



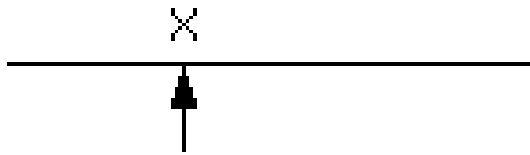
13. 배열

- 이 장의 주제는 배열(array)입니다.
- 배열을 고려하기 전에 먼저 **차원(dimension)**의 개념을 살펴보는 것이 필요합니다.



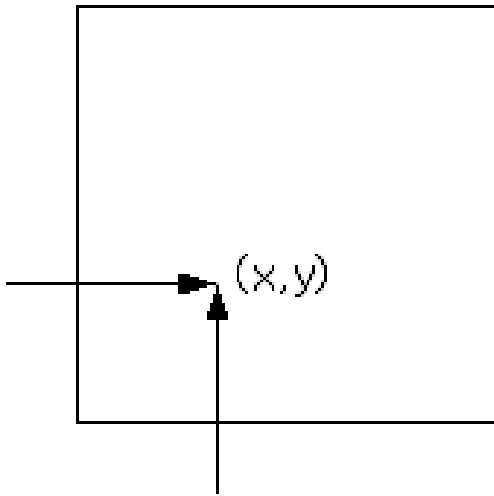
차원(dimension)

- 1차원(one dimension)은 기하학적으로 **직선(line)**을 의미합니다.
- 직선상의 임의의 위치를 기술하는데 필요한 파라미터의 수는 1개입니다.



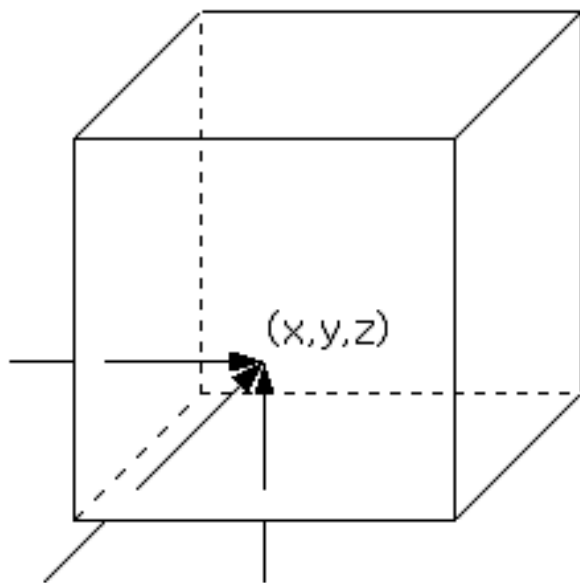
1차원상에서 위치를 기술하는데 필요한 파라미터는 1개뿐입니다.

- 2차원(2 dimension)은 기하학적으로 **평면(plane)**입니다. 평면상의 임의의 위치를 기술하는데 필요한 파라미터의 수는 2개입니다.



2차원상에서 위치를 기술하는데 필요한 파라미터는 2개입니다.

- 3차원(three dimension)은 기하학적으로 **입체(cube)**입니다. 입체상의 임의의 한 점을 기술하는데 필요한 파라미터는 3개입니다.



3차원상에서 위치를 기술하는데 필요한 파라미터는 3개입니다.

-
- 위치 함수(position function)에 필요한 파라미터의 수에 따라 차원의 수가 구분됩니다.
 - 어떤 위치 함수의 출력 값을 결정하는데 3개의 파라미터가 필요하다면, 그 함수는 3차원 공간의 위치를 나타내는 함수가 되는 셈입니다.



같은 형의 변수를 여러 개 선언하는 법

- 성적 처리를 하는 프로그램을 코딩한다고 가정해 봅시다.
- 학생의 수가 50명이고 과목의 수가 4과목이라면, 최소한 정수형 변수가 200개 필요할 것입니다.
- 그러면 프로그램은 다음과 같은 긴 변수 선언문을 가질 것입니다.

```
#include <stdio.h>
void main() {
    int i001,i002,i003,i004,i005,i006,i007,i008,i009,i010,
        i011,i012,i013,i014,i015,i016,i017,i018,i019,i020,
        i021,i022,i023,i024,i025,i026,i027,i028,i029,i030,
        i031,i032,i033,i034,i035,i036,i037,i038,i039,i040,
        i041,i042,i043,i044,i045,i046,i047,i048,i049,i050,
        i051,i052,i053,i054,i055,i056,i057,i058,i059,i060,
        i061,i062,i063,i064,i065,i066,i067,i068,i069,i070,
        i071,i072,i073,i074,i075,i076,i077,i078,i079,i080,
        i081,i082,i083,i084,i085,i086,i087,i088,i089,i090,
```

```
i091, i092, i093, i094, i095, i096, i097, i098, i099, i100,  
i101, i102, i103, i104, i105, i106, i107, i108, i109, i110,  
i111, i112, i113, i114, i115, i116, i117, i118, i119, i120,  
i121, i122, i123, i124, i125, i126, i127, i128, i129, i130,  
i131, i132, i133, i134, i135, i136, i137, i138, i139, i140,  
i141, i142, i143, i144, i145, i146, i147, i148, i149, i150,  
i151, i152, i153, i154, i155, i156, i157, i158, i159, i160,  
i161, i162, i163, i164, i165, i166, i167, i168, i169, i170,  
i171, i172, i173, i174, i175, i176, i177, i178, i179, i180,  
i181, i182, i183, i184, i185, i186, i187, i188, i189, i190,  
i191, i192, i193, i194, i195, i196, i197, i198, i199, i200;
```

```
}
```

- 이제 모두의 성적을 0으로 초기화해야 한다고 합시다. 이런! 대입문을 200번 사용해야 할 것입니다!
- 여기에 대한 좋은 해결책은 바로 **배열**을 사용하는 것입니다.

