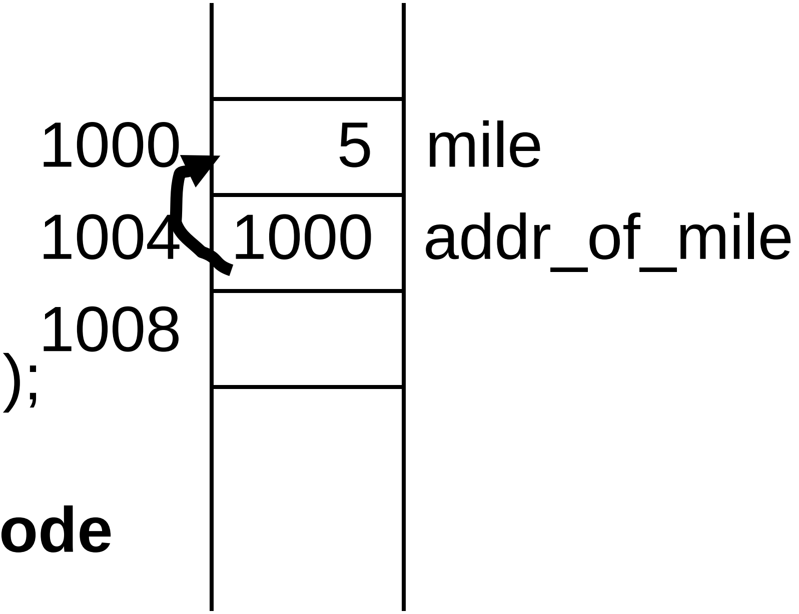
Arrays & Pointers

Pointer

- **Pointer variable**
  => variable to hold address of a memory

```
int    mile = 5;
int    *addr_of_mile;

addr_of_mile = &mile;
printf("%d", *addr_of_mile);
```

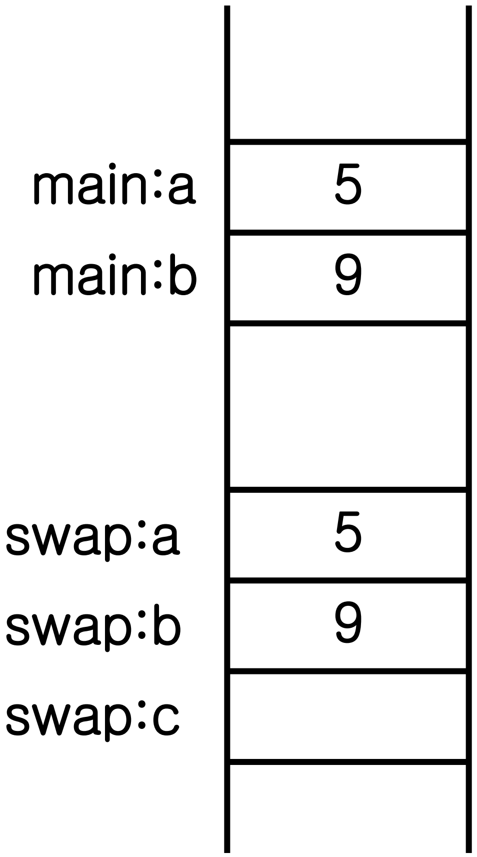


Indirect addressing mode

This is wrong

```
int  main(void)  {
    int  a = 5;
    int  b = 9;
    swap(a, b);
    printf(“%d  %d”, a, b);
}

void swap (int a, int b)  {
    int c = a;
    a = b;
    b = c;
}
```



