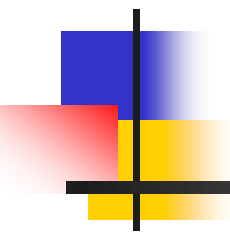


C Programming (CSE2035) (Chap7. Array)



Sungwon Jung, Ph.D.

Bigdata Processing & DB LAB
Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea

Tel: +82-2-705-8930

Email : jungsung@sogang.ac.kr



Array

- **Concepts**
- **Using Arrays in C**
- **Arrays and Functions**
- **Two-dimensional Arrays**
- **Multidimensional Arrays**



Concepts

■ 여러 학생의 성적관리

- 성적 관리를 위해서 한 명의 학생에게 하나의 변수를 지정하는 프로그램을 생각해 보자. 만약 학생 수가 많다면 변수를 여럿 만들어야 할 것이다.

```
float student1 = 4.3;  
float student2 = 4.0;  
  
...  
float student9 = 3.3;  
float student10 = 3.0;
```



만약 관리하는 학생의 수가 매우 많다면?

→ 변수의 수도 비례해서 증가하므로
프로그램을 만들기도 관리하기도 힘들다.

- 대신에 동일한 형을 갖는 연속적인 변수를 생성할 수 있다면, 위 문제를 더욱 쉽게 해결 할 수 있다.



Concepts

■ 배열(array)

- 같은 형의 여러 변수들에 대한 저장소 집합
- 배열의 구성 요소는 index(첨자)로 접근됨
 - 가장 낮은 주소는 배열의 첫 번째 요소이며
가장 높은 주소는 마지막 요소에 대응
- C에서 가장 일반적인 배열은 널(null)로 끝나는 문자들의 배열인 문자열(string)이다. 예를 들어 “sogang”이라는 문자열을 C언어에서 다루려면 아래와 같은 char 형의 배열이 필요하다.

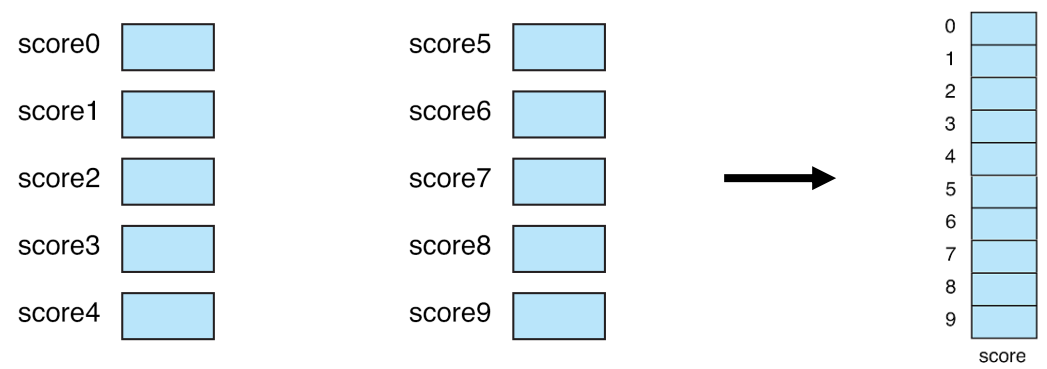
s	o	g	a	n	g	\0
---	---	---	---	---	---	----

- C에서 배열은 사용형식에 따라 여러 차원으로 구성할 수 있다.

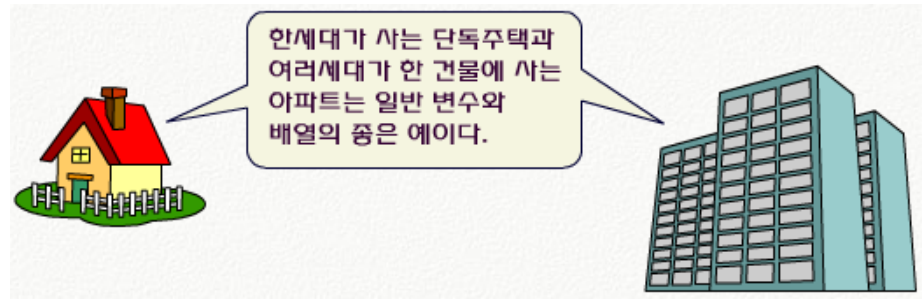
Concepts

- 배열을 사용할 수 있는 예

- 연속적인 항목들이 동일한 크기로 메모리에 저장되는 구조

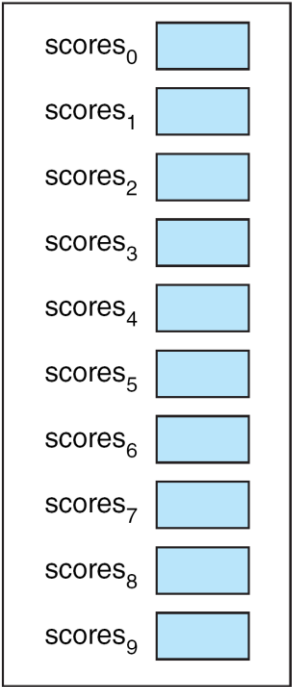


- 동일한 자료 유형이 여러 개 필요한 경우 이용할 수 있는 자료구조
 - 일반 변수가 단독주택이라면 배열은 아파트

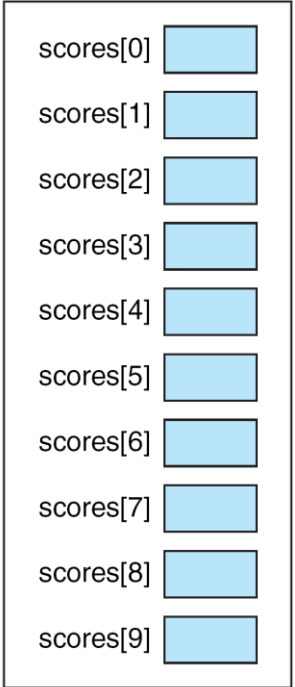


Concepts

- 즉, 같은 정수형(int)인 10개의 수를 관리할 때, 10개의 변수를 따로 정의하기 보다는 한 개 변수 이름에 10개 index를 주어 관리하는 것이 편리하다.



