Fundamentals of Computer System
- chapter 10. Arrays and Pointers

민기복

Ki-Bok Min, PhD

서울대학교 에너지자원공학과 조교수
Assistant Professor, Energy Resources Engineering

SEOUL NATIONAL UNIVERSITY

Last week
Chapter 9. Functions (함수)

- Functions and how to define them
- Arguments and return values (전달인자와 리턴값)
- Function types (함수의 데이터형)
- Declaring a function (함수 선언)
- Recursion (재귀)
- Pointers : a first look

Functions
What is it?

- Functions: a self-contained unit of program code designed to accomplish a particular task.
- 함수: 하나의 특정 작업을 수행하도록 독립적으로 설계된 프로그램 코드의 한 단위
  - Ex) printf(): causes data to be printed on your screen
  - Strlen(): tells a program how long a certain string is.
- Why do we need it?
  - Only one function can be used many times for repeated work.
  - Imagine we write a program for printf() each time you need it. – terrible!

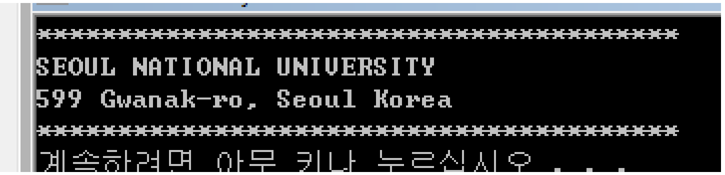
Analyzing the program

```c
/* lethead1.c */
#include <stdio.h>
#define NAME "SEOUL NATIONAL UNIVERSITY"
#define ADDRESS "599 Gwanak-ro, Seoul Korea"
#define WIDTH 40

void starbar(void);   /* prototype the function */

int main(void)
{
    starbar();
    printf("%s₩n", NAME);
    printf("%s₩n", ADDRESS);

    starbar();          /* use the function       */

    return 0;
}

void starbar(void)   /* define the function     */
{
    int count;

    for (count = 1; count <= WIDTH; count++)
        putchar('*');
    putchar('₩n');
}
```

Function prototype (함수 프로토타입): tells the compiler what sort of function starbar() is

Function call (함수호출): causes the function to be executed

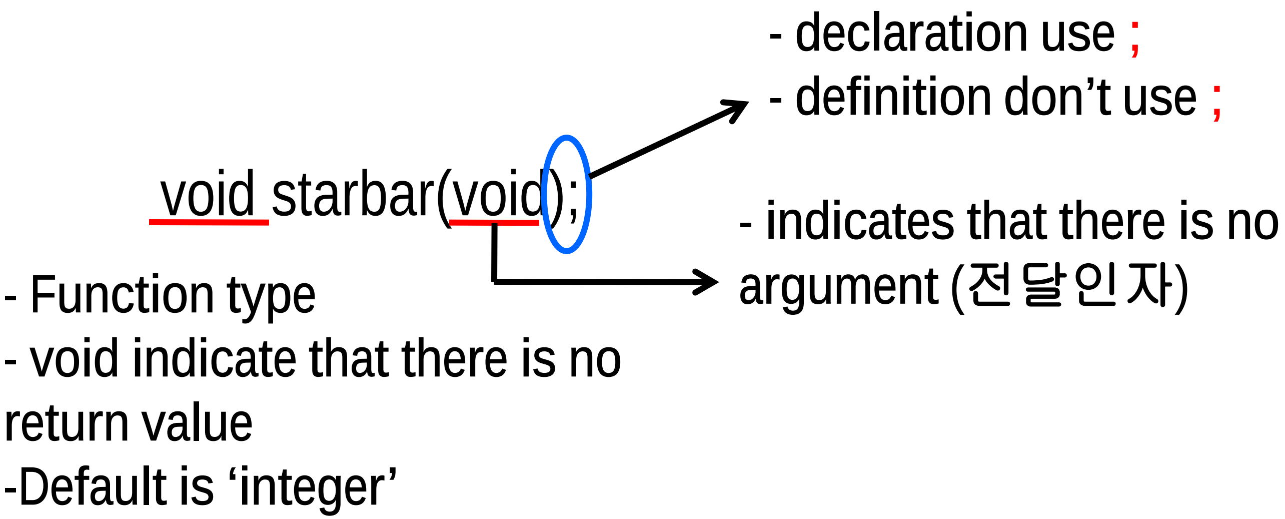
Function definition (함수정의): specifies exactly what the function does



```
****************************************
SEOUL NATIONAL UNIVERSITY
599 Gwanak-ro, Seoul Korea
****************************************
계속하려면 아무 키나 누르십시오 . . .
```

Functions
Declaration

- Any program that uses a function should declare the type before it is used – 함수의 데이터형 미리 선언.



void starbar(void);

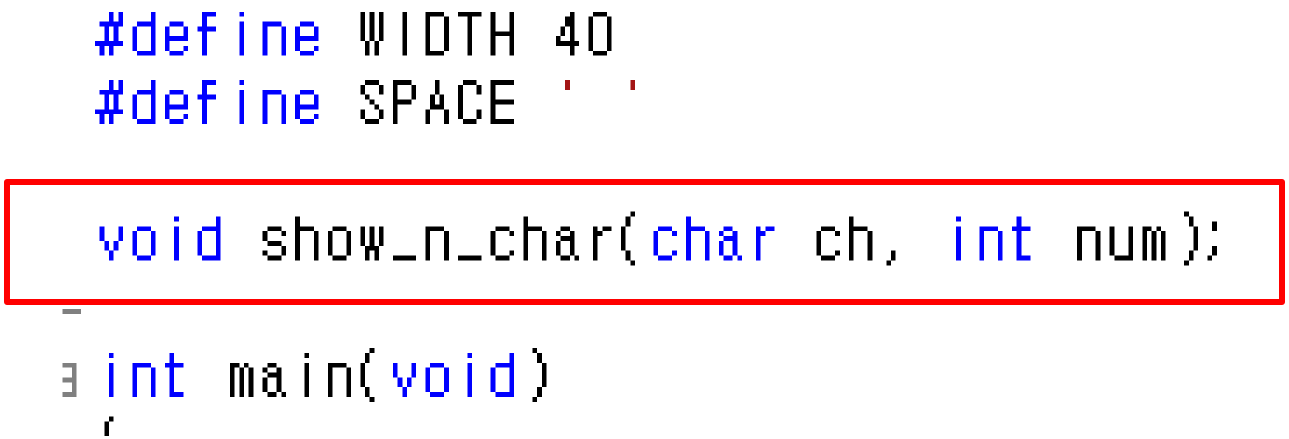
- declaration use ;
- definition don't use ;

- indicates that there is no argument (전달인자)

- Function type
- void indicate that there is no return value
-Default is 'integer'

  – This line announces that the program uses a type void function called starbar() & compiler expect to find the definition for this function elsewhere. May put this inside main()

Functions
Argument (전달인자)



```
#define WIDTH 40
#define SPACE ' '

void show_n_char(char ch, int num);

int main(void)
{
```

- ch & num: argument (전달인자), formal argument (형식전달인자) or formal parameter (형식매개변수)
- ch: char type, num: int

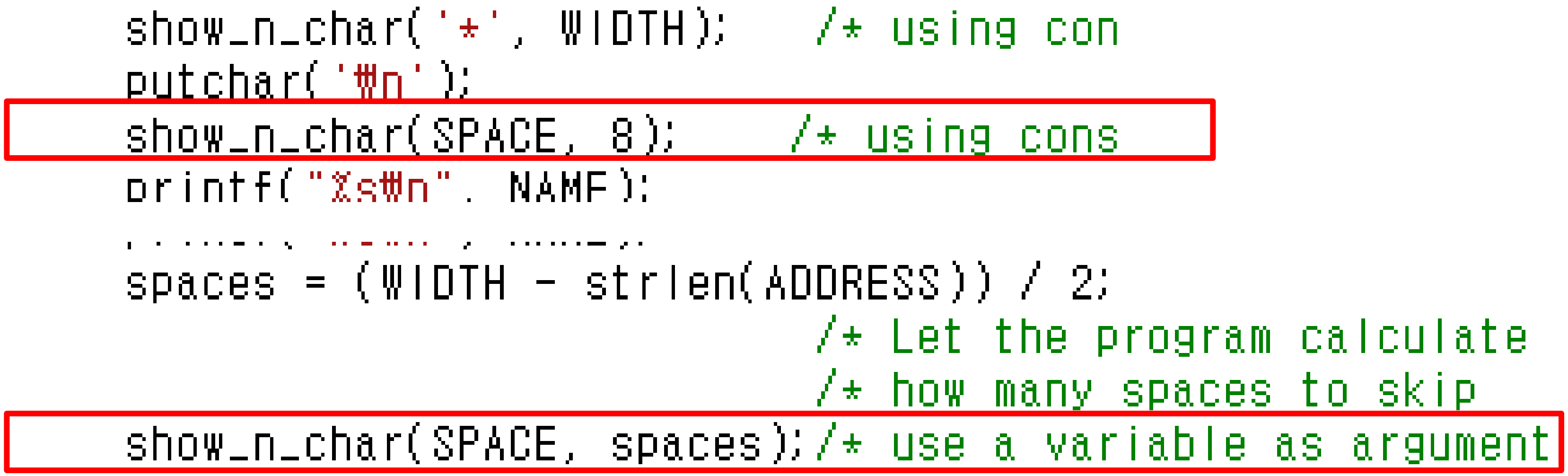
void show_n_char(ch, num);

char ch

int num

Valid (but old style)

Functions
Arguments (전달인자)

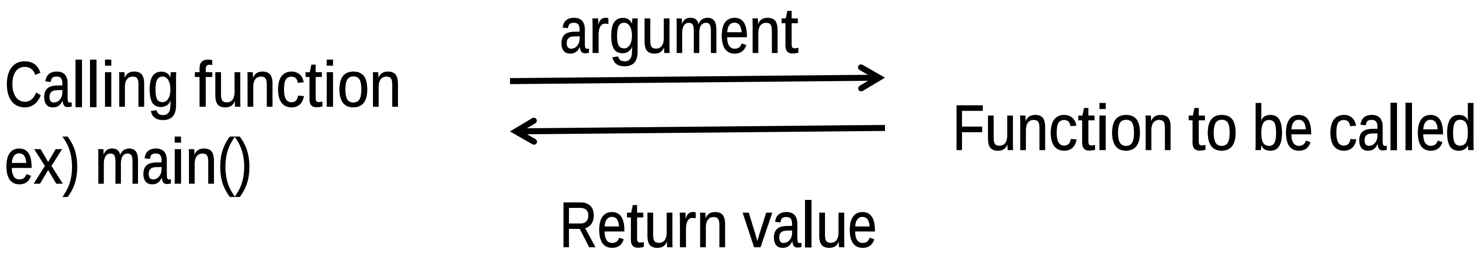


```
show_n_char('*', WIDTH);    /* using con
putchar('₩n');
show_n_char(SPACE, 8);     /* using cons
printf("%s₩n", NAME);
spaces = (WIDTH - strlen(ADDRESS)) / 2;
                             /* Let the program calculate
                             /* how many spaces to skip
show_n_char(SPACE, spaces);/* use a variable as argument
```

- SPACE, 8: actual argument (실전달인자)
- Formal parameter (형식매개변수): called functioin – show_n_char()
- Actual argument (실질전달인자): calling function – main()

Functions
Arguments (전달인자) & return value

- Argument send information to functions.

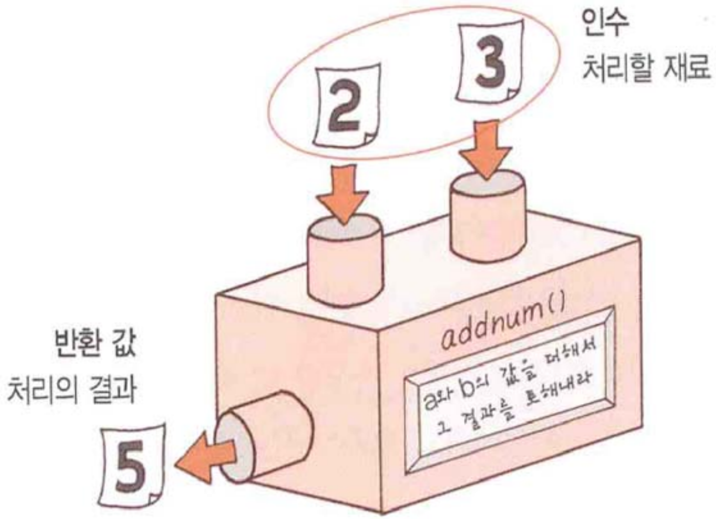


Calling function
ex) main()

argument

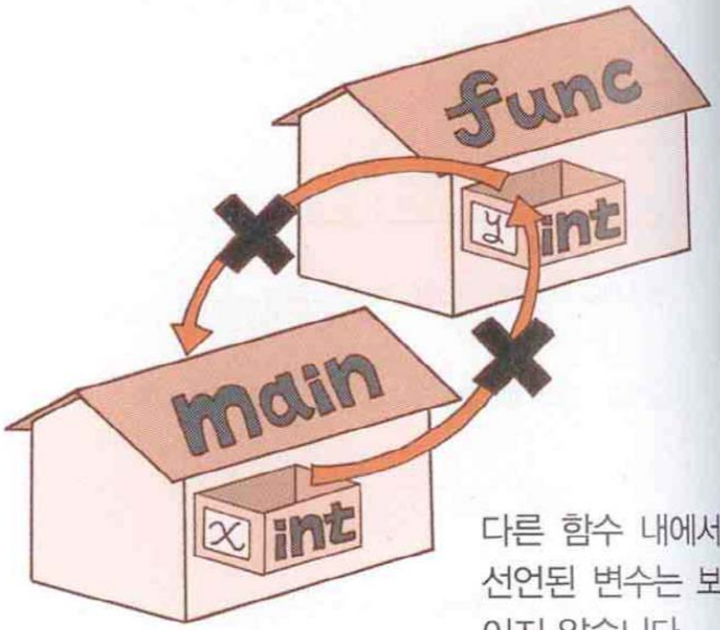
Return value

Function to be called



Functions

Local and global variable (지역변수와 전역변수)

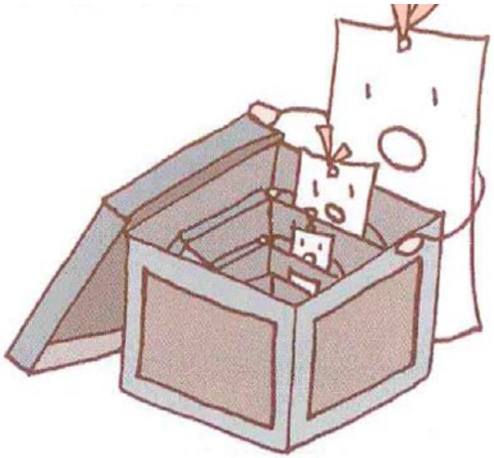


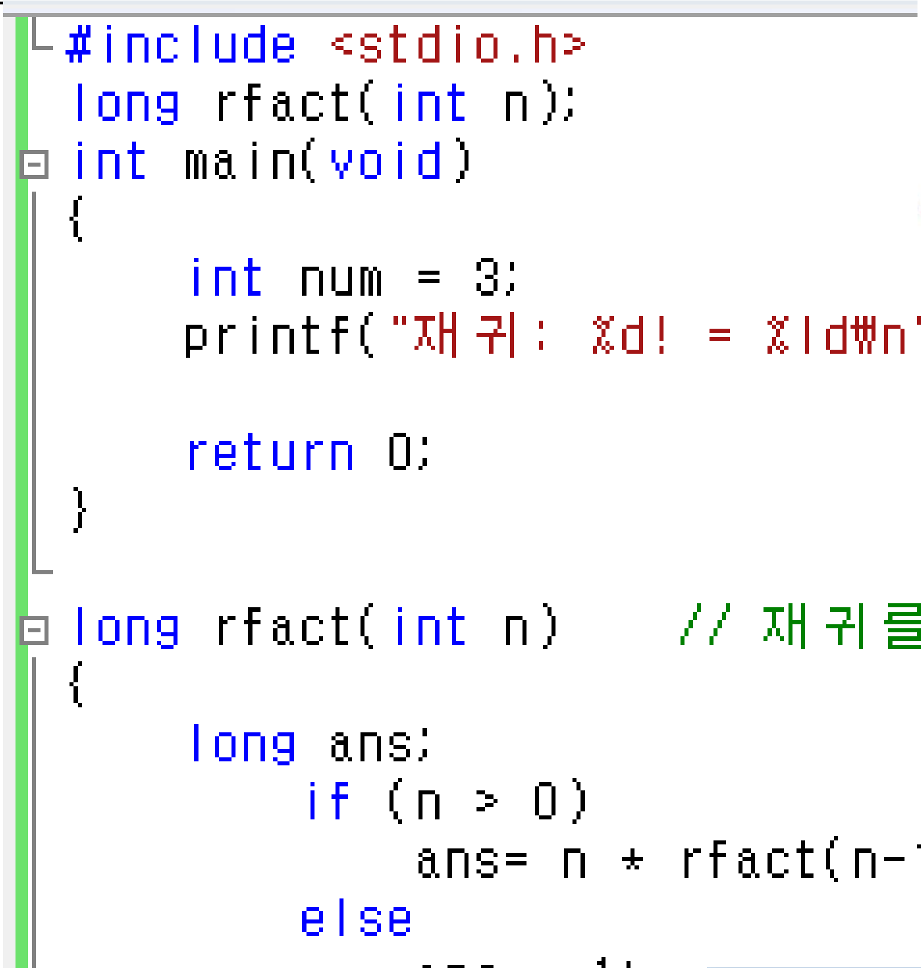
다른 함수 내에서
선언된 변수는 보
이지 않습니다.