-
- 만약 정수형 변수 200개를 선언하고 싶다면, 위의 선언에서 변수이름 뒤에 여는 브래킷([)과 닫는 브래킷(])을 쓰고, 그 안에 변수의 개수를 명시합니다.

```
int i[200];
```

- 여기서 대괄호([,])안에 쓰는 숫자를 **색인(index)**이라고 합니다.
- 위의 선언에서 색인의 크기는 200인데 이것은 정수형 변수 200개를 선언함을 의미합니다.
- 주의해야 하는 사항은 색인의 범위는 0부터 199까지라는 것입니다.
- 그러므로 위의 선언은 다음과 같은 변수 200개를 선언한 것으로 생각할 수 있습니다.

```
i[0], i[1], i[2], ... ,i[199]
```


-
- 배열을 사용하는 진짜 이유는 여러 개의 변수를 간결하게 선언할 수 있어서가 아닙니다. 배열을 사용하는 이유는, 사용해야 하는 이유는 다음과 같습니다.

“변수를 이용하여 변수를 참조할 수 있습니다.”

- 위의 예에서 i배열의 인덱스의 범위는 0 ~ 199입니다.
- 배열의 모든 요소를 초기화하기 위해서는 다음과 같은 for문을 사용하는 것이 가능합니다.

```
...  
int i[200], j;  
for (j=0;j<200;++j)  
    i[j]=0;  
...
```



내용 연산자(content-of operator) []

- 5개의 요소를 가지는 정수형 배열 a[]를 고려해 봅시다. 배열 a[]의 요소는 각각 1,3,5,7과 9로 초기화합니다.

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

```
void main() {  
    int a[5], i;  
    a[0]=1;  
    a[1]=3;  
    a[2]=5;  
    a[3]=7;  
    a[4]=9;  
    for (i=0; i<5; ++i)  
        printf("%d, ", a[i]);  
}
```

-
- 정수를 선언하면서 초기화하듯이, 배열을 선언하면서 초기화할 수 없을까요?

```
int a[5]={1,3,5,7,9};
```

- 배열의 초기 값이 모두 명시된 경우 배열의 크기를 생략하는 것이 가능합니다.

```
int a[]={1,3,5,7,9};
```

-
- 배열의 크기가 개발 시간동안, 빈번하게 변하는 경우라면 어떻게 할까요? 바로 매크로를 이용하여 해결할 수 있습니다.
 - `sizeof(a)`는 배열 `a[]`가 차지하는 전체 바이트의 크기를 리턴합니다. 즉 20(정수가 4바이트를 차지하는 경우)입니다. `sizeof(a[0])`는 얼마일까요? 4입니다.
 - 그러면 배열의 크기 - 첨자의 범위 - 를 계산하는 방법이 떠오를 것입니다. 그것은 다음과 같습니다.

`sizeof(a)/sizeof(a[0])`

-
- 이제 좀 더 유지/보수가 쉽게 소스를 수정해 봅시다.

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

```
#define SIZE sizeof(a)/sizeof(a[0])
```

```
void main() {  
    int a[]={1,3,5,7,9},i;  
    for (i=0;i<SIZE;++i)  
        printf("%d, ",a[i]);  
}
```

- 배열 `a[]`가 실제 어떻게 메모리에 할당되는지 살펴봅시다.

		symbol table		
		symbol	address	
[1000]	<code>a[0]=1</code>	<code>a</code>	[1000]	
[1004]	<code>a[1]=3</code>	<code>i</code>	[1020]	
	<code>a[2]=5</code>	<code>main</code>	?	
	<code>a[3]=7</code>			
	<code>a[4]=9</code>			
[1020]	<code>i=?</code>			



배열의 요소는 낮은 첨자의 값부터 차례대로 할당됩니다. 배열의 이름 `a`는 배열의 시작 주소를 의미합니다.

```
#include <stdio.h>

#define SIZE sizeof(a)/sizeof(a[0])

void main() {
    int a[]={1,3,5,7,9};

    printf("%d,%d,%d\n", a[2], 2[a], *(a+2));
    printf("%p,%p\n", a+2, &a[2]);
}
```

- 출력 결과는 다음과 같습니다.

5,5,5

1B872026,1B872026

-
- **문자 배열(character array)**에 대해서 살펴봅시다.
 - 문자열을 관리하기 위해 다음과 같은 배열을 사용한다고 합시다. 문자열의 최대 길이가 5라면 EOS(end of string) 0을 포함하여 길이가 6인 문자열 배열을 사용할 수 있습니다.

```
char s[6];
```

- 이제 문자열 배열을 "hello"로 초기화 해 봅시다.

```
char s[6]={'h','e','l','l','o',0};
```

- 초기화가 된 경우 배열의 크기를 생략할 수 있으므로, 다음과 같이 적을 수 있습니다.

```
char s[]={'h','e','l','l','o',0};
```


- 아래의 예제는 문자열 "hello"를 출력합니다.

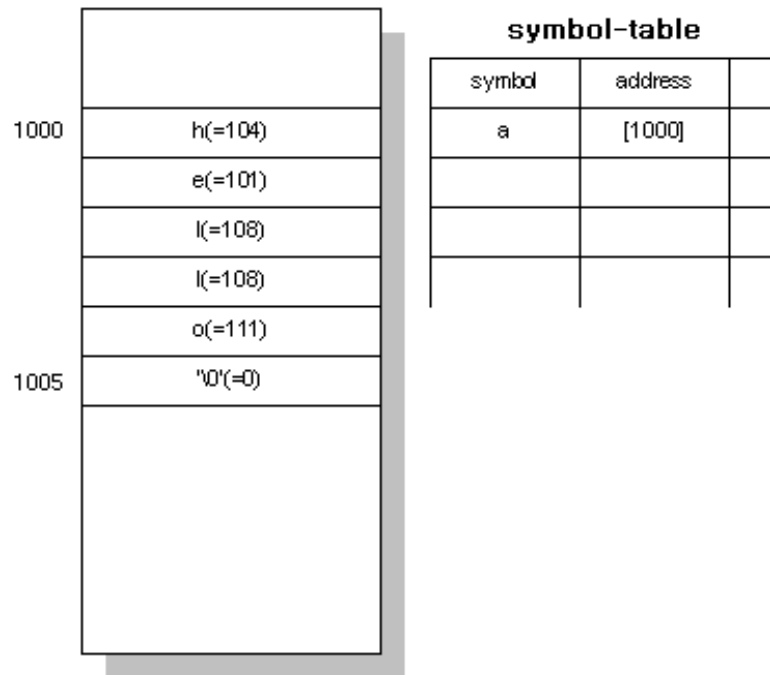
```
#include <stdio.h>

void main() {
    char a[]={'h','e','l','l','o',0};
    printf("%s\n",a);
}
```

- 문자 배열의 용도가 문자열을 관리하기 위해서라면, 특별한 형태로 배열을 초기화하는 것이 가능합니다.

```
#include <stdio.h>

void main() {
    char a[]="hello";
    printf("%s\n",a);
}
```



`a[]="hello"`의 메모리 상태: 배열 `a[]`는 6바이트의 메모리가 할당되며, 각각의 요소가 문자열 "hello" 및 0으로 초기화됩니다. 'h'가 아니라 실제로 메모리에는 ASCII (혹은 ANSI) 코드 값인 104가 2진수 2의 보수 형태로 저장된다는 사실에 주의하세요.