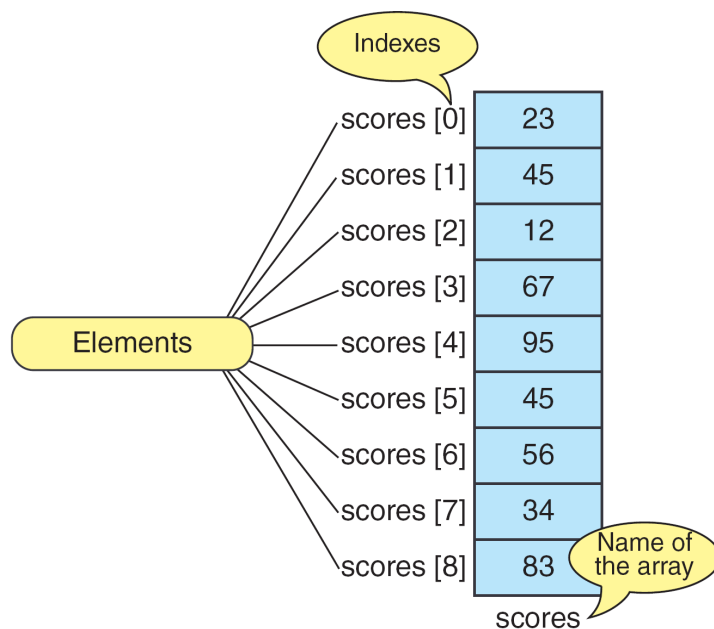
scores
(a) Subscript Format



scores
(b) Index Format

Using Arrays in C

- C언어는 배열을 연속된 메모리 위치로 봄
 - 실제 메모리 위치배정은 다를 수 있으나 연속된 메모리로 인식한다.
 - Index는 0번부터 위치가 부여된다. (index는 0 ~ (배열크기-1))
 - 예) score라는 이름으로 9개의 elements가 array로 사용되고 있다.



Using Arrays in C

■ Declaration and Definition

type-specifier

데이터형의 이름 (int, float 등)

array-name [constant expression];

배열의 이름 [배열 크기];

- 원소의 개수를 나타내는 **배열크기** 는 10과 같은 상수이거나, 또는 컴파일 시 값이 정확히 결정되는 $8 * \text{sizeof}(\text{int})$ 와 같은 상수수식이어야 한다.
- 프로그램이 실행되는 동안에 값이 설정되는 변수는 배열의 크기가 될 수 없다.
- 배열크기는 공란으로 둘 수도 있다. 단 이 경우 컴파일러가 배열크기를 알 수 있게 배열 선언시 배열의 크기만큼 초기화가 일어나야 한다.



Using Arrays in C

■ Declaration and Definition

- 예를 들어 다음과 같은 배열 선언문은

```
int months[12]; // 12개의 int형 값을 가진 배열을 생성
```

배열 이름이 months이고, 그 배열은 12개의 원소를 가지며, 각 원소는 int형 값을 저장할 수 있다는 것을 나타낸다.

- Index 0 부터 첫번째 원소가 시작된다.

```
month[0] = 6; // 첫번째 원소에 6을 저장  
month[4] = 8; // 다섯번째 원소에 8를 저장
```

- 실제로 배열의 각 원소는 일반변수처럼 사용될 수 있다.
- 변수를 배열의 크기로 설정하면 프로그램 실행시 에러를 출력한다.

```
int months[two+ten]; // error  
int months[2+10];    // ok
```

Using Arrays in C

- 배열 선언의 모습

```
int scores [9];
```

type of each
element

[0] [1] [2] [3] [4] [5] [6] [7] [8]

scores

```
char name [10];
```

name of
the array

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

name

```
float gpa [40];
```

number of
elements

[0] [1] [2] ... [37][38][39]

gpa

Using Arrays in C

■ Accessing Elements in Arrays

- 배열에서 한 특정 원소 값에 접근은 '배열이름[index]' 로 된다.
 - 예) scores[3], name[2], gpa[39]
- 배열의 정의된 범위를 넘는 index는 오류로 나타나지 않지만 쓰레기 값에 접근 되거나 system memory 침범 오류와 같은 심각한 오류가 있을 수 있다.
 - 예) 배열의 크기가 10이나 index가 10 이상일 때

```
int score[10];
score[10] = 7;
```

index가 음수일 때

```
int score[10];
score[-1] = 7;
```



Using Arrays in C

■ 예제 프로그램 - 배열의 할당

- 100개의 배열을 생성 후 $x[0] \sim x[99]$ 까지의 각 배열원소에 0~ 99까지의 값을 할당한다.
 $x[5]$ 의 값과 $x[55]$ 의 값을 저장된 값을 확인해보기 위해 출력하여 본다.