Function
Function types

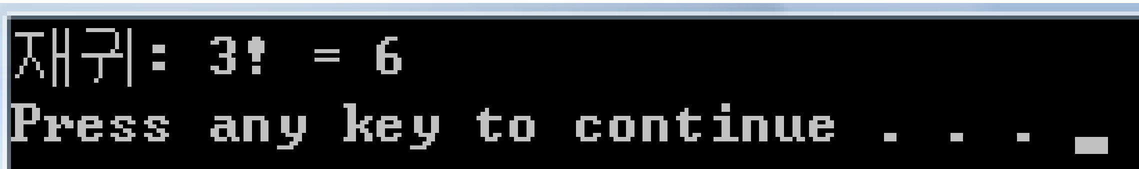
- You generally inform the compiler about functions by declaring them in advance – should come before it is used.
- Function should be declared by type.
- A function with a return value should be declared the same type as the return value.
- If no type is given → default is ‘integer’.

Recursion (재귀)
함수가 자기자신을 호출



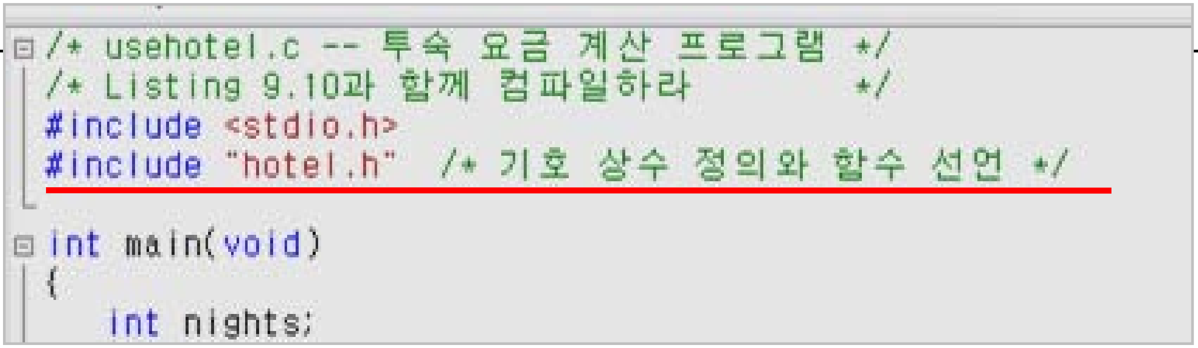


```
#include <stdio.h>
long rfact(int n);
int main(void)
{
        int num = 3;
        printf("재귀: %d! = %ld₩n",num, rfact(num));

        return 0;
}

long rfact(int n)     // 재귀를 사용하는 함수
{
        long ans;
                if (n > 0)
                        ans= n * rfact(n-1);
                else
                        ans = 1;
         return ans;
}
```



```
재귀: 3! = 6
Press any key to continue . . . _
```

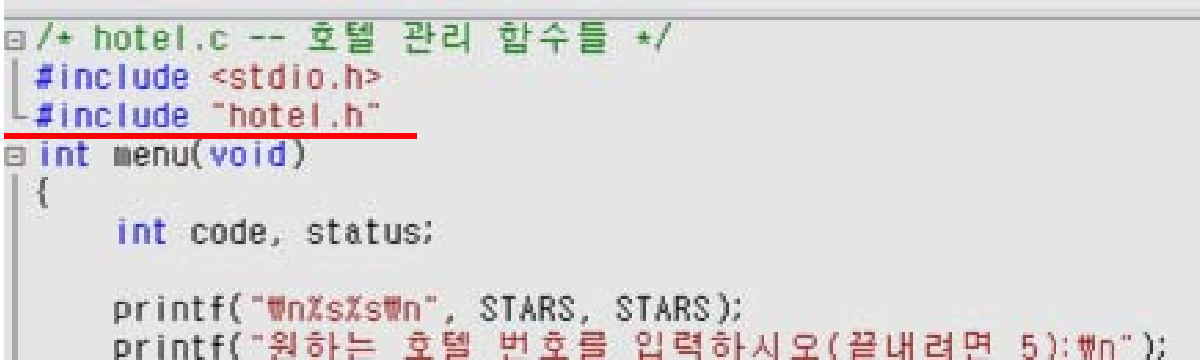
Compiling programs with two or more source code files



```
/* usehotel.c -- 투숙 요금 계산 프로그램 */
/* Listing 9.10과 함께 컴파일하라        */
#include <stdio.h>
#include "hotel.h"  /* 기호 상수 정의와 함수 선언 */

int main(void)
{
    int nights;
```

usehotel.c

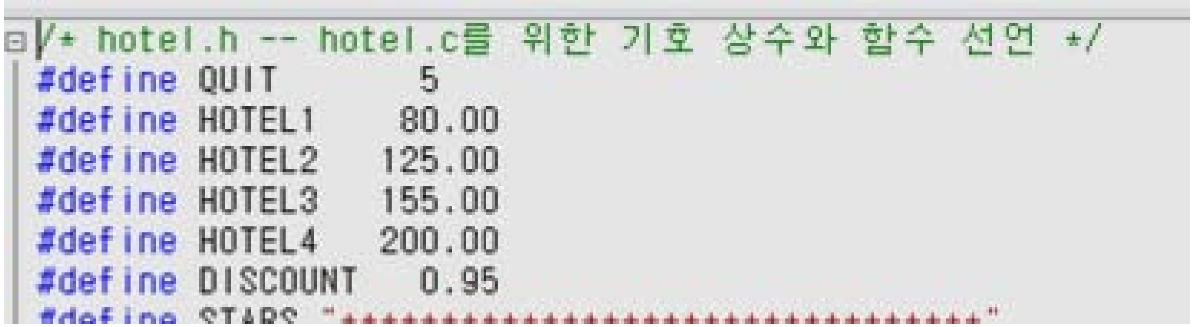
- Contains main()



```
/* hotel.c -- 호텔 관리 함수들 */
#include <stdio.h>
#include "hotel.h"
int menu(void)
{
    int code, status;

    printf("₩n%s%s₩n", STARS, STARS);
    printf("원하는 호텔 번호를 입력하시오(끝내려면 5):₩n");
```

hotel.c

- function definitions



```
/* hotel.h -- hotel.c를 위한 기호 상수와 함수 선언 */
#define QUIT       5
#define HOTEL1    80.00
#define HOTEL2   125.00
#define HOTEL3   155.00
#define HOTEL4   200.00
#define DISCOUNT   0.95
#define STARS "**********************************"
```

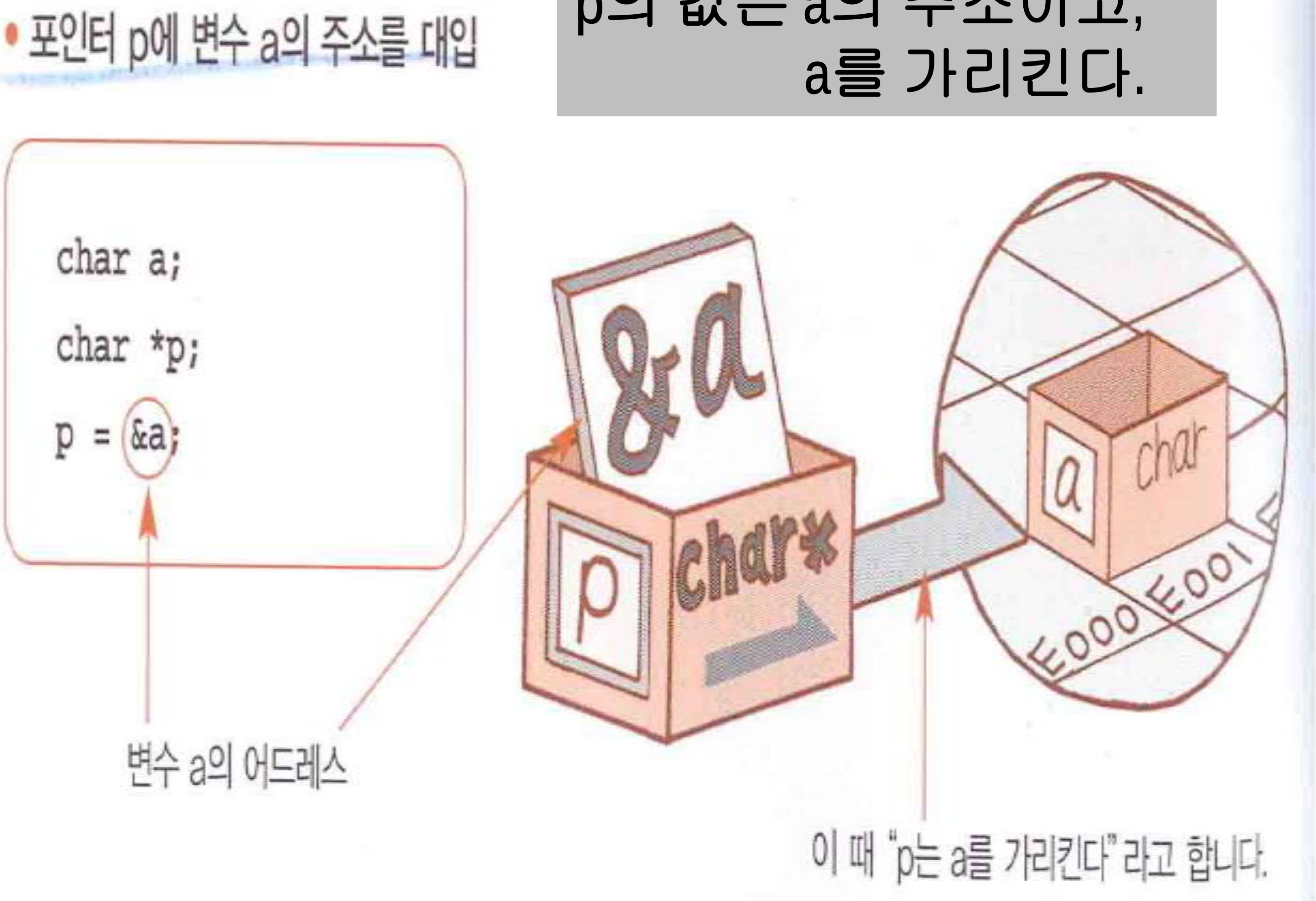
hotel.h

Pointers: a first look
What is it?

- Pointer: a variable whose value is a memory address
- 포인터는 주소를 값으로 가지는 변수이다.
  - Char 형 변수 → 문자
  - int형 변수 → 정수
  - 포인터변수 →
- People: a name and a value
- Computer: an address (a computer's version of name) and a value

Pointers: a first look
What is it?

p의 값은 a의 주소이고,
a를 가리킨다.



• 포인터 p에 변수 a의 주소를 대입

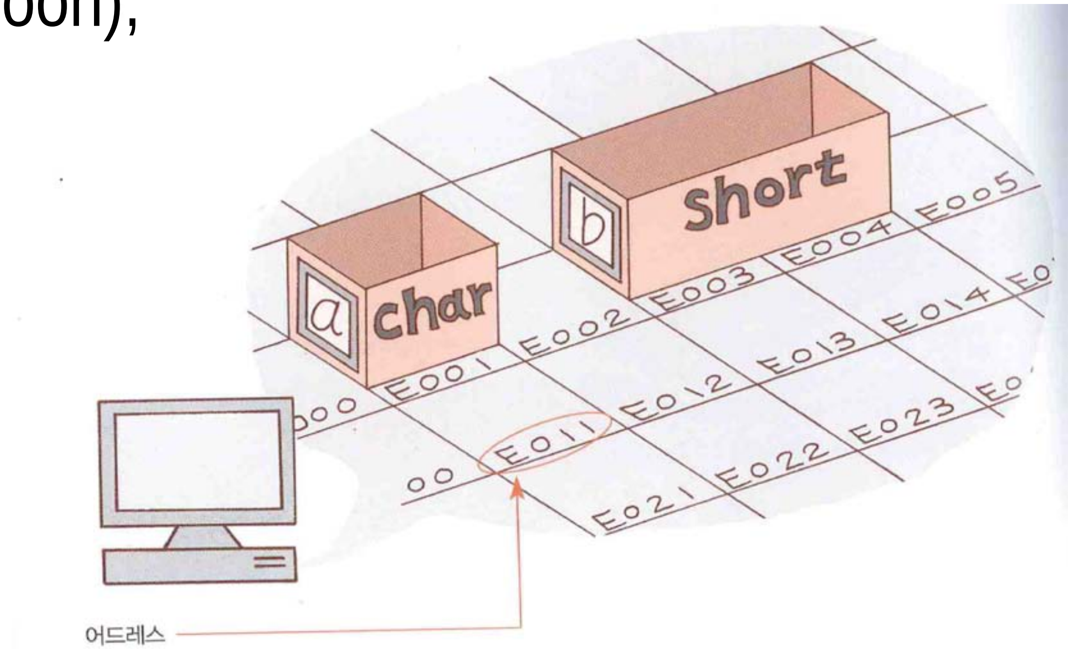
```
char a;
char *p;
p = &a;
```

변수 a의 어드레스

이 때 "p는 a를 가리킨다"라고 합니다.

Pointers: a first look
What is it?

- & operator gives you the address for a variable
- pooh = 24;
- pooh: 변수의 이름, &pooh : 변수의 주소(ex. 0B76)
- Printf(“%d %p\n”,pooh,&pooh);
- 24 0B76



Pointers: a first look
the indirection operator(간접연산자)

```
p = &a;
b = *p;
```

←→

```
b = a
```

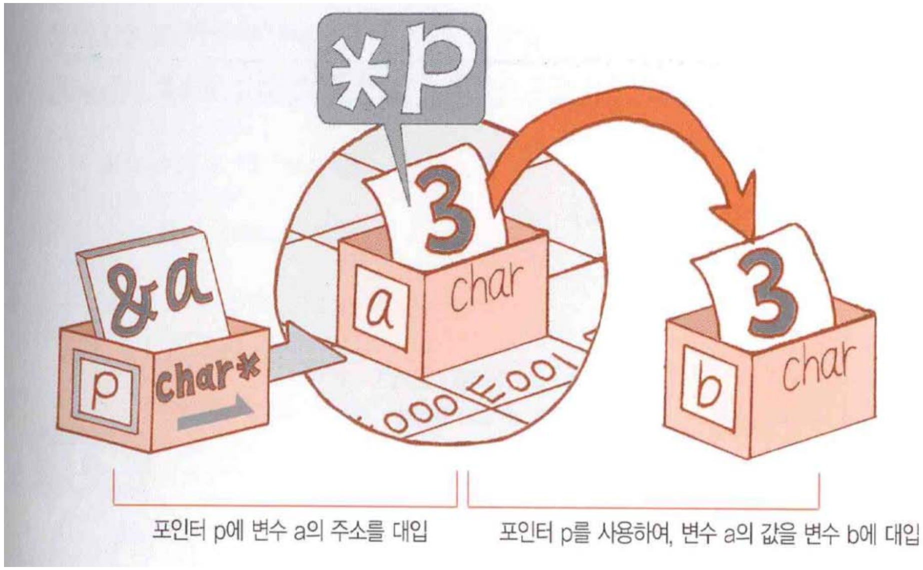
p가 가리키는 주소의 값

ex)

a = 3;

p = &a; //a를 가리키는 포인터

b = *p; //p이 가리키고 있는주소의 값을 b에 대입


*p
3
&a
p
char*
a
char
3
b
char
포인터 p에 변수 a의 주소를 대입
포인터 p를 사용하여, 변수 a의 값을 변수 b에 대입

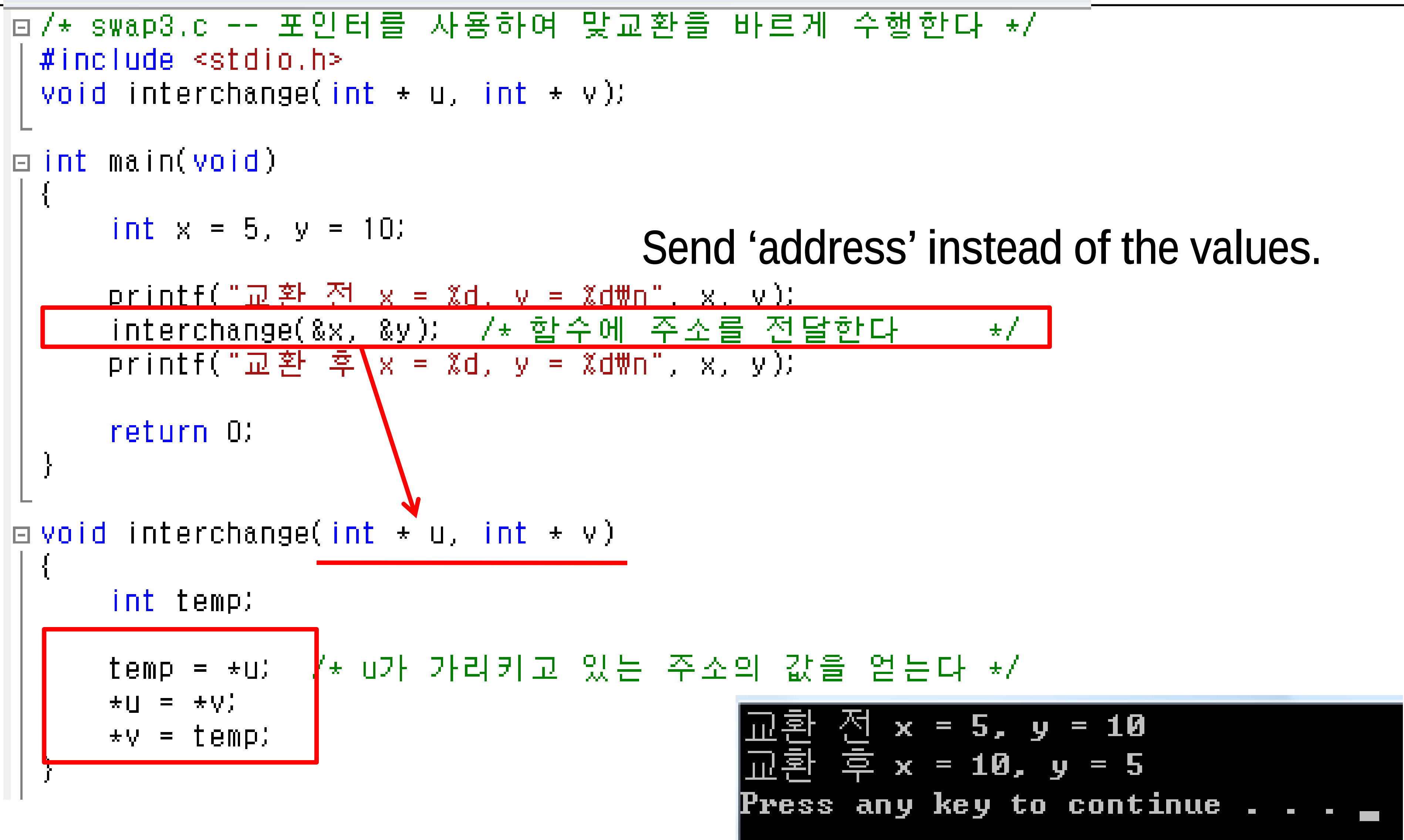
Pointers: a first look
declaring pointers

SEOUL NATIONAL UNIVERSITY

~~pointer a;~~

- Above does not provide sufficient information.
- Various types take different amount of memory and some pointer operations require knowledge of that size.
  - Int *a
  - Char *a
  - Float *a
- Space between * and the pointer name is optional.