main:a 5

main:b 9

swap:a 5

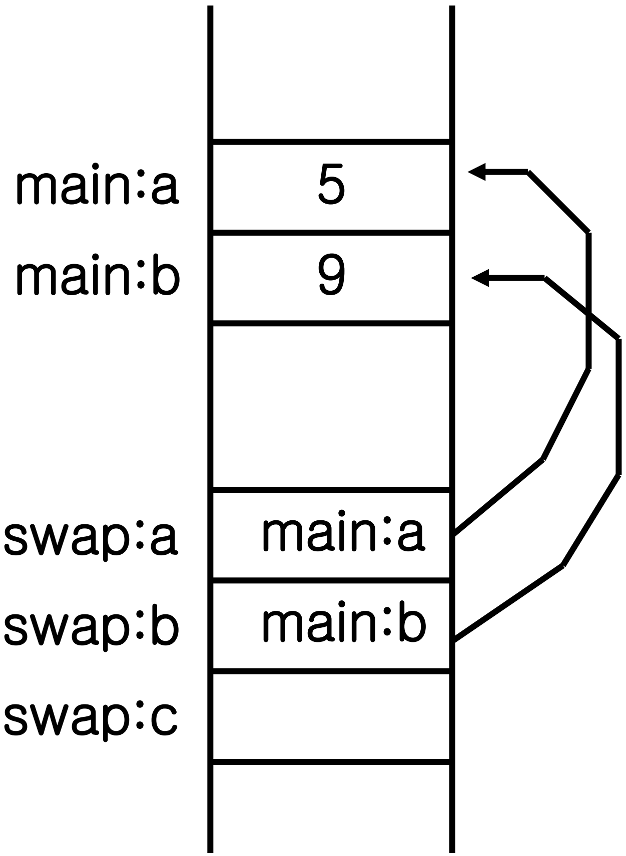
swap:b 9

swap:c

This is okay

```
int  main(void)  {
     int  a = 5;
     int  b = 9;
     swap(&a, &b);
     printf(“%d  %d”, a, b);
}

void swap (int *a, int *b)  {
     int c = *a;
     *a = *b;
     *b = c;
}
```



Why Pointer ?

- **Function arguments pass by value**
  - **All the function arguments are input to the function.**
  - Return value (output) is only one.
  - What would you do if you **need 2 or more outputs**?
- **Dynamic memory allocation**
- **Passing arguments of massive structure**

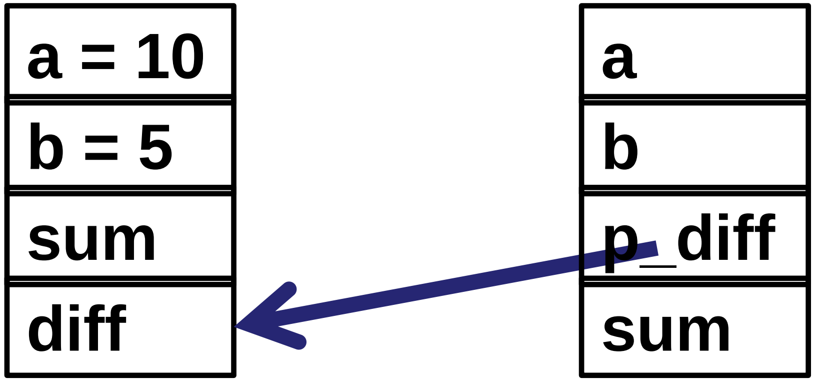
Function Arguments

- 두 정수의 합(sum)과 차이(diff)를 구하기

```
int main(vodi)
{
    int a = 10, b = 5;
    int sum, diff;

    sum = add_sub(a, b, &diff)
    printf(“%d %d”, sum, diff);
}
```

```
int add_sub(int a, int b, int *p_diff)
{
    int sum = a + b;
    *p_diff = a –b;
    return sum;
}
```



a = 10
b = 5
sum
diff

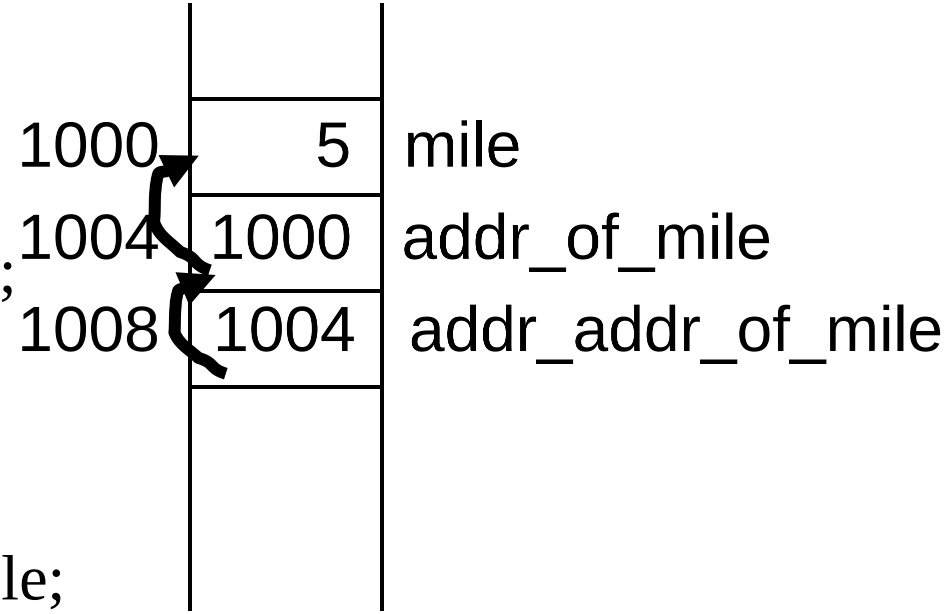
a	10
b	5
p_diff	&diff
sum	

when called

Double Pointer

- Double Pointer variable

  => variable to hold address of address of a memory



```
int    mile = 5;
int    *addr_of_mile;
int    **addr_addr_of_mile;

addr_of_mile = &mile;
addr_addr_of_mile
              = &addr_of_mile;
printf(“%d”, **addr_addr_of_mile);
```

Tripple pointer, quad … etc is the same.