-
- 위의 메모리 맵을 참고하여 다음 프로그램의 출력 결과를 예측해 보세요.

```
#include <stdio.h>

void main() {
    char a[]="hello";
    printf("%c,%d,%d\n",*(a+1),*(a+1),*(a+5));
}
```

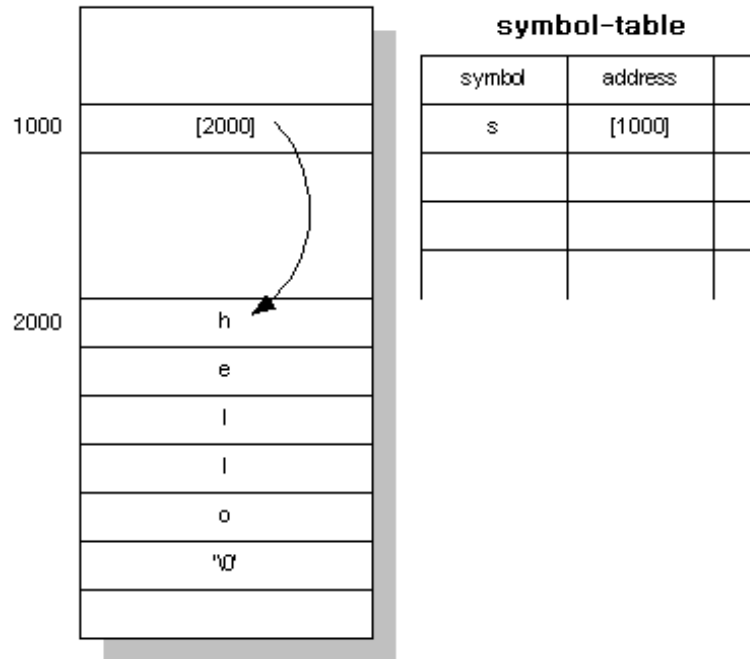
- 출력 결과는 다음과 같습니다(%c는 ASCII 문자를 %d는 십진수를 출력합니다).

e,101,0

-
- "할당되지 않은 메모리에 문자열을 대입하지 마세요."

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

```
void main() {  
    char* s;  
    s="hello";//이 문장은 타당한가요?  
    printf("%s\n",s);  
}
```

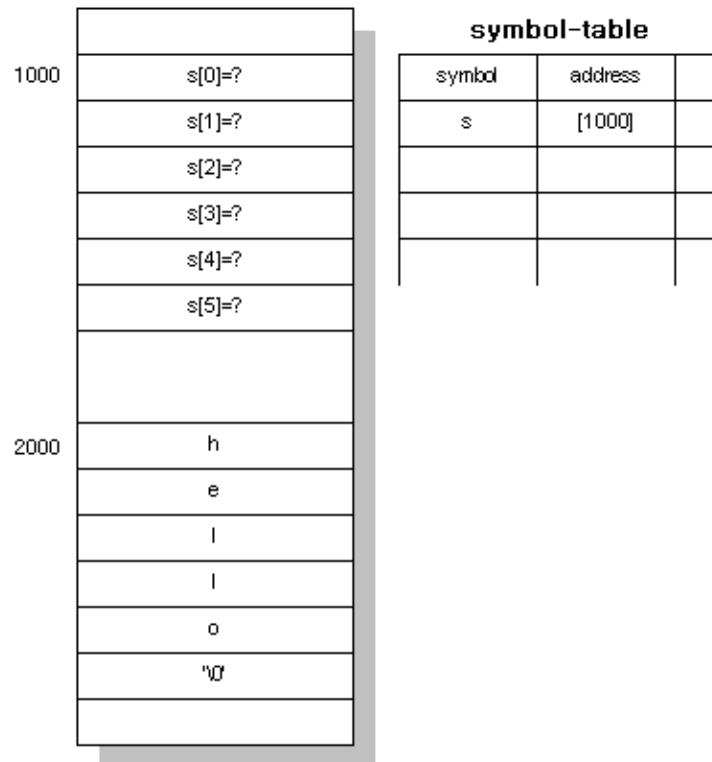


할당된 메모리는 s를 위한 4바이트뿐이지, 문자열 "hello"를 위해서는 메모리가 할당되지 않았습니다. "hello"라는 표현은 메모리의 특정한 곳에 문자열 hello와 0을 차례대로 집어넣고 'h'의 시작 주소인 2000을 리턴합니다. s는 2000이라는 값을 가지게 되지만, 2000번지부터 2005번지까지가 메모리 할당된 것은 아닙니다.

-
- 이번에는 배열을 이용하여 명시적인 메모리 할당 후에 문자열 대입을 시도하였습니다.
 - 이번에는 컴파일 시간 에러가 발생합니다.

