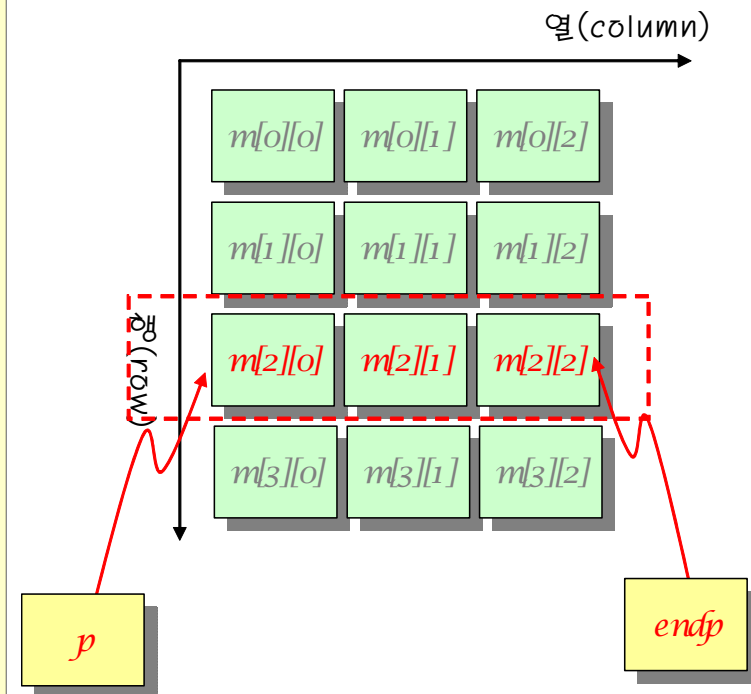
- 행의 평균을 구하는 경우

```
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

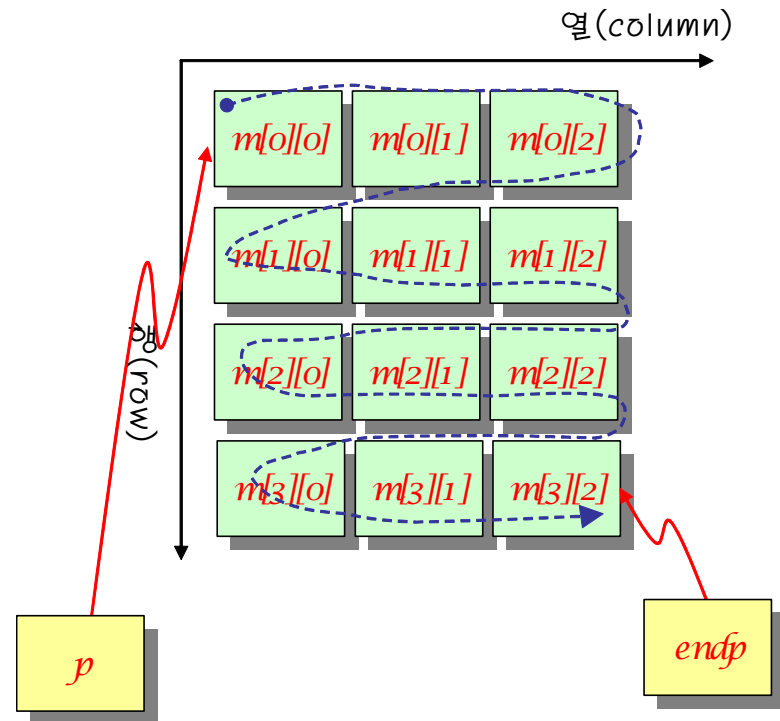
- 전체 원소의 평균을 구하는 경우

```
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

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





# const 포인터

- const를 붙이는 위치에 따라서 의미가 달라진다.

p가 가리키는 내용이 변경되지 않음을 나타낸다.

*const char \*p;*

포인터 p가 변경되지 않음을 나타낸다.

*char \*const p;*





# 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';
```

p가 가리키는 곳의 내용을 변경할 수 없다.

```
    p = t;
```

하지만 p는 변경이 가능하다.

```
    q[3] = 'a';
```

q가 가리키는 곳의 내용은 변경할 수 있다.

```
    //q = t;
```

하지만 q는 변경이 불가능하다.