Double Pointer

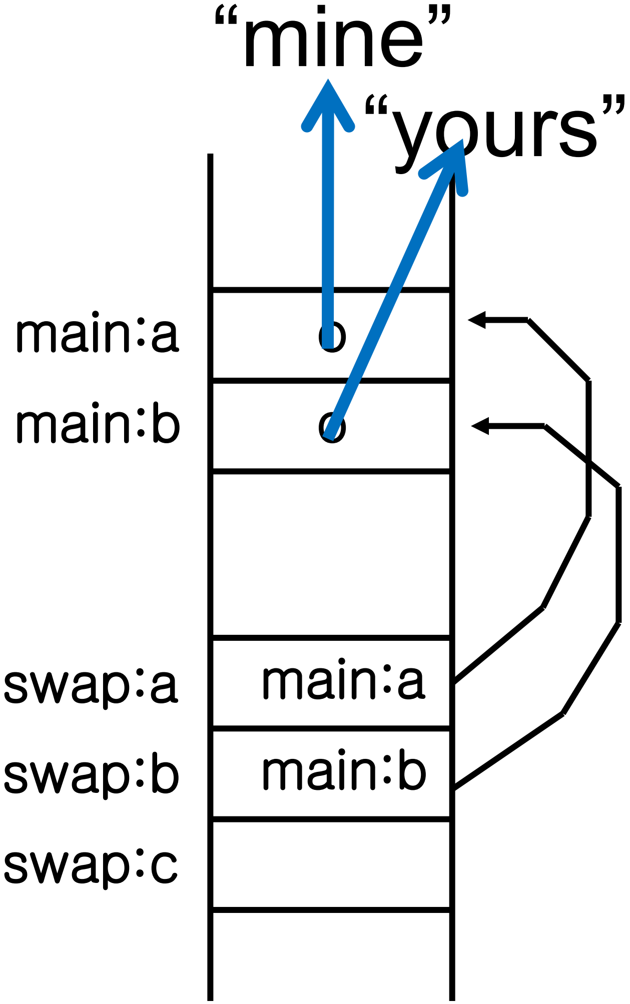
- Typical application is a modification of pointer variable within a function.
- For example, consider a function that swaps two strings
- swap(s1, s2)

- Another : array of arrays

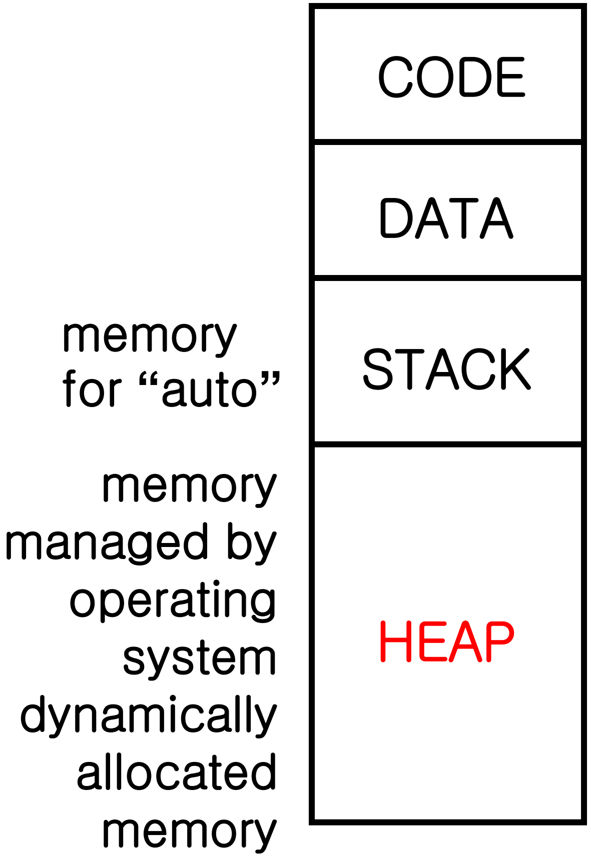
Swap strings

```
int  main(void)  {
     char *a = “mine”;
     char *b = “yours”;
     swap(&a, &b);
     printf(“%s  %s”, a, b);
}

void swap (char **s1, char **s2)  {
     char *str = *s1;
     *s1 = *s2;
     *s2 = str;
}
```