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int x[100];
6     int t;
7
8     for(t=0; t<100; ++t)
9         x[t] = t;
10
11     printf("x[5] = %d, x[55] = %d\n", x[5], x[55]);
12     return 0;
13 }
14

```

100개의 정수를
위한 배열 선언

배열의 할당

```

[root@mclab chap8]# vi chap8-1.c
[root@mclab chap8]# gcc -o chap8-1 chap8-1.c
[root@mclab chap8]# ./chap8-1
x[5] = 5, x[55] = 55
[root@mclab chap8]#

```

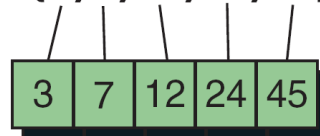
배열의 초기화

■ Initialization

- 초기화는 배열 정의 시에 가능하고 이후는 할 수 없다.

Basic Initialization

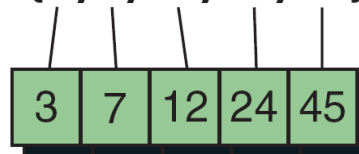
```
int numbers[5] = {3, 7, 12, 24, 45};
```



- Static, Global variables는 선언 시 컴파일러가 0으로 초기화한다.
- 배열의 크기가 정의되지 않았다면 초기화를 통해 배열의 크기를 정의 할 수 있다.

Initialization without Size

```
int numbers[ ] = {3, 7, 12, 24, 45};
```





배열의 초기화

■ 배열 초기화에 대한 보충 설명

- 초기화 형식은 배열을 정의하는 곳에서만 사용할 수 있으며,
- 초기화를 나중에 할 수 없고,
- 어떤 배열을 다른 배열에 통째로 대입할 수도 없다.

Ex)

```
int cards[4] = {3, 6, 8, 10};    /* 맞음 */  
int hand[4];                    /* 맞음 */  
hand[4] = {5, 6, 7, 8};         /* 틀림 */  
hand = cards;                   /* 틀림 */
```

- 그러나 첨자를 사용하여 개별적으로 배열 원소에 값을 대입하는 것은 언제든지 가능하다.

Ex)

```
hand[3] = cards[0];  
hand[0] = cards[3];
```

배열의 초기화

- 배열을 초기화할 때, 초기화 값의 개수를 배열 원소의 개수보다 적게 제공할 수 있는데, 이러한 경우에 배열의 앞 부분만이 초기화한다.
- 배열이 부분적으로 초기화할 경우, 컴파일러가 나머지 원소들을 0으로 설정한다.

Partial Initialization

```
int numbers[5] = {3, 7};
```



The rest are filled with 0s

Initialization to All Zeros

```
int lotsOfNumbers [1000] = {0};
```



All filled with 0s

- 따라서 모든 배열을 0으로 초기화 하고자 할 경우 처음 원소만 0으로 설정하면 된다.



배열의 초기화

- 원소의 개수를 컴파일러가 결정하는 것에 대해서
 - 컴파일할 때 배열 원소의 개수를 컴파일러가 스스로 결정하도록 하는 것은 좋지 않은 방법이다.
 - 그 이유는 프로그래머가 생각한 개수와 컴파일러가 결정하는 개수가 다를 수 있기 때문이다.
 - 그러나 `char` 형 배열을 문자열(string)으로 초기화할 때에는 이 방법이 오히려 안전하며, 프로그래머에게 배열 원소의 개수를 알려 주어야 한다면 다음과 같이 할 수 있다.

Ex)

```
short things[] = {1, 5, 3, 8};  
int num_elements = sizeof(things) / sizeof(short);
```


배열의 초기화

- 예제 프로그램 - 배열의 초기화1
 - 5개의 배열 원소를 초기화하고 이를 출력한다.
x[5]는 선언되지 않은 배열 원소이다. 따라서 x[5]는 쓰레기값을 유지하고 있다.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int x[5] = {1, 2, 3, 4};
6
7     printf("x[2] = %d, x[4] = %d, x[5] = %d\n", x[2], x[4], x[5]);
8
9     return 0;
10 }
```

```
[root@mclab chap8]# vi chap8-2.c
[root@mclab chap8]# gcc -o chap8-2 chap8-2.c
[root@mclab chap8]# ./chap8-2
x[2] = 3, x[4] = 0, x[5] = 6671520
[root@mclab chap8]#
```

배열의 초기화

■ 예제 프로그램 - 배열의 초기화2

- 5개의 배열 원소가 생성하고 x[0]과 x[2]에 1, 2를 할당한다.
x[4]는 초기화되지도 값이 할당되지 않았다. 또한 x[5]는 선언되지 않은 배열 원소이다. 즉, x[4], x[5]는 쓰레기 값을 가진다.

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int x[5];
6     // x[5] = {1, 2, 3, 4, 5};
7     // x[5] = 9;
8     x[0] = 1;
9     x[2] = 2;
10
11     printf("x[2] = %d, x[4] = %d, x[5] = %d\n", x[2], x[4], x[5]);
12
13     return 0;
14 }

```

배열 선언 이후의 초기화는 컴파일 에러

다음과 같은 배열 원소에 대한 값의 할당만 가능

```

[root@mclab chap8]# vi chap8-3.c
[root@mclab chap8]# gcc -o chap8-3 chap8-3.c
[root@mclab chap8]# ./chap8-3
x[2] = 2, x[4] = 134513608, x[5] = -1074547336
[root@mclab chap8]#

```

배열 원소의 swap

■ Exchanging Values

- 배열 또한 일반 변수와 마찬가지로 첨자를 사용하여 배열 원소에 값을 대입하는 것이 가능하다.

Ex)