Pointers: a first look
Why do we need it? (왜 필요할까?)



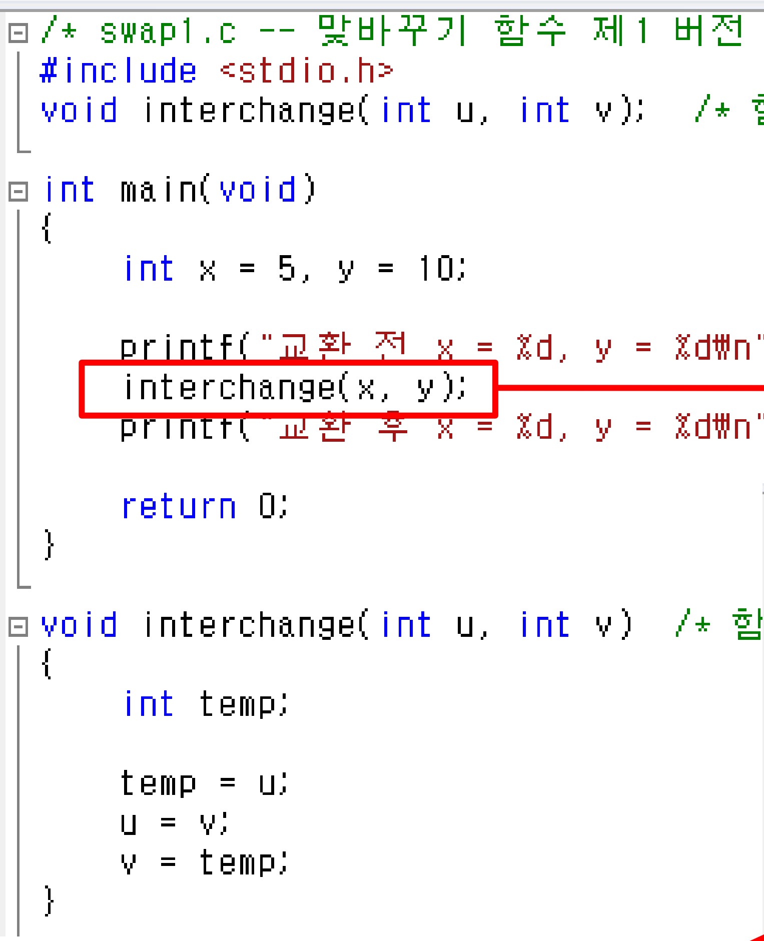
```
/* swap3.c -- 포인터를 사용하여 맞교환을 바르게 수행한다 */
#include <stdio.h>
void interchange(int * u, int * v);

int main(void)
{
    int x = 5, y = 10;

    printf("교환 전 x = %d, y = %d₩n", x, y);
    interchange(&x, &y);  /* 함수에 주소를 전달한다      */
    printf("교환 후 x = %d, y = %d₩n", x, y);

    return 0;
}

void interchange(int * u, int * v)
{
    int temp;

    temp = *u;   /* u가 가리키고 있는 주소의 값을 얻는다 */
    *u = *v;
    *v = temp;
}
```

Send ‘address’ instead of the values.

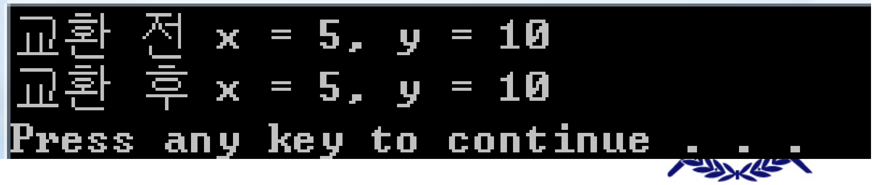
```
교환 전 x = 5, y = 10
교환 후 x = 10, y = 5
Press any key to continue . . .
```

Pointers: a first look
This is why...(이래서 필요하다)

- X 와 y의 주소를 전달함으로써 interchange()가 그변수들에 접근할 수 있게 한다.
- 포인터와 간접연산자 *를 사용함으로써 함수는 각각의 주소에 저장되어 있는 값들을 구하고 변경할 수 있게 된다.
- 포인터는 interchange()함수의 변수들이 지역적이라는 사실을 극복해준다. Main()에 손을뻗쳐 저장되어 있는 값을 변경.

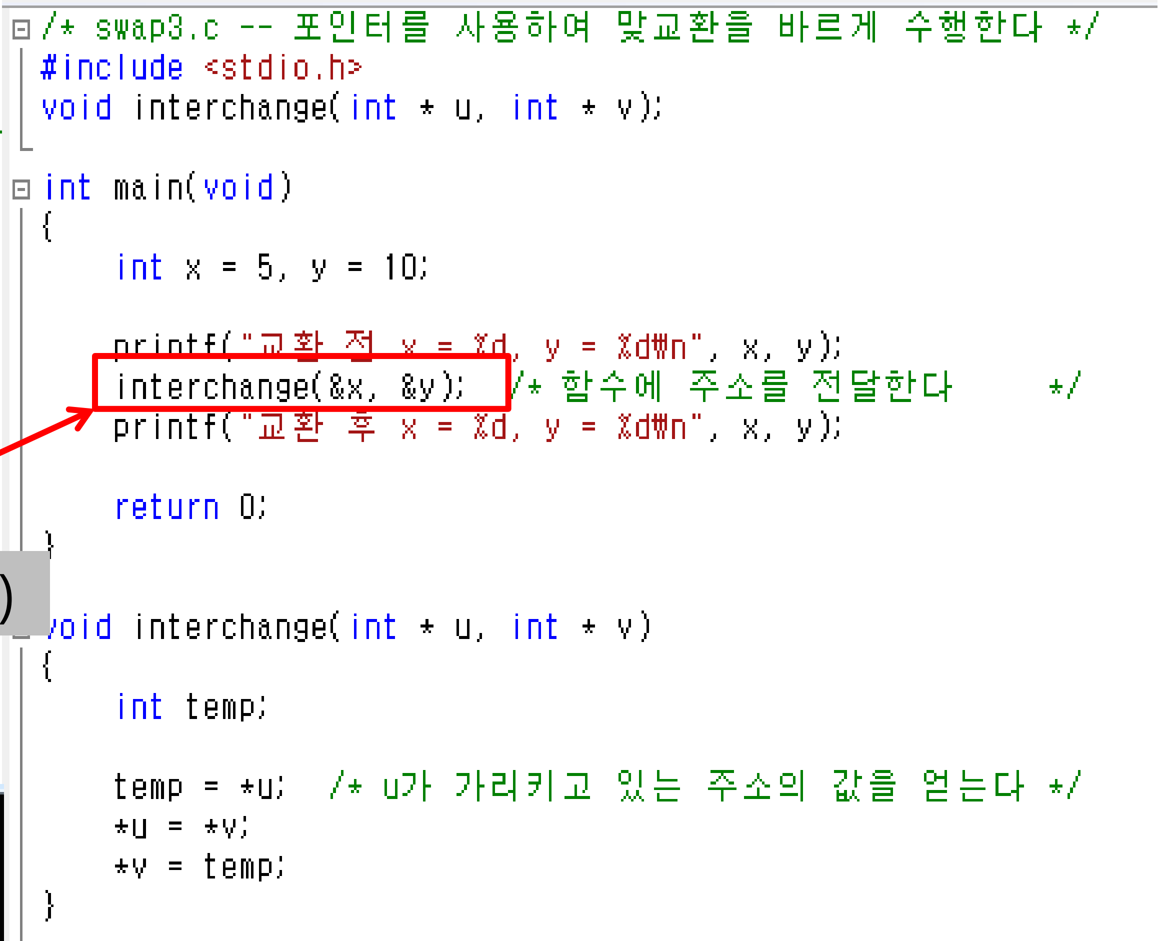


```
/* swap1.c -- 맞바꾸기 함수 제1 버전  */
#include <stdio.h>
void interchange(int u, int v);  /* 함수 선언

int main(void)
{
    int x = 5, y = 10;

    printf("교환 전 x = %d, y = %d₩n", x, y);
    interchange(x, y);
    printf("교환 후 x = %d, y = %d₩n", x, y);

    return 0;
}

void interchange(int u, int v)  /* 함
{
    int temp;

    temp = u;
    u = v;
    v = temp;
}
```



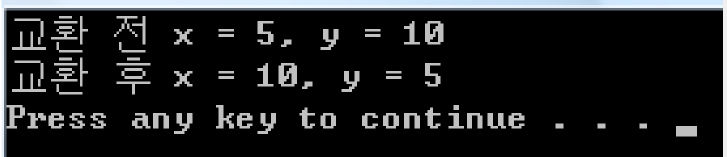
```
교환 전 x = 5, y = 10
교환 후 x = 5, y = 10
Press any key to continue . . .
```

Call by value (값에 의한 호출)



```
/* swap3.c -- 포인터를 사용하여 맞교환을 바르게 수행한다 */
#include <stdio.h>
void interchange(int * u, int * v);

int main(void)
{
    int x = 5, y = 10;

    printf("교환 전 x = %d, y = %d₩n", x, y);
    interchange(&x, &y);  /* 함수에 주소를 전달한다     */
    printf("교환 후 x = %d, y = %d₩n", x, y);

    return 0;
}

void interchange(int * u, int * v)
{
    int temp;

    temp = *u;  /* u가 가리키고 있는 주소의 값을 얻는다 */
    *u = *v;
    *v = temp;
}
```

Call by address (주소에 의한 호출)



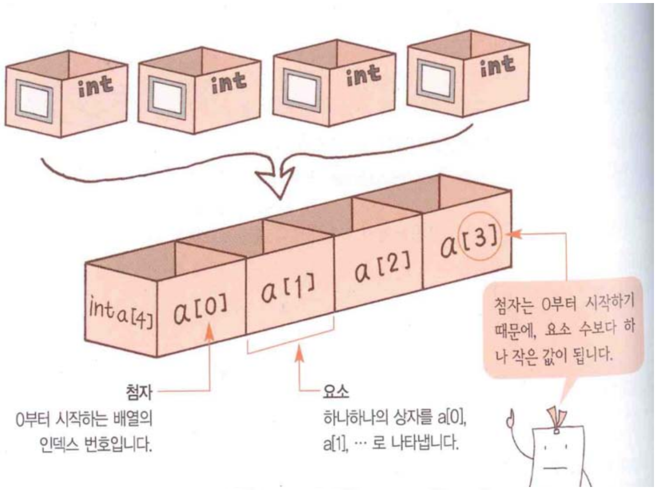
```
교환 전 x = 5, y = 10
교환 후 x = 10, y = 5
Press any key to continue . . .
```

Today

Chapter 10. Arrays and Pointers (배열과 포인터)

SEOUL NATIONAL UNIVERSITY

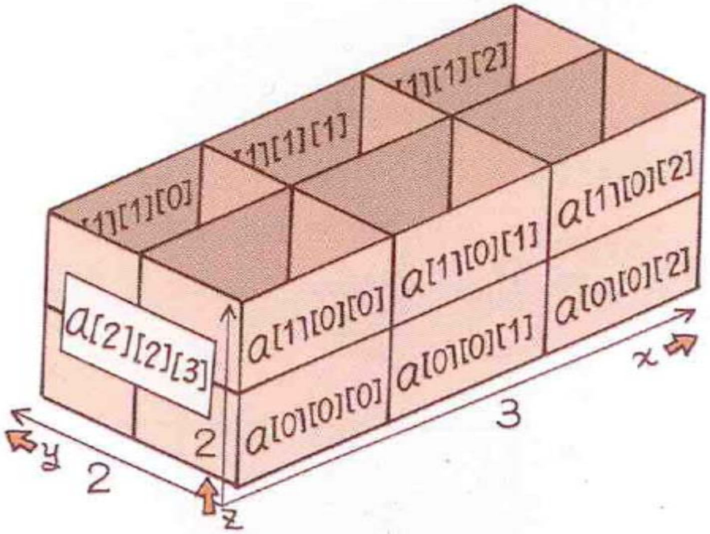
- Arrays – initialization, assignment,
- Multidimensional arrays
- Pointers and arrays
- Functions, Arrays, and Pointers
- Pointer operations



첨자는 0부터 시작하기 때문에, 요소 수보다 하나 작은 값이 됩니다.

첨자
0부터 시작하는 배열의 인덱스 번호입니다.

요소
하나하나의 상자를 a[0], a[1], … 로 나타냅니다.



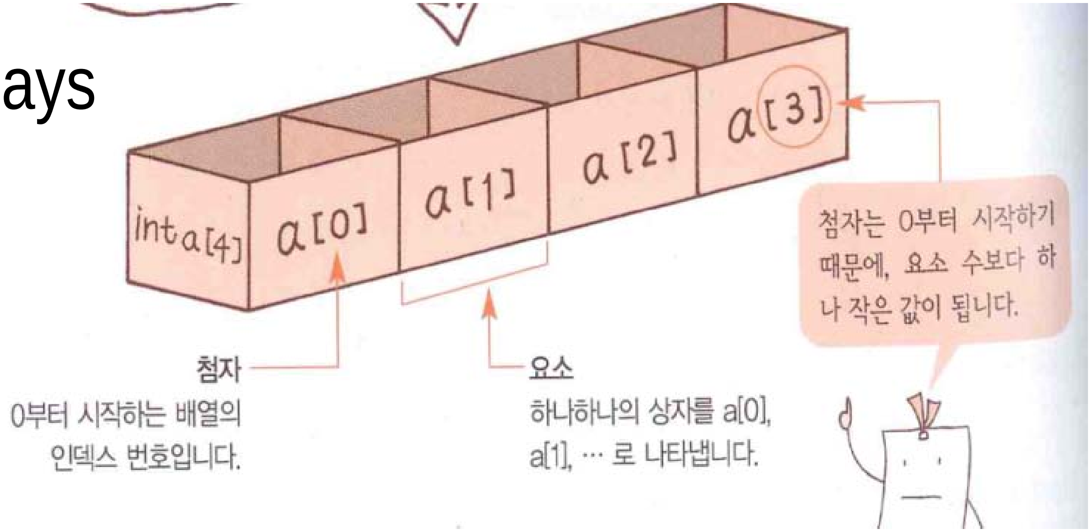
Array (배열)
What is it?

- Array (배열): a series of elements of one data type
  동일한 하나의 데이터형을 가진 연속된 원소
- Array declaration (배열선언): tells the compiler how many elemnts the array contains and what the type is.
  그 배열이 몇 개의 원소를 가지고 있으며, 원소들의 데이터형이 무엇인지 컴파일러에게 알려준다.
- Array elements can have the same types as ordinary variables.

Array (배열)
Declaration

$$\text{int a}\underset{\text{index}}{\underline{[4]}}\text{;}$$

- [ ] identify candy as arrays
- 1st element of a: a[0]
- 2nd element of a: a[1]
- 3rd element of a: a[2]
- 4th element of a: a[3]
- ~~a[4]~~


int a[4]
a[0]
a[1]
a[2]
a[3]
첨자는 0부터 시작하기 때문에, 요소 수보다 하나 작은 값이 됩니다.
첨자
0부터 시작하는 배열의 인덱스 번호입니다.
요소
하나하나의 상자를 a[0], a[1], … 로 나타냅니다.