Dynamic Memory



memory for program codes

memory for "static" and "extern"

```
void myfunc (int a[], int size)  {
     int  k;
     int *array = (int *)malloc(size*sizeof(int));
     assert(array);
     for(k = 0 ; k < size ; k++)
          array[k] = a[k];
     free(array);
}
```

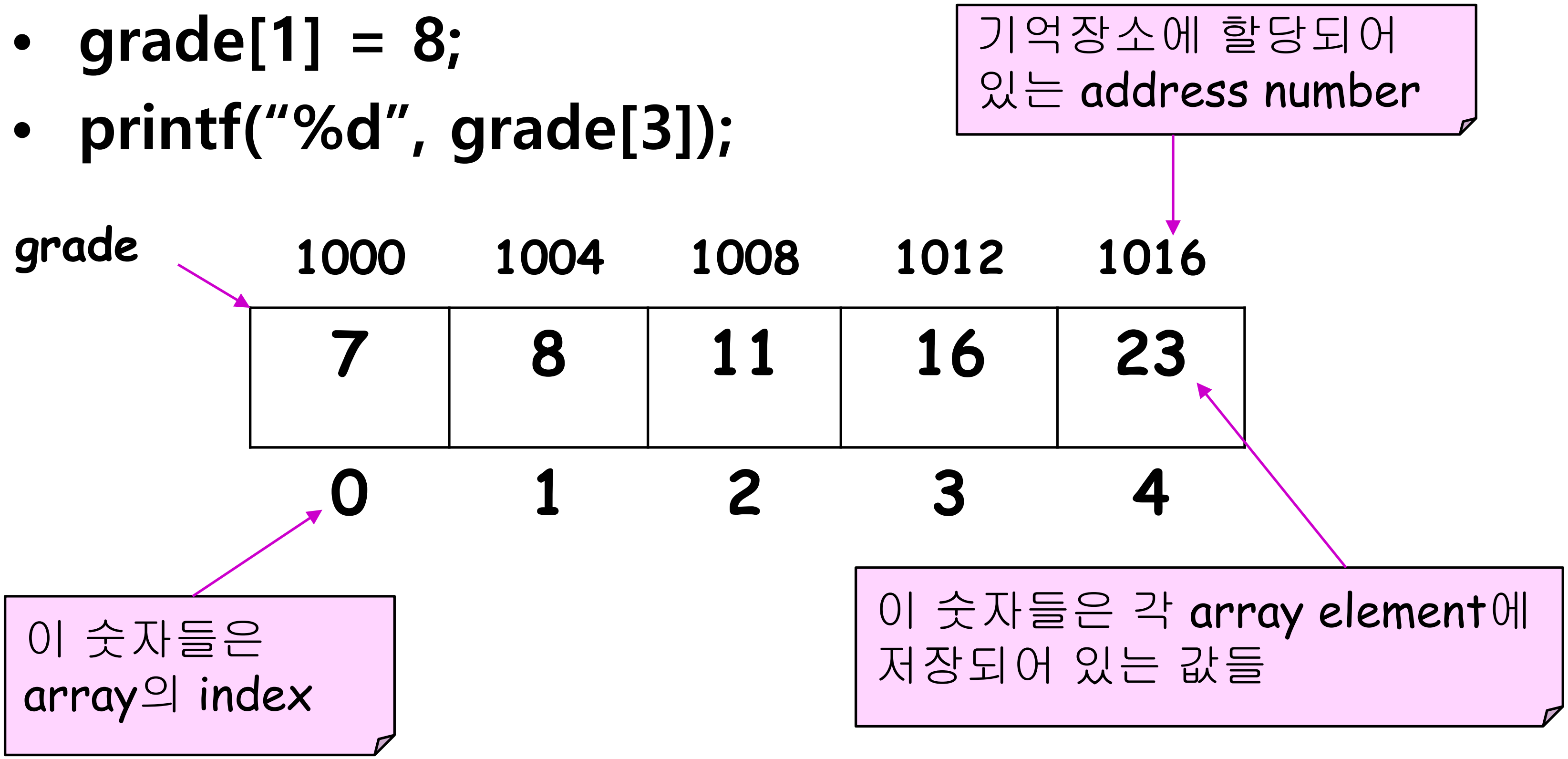
Dynamic Memory

- Use **“void *malloc(size_t size**);” to get memory from operating system.
- Use“**void free(void *ptr);**” to return the memory to operating system.
- **“malloc()” and “free()” should be a pair,** i.e, all the memory obtained from operating system should be returned to operating system after use.