```
numbers[3] = numbers[1];
numbers[1] = numbers[3];
```

- 그러나 위 방식과 같이 코드를 작성하면 결국 number[1]과 number[3]은 같은 값을 가지게 된다.

before ;

[0]	[1]	[2]	[3]	[4]
3	7	12	24	45

number[3] = number[1] ;

[0]	[1]	[2]	[3]	[4]
3	7	12	7	45

number[1] = number[3] ;

[0]	[1]	[2]	[3]	[4]
3	7	12	7	45

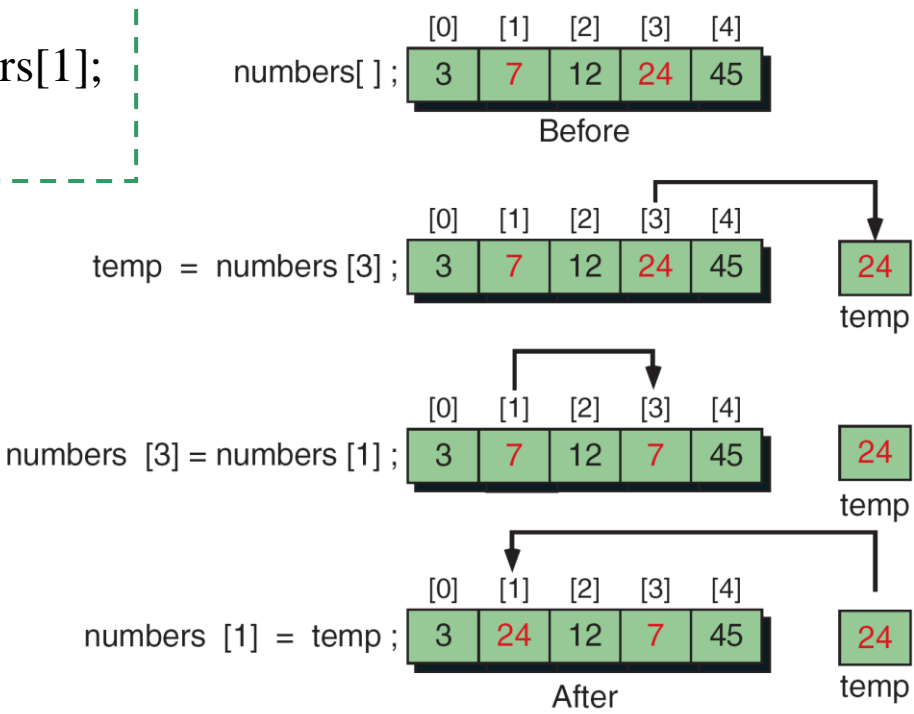


배열 원소의 swap

- 따라서 만약 `numbers[3]`과 `numbers[1]` 값을 바꾸려 할 때는 아래와 같아야 한다.

Ex)