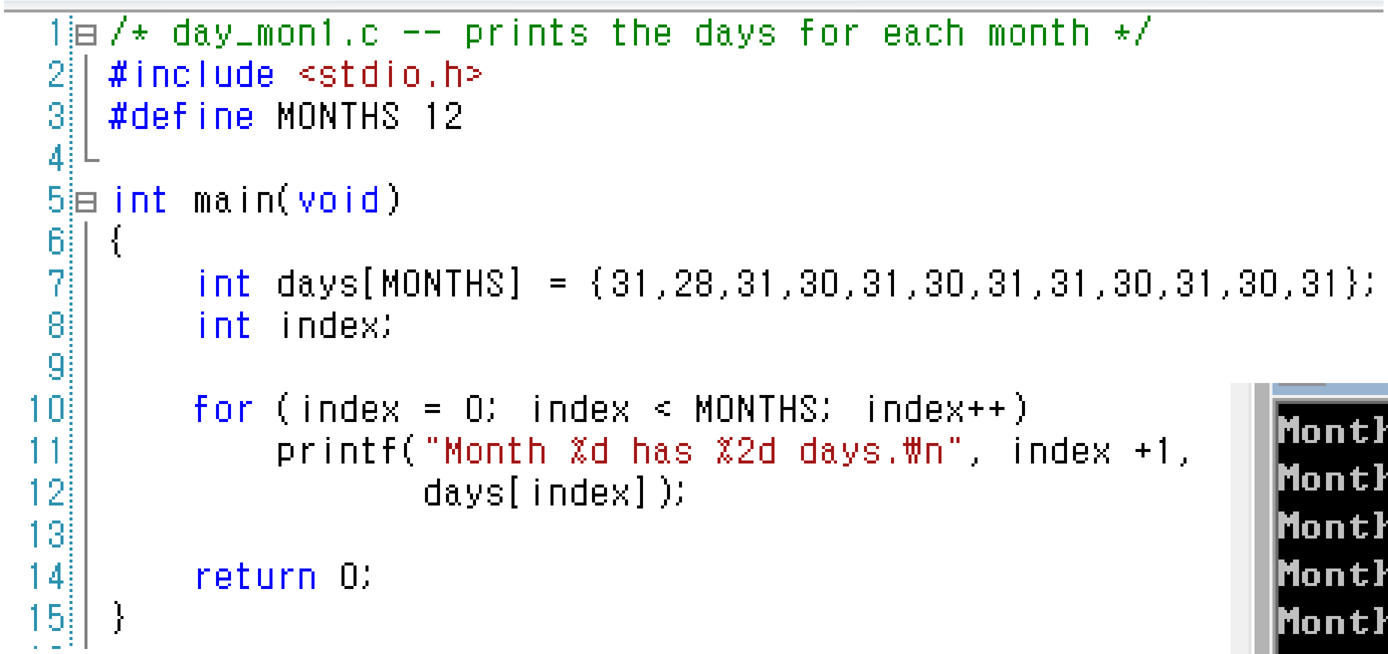
Array (배열)
initialization (초기화)

- Use { } and , to initialize.

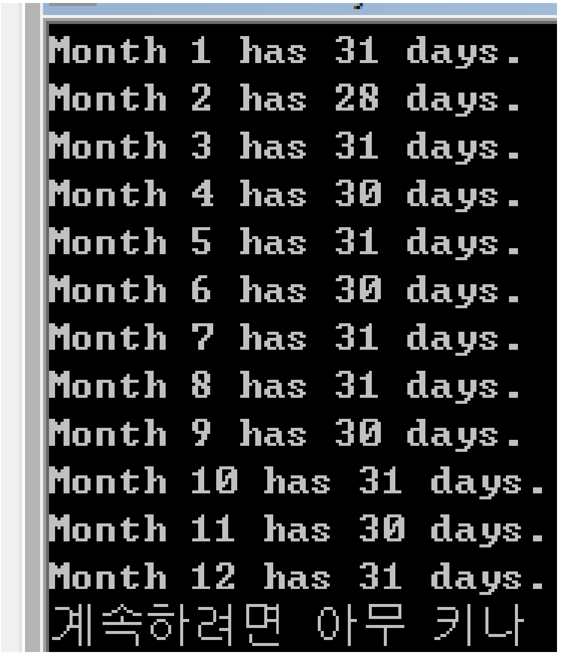
```
int main(void)
{
Int a[4] = {1, 5, 8, 11};
…
}
```

- If you don’t initialize them, they might have any value.

Array (배열) initialization (초기화)



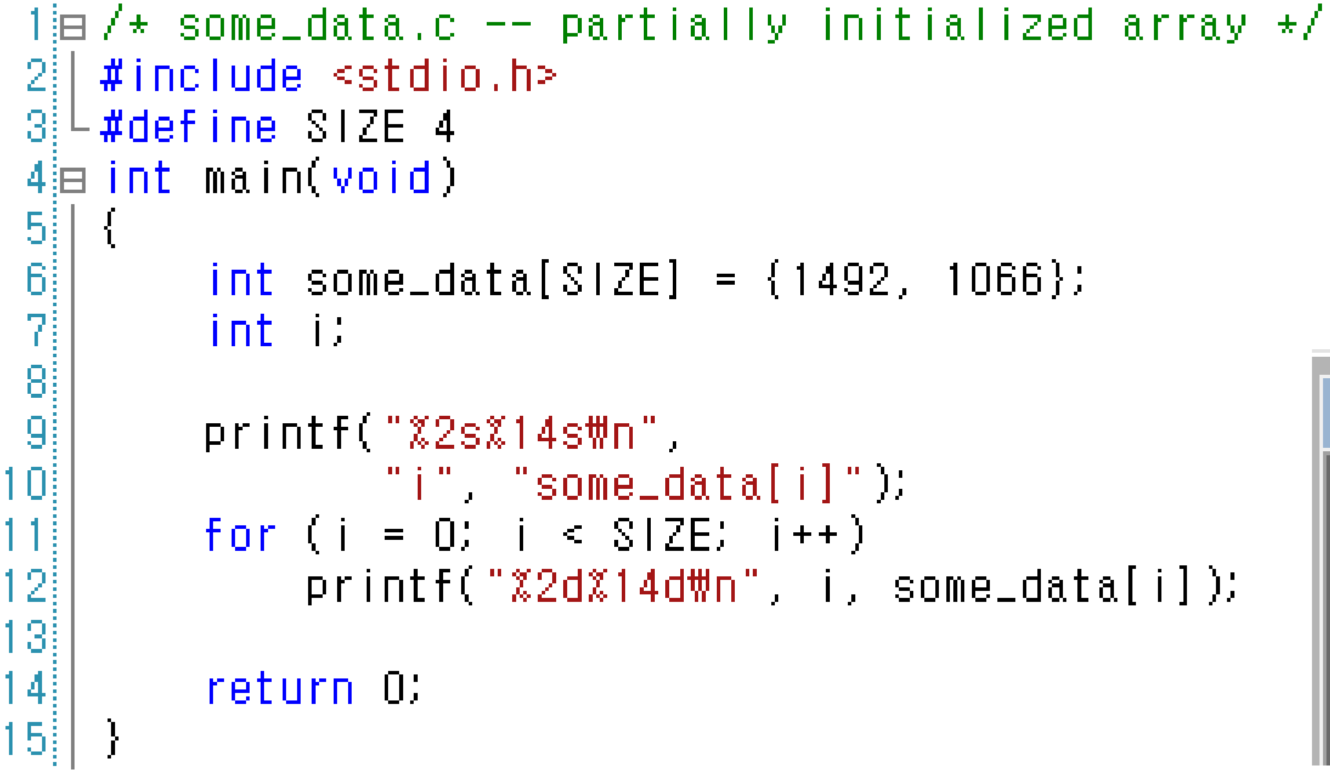
```c
/* day_mon1.c -- prints the days for each month */
#include <stdio.h>
#define MONTHS 12

int main(void)
{
    int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
    int index;

    for (index = 0; index < MONTHS; index++)
        printf("Month %d has %2d days.₩n", index +1,
               days[index]);

    return 0;
}
```



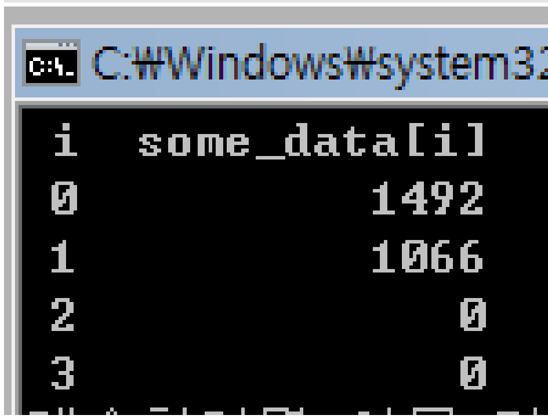
```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
계속하려면 아무 키나
```

Array (배열) initialization (초기화)

- Partial initialization → compiler initialize the remaining elements to 0.

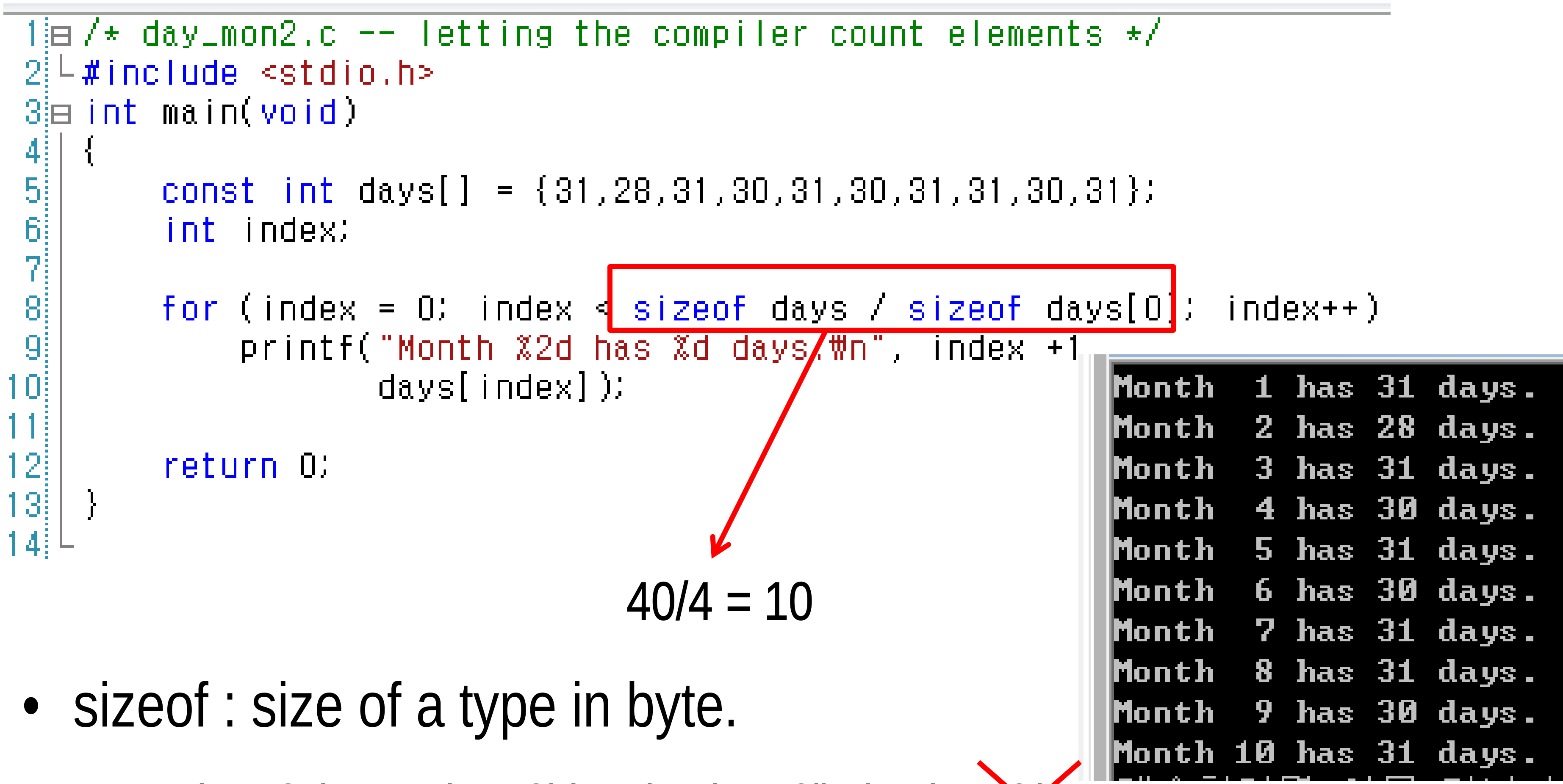


```c
/* some_data.c -- partially initialized array */
#include <stdio.h>
#define SIZE 4
int main(void)
{
    int some_data[SIZE] = {1492, 1066};
    int i;

    printf("%2s%14s₩n",
           "i", "some_data[i]");
    for (i = 0; i < SIZE; i++)
        printf("%2d%14d₩n", i, some_data[i]);

    return 0;
}
```



```
 i  some_data[i]
 0          1492
 1          1066
 2             0
 3             0
```

Array (배열) initialization (초기화)



```
/* day_mon2.c -- letting the compiler count elements */
#include <stdio.h>
int main(void)
{
    const int days[] = {31,28,31,30,31,30,31,31,30,31};
    int index;

    for (index = 0; index < sizeof days / sizeof days[0]; index++)
        printf("Month %2d has %d days.₩n", index +1
               days[index]);

    return 0;
}
```

```
Month  1 has 31 days.
Month  2 has 28 days.
Month  3 has 31 days.
Month  4 has 30 days.
Month  5 has 31 days.
Month  6 has 30 days.
Month  7 has 31 days.
Month  8 has 31 days.
Month  9 has 30 days.
Month 10 has 31 days.
```

40/4 = 10

- sizeof : size of a type in byte.
  - sizeof days, sizeof(days), sizeof(int), ~~sizeof int~~

Array (배열)
initialization (초기화)

- designated initializer (지정초기화자)
  - To pick and choose which elements are initialized

```
int arr[6] = {0,0,12,0};
int arr[6] = { [2] = 12};
```

identical

  - After you initialize at least one element, the uninitialized elements are set to 0

Array (배열)
assigning (배열에 값 대입하기)

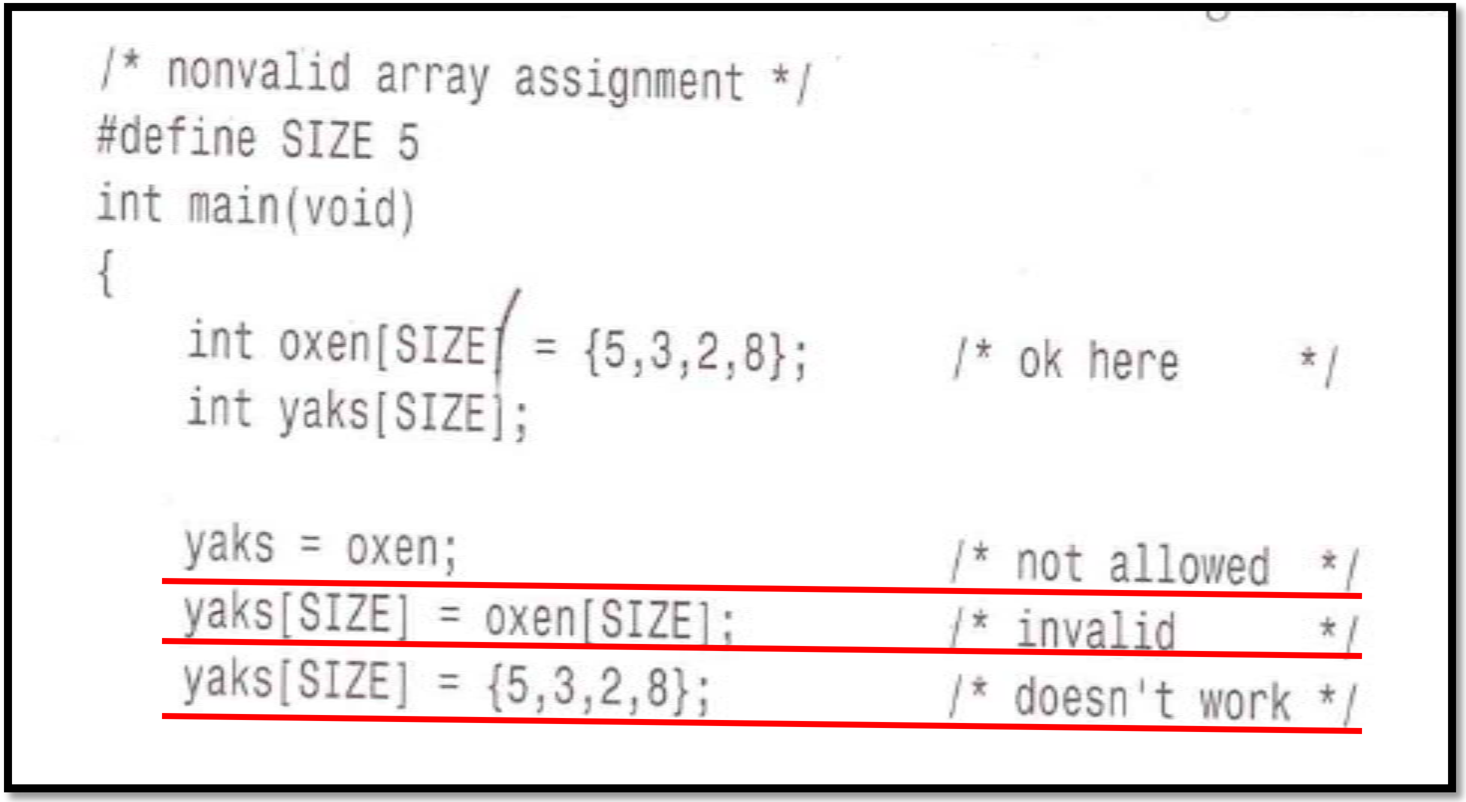
- After an array has been declared, you can assign values to array members by using an array index.
  - Ex) evens[0] = 2; evens[1] = 8;

```
/*배열에 값을 대입*/
#include <stdio.h>
#define SIZE 50
Int main(void)
{
    int counter, evens[SIZE]
    for (counter = 0; counter < SIZE; counter++)
                    evens[counter] = 2 * counter;
    ...
}
```