Array

Consecutive memories of the Same Type

- int grade[5];
- grade[1] = 8;
- printf(“%d”, grade[3]);



기억장소에 할당되어 있는 address number

grade

	1000	1004	1008	1012	1016
grade	7	8	11	16	23
	0	1	2	3	4

이 숫자들은 array의 index

이 숫자들은 각 array element에 저장되어 있는 값들

Arrays and Pointers

int array1[50]; 상수 data를 담을 공간이 있음

int *p_array 변수 data를 담을 공간이 없음

p_array = array1;

int var;

int array1[50];

Pointer may have…

- Uninitialized
- 0
- Valid address

Pointers as Arrays

- No data storage space at a pointer
- Two ways to get storage
  - Points pre-existing space
  - Dynamic memory allocation

```
int var, array[100];
int *prr1 = &var;   *ptr1 = 23;
int *ptr2 = array;  ptr2[4] = 23;
int *ptr3 = malloc(10*sizeof(int));
ptr3[4] = 23;
```

Array as Arguments

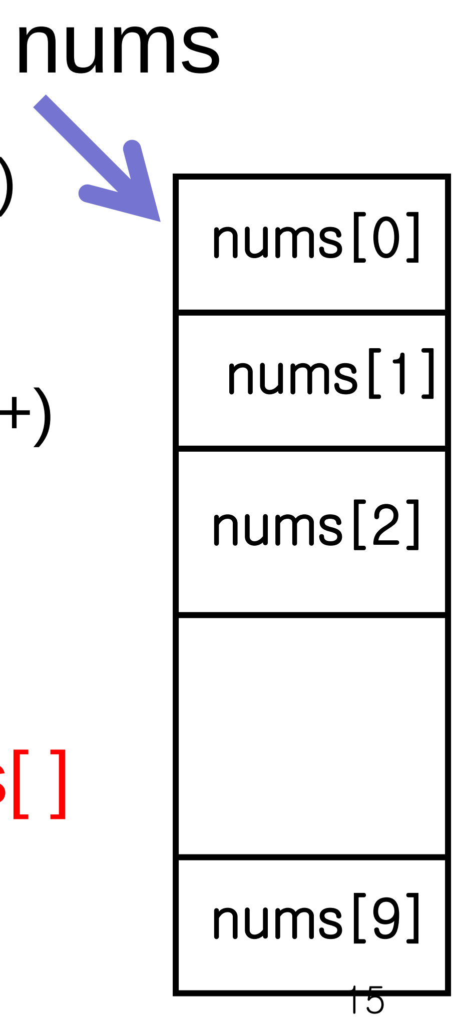
- Fill 10 integers in an array

```
int main(void)
{
    int nums[10];
    int  k;

    fill (nums, 10);
    for(k  = 0 ; k < 10 ; k++)
        printf(“ %d”, nums[k]);
}
```

```
void fill (int *nums, int len)
{
    int k;
    for (k = 0 ; k < len ; k++)
        nums[k] = k*10;
}
```

int nums[ ]



String as an Array

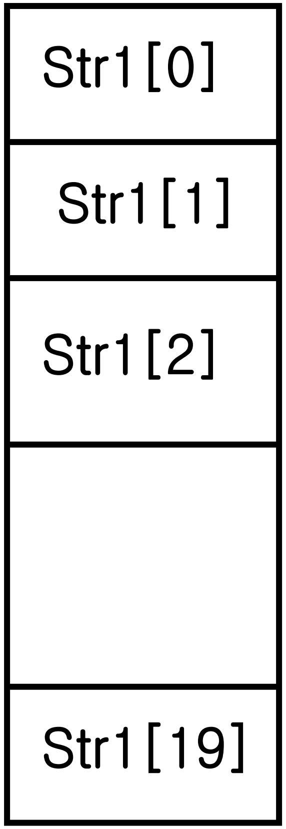
- Character array
- Array의 마지막 character는 항상 '\0' [혹은 (char)0]

char str1[20];

(1) 초기화 : char str1[20] = "Hello";

(2) strcpy(str1, "Hello");

(3) str1 = "Hello"; **(WRONG)**



Str1[0]

Str1[1]

Str1[2]

Str1[19]