```
temp = numbers[3];
numbers[3] = numbers[1];
numbers[1] = temp;
```

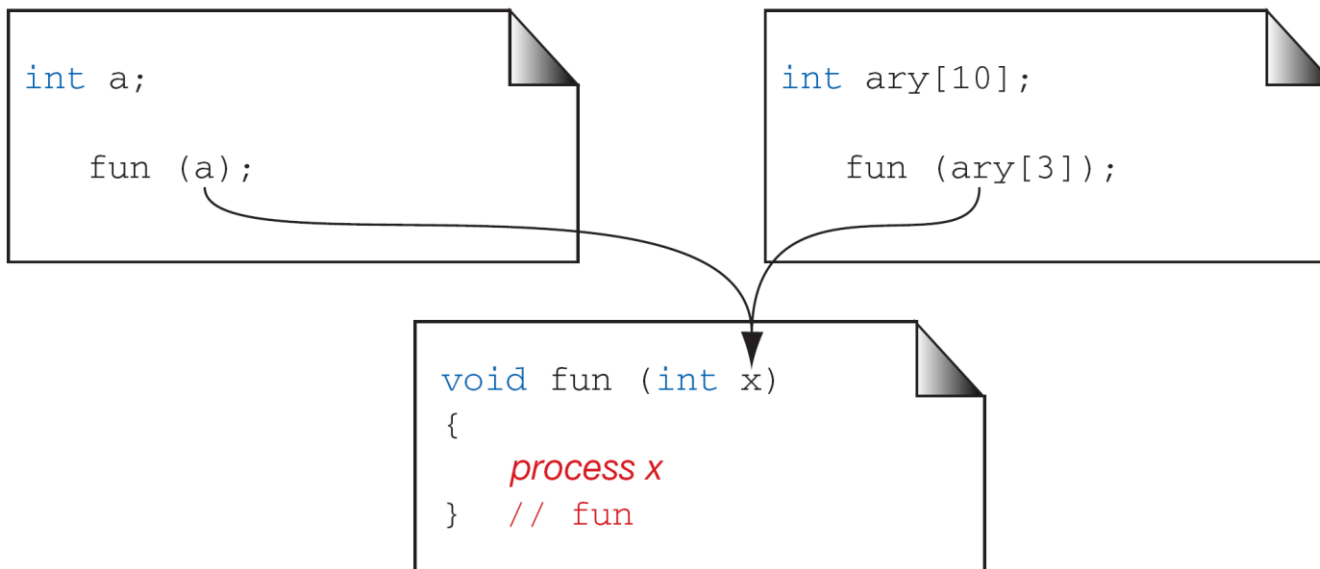


Arrays and Functions

■ Passing individual elements

■ Passing Data Values

일반 변수 값을 전달하듯 배열 원소를 함수 전달 인수로 사용할 수 있다.

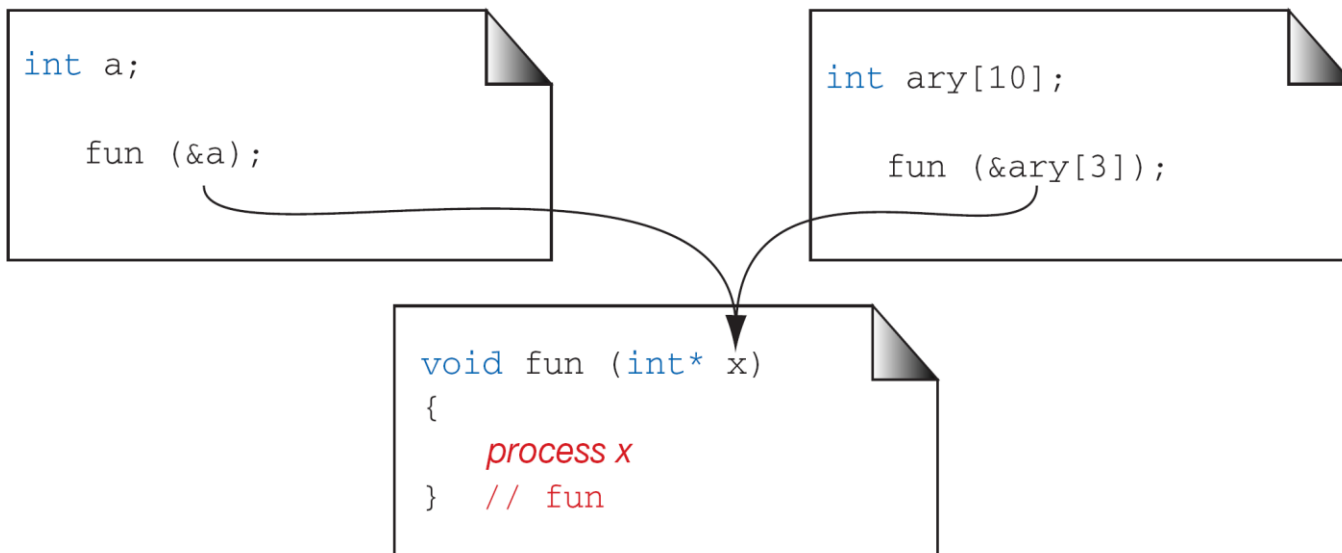


Arrays and Functions

■ Passing individual elements

■ Passing Addresses

일반 변수 값의 주소를 전달하는 방법과 동일하게
배열 원소의 주소를 함수 전달 인수로 넘겨 사용한다.



Arrays and Functions

- 예제 프로그램 - 함수의 배열 인자 전달1
 - 5개의 배열 원소가 각각 1,2,3,4,5로 초기화된다.
x[3]의 값을 인자로 해당 값을 출력하는 함수 arrayprt 호출하면
함수에는 x[3]의 값 4가 전달, "value is 4"가 출력 된다.

```

1  #include <stdio.h>
2
3  void arrayprt(int x);
4
5  int main(void)
6  {
7      int x[5] = {1, 2, 3, 4, 5};
8      arrayprt(x[3]);
9
10     return 0;
11 }
12
13 void arrayprt(int x)
14 {
15     printf("value is %d\n", x);
16 }
    
```

```

[root@mclab chap8]# vi chap8-4.c
[root@mclab chap8]# gcc -o chap8-4 chap8-4.c
[root@mclab chap8]# ./chap8-4
value is 4
[root@mclab chap8]#
    
```

Arrays and Functions

■ Passing the whole array

- 개별 원소가 아닌 배열 전체를 모두 parameter로 전달하기 위해서는 배열 이름을 사용하면 된다.
- 배열의 이름은 배열의 시작 주소를 가리킨다.
- 함수에서 값이 바뀌면 원 배열 값 역시 바뀌게 된다.

```
int ary[10];
```

```
fun (ary);
```

```
void fun (int fAry[ ])
```

```
{
```

```
    process x
```

```
} // fun
```


Arrays and Functions

■ 예제 프로그램 - 함수의 배열 주소 전달

- 5개의 배열 원소가 각각 1,2,3,4,5로 초기화된다. i의 값을 거듭제곱하여 저장하는 함수 mularray에 배열 이름을 인자로(즉 배열의 주소 값을 인자로) 호출한다.
배열 a의 각 원소에는 i의 거듭제곱 값이 저장된다.

```
1 #include <stdio.h>
2
3 void mularray(int x[]);
4
5 int main(void)
6 {
7     int a[5] = {1, 2, 3, 4, 5};
8     mularray(a);
9     printf("a[0]=%d, a[1]=%d, a[2]=%d, a[3]=%d, a[4]=%d\n", a[0], a[1], a[2], a[3], a[4]);
10    return 0;
11 }
12
13 void mularray(int x[]);
14 {
15     int i;
16     for(i=0; i<5; i++)
17         x[i] = i * i;
18 }
```

```
[root@mclab chap8]# vi chap8-5.c
[root@mclab chap8]# gcc -o chap8-5 chap8-5.c
[root@mclab chap8]# ./chap8-5
a[0]=0, a[1]=1, a[2]=4, a[3]=9, a[4]=16
[root@mclab chap8]#
```

Arrays and Functions

■ 예제 프로그램 - 함수의 배열 인자 전달2

- 5개의 배열 원소가 각각 1,2,3,4,5로 초기화된다. i 의 값($i=4$)을 거듭제곱하여 저장하는 함수 mularray에 이중 $a[4]$ 의 값을 인자로 넘겨주어 호출한다.

배열 $a[4]$ 에는 i 의 거듭제곱 값이 저장되길 기대하지만, $a[4]$ 의 주소를 인자로 넘겨주지 않았으므로 결과는 그렇지 않다.