Array (배열)

assigning (배열에 값 대입하기)

- 하나의 배열을 다른 배열로 통째로 → No
- { } 를 이용해서? → No

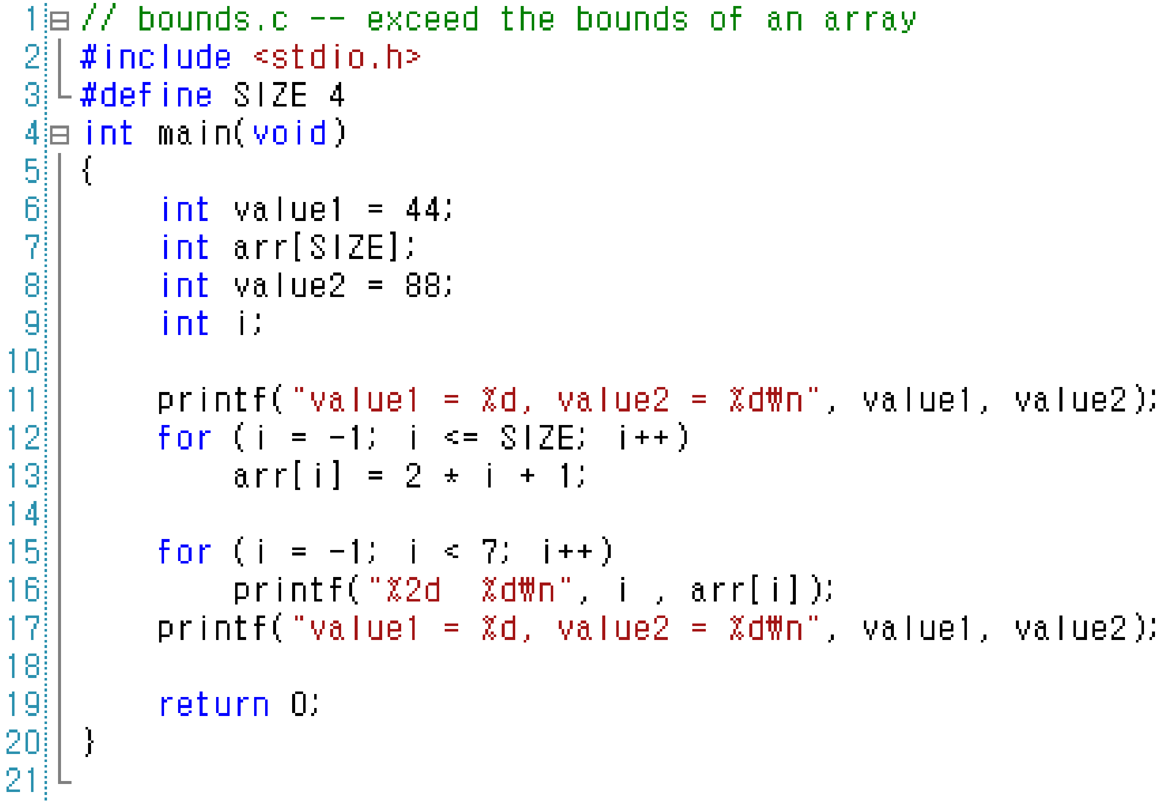


```
/* nonvalid array assignment */
#define SIZE 5
int main(void)
{
    int oxen[SIZE] = {5,3,2,8};          /* ok here      */
    int yaks[SIZE];

    yaks = oxen;                         /* not allowed  */
    yaks[SIZE] = oxen[SIZE];             /* invalid      */
    yaks[SIZE] = {5,3,2,8};              /* doesn't work */
```

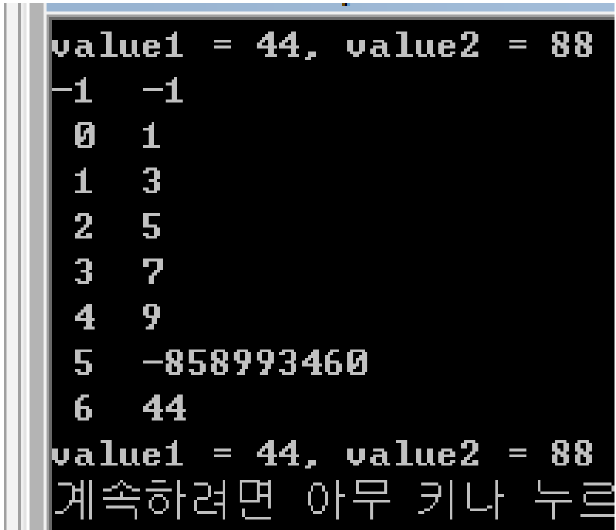
Array (배열)
array bounds (배열의 범위)

SEOUL NATIONAL UNIVERSITY

- The compiler doesn't check if the indices are valid.
  - When invalid indices were used,
  - Computer work oddly, it might abort (먹통), alter the value of other variables



```c
// bounds.c -- exceed the bounds of an array
#include <stdio.h>
#define SIZE 4
int main(void)
{
    int value1 = 44;
    int arr[SIZE];
    int value2 = 88;
    int i;

    printf("value1 = %d, value2 = %d₩n", value1, value2);
    for (i = -1; i <= SIZE; i++)
        arr[i] = 2 * i + 1;

    for (i = -1; i < 7; i++)
        printf("%2d  %d₩n", i , arr[i]);
    printf("value1 = %d, value2 = %d₩n", value1, value2);

    return 0;
}
```



```
value1 = 44, value2 = 88
-1  -1
 0  1
 1  3
 2  5
 3  7
 4  9
 5  -858993460
 6  44
value1 = 44, value2 = 88
계속하려면 아무 키나 누르
```

Array (배열)
array bounds (배열의 범위)

- C doesn’t check if the indices are valid.
  - Allows a C program to run faster
  - C trusts the programmer to do the coding correctly → rewards programmer with a faster program

- Remember that index start from 0

Array (배열)
Variable Length Array (VLA)

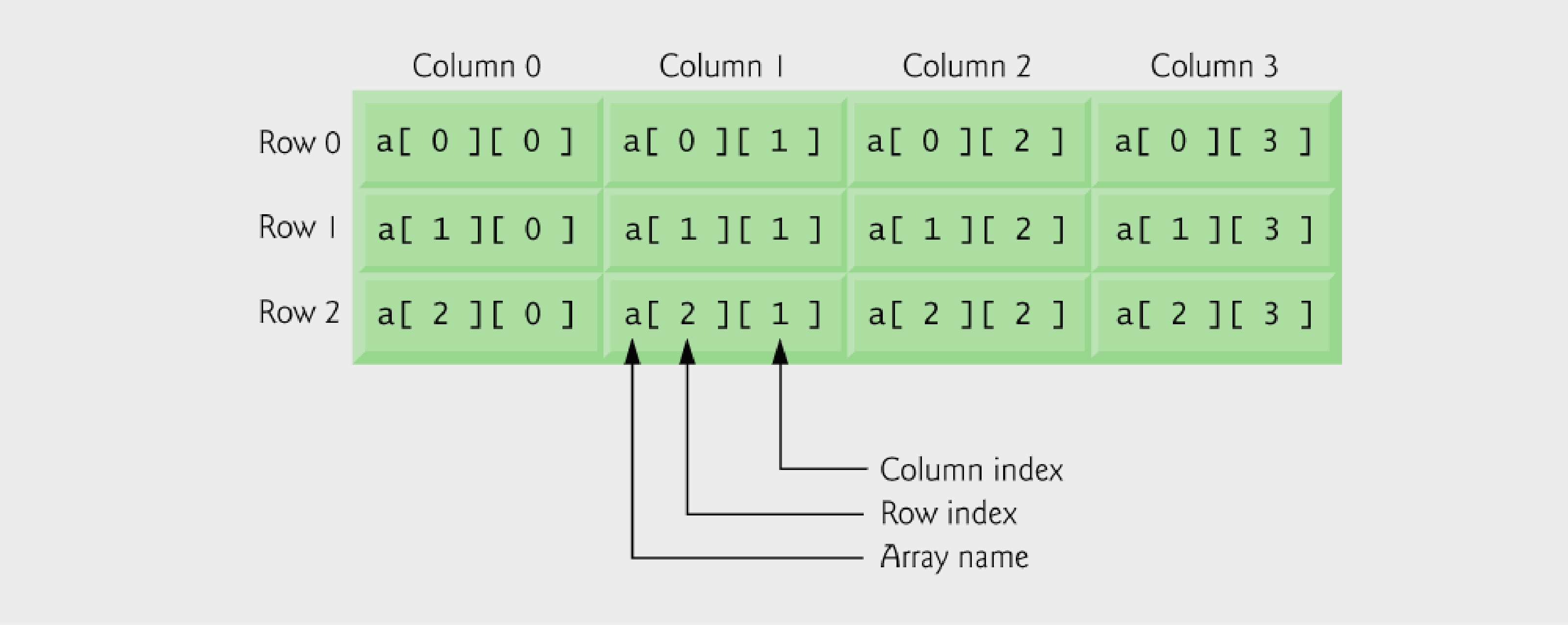
- New in C99
- Visual C++ doesn’t support this.
- You may use ‘gcc’ for this function.

Array (배열)
multidimensional arrays

- Using array of arrays can be useful.
  - Ex) five years of monthly rainfall data
  - float rain[5][12]   is much better than float rain[60]

rain[0][0]	rain[0][1]	rain[0][2]	rain[0][3]	rain[0][4]	…	rain[0][10]	rain[0][11]	rain[0]
Rain[1[0]	rain[1][1]	rain[1][2]	rain[1][3]	rain[1][4]	…	rain[1][10]	rain[1][11]	rain[1]
rain[2][0]	rain[2][1]	rain[2][2]	rain[2][3]	rain[2][4]	…	rain[2][10]	rain[2][11]	rain[2]
rain[3][0]	rain[3][1]	rain[3][2]	rain[3][3]	rain[3][4]	…	rain[3][10]	rain[3][11]	rain[3]
rain[4][0]	rain[4][1]	rain[4][2]	rain[4][3]	rain[4][4]	…	rain[4][10]	rain[4][11]	rain[4]

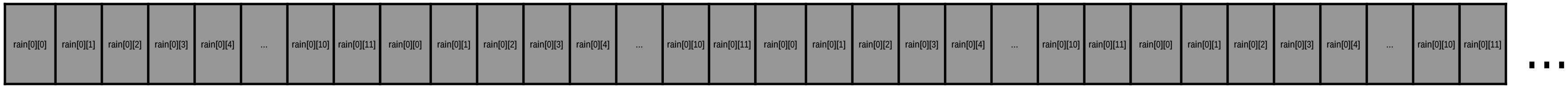
Array (배열)
multidimensional arrays



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Array (배열)
multidimensional arrays

- 2D view is merely convenient way to visualize. 2D array is actually stored sequentially,



rain[0][0]	rain[0][1]	rain[0][2]	rain[0][3]	rain[0][4]	...	rain[0][10]	rain[0][11]	rain[0][0]	rain[0][1]	rain[0][2]	rain[0][3]	rain[0][4]	...	rain[0][10]	rain[0][11]	rain[0][0]	rain[0][1]	rain[0][2]	rain[0][3]	rain[0][4]	...	rain[0][10]	rain[0][11]	rain[0][0]	rain[0][1]	rain[0][2]	rain[0][3]	rain[0][4]	...	rain[0][10]	rain[0][11]	...

Array (배열)
multidimensional arrays - initialization

- Similar to 1D, but use { two times.

int mat[2][3] = {{1,2,3},{4,5,6}}; Or int mat[2][3] = {1,2,3,4,5,6}

mat[0][0] = 1 mat[0][1] = 2 mat[0][2] = 3

mat[1][0] = 4 mat[1][1] = 5 mat[1][2] = 6

1	2	3
4	5	6

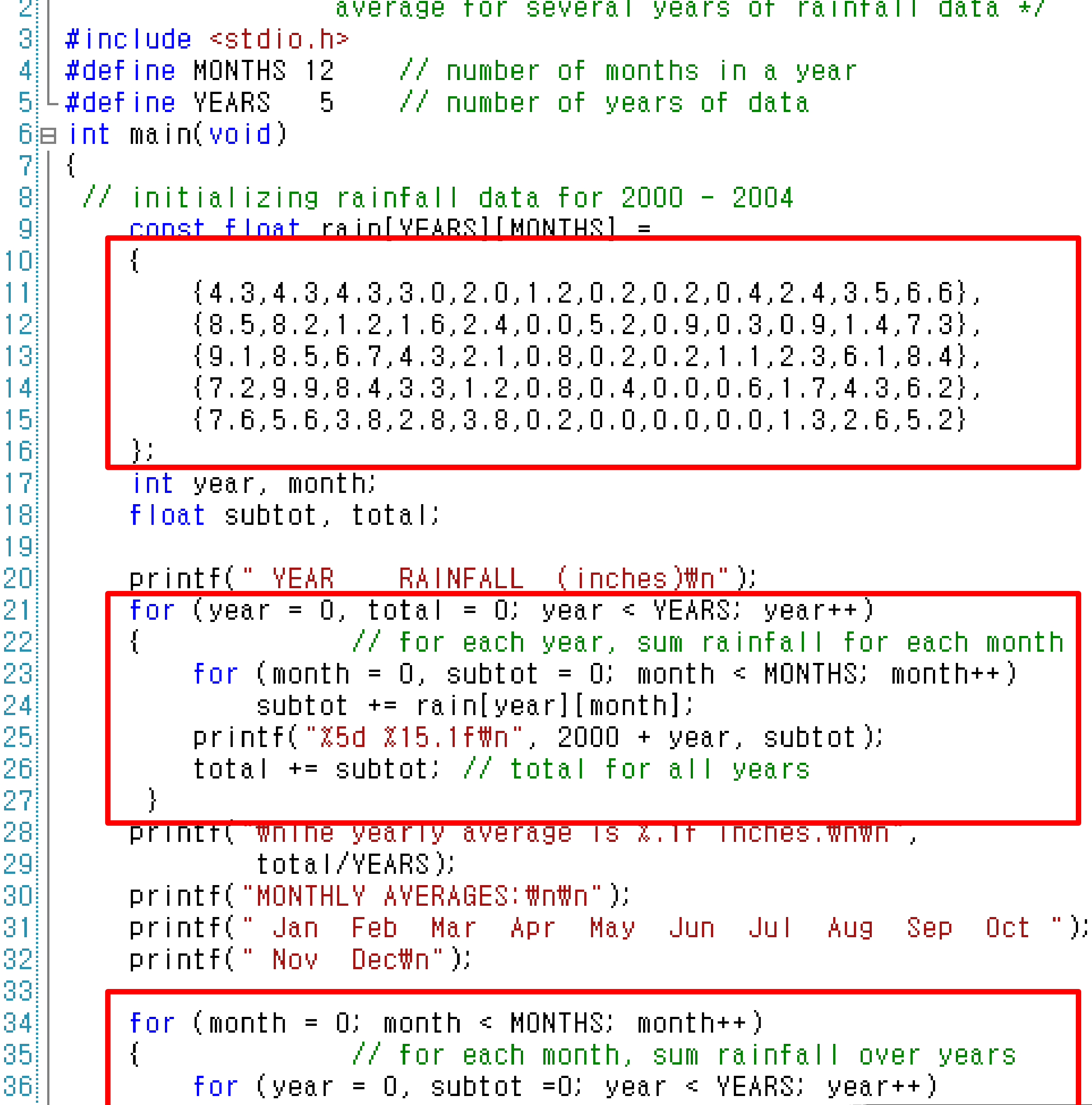
5	6	0
7	8	0

int mat[2][3] = {{5,6},{7,8}}

5	6	7
8	0	0

int mat[2][3] = {5,6,7,8}

SEOUL NATIONAL UNIVERSITY

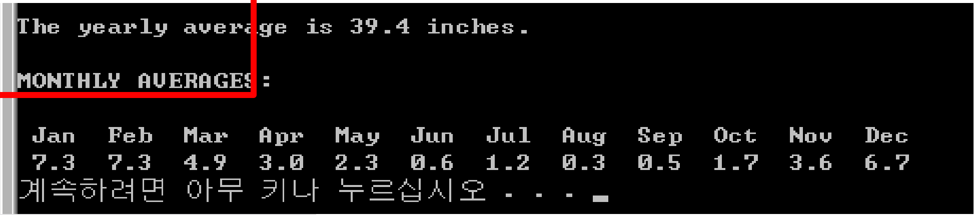


```
                    average for several years of rainfall data */
#include <stdio.h>
#define MONTHS 12     // number of months in a year
#define YEARS   5     // number of years of data
int main(void)
{
 // initializing rainfall data for 2000 - 2004
  const float rain[YEARS][MONTHS] =
  {
    {4.3,4.3,4.3,3.0,2.0,1.2,0.2,0.2,0.4,2.4,3.5,6.6},
    {8.5,8.2,1.2,1.6,2.4,0.0,5.2,0.9,0.3,0.9,1.4,7.3},
    {9.1,8.5,6.7,4.3,2.1,0.8,0.2,0.2,1.1,2.3,6.1,8.4},
    {7.2,9.9,8.4,3.3,1.2,0.8,0.4,0.0,0.6,1.7,4.3,6.2},
    {7.6,5.6,3.8,2.8,3.8,0.2,0.0,0.0,0.0,1.3,2.6,5.2}
  };
  int year, month;
  float subtot, total;

  printf(" YEAR    RAINFALL  (inches)₩n");
  for (year = 0, total = 0; year < YEARS; year++)
  {             // for each year, sum rainfall for each month
    for (month = 0, subtot = 0; month < MONTHS; month++)
      subtot += rain[year][month];
    printf("%5d %15.1f₩n", 2000 + year, subtot);
    total += subtot; // total for all years
   }
  printf("₩nThe yearly average is %.1f inches.₩n₩n",
      total/YEARS);
  printf("MONTHLY AVERAGES:₩n₩n");
  printf(" Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct ");
  printf(" Nov  Dec₩n");

  for (month = 0; month < MONTHS; month++)
  {             // for each month, sum rainfall over years
    for (year = 0, subtot =0; year < YEARS; year++)
      subtot += rain[year][month];
    printf("%4.1f ", subtot/YEARS);
  }
  printf("₩n");

  return 0;
```