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

```
void main() {  
    char s[6];  
    s="hello";  
    printf("%s\n",s);  
}
```



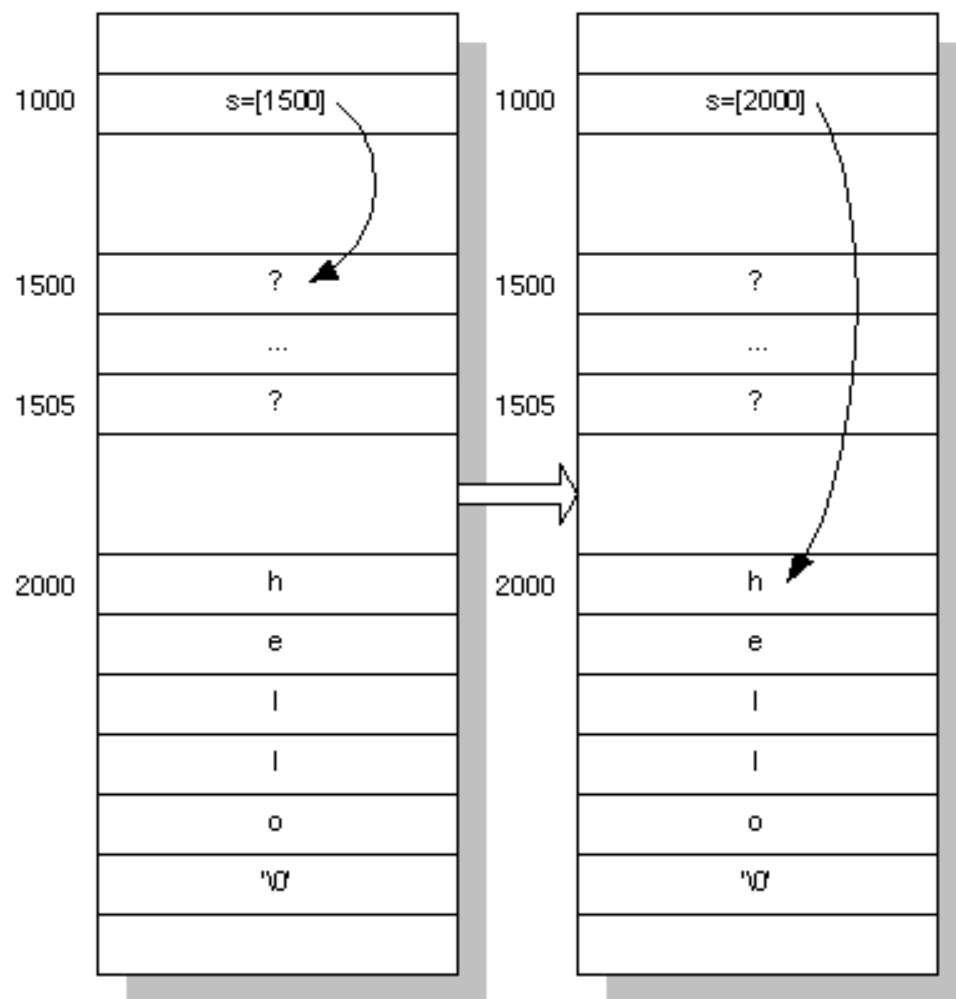
메모리 할당: `s`는 메모리의 어느 곳에도 할당되지 않았습니다. 다만, `s[]`의 요소(element)들이 메모리에 할당되었을 뿐입니다. 즉 `s`는 상수 주소 - 배열의 시작을 가리키는 - 로서 존재할 뿐입니다.

-
- 회심의 미소를 머금고 프로그램을 아래처럼 수정하였습니다.

```
#include <stdio.h>

void main() {
    char* s;

    s=new char[6];
    s="hello";
    printf("%s\n",s);
    delete[] s;
}
```

-
- s는 할당된 메모리의 시작 주소를 가리킵니다.
 - 만약 new가 1500번지부터 6바이트를 할당했다면, s는 [1500]이 됩니다.
 - 그러므로, s가 가리키는 1500번지에서부터 문자열 "hello"의 문자들을 차례대로 집어넣어야 합니다.
 - 하지만 s="hello"란 문장에 의해 s의 값은 2000으로 변경되어 할당되지 않은 메모리를 가리킬 뿐 만 아니라, 1500을 가리키는 포인터가 사라져서 s는 댕글링(dangling) 포인터가 됩니다.
 - 즉, 1500번지부터 6바이트의 메모리에 누수가 발생합니다(leak).

-
- 최종적으로 완성된 완전한 소스를 작성할 수 있습니다.

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

void main() {
    char* s;

    s=new char[6];
    //s="hello";
    strcpy(s, "hello");
    printf("%s\n", s);
    delete[] s;
}
```



이차원 배열(2-dimensional array)

- 이차원 배열은 배열의 요소를 참조하기 위해 2개의 인덱스를 사용합니다.

	국어	영어	수학	과학
1				
2				
3				

	국어	영어	수학	과학
49				
50				



50명의 성적 처리를 위한 (2차원) 테이블: 특정한 성적을 참조하기 위해서는 (번호, 과목)이 필요합니다. 2차원 배열 `score[50][4]`를 선언해서 번호는 줄(row)로 과목은 열(column)로 참조합니다.

-
- 위의 그림처럼 50행(row), 4열(column)의 테이블을 만들기 위해서 배열을 다음과 같이 선언합니다.

```
int score[50][4];
```

- 첫 번째 인덱스가 행의 수를, 두 번째 인덱스가 열의 수를 지정하는 것에 주의하세요.

- 2×4의 score[2][4] 배열을 생각해 봅시다. 테이블은 다음과 같이 설정되어 있습니다.

	0	1	2	3
0	1	3	5	7
1	10	30	50	70



score[2][4] 배열: 첫 번째 인덱스 2가 행(row)의 수를, 두 번째 인덱스 4가 열(column)의 수를 나타냅니다.

- 이것은 배열로 다음과 같이 선언할 수 있습니다.