```
    return 0;
```

```
}
```



# 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
- 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
```





# vp.c

```
#include <stdio.h>

int main(void)
{
    int a[] = { 10, 20, 30, 40, 50 };
    void *vp;

    vp = a;           // 가능
    vp = &a[2];        // 가능

    //*vp = 35;         // 오류
    //vp++;             // 오류

    *(int *)vp = 35;   // 가능

    return 0;
}
```



## 중간 점검

- void형 포인터 vp를 int형 포인터 ip로 형변환하는 문장을 작성하라.





# main() 함수의 인수

- 지금까지의 main() 함수 형태

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

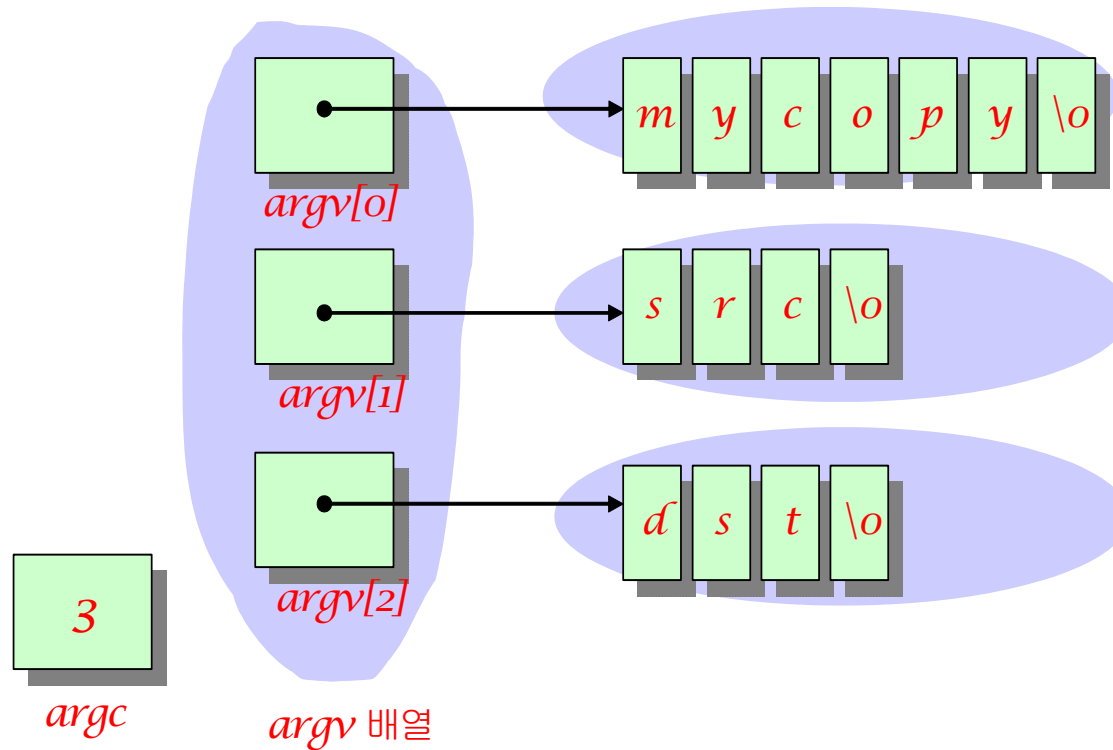
- 외부로부터 입력을 받는 main() 함수 형태

```
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;
}
```