initialization

Nested loop

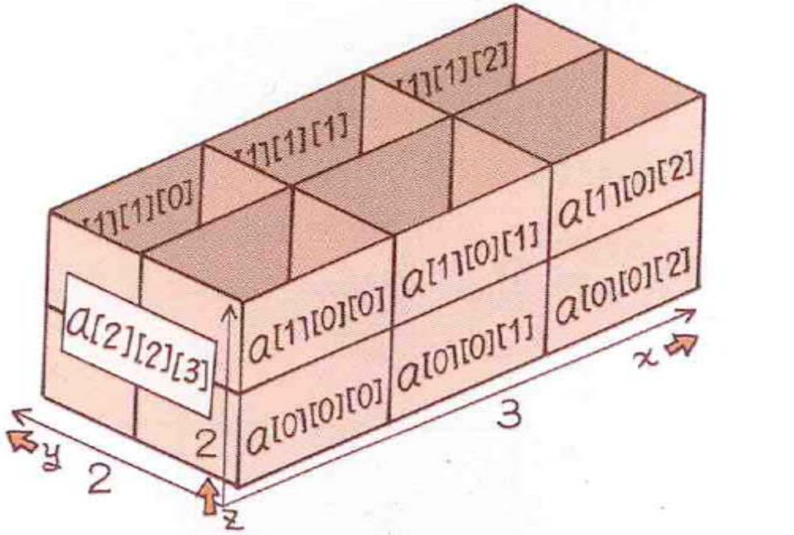
Print out monthly average



```
The yearly average is 39.4 inches.

MONTHLY AVERAGES:

 Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
 7.3  7.3  4.9  3.0  2.3  0.6  1.2  0.3  0.5  1.7  3.6  6.7
계속하려면 아무 키나 누르십시오 . . .
```

Array (배열)
multidimensional arrays – 3rd or higher

- Can be also used
- Int box[10][20][30]



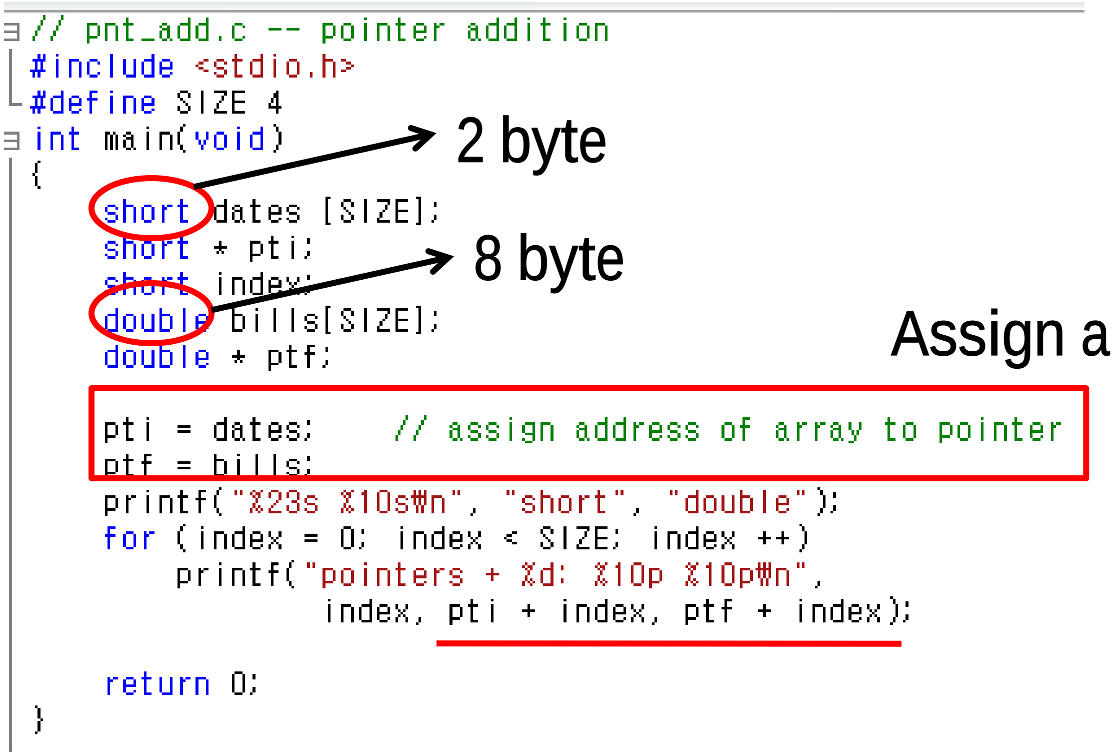
Pointers

Pointers and Arrays (포인터와 배열)

- Array is simply a disguised use of pointers. (배열표기는 실제로는 포인터의 변장된 사용에 불과하다)
- 배열명은 곧 그 배열명의 시작주소이다.
- Pointers can do array subscripting operations.

- For an array flizny, the following is true.
  - flizny == &flizny[0]
  - Both flizny and &flizny[0] represent the memory address of first element. Both are constants because they remain fixed.

Pointers
pointers and arrays

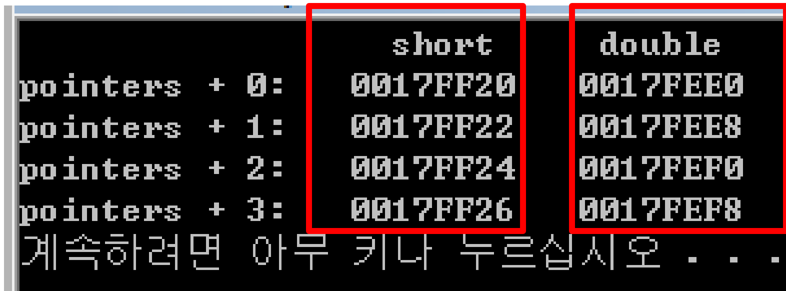


```
// pnt_add.c -- pointer addition
#include <stdio.h>
#define SIZE 4
int main(void)
{
    short dates [SIZE];
    short * pti;
    short index;
    double bills[SIZE];
    double * ptf;

    pti = dates;     // assign address of array to pointer
    ptf = bills;
    printf("%23s %10s₩n", "short", "double");
    for (index = 0; index < SIZE; index ++)
        printf("pointers + %d: %10p %10p₩n",
                index, pti + index, ptf + index);

    return 0;
}
```

2 byte

8 byte

Assign address of array to pointer



```
                          short      double
pointers + 0:          0017FF20   0017FEE0
pointers + 1:          0017FF22   0017FEE8
pointers + 2:          0017FF24   0017FEF0
pointers + 3:          0017FF26   0017FEF8
계속하려면 아무 키나 누르십시오 . . .
```

Pointers

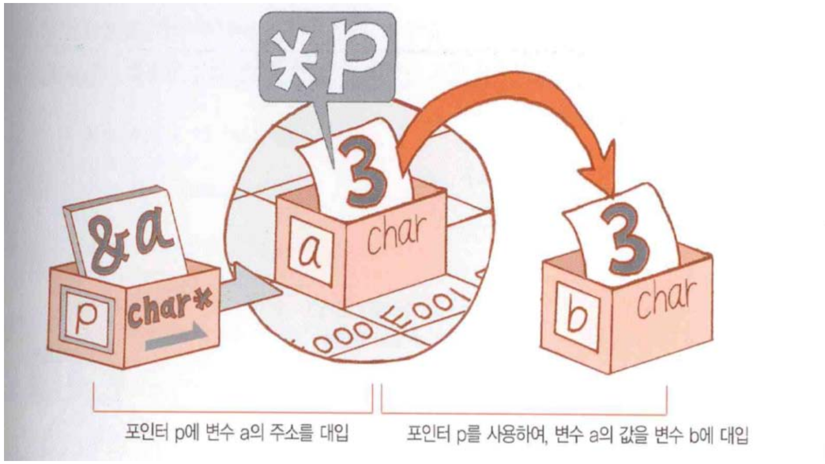
pointers and arrays

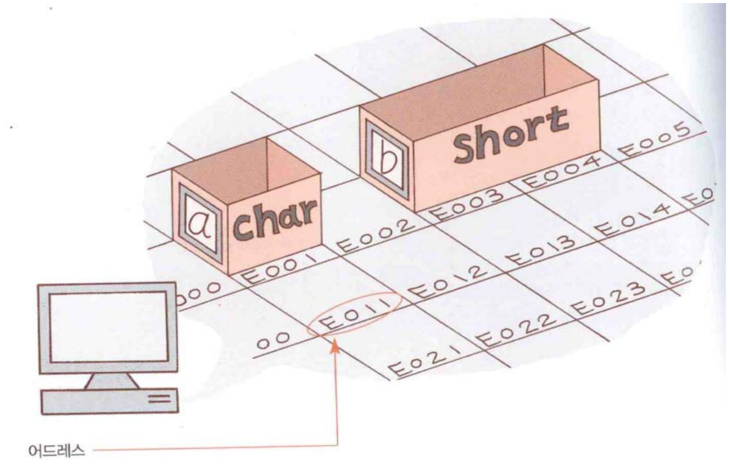
- 우리가 사용하는 시스템은 바이트 (byte)단위로 주소가 매겨진다.
- 포인터에 1을 더하면 C는 하나의 기억단위를 더한다 (short – 2 byte,int – 4 byte, double – 8 byte).
- 즉, 주소가 다음 바이트가 아니라 다음 원소(element)의 주소로 증가한다 – 포인터가 가리키는 객체의 종류를 선언해야 하는 이유.
- 데이터 객체(data object)는 값을 저장하는 데 사용할 수 있는 데이터 저장 영역을 일반적으로 지칭하는 용어

Pointers
pointers and arrays

SEOUL NATIONAL UNIVERSITY

- 포인터의 값은 그것이 가리키는 객체의 주소이다.
- 포인터에 * 연산자를 적용하면 그 포인터가 가리키는 객체에 저장되어 있는 값을 얻는다.
- 포인터에 1을 더하면 그 포인터가 가리키는 객체의 바이트 수 크기만큼 포인터 값이 증가한다.


*p
&a
p
char*
a
char
3
3
b
char
000
E001
포인터 p에 변수 a의 주소를 대입
포인터 p를 사용하여, 변수 a의 값을 변수 b에 대입



Pointers
pointers and arrays

- With an array *dates*,

  dates + 2 == &dates[2] /* 주소가 같다.*/

  (dates + 2) == dates[2] / 값이 같다.*/

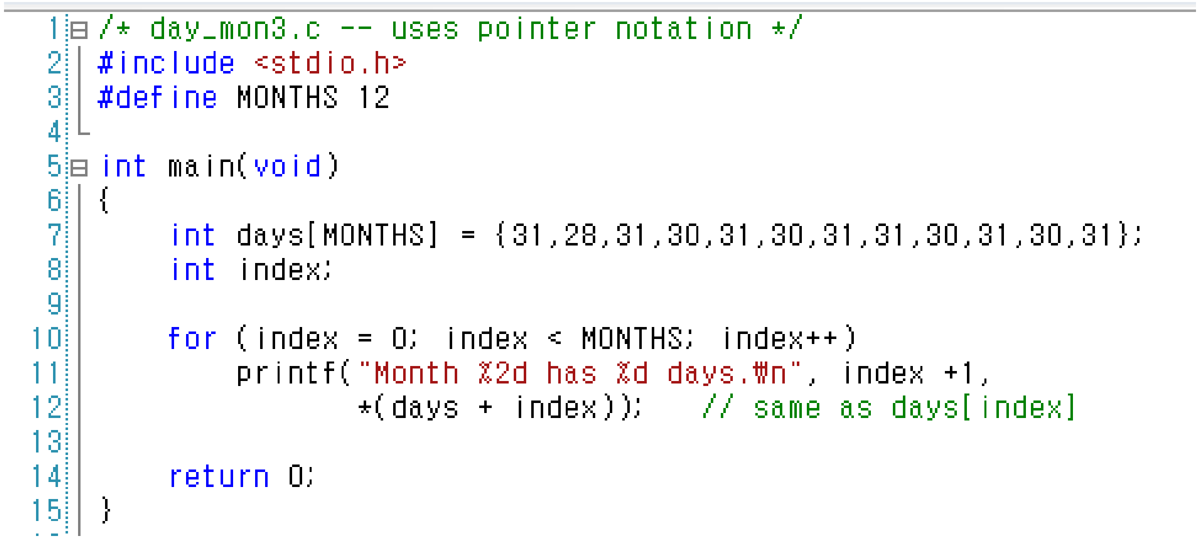
- Close connection between arrays and pointers!!!
- Use a pointer to identify an individual element of an array and to obtain its value.
- Two different notations for the same thing.
- 실제로 C언어 표준은 배열표기를 포인터로 서술한다. ar[n] → *(ar+n)

Pointers

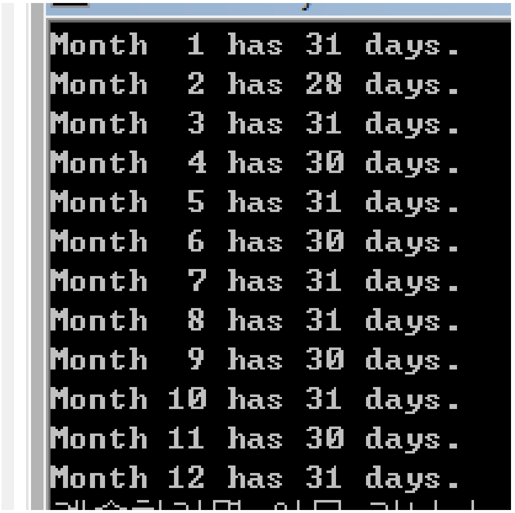
pointers and arrays

```
*(dates + 2)   /* dates의 세번째 원소의 값 */
*dates + 2     /* dates의 첫 번째 원소의 값에 2를 더한다. */
```