Initialize Character Array

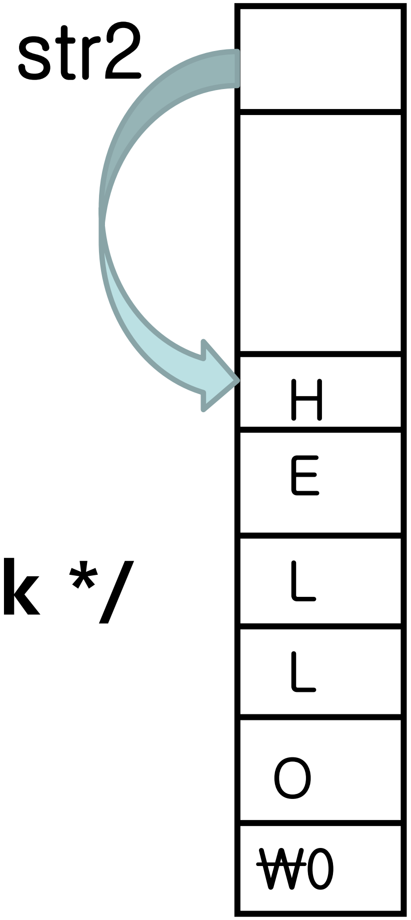
```
char str1[20] = “hello”;
               // 6 of 20
char str2[] = “hello”;
               // 6 of 6
char str3[20] = { ‘h’, ‘e’, ‘l’,
  ‘l’, ‘o’ };    // 6 of 20
char str4[] = { ‘h’, ‘e’, ‘l’,
  ‘l’, ‘o’ };    // 5
```

String as a Pointer

```
char *str2 = (char *)malloc(10*sizeof(char));

str2[0] = ‘P’;
strcpy(str2, “Min”);
strcpy(str2, “Hello”);

str2 = “Kim”;   /* memory leak */
```



Initialize Pointers as Strings

```
char *str1 = “hello”;
const char *str2 = “hello”;
~~char *str3 = {‘h’, ‘e’, ‘l’};~~
char *str4 = str1;
char *str5 =
  (char *)malloc(20*sizeof(char));
strcpy(str5, “hello”);
```

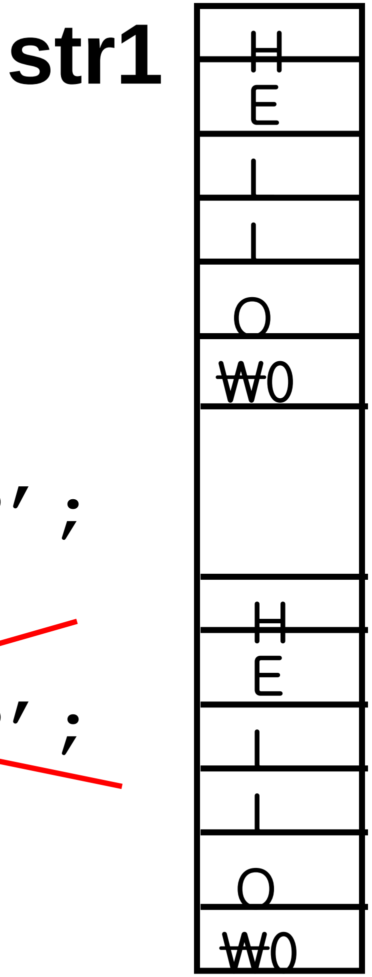
Character Array vs Pointer

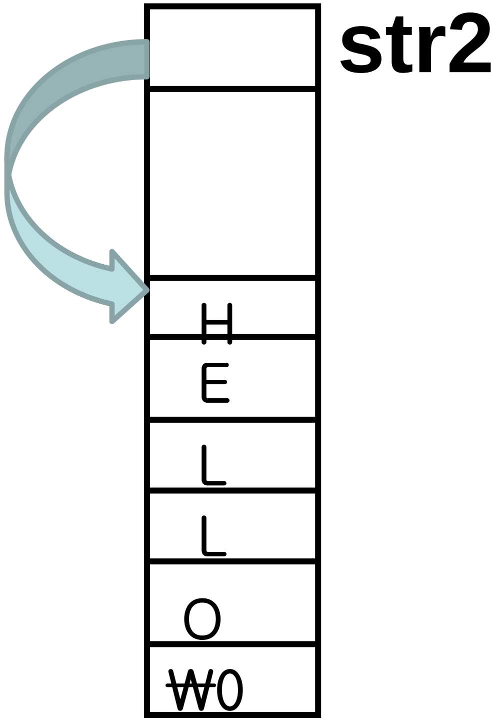
- Can you tell the difference between the following 2 declarations ?

```
char str1[] = “HELLO”;
char *str2 = “HELLO”;
```

Immutable vs Mutable

```
char str1[] = “HELLO”;
char *str2 = “HELLO”;
```





```
str1[0] = ‘P’;
```

~~`Str2[0] = ‘P’;`~~

Program 연습 (1)

- 2개의 part로 구성되어 있습니다.
- Part 1
  - 주어진 program의 comment를 참고하여 동작하도록 만듭니다.
  - main()에서 call하는 function의 대부분이 없습니다. 여러분이 추가하여야 합니다.
  - 이미 존재하는 code는 추가만 가능하며 제거할 수 없습니다.