```

1 #include <stdio.h>
2
3 void mularray(int x);
4
5 int main(void)
6 {
7     int a[5] = {1, 2, 3, 4, 5};
8     mularray(a[4]);
9     printf("a[0]=%d, a[1]=%d, a[2]=%d, a[3]=%d, a[4]=%d\n", a[0], a[1], a[2], a[3], a[4]);
10    return 0;
11 }
12
13 void mularray(int x)
14 {
15     int x=4;
16     x=i*i;
17 }

```

```

[root@mclab chap8]# vi chap8-6.c
[root@mclab chap8]# gcc -o chap8-6 chap8-6.c
[root@mclab chap8]# ./chap8-6
a[0]=1, a[1]=2, a[2]=3, a[3]=4, a[4]=5
[root@mclab chap8]#

```

Two-dimensional Arrays

■ 2차원 배열

- C는 다차원 배열을 제공
- 다차원 배열의 가장 간단한 형태가 2차원 배열
- 2차원 배열은 1차원 배열들에 대한 배열로 행(row)와 열(column)로 구성

■ 2차원 배열의 선언

- 다음은 10행 20열 크기의 2차원 정수 배열 d를 선언한 것

```
int d[10][20];
```

■ 2차원 배열과 메모리

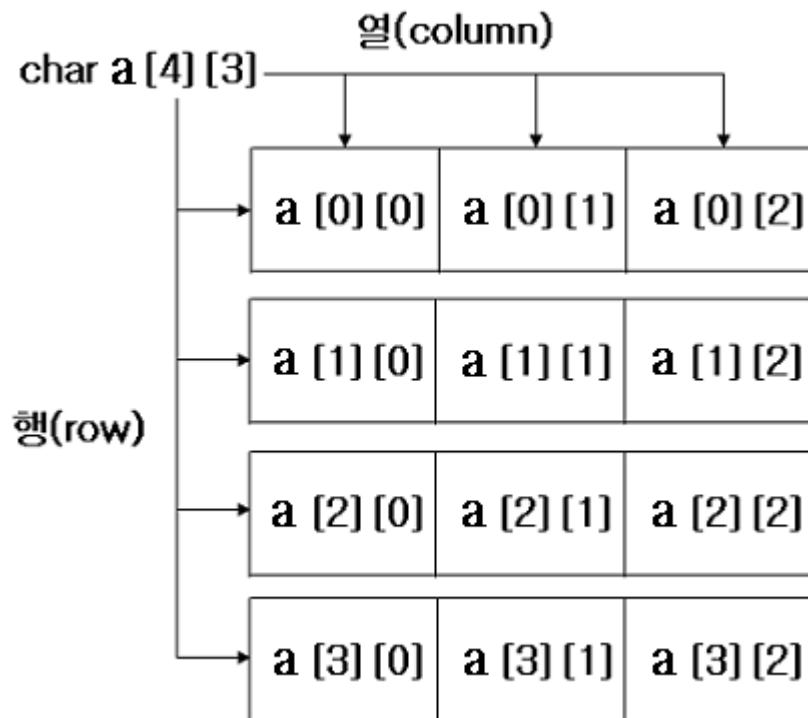
- 배열을 저장하는데 필요한 메모리의 양

바이트수 = 첫번째 인덱스의 크기 * 두번째 인덱스의 크기 * sizeof(기본 형)

Two-dimensional Arrays

- 2차원 배열과 메모리

```
char a[4][3];
```



Two-dimensional Arrays

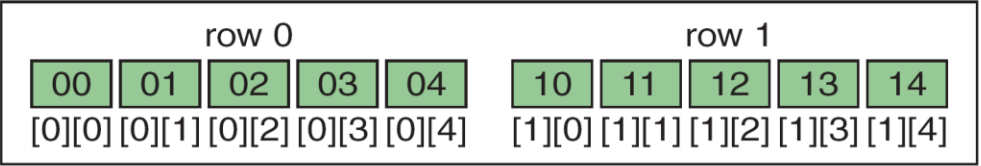
- 그러나 C언어에서는 1차원 배열 만이 존재하고 엄밀한 의미에서 2차원 이상의 배열은 존재하지 않는다.

```
Ex)      int c[4];      // int 4개의 배열
         int x[4][3];   // int[3] 4개의 배열
```

- 예에서 c[4] 는 <int형 변수> 를 한 개의 요소로 하는 크기 4개의 배열 c 를 선언한 것이다.
- x[4][3] 는 엄밀히 말하면 4*3의 배열을 선언하고 있는 것이 아니라 <int형 3개의 크기 배열> 을 한 개의 요소로 하는 크기 4개의 배열을 선언한 것이다. 즉, int x[4][3]은 x[0]부터 x[3] 이라는 4개의 요소로 된 배열을 선언한 것이다.

00	01	02	03	04
10	11	12	13	14

User's View



Memory View

Two-dimensional Arrays

■ 예제 프로그램 - 2차원 배열

- 12개의 배열 원소가 a[0][0]부터 a[3][2]까지 선언되어 있다.
각각의 배열 원소는 for 루프에 의해 값이 할당되고 이어서 다음 for loop에 의해 값이 출력된다.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a[4][3];
6     int i, j;
7
8     for(i=0; i<4; i++)
9         for(j=0; j<3; j++)
10             a[i][j]=i*j;
11
12     for(i=0; i<4; i++)
13         for(j=0; j<3; j++)
14             printf("a[%d][%d]=%d\n", i, j, a[i][j]);
15
16     return 0;
17 }
```

```
[root@mclab chap8]# vi chap8-7.c
[root@mclab chap8]# gcc -o chap8-7 chap8-7.c
[root@mclab chap8]# ./chap8-7
a[0][0]=0
a[0][1]=0
a[0][2]=0
a[1][0]=0
a[1][1]=1
a[1][2]=2
a[2][0]=0
a[2][1]=2
a[2][2]=4
a[3][0]=0
a[3][1]=3
a[3][2]=6
[root@mclab chap8]#
```