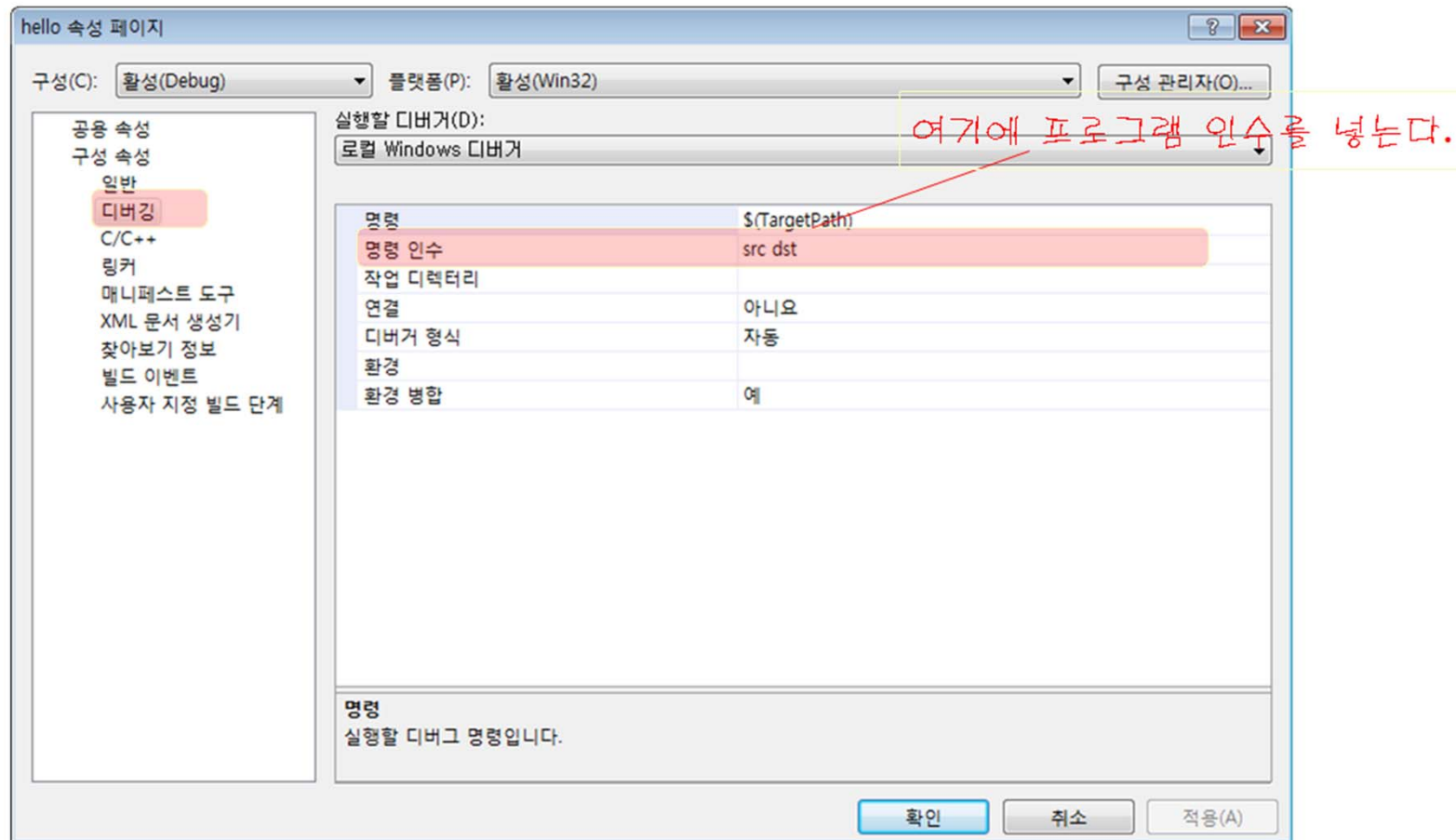
```
c:\cprogram\mainarg\Debug>mainarg src dst
명령어 라인에서 0번째 문자열 = mainarg
명령어 라인에서 1번째 문자열 = src
명령어 라인에서 2번째 문자열 = dst
c:\cprogram\mainarg\Debug>
```





# 비주얼 C++ 프로그램 인수 입력 방법

- [프로젝트]->[main\_arg.exe 속성] 선택





# mile2km.c

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

int main(int argc, char *argv[])
{
    double mile, km;

    if( argc != 2 ){
        printf("사용 방법: mile2km 거리\n");
        return 1;
    }
    mile = atof(argv[1]);
    km = 1.609 * mile;
    printf("입력된 거리는 %f km입니다. \n", km);