```
int score[2][4];
```

- 1차원 배열을 초기화하듯이, 2차원 배열도 초기화할 수 있는데, 각각의 행을 블록으로 묶어 나타냅니다.

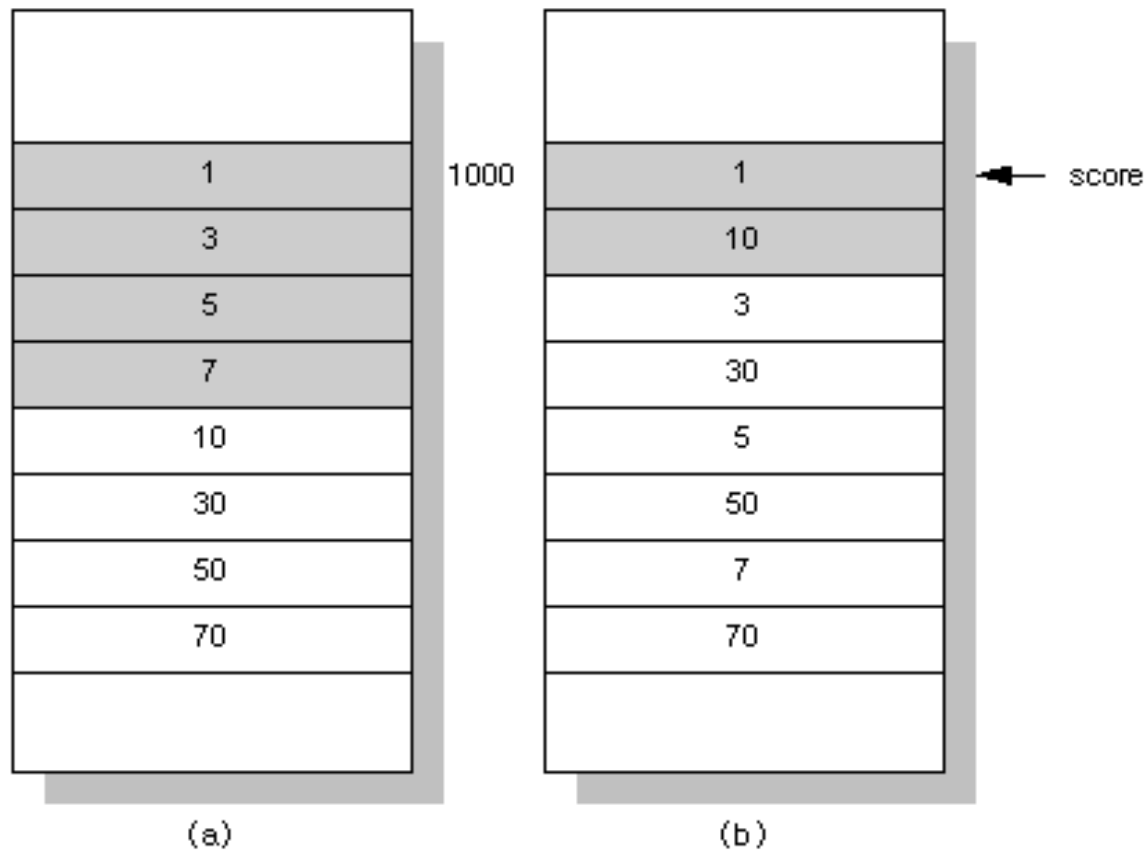
```
int score[2][4]={1,3,5,7},{10,30,50,70};
```

- 배열의 행을 먼저 저장하는 방식을 **행 중요 순서(row major order)**라 하고, 열을 먼저 저장하는 방식을 **열 중요 순서(column major order)**라 합니다.

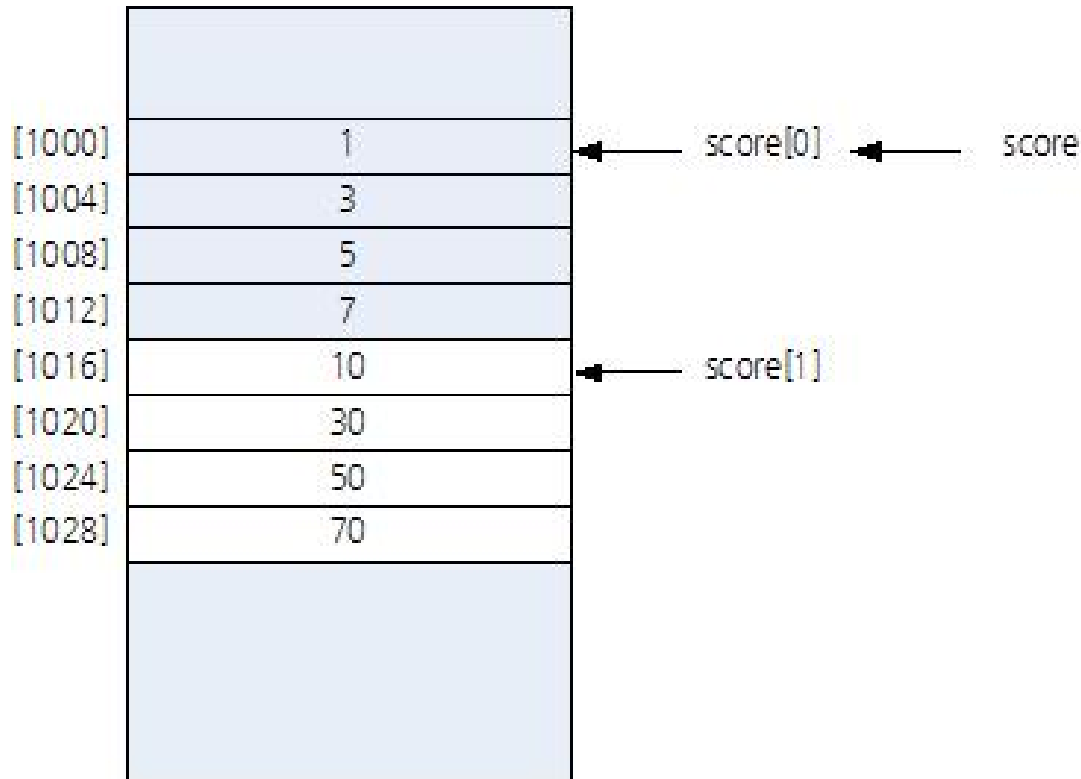
	0	1	2	3
0	1	3	5	7
1	10	30	50	70



- score[] 배열은 실제로 메모리에 아래 그림의 (a)처럼 저장됩니다.



-
- 1차원 배열에서 배열의 인덱스를 생략하면, 배열의 시작 주소를 의미(상수 포인터)한다고 했습니다.
 - 2차원 배열에서도 배열의 인덱스를 생략할 수 있는데, 생략된 인덱스를 0으로 한 곳의 시작 주소를 의미합니다.
 - score[0]은 score[0][0]의 시작 주소입니다. score[1]은 score[1][0]의 시작 주소입니다.
 - 2개의 인덱스를 모두 생략한 score는 어떨까요?
 - 물론 score[0][0]의 시작 주소입니다.
 - 하지만, 첨자를 2개 생략했으므로, 포인터의 포인터(pointer to pointer)로 해석을 합니다.
 - 그러므로 *score는 score[0]을 의미하며, **score가 score[0][0]을 의미합니다.



score의 해석: score[0]은 pointer to score[0][0] 즉, &score[0][0]을 의미합니다. score는 pointer to score[0] 즉, &score[0]으로 해석을 합니다. 그러므로, 배열의 요소 정수를 참조하기 위해서는 두 번 재참조(indirection)해야 합니다.

-
- 아래 프로그램의 출력 결과를 예측해 보기 바랍니다.