Two-dimensional Arrays

■ 예제 프로그램 - 2차원 배열의 초기화

- 12개의 배열 원소가 a[0][0]부터 a[3][2]까지 선언되면서 초기화 된다.
초기화된 2차원 배열의 각 원소의 값이 for loop에 의해 출력된다.

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a[4][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};
6     int i, j;
7
8     for(i=0; i<4; i++)
9         for(j=0; j<3; j++)
10             printf("a[%d][%d]=%d\n", i, j, a[i][j]);
11
12     return 0;
13 }

```

```

[root@mclab chap8]# vi chap8-8.c
[root@mclab chap8]# gcc -o chap8-8 chap8-8.c
[root@mclab chap8]# ./chap8-8
a[0][0]=1
a[0][1]=2
a[0][2]=3
a[1][0]=4
a[1][1]=5
a[1][2]=6
a[2][0]=7
a[2][1]=8
a[2][2]=9
a[3][0]=10
a[3][1]=11
a[3][2]=12
[root@mclab chap8]#

```

Two-dimensional Arrays

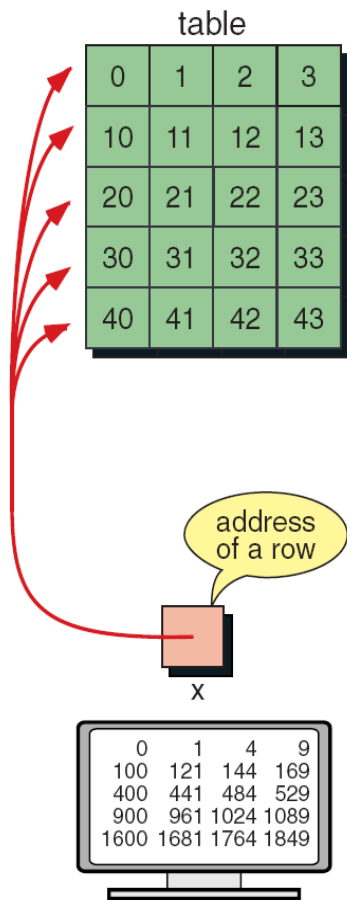
■ Passing a Two-dimensional Array to a Function

- Passing Individual Elements
 - 1차원 배열과 마찬가지로 개별 값을 전달할 수 있다.
- Passing a Row
 - 1차원 배열과 마찬가지로 각 행의 이름을 이용하여 행 단위로 전달할 수 있다.
 - 함수 안에서 배열 값이 바뀌면 원래 배열 값 역시 바뀐다.

```
#define MAX_ROWS 5
#define MAX_COLS 4
// Function Declarations
void print_square (int []);
int main (void)
{
    int table [MAX_ROWS][MAX_COLS] =
        {
            { 0, 1, 2, 3 },
            { 10, 11, 12, 13 },
            { 20, 21, 22, 23 },
            { 30, 31, 32, 33 },
            { 40, 41, 42, 43 }
        }; /* table */

    ...
    for (int row = 0; row < MAX_ROWS; row++)
        print_square (table [row]);
    ...
    return 0;
} // main
```

```
void print_square (int x[])
{
    for (int col = 0; col < MAX_COLS; col++)
        printf("%6d", x[col] * x[col]);
    printf ("\n");
    return;
} // print_square
```



Two-dimensional Arrays

- Passing the whole Array
 - 전체 배열은 배열 이름을 이용하여 전달할 수 있다.
 - 함수 안에서 배열 값이 바뀌면 원래 배열 값 역시 바뀐다.

```
const int cMAX_ROWS    = 5;
const int cMAX_COLS    = 4;
double average ( int [ ][cMAX_COLS] );
int main ( )
{
    int
        ave;
    int
        table [ MAX_ROWS ] [cMAX_COLS] =
            {
                { 0, 1, 2, 3 },
                { 10, 11, 12, 13 },
                { 20, 21, 22, 23 },
                { 30, 31, 32, 33 },
                { 40, 41, 42, 43 }
            };

    ...
    ave = average ( table );
    ...
    return 0 ;
} // main
```

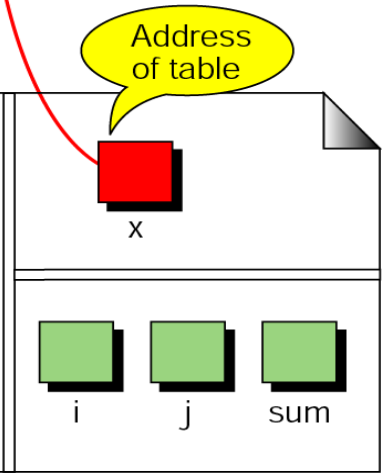
table

0	1	2	3
10	11	12	13
20	21	22	23
30	31	32	33
40	41	42	43