    return 0;
}
```



```
c:\cprogram\mainarg\Debug>mainarg 10
입력된 거리는 16.090000 km입니다.
c:\cprogram\mainarg\Debug>
```



## 중간 점검

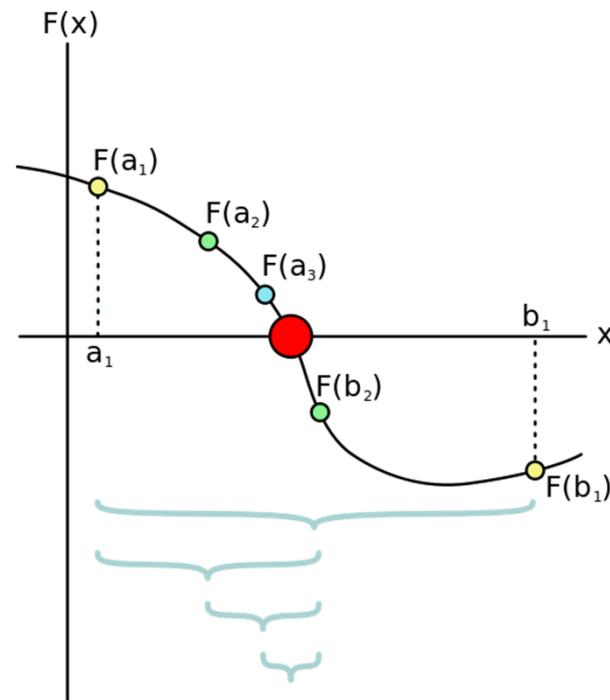
- C>main arg1 arg2 arg3와 같이 실행시킬 때 argv[0]가 가리키는 것은 ?
- C>main arg1 arg2 arg3와 같이 실행시킬 때 argc의 값은?





# 실습: 이분법으로 근 구하기

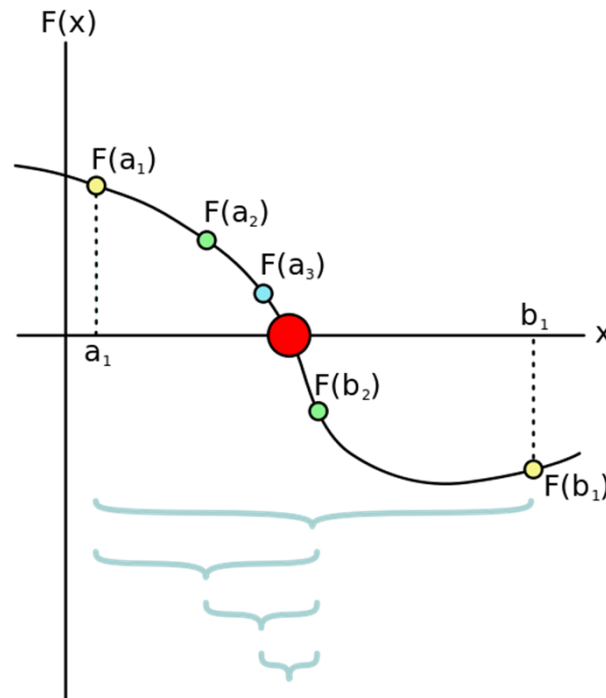
- 2차식의 경우에는 공식이 있지만 일반적인  $n$ 차식의 경우 공식이 존재하지 않는다.
- 이때 사용할 수 있는 방법이 이분법(bisection)이다





# 이분법

- 구간  $[a, b]$ 에서 근을 가지는 것이 확실하면 구간  $[a, b]$ 의 중점  $m$ 을 구하여 구간  $[a, m]$ 과 구간  $[m, b]$ 로 나눈다.
- 각각의 구간에서 다시  $f(a)$ 와  $f(b)$ 의 부호를 조사하여 근이 어떤 구간에 위치하는지를 결정한다. 다시 그 구간에 대하여 동일한 과정을 되풀이한다.





## 실행 결과

- 다음과 같은 함수에 대하여 근을 구한다.

$$f(x) = x^3 + x^2 + x + 7$$

- 함수 포인터를 받는 `get_root()`를 작성한다.
- `double get_root(double (*fp)(double), double a, double b);`



a의 값을 입력하시오 -200  
b의 값을 입력하시오 200  
값은 -2.104950



## 예제 소스

```
#include<stdio.h>
#include <math.h>
#define ESP 0.001
double get_root(double (*f)(double), double a, double b);
double func(double x)
{
    return (x)*(x)*(x) + (x)*(x) + (x) + 7;
}
void main()
{
    double x0,x1;
    double r;
    printf("a의 값을 입력하시오");
    scanf("%lf",&x0);
    printf("b의 값을 입력하시오");
    scanf("%lf",&x1);
    r = get_root(func, x0, x1);
    printf("값은 %f\n", r);
    return 0;
}
```



## 예제 소스

```
double get_root(double (*f)(double), double x0, double x1)
{
    float x2;
    int i = 1;
    double f1, f2, f0;
    do
    {
        x2=(x0+x1)/2;
        f0=f(x0);
        f1=f(x1);
        f2=f(x2);
        if(f0*f2<0)
            x1=x2;
        else
            x0=x2;
        i++;
    } while(fabs(f2)>ESP);

    return x2;
}
```

부호가 다르면

f2의 절대값이 무시할 만큼 작아지면, 즉 0에 가까워지면





# 도전문제

- 사용자로부터 함수를 입력받도록 프로그램을 수정할 수 있는가? 다항식의 최고차수는 3으로 제한하도록 하자.





# Q & A