```
#include <stdio.h>

void main() {
    int score[2][4]={{1,3,5,7},{10,30,50,70}};

    printf("%d,%d\n", score[0][1], score[1][2]);
    printf("%d,%d\n", *(score[0]+1), *(score[1]+2));
    printf("%d,%d\n", *(*score+1), *(*score+6));
}
```

-
- 아래와 같은 2차원 배열의 초기화를 생각해 봅시다.

```
int score[2][4]={{1,3,5,7},{10,30,50,70}};
```

- 위의 초기화는 다음과 같이 쓸 수 있습니다.

```
int score[2][4]={1,3,5,7,10,30,50,70};
```



다음 두 개의 초기화는 어떻게 다른가요?

```
int score[2][4]={{1,3,5},{10,30,50,70}};
```

```
int score[2][4]={{1,3,5},{10,30,50,70}};
```



첫 번째 초기화에서 `score[0][3]`은 초기화되지 않습니다. 하지만 두 번째 초기화는 `score[0][3]`을 0으로 초기화합니다.

- 2차원 문자 배열은 1차원처럼 간단하게 쓸 수 있습니다.

```
#include <stdio.h>

void main() {
    char str[4][10]={{'g','o','l','d',0},
                     {'s','i','l','v','e','r',0},
                     {'c','o','p','p','e','r',0},
                     {'n','e','c','k',0}};

    int i;

    for (i=0;i<4;++i) {
        printf("%s\n",str[i]);
    }//for
}
```

-
- 위의 예에서 문자 배열 `str[]`은 아래와 동일합니다.

```
char str[4][10]={{"gold"},
                  {"silver"},
                  {"copper"},
                  {"neck"}};
```

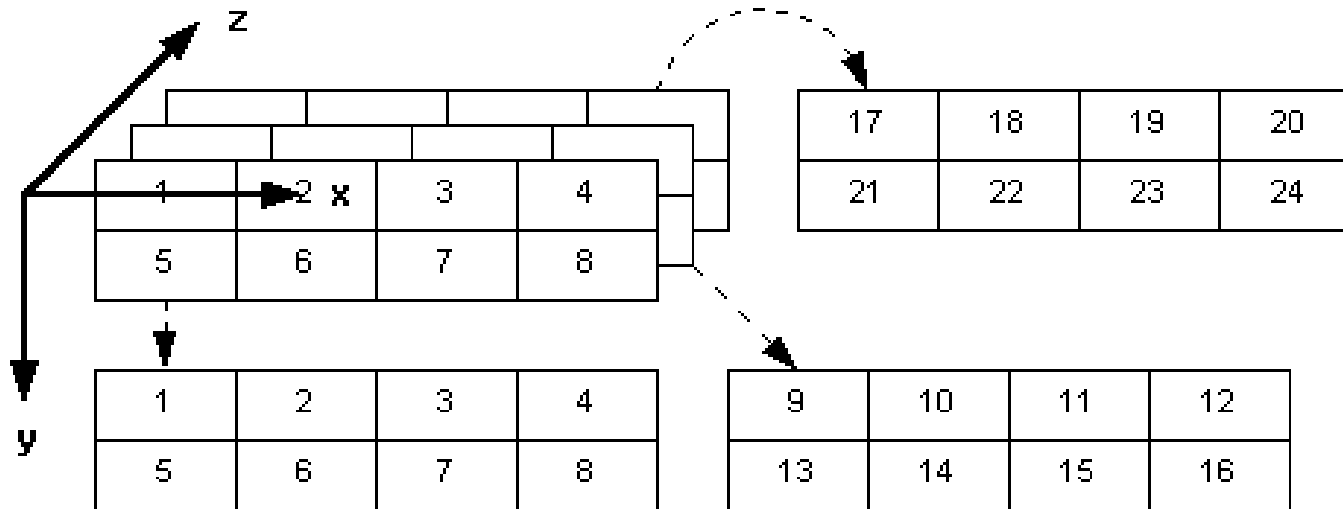
- 위 문장은 아래와 같이 초기화 할 수 있으며 이것이 좀 더 일반적인 방법입니다.

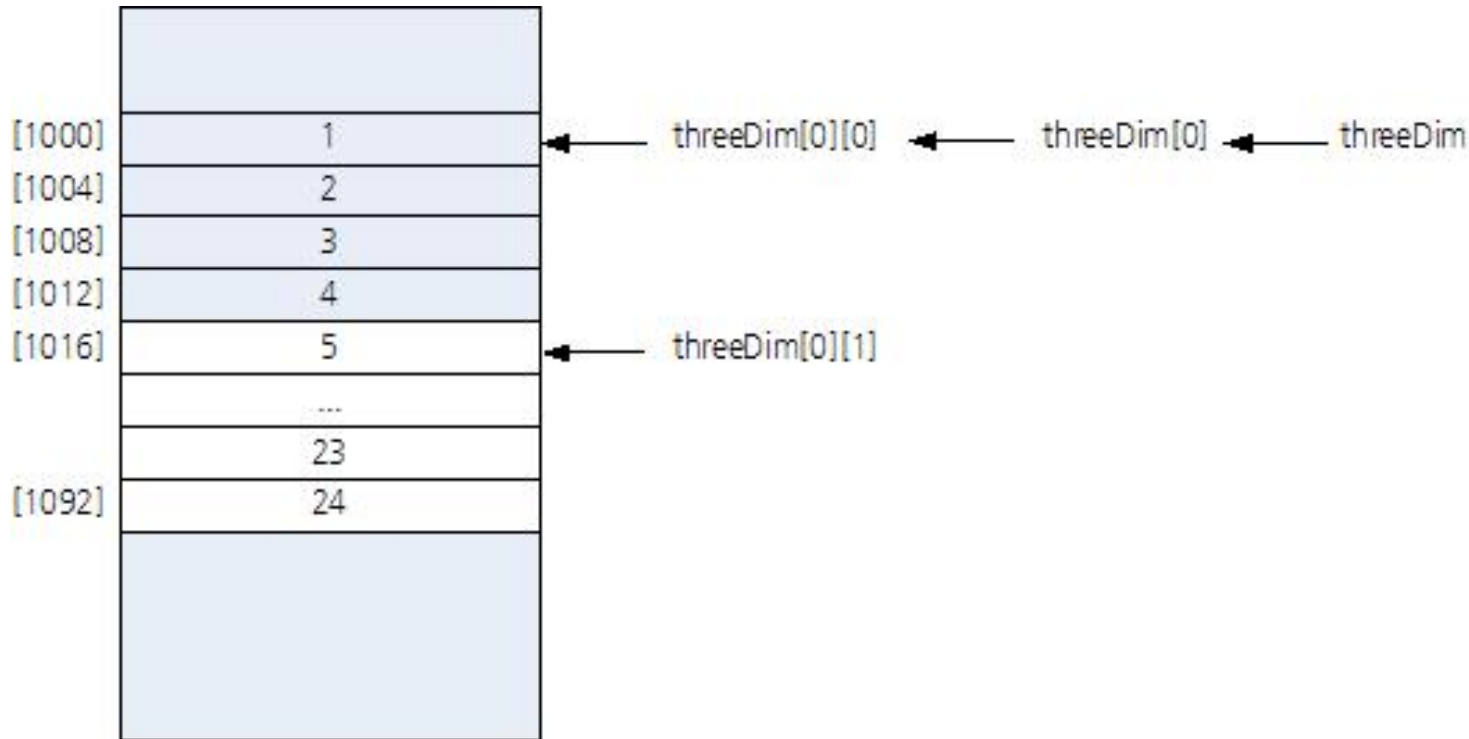
```
char str[4][10]={"gold",
                  "silver",
                  "copper",
                  "neck"};
```

3차원 이상의 배열