Pointers
pointers and arrays



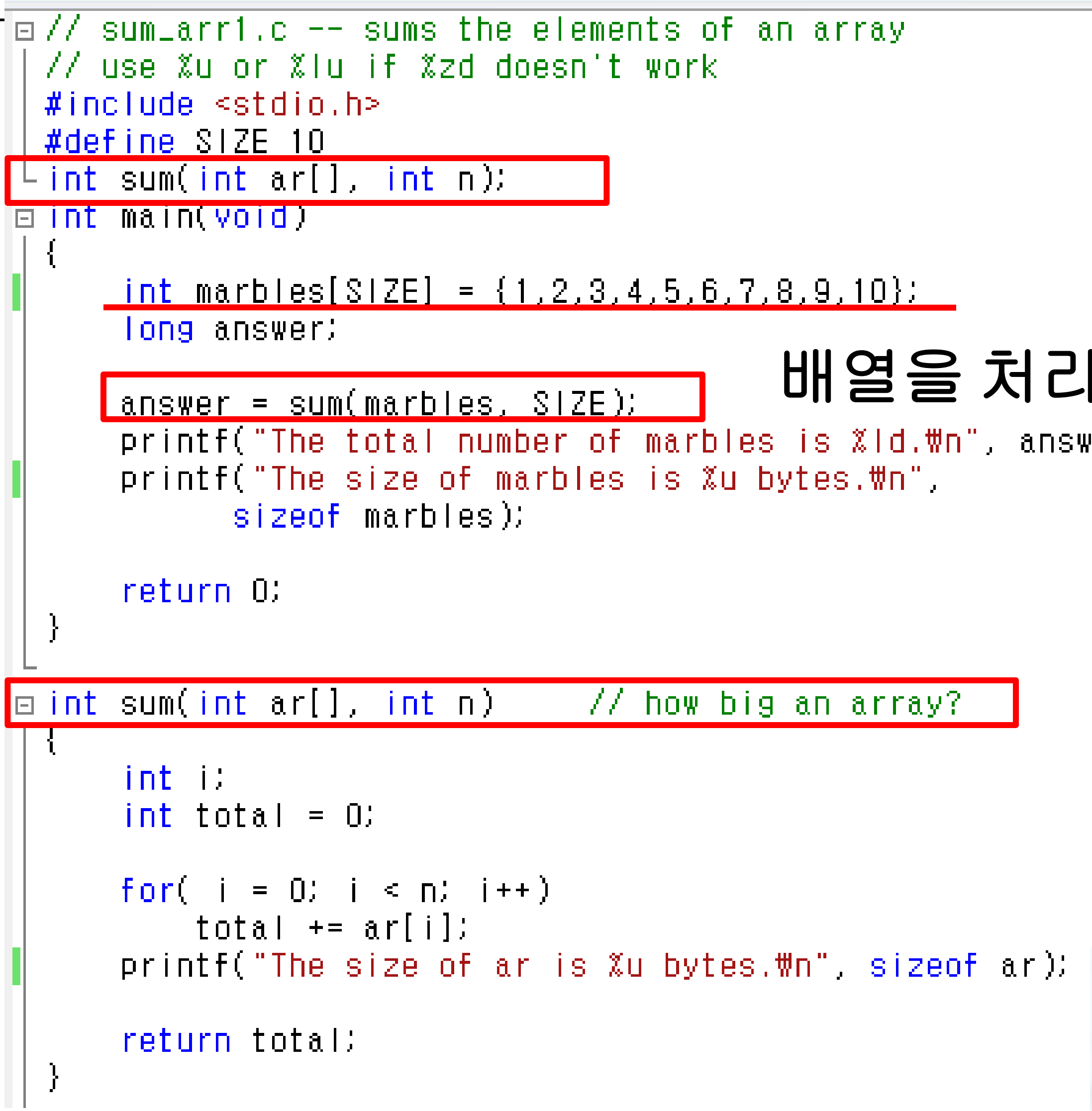
```c
/* day_mon3.c -- uses pointer notation */
#include <stdio.h>
#define MONTHS 12

int main(void)
{
    int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
    int index;

    for (index = 0; index < MONTHS; index++)
        printf("Month %2d has %d days.₩n", index +1,
               *(days + index));    // same as days[index]

    return 0;
}
```



```
Month  1 has 31 days.
Month  2 has 28 days.
Month  3 has 31 days.
Month  4 has 30 days.
Month  5 has 31 days.
Month  6 has 30 days.
Month  7 has 31 days.
Month  8 has 31 days.
Month  9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

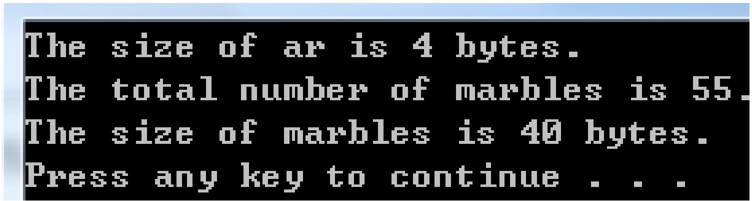
- days : 첫 번째 원소의 시작주소
- days + index : days[index]의 주소
- *(days + index) : 그 원소의 값, days[index]

Pointers
Functions that operates on an array



```
// sum_arr1.c -- sums the elements of an array
// use %u or %lu if %zd doesn't work
#include <stdio.h>
#define SIZE 10
int sum(int ar[], int n);
int main(void)
{
    int marbles[SIZE] = {1,2,3,4,5,6,7,8,9,10};
    long answer;

    answer = sum(marbles, SIZE);
    printf("The total number of marbles is %ld.₩n", answer);
    printf("The size of marbles is %u bytes.₩n",
          sizeof marbles);

    return 0;
}

int sum(int ar[], int n)     // how big an array?
{
    int i;
    int total = 0;

    for( i = 0; i < n; i++)
        total += ar[i];
    printf("The size of ar is %u bytes.₩n", sizeof ar);

    return total;
}
```

배열을 처리하는 함수



```
The size of ar is 4 bytes.
The total number of marbles is 55.
The size of marbles is 40 bytes.
Press any key to continue . . .
```

Pointers
Functions that operates on an array

- Suppose you want a function that returns the sum of the elements of an array, *marbles*

- Calling a function
  - total = sum(marbles); // 가능한 함수 호출의 예
- Prototype (declaration)
  - int sum(int * ar) // 대응하는 함수 프로토타입

배열이름은 첫 번째 원소의 주소이기 때문에 배열 이름을 실전달인자로 사용하려면 대응하는 형식매개변수가 포인터여야 한다.

Pointers
Functions that operates on an array

- Definition

```
int sum(int * ar)
{
    int i;
    int total = 0;
    for (i = 0; i < 10; i++)     // 원소가 10개라고 가정
        total += ar[ i ];        // ar[ i ]는 *(ar + i)와 같다.
    return  total;
}
```

형식매개변수의 선언에 사용될때 다음과 같이 써도된다.
int ar[ ]

Use array notation with a pointer
포인터에 배열표기를 사용

Pointers
Functions that operates on an array

- Definition (with 2nd argument)

```
int sum(int * ar, int n)
{
    int i;
    int total = 0;
    for (i = 0; i < n; i++)     // 원소가 10개라고 가정
        total += ar[ i ];          // ar[ i ]는 *(ar + i)와 같다.
    return  total;
}
```

Number of elements

location of the array & type of it

Pointers
Usinig Pointer Parameters

```
int sum(int * ar, int n);
int sum(int *   , int   );
int sum(int  ar[ ], int n);
int sum(int  [ ], int) ;
```

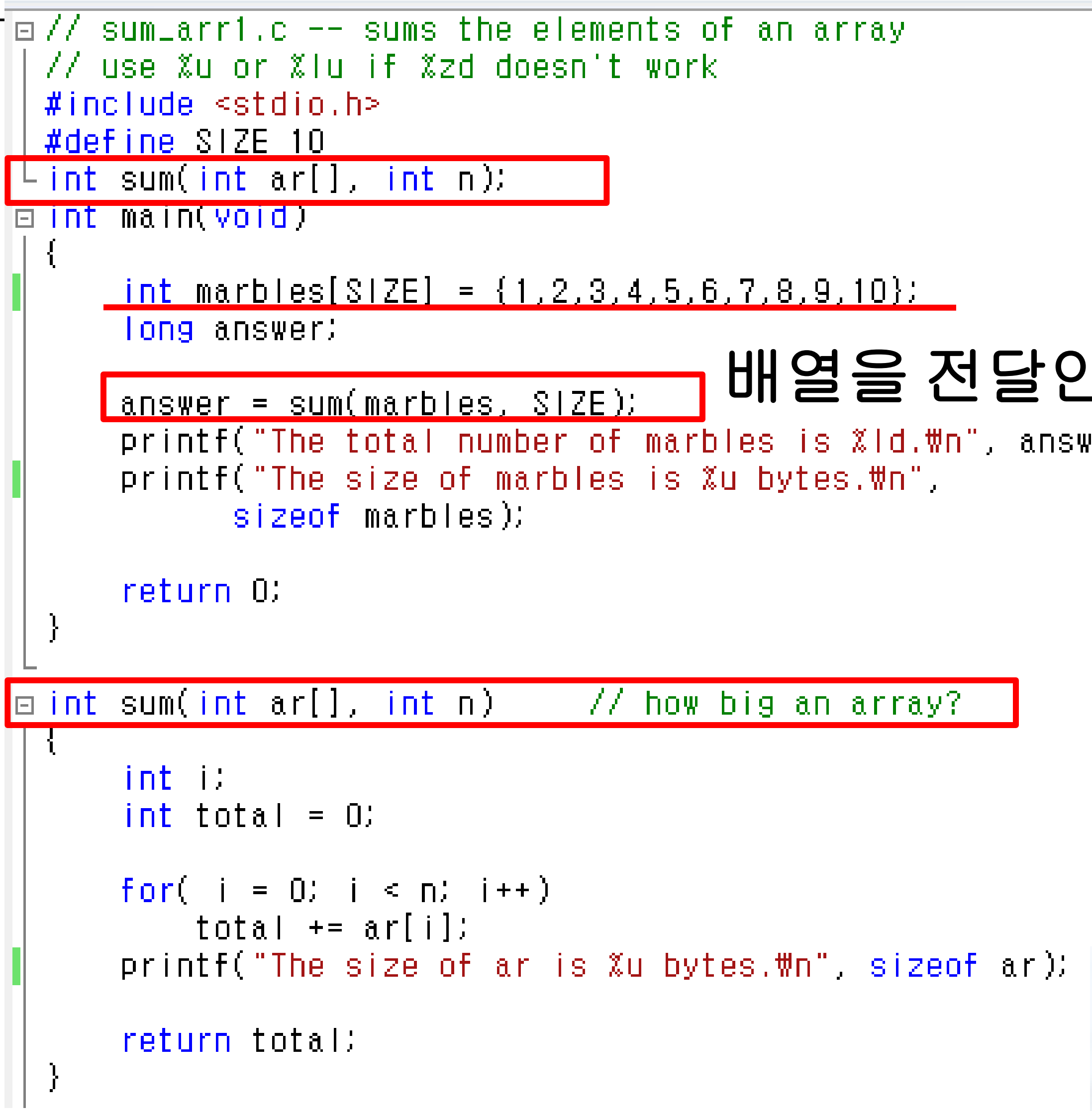
Four prototypes are identical

```
int sum(int * ar, int n)
{ …}

int sum(int ar[ ], int n)
{ …}
```

Two definitions are identical

Pointers
Functions that operates on an array

SEOUL NATIONAL UNIVERSITY

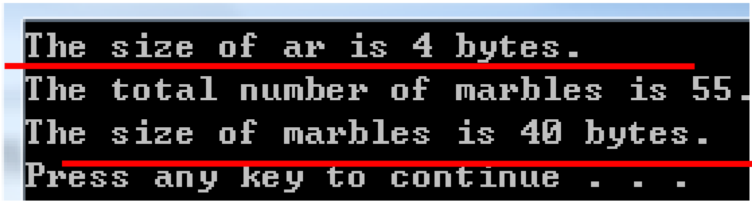


```
// sum_arr1.c -- sums the elements of an array
// use %u or %lu if %zd doesn't work
#include <stdio.h>
#define SIZE 10
int sum(int ar[], int n);
int main(void)
{
    int marbles[SIZE] = {1,2,3,4,5,6,7,8,9,10};
    long answer;

    answer = sum(marbles, SIZE);
    printf("The total number of marbles is %ld.₩n", answer);
    printf("The size of marbles is %u bytes.₩n",
          sizeof marbles);

    return 0;
}

int sum(int ar[], int n)      // how big an array?
{
    int i;
    int total = 0;

    for( i = 0; i < n; i++)
        total += ar[i];
    printf("The size of ar is %u bytes.₩n", sizeof ar);

    return total;
}
```

배열을 전달인자로 사용하는 함수

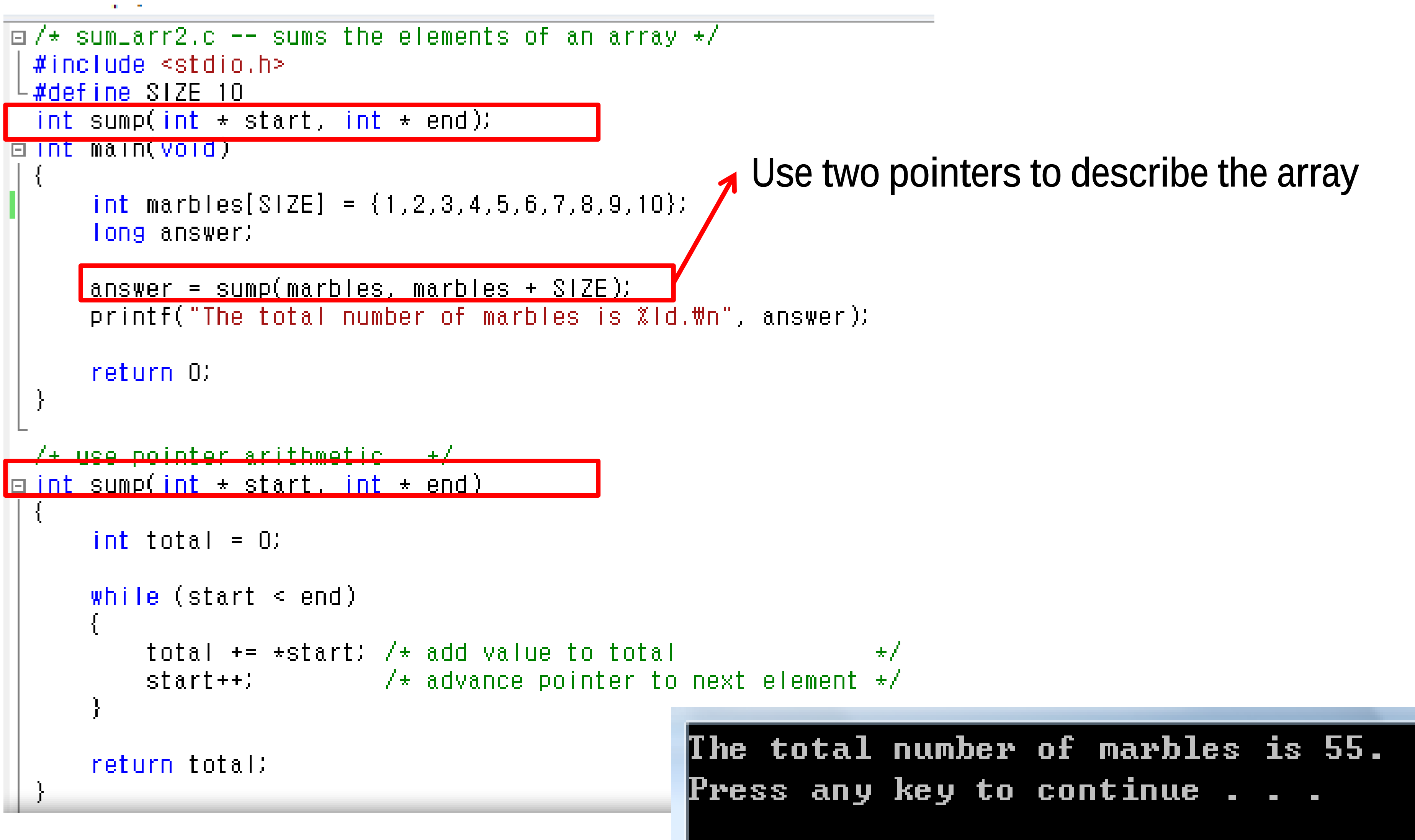
Size of marble is 40 bytes

Size of ar is 4 bytes



```
The size of ar is 4 bytes.
The total number of marbles is 55.
The size of marbles is 40 bytes.
Press any key to continue . . .
```

Pointers
Using Pointer Parameters

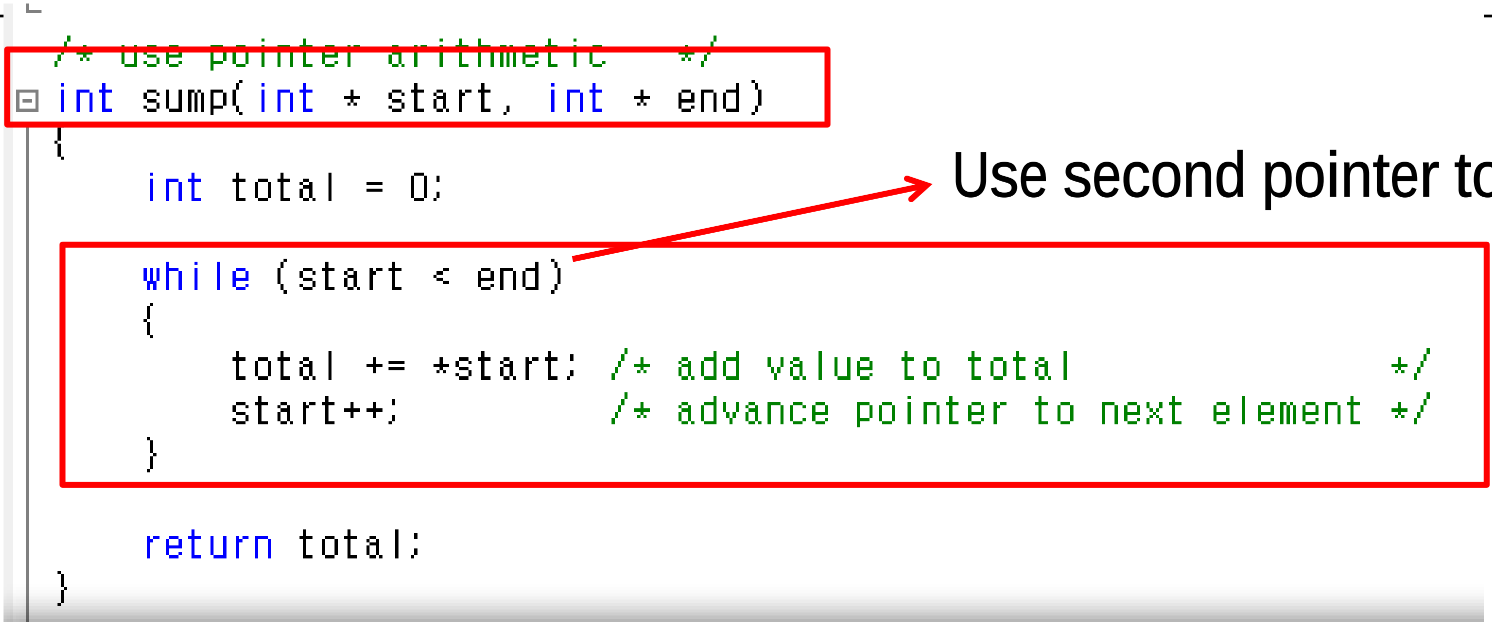