```
double average ( int x [ ][cMAX_COLS] )
{
    double sum = 0 ;

    for (int i = 0 ; i < cMAX_ROWS ; i++)
        for (int j = 0 ; j < cMAX_COLS ; j++)
            sum += x [ i ] [ j ];

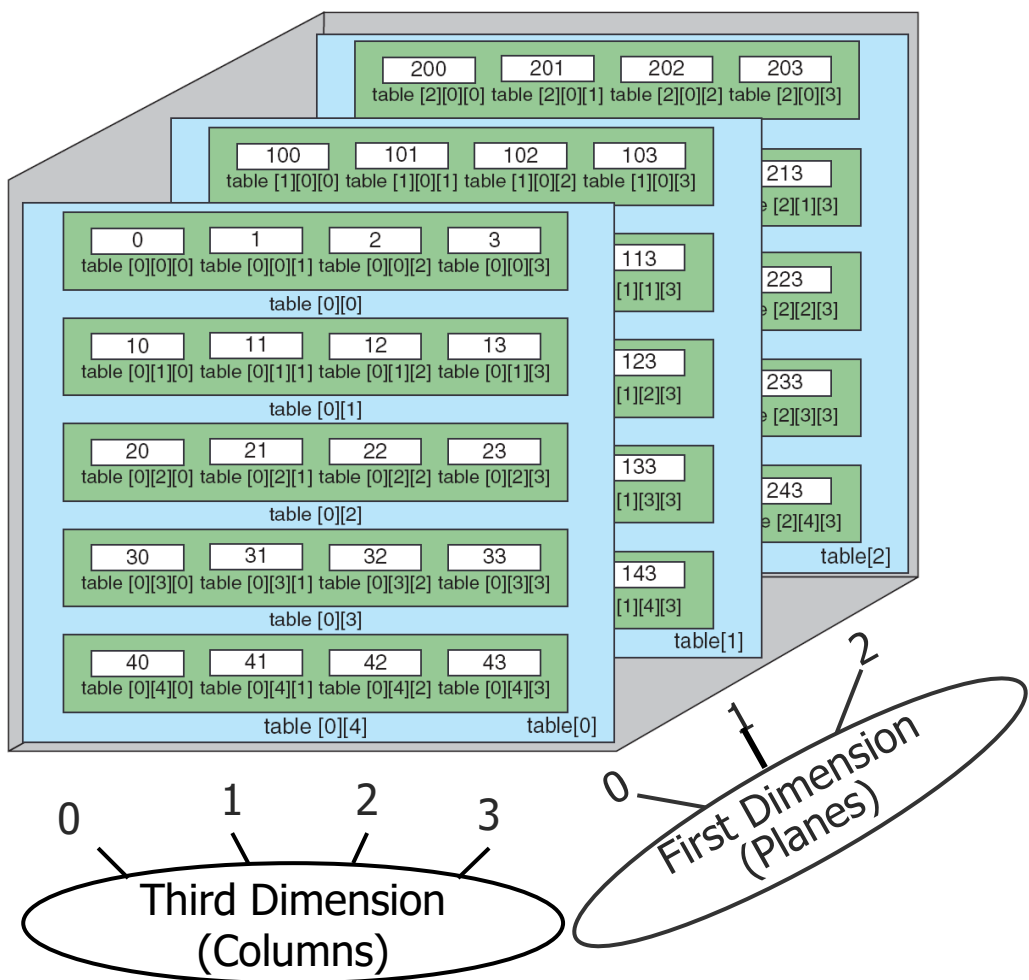
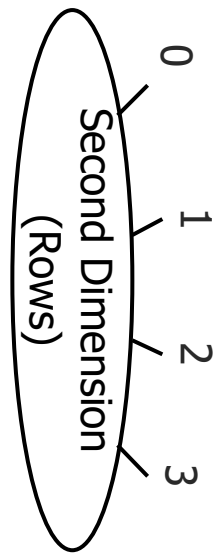
    return( sum / ( cMAX_ROWS * cMAX_COLS ) );
} // average
```



Multi-dimensional Arrays

- 다차원 배열은 2차원, 3차원 혹은 그 이상의 차원을 가질 수 있다.
- 이 배열의 선언은

```
int table[3][5][4];
```



Multi-dimensional Arrays

■ Initialization

- 초기화 방법은 1차 배열과 동일하다. 단 선언한 배열의 row, column등의 개수와 초기값의 row, column등의 개수가 동일해야 한다.

```
int table [3][5][4] = {0};
```

☞ 모든 element 를 0으로 초기화하는 경우

```
int table[3][5][4] =
{
    { /* Plane 0*/
        {0, 1, 2, 3},          /* Row 0 */
        {10, 11, 12, 13},     /* Row 1 */
        {20, 21, 22, 23},     /* Row 2 */
        {30, 31, 32, 33},     /* Row 3 */
        {40, 41, 42, 43}      /* Row 4 */
    },
    { /* Plane 1*/
        {100, 101, 102, 103},  /* Row 0 */
        {110, 111, 112, 113}, /* Row 1 */
        {120, 121, 122, 123}, /* Row 2 */
        {130, 131, 132, 133}, /* Row 3 */
        {140, 141, 142, 143}  /* Row 4 */
    },
    { /* Plane 2*/
        {200, 201, 202, 203},  /* Row 0 */
        {210, 211, 212, 213}, /* Row 1 */
        {220, 221, 222, 223}, /* Row 2 */
        {230, 231, 232, 233}, /* Row 3 */
        {240, 241, 242, 243}  /* Row 4 */
    }
}; /* table */
```

☞ 모든 element 에 개별적인 초기값을 선언하는 경우.