```
int threeDim[3][2][4]={{{1,2,3,4},{5,6,7,8}},
                        {{9,10,11,12},{13,14,15,16}},
                        {{17,18,19,20},{21,22,23,24}}};
```

- $3 \times 2 \times 4$ 의 3차원 배열은 위처럼 선언합니다.





3차원 배열의 이름의 해석: `threeDim[0][1]`은 `&threeDim[0][1][0]`를 의미합니다. `threeDim[0]`은 `&threeDim[0][0]`, 즉 `threeDim[0][0][0]`에 대한 포인터의 포인터를 의미합니다. `threeDim`은 `&threeDim[0]`, 즉 `threeDim[0][0][0]`에 대한 포인터의 포인터의 포인터(pointer to pointer to pointer)를 의미합니다.



배열의 전달

- 배열을 함수의 파라미터로 어떻게 전달할까요?

```
#include <stdio.h>

void f(int score[][4]) {
    score[1][1]++;
} //score

void main() {
    int score[][4]={{1,3,5,7},{10,30,50,70}};

    printf("%d\n", score[1][1]);
    f(score);
    printf("%d\n", score[1][1]);
}
```

- 1차원인 경우는 1개의 첨자로 배열의 요소를 결정하므로, 정수 배열의 시작 주소를 `int* score` 형태로 받는 것이 가능합니다.
- 함수의 파라미터 선언에서 `int* score`는 `int score[]`와 동일합니다. 아래의 예를 참고하세요.

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

```
void f(int score[]) {  
    ++(*(score+2));  
} //score
```

```
void main() {  
    int score[4]={1,3,5,7};  
  
    printf("%d\n", score[2]);  
    f(score);  
    printf("%d\n", score[2]);  
}
```



포인터 배열

```
char c;
```

- 위의 문장은 1바이트 정수형(문자형) 변수 c를 선언한 것입니다.

```
char c[10];
```

- 위의 문장은 크기가 10인 문자형 배열을 선언한 것입니다.

```
char* c;
```

- 위의 문장은 문자형 포인터 변수를 선언한 것이며, 메모리에는 4바이트가 할당됩니다.

- 크기가 4인 문자형 포인터 배열은 다음과 같이 선언합니다.

```
char* str[4];
```

- 이것은 str[0], str[1], str[2]와 str[3]이 char* 형인 배열을 선언한 것입니다.

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

```
void main() {  
    char* str[4]={ "gold",  
                  "silver",  
                  "copper",  
                  "neck"};  
  
    int i;  
  
    for (i=0;i<4;++i)  
        printf("%s\n",str[i]);  
}
```

-
- str을 파라미터로 받는 함수 f()는 다음과 같은 다양한 방법으로 선언할 수 있습니다.

```
void f(char* str[4]);
```

```
void f(char* str[]);
```

```
void f(char** str);
```

```
#include <stdio.h>

void f(char** str) {
    int i;

    for (i=0;i<4;++i)
        printf("%s\n",str[i]);
}

void main() {
    char* str[4]={"gold",
                  "silver",
                  "copper",
                  "neck"};