```
/* sum_arr2.c -- sums the elements of an array */
#include <stdio.h>
#define SIZE 10
int sump(int * start, int * end);
int main(void)
{
    int marbles[SIZE] = {1,2,3,4,5,6,7,8,9,10};
    long answer;

    answer = sump(marbles, marbles + SIZE);
    printf("The total number of marbles is %ld.₩n", answer);

    return 0;
}

/* use pointer arithmetic   */
int sump(int * start, int * end)
{
    int total = 0;

    while (start < end)
    {
        total += *start;  /* add value to total              */
        start++;          /* advance pointer to next element */
    }

    return total;
}
```

Use two pointers to describe the array

```
The total number of marbles is 55.
Press any key to continue . . .
```

Pointers
Using Pointer Parameters

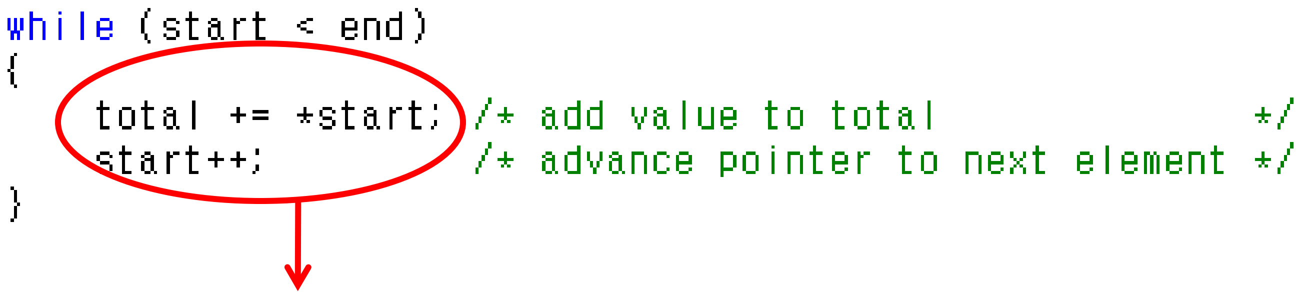


```
/* use pointer arithmetic     */
int sump( int * start, int * end)
{
    int total = 0;

    while (start < end)
    {
        total += *start; /* add value to total           */
        start++;         /* advance pointer to next element */
    }

    return total;
}
```

Use second pointer to finish loop

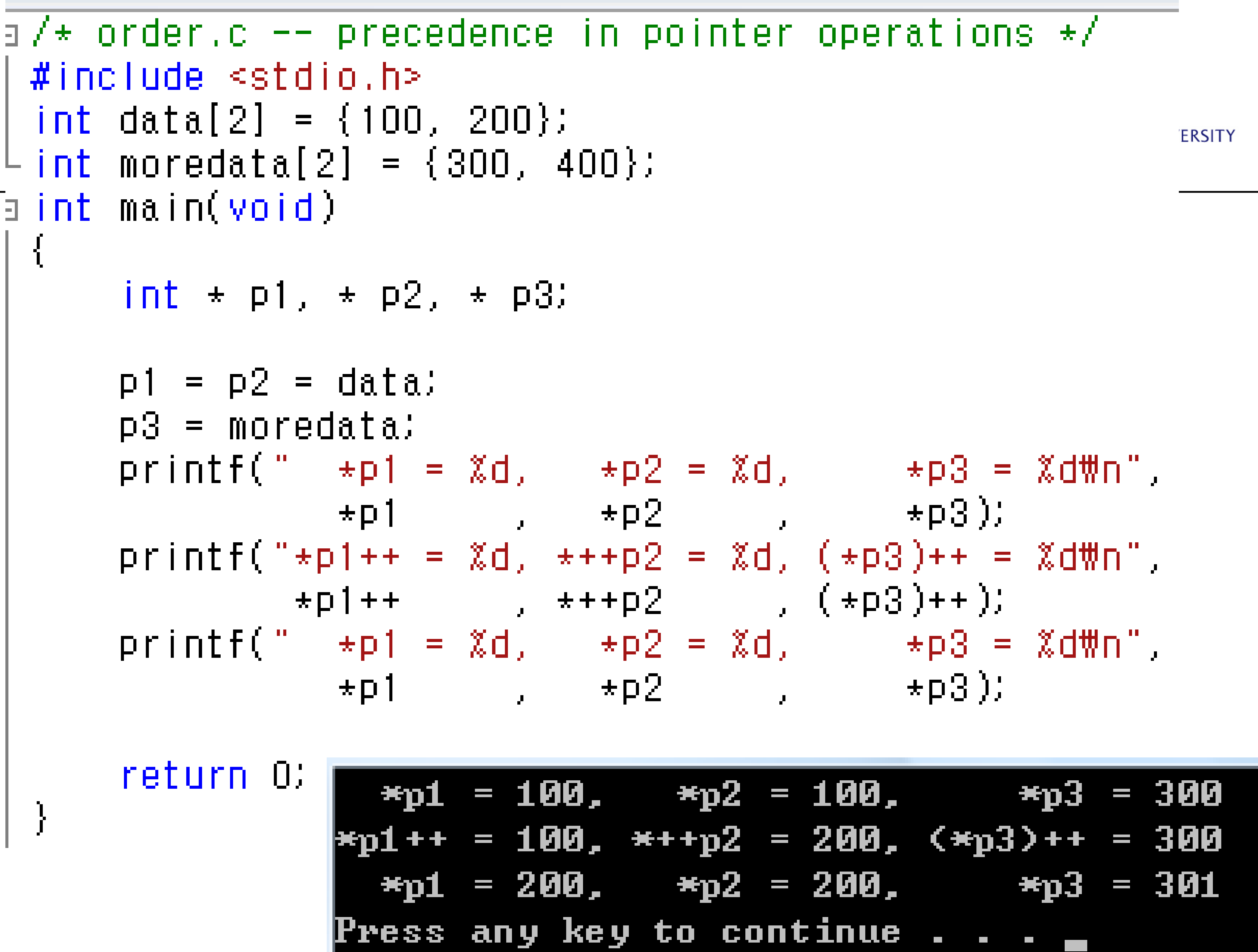
- 마지막으로 처리되는 원소는 end가 가리키는 원소 바로앞에 있는 원소임.
- C는 배열을 위한 공간 할당시 배열의 끝 바로 다음 첫 번째 위치를 가리키는 포인터가 유효함을 보장.
- answer = sump(marbles, marlbes + SIZE -1) ← 덜 이쁘다.

Pointers
Using Pointer Parameters



```
while (start < end)
{
    total += *start;  /* add value to total            */
    start++;          /* advance pointer to next element */
}
```

- total += *start++
- * 와 ++는 우선순위가 같지만 오른쪽에서 왼쪽으로 결합 → 포인터 자체(값이 아니고)가 증가



```c
/* order.c -- precedence in pointer operations */
#include <stdio.h>
int data[2] = {100, 200};
int moredata[2] = {300, 400};
int main(void)
{
    int * p1, * p2, * p3;

    p1 = p2 = data;
    p3 = moredata;
    printf("  *p1 = %d,   *p2 = %d,     *p3 = %d₩n",
              *p1     ,   *p2     ,     *p3);
    printf("*p1++ = %d, *++p2 = %d, (*p3)++ = %d₩n",
            *p1++     , *++p2     , (*p3)++);
    printf("  *p1 = %d,   *p2 = %d,     *p3 = %d₩n",
              *p1     ,   *p2     ,     *p3);

    return 0;
}
```

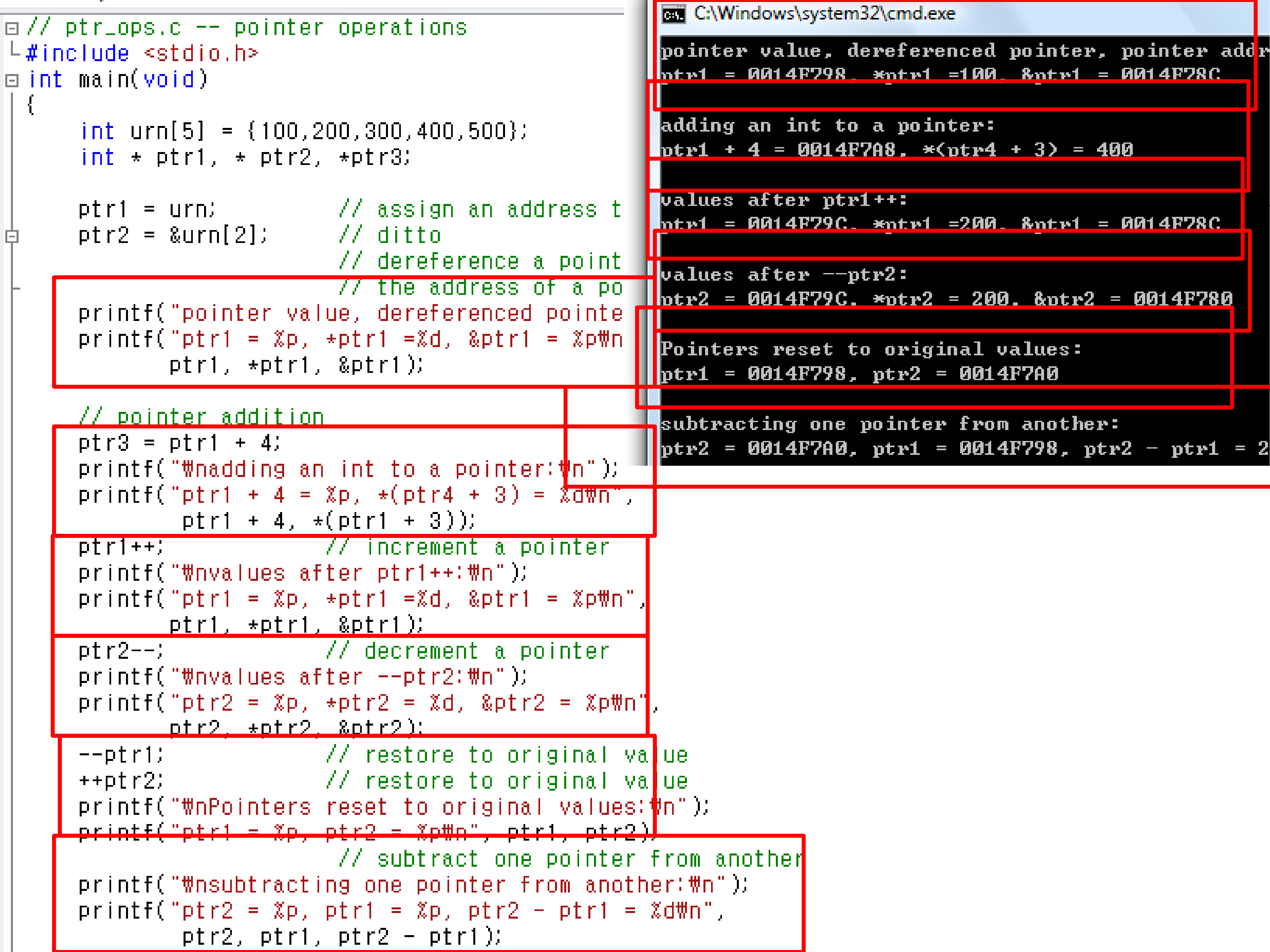
```
  *p1 = 100,   *p2 = 100,     *p3 = 300
*p1++ = 100, *++p2 = 200, (*p3)++ = 300
  *p1 = 200,   *p2 = 200,     *p3 = 301
Press any key to continue . . .
```

Pointers
Pointer operations

- Assignment
  - ptr1 = urn; //assign an address to a pointer
- Value finding
- Taking a pointer address
- Adding an integer to a pointer
- Incrementing a pointer
- Subtracting an integer from a pointer
- Decrementing a pointer

Pointers
Pointer operations

- Differencing
- Comparisons



```
// ptr_ops.c -- pointer operations
#include <stdio.h>
int main(void)
{
    int urn[5] = {100,200,300,400,500};
    int * ptr1, * ptr2, *ptr3;

    ptr1 = urn;            // assign an address t
    ptr2 = &urn[2];        // ditto
                           // dereference a point
                           // the address of a po
    printf("pointer value, dereferenced pointe
    printf("ptr1 = %p, *ptr1 =%d, &ptr1 = %p₩n
           ptr1, *ptr1, &ptr1);

    // pointer addition
    ptr3 = ptr1 + 4;
    printf("₩nadding an int to a pointer:₩n");
    printf("ptr1 + 4 = %p, *(ptr4 + 3) = %d₩n",
            ptr1 + 4, *(ptr1 + 3));
    ptr1++;               // increment a pointer
    printf("₩nvalues after ptr1++:₩n");
    printf("ptr1 = %p, *ptr1 =%d, &ptr1 = %p₩n",
           ptr1, *ptr1, &ptr1);
    ptr2--;               // decrement a pointer
    printf("₩nvalues after --ptr2:₩n");
    printf("ptr2 = %p, *ptr2 = %d, &ptr2 = %p₩n",
           ptr2, *ptr2, &ptr2);
    --ptr1;               // restore to original value
    ++ptr2;               // restore to original value
    printf("₩nPointers reset to original values:₩n");
    printf("ptr1 = %p, ptr2 = %p₩n", ptr1, ptr2);
                           // subtract one pointer from another
    printf("₩nsubtracting one pointer from another:₩n");
    printf("ptr2 = %p, ptr1 = %p, ptr2 - ptr1 = %d₩n",
            ptr2, ptr1, ptr2 - ptr1);
```

```
pointer value, dereferenced pointer, pointer addr
ptr1 = 0014F798, *ptr1 =100, &ptr1 = 0014F78C

adding an int to a pointer:
ptr1 + 4 = 0014F7A8, *(ptr4 + 3) = 400

values after ptr1++:
ptr1 = 0014F79C, *ptr1 =200, &ptr1 = 0014F78C

values after --ptr2:
ptr2 = 0014F79C, *ptr2 = 200, &ptr2 = 0014F780

Pointers reset to original values:
ptr1 = 0014F798, ptr2 = 0014F7A0

subtracting one pointer from another:
ptr2 = 0014F7A0, ptr1 = 0014F798, ptr2 - ptr1 = 2
```

Pointers
uninitialized pointer

- 초기화 하지 않은 포인터의 내용을 참조하지 말 것.
- Ex)

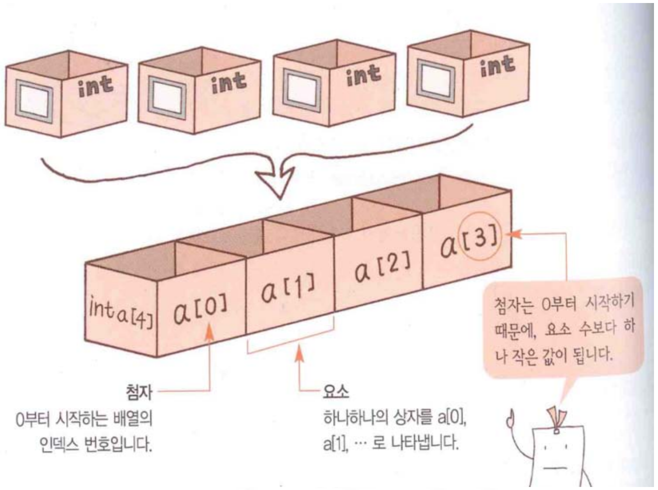
```
int * pt;      //초기화 하지 않은 포인터
*pt = 5;       //지독한 에러
```

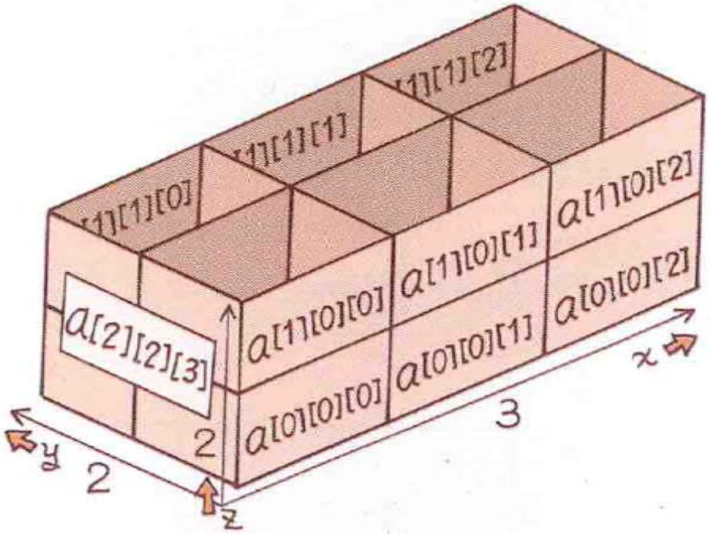
- 5가 어디에 저장될 지 모르고, 어딘가에 해를 입힐 수 있다!
- 포인터를 생성하면 포인터 자체를 저장하기 위한 메모리만 할당되고, 데이터를 저장하기 위한 메모리는 할당되지 않는다.

Today

Chapter 10. Arrays and Pointers (배