Program 연습 (2)

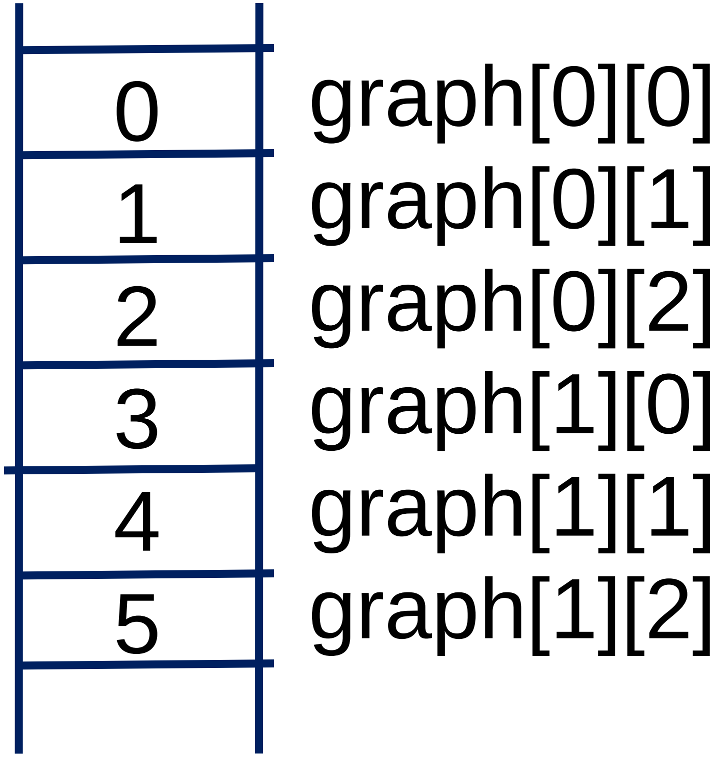
Part 2 : 정현파 함수의 graph 그리기

- void drawSineWave(char graph[][70])
  - 0 ~ 90도 사이의 sine 함수를 그립니다
  - 고정된 크기의 2차원 array를 사용
- void drawCosineWave(char **graph)
  - 0 ~ 90도 사이의 cosine 함수를 그립니다.
  - dynamic memory 사용
- 이번 실습에 한하여, Flow Chart를 작성하지 않습니다.

2-Dimensional Array

int graph[2][3] = { {0, 1, 2}, {3, 4, 5} };

0	1	2
3	4	5



0 graph[0][0]

1 graph[0][1]

2 graph[0][2]

3 graph[1][0]

4 graph[1][1]

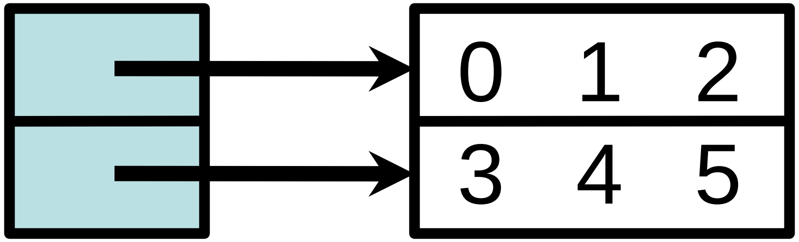
5 graph[1][2]

The 2nd dimension size should be a constant.

Type of “graph” :
int [][3]

Pointer Array for 2-Dimensional Array

```
int  *graph[2];
graph[0] = (int *)malloc(3*sizeof(int));
graph[1] = (int *)malloc(3*sizeof(int));
```



0	1	2
3	4	5

You can use array expression such as graph[1][2]

Double Pointer for 2-Dimensional Array

```
int  **graph;        /* int  *graph[2]; */
graph = (int **)malloc(2*sizeof(int *));


graph[0] = (int *)malloc(3*sizeof(int));
graph[1] = (int *)malloc(3*sizeof(int));
```