    f(str);
}
```

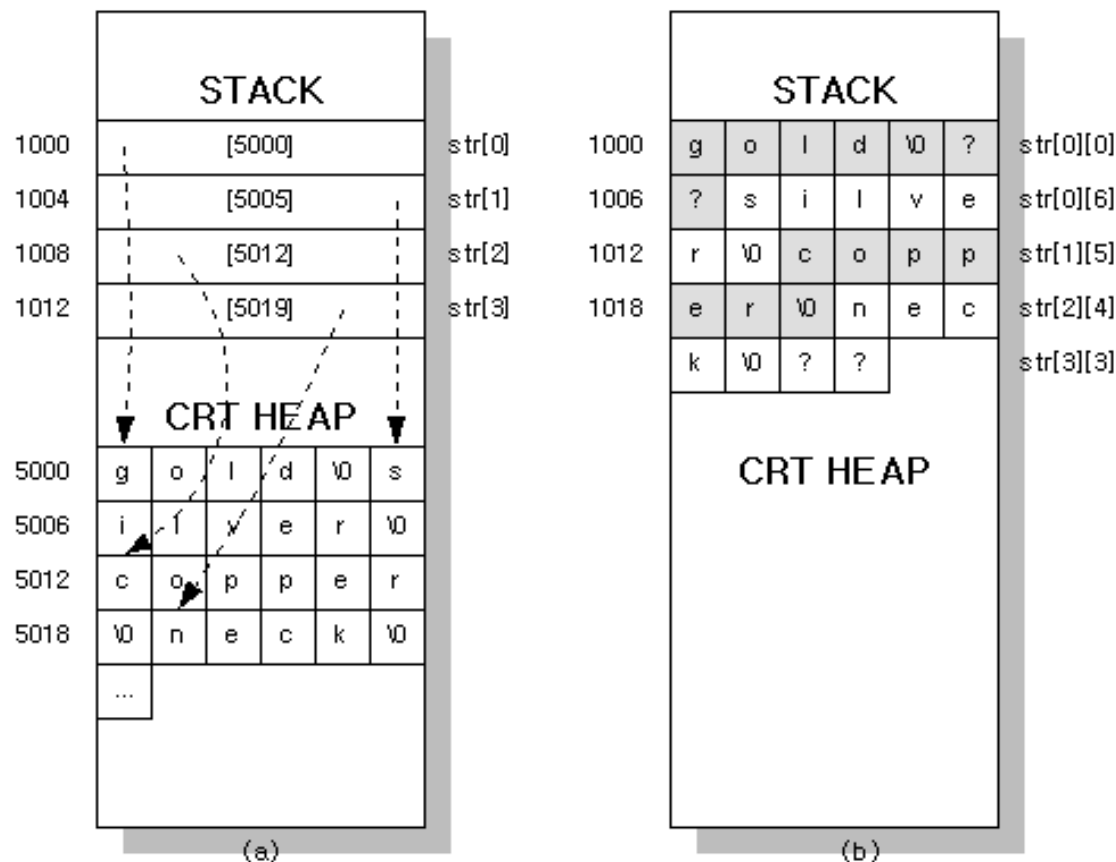
-
- 우리는 위의 main()안에서 선언된 아래코드,

```
char* str[4]={ "gold",  
               "silver",  
               "copper",  
               "neck"};
```

- 그리고, 2차원 배열을 사용한 이전의 아래 코드를 구분해야 합니다.

```
char str[4][7]={ "gold",  
                 "silver",  
                 "copper",  
                 "neck"};
```

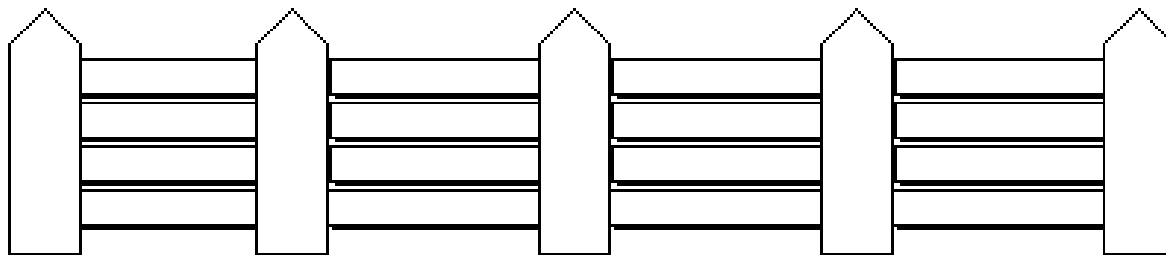

- 아래 그림의 (a)와 (b)는 각각 2가지 경우의 메모리 상태를 보여줍니다. 명시적으로 할당되지 않은, 메모리 공간을 사용하지 마세요.





울타리 막대기 문제: 가장 자리 문제

- 4개의 울타리를 만들기 위해서 필요한 막대기는 5개라는 것을 주의하세요.



울타리 막대기 문제: 울타리보다 1개 많은 막대기가 필요합니다.

- 예를 들면 크기가 5인 배열의 인덱스는 1 ~ 5가 아니라, 0 ~ 4입니다.
- 길이가 5인 문자열을 관리하기 위해서는 크기가 6인 문자열 배열이 선언되어야 합니다.
- 배열의 요소를 계산하기 위해, 앞뒤로 인접한 배열의 셀(cell)을 검사할 때, 첫 요소에서는 바로 앞의 요소를 접근하면 안 되며, 마지막에서는 바로 뒤의 요소를 접근해서는 안 됩니다.
- 배열의 모든 요소를 1로 초기화하는 for문을 수행했을 때, for문 수행이 끝났을 때의 인덱스는 유효하지 않습니다.

```
int a[5], i;  
for (i=0; i<5; ++i)  
    a[i]=1;
```



실습문제

1. (**Zeller의 공식**) 아래의 프로그램에서 DayOfWeek()는 년, 월, 일을 파라미터로 받아 주일, 월요일, ... ,토요일에 대해 0,1, ... ,6을 리턴합니다. DayOfWeek()를 자세히 설명하세요. 또한 main()에서 2차원 배열의 역할에 대해서도 자세히 설명하세요(힌트: 윤년(leap year)은 4의 배수면서, 100의 배수가 아니면서, 400의 배수이면 성립함. 또한 0년 1월 1일은 1주일의 시작인 일요일 임).

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

```
int DayOfWeek(int nYear, int nMonth, int nDay) {  
    if (nMonth < 3) {  
        nYear -= 1;  
        nMonth += 12;  
    }  
    return (nYear + nYear/4 - nYear/100 + nYear/400 + (13*nMonth + 8)/5 + nDay) % 7;  
}
```

```
void main() {
    char s[7][20]={"SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"};

    printf("%s\n", s[ DayOfWeek(0,1,1) ]);
    printf("%s\n", s[ DayOfWeek(1999,3,9) ]);
    printf("%s\n", s[ DayOfWeek(1999,3,10) ]);
}
```

2. 잘못된 부분을 수정하고, 이유를 구체적으로 설명하세요.

```
char* s[6]="hello";//에러가 발생합니다.
```

```
char* t[2]={"world","every"};//잘못된 예를 드세요.
```