[1] 고정 크기 array 사용

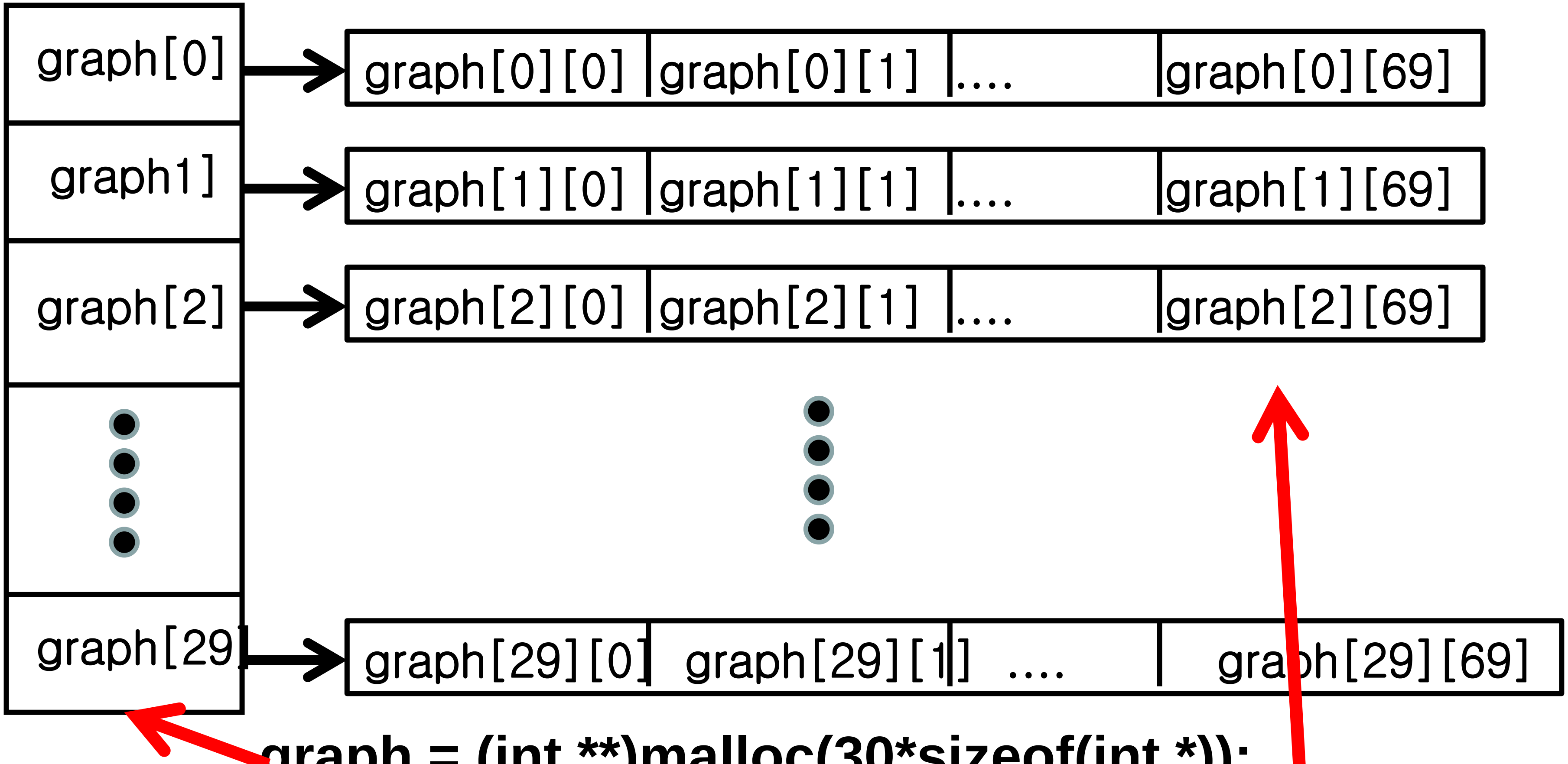
char graph[30][70];

y = sin (x)

x

```
100000000000000000000000000000
110000000000000000000000000000
111100000000000000000000000000
111111111100000000000000000000
111111111111111110000000000000
111111111111111111111110000000
```

[2] Dynamic memory 사용



graph[0]
graph1]
graph[2]
graph[29]

graph[0][0] graph[0][1] .... graph[0][69]
graph[1][0] graph[1][1] .... graph[1][69]
graph[2][0] graph[2][1] .... graph[2][69]
graph[29][0] graph[29][1] .... graph[29][69]

```
graph = (int **)malloc(30*sizeof(int *));
for (k = 0 ; k < 30 ; k++)
     graph[k] = (int *)malloc(70*sizeof(int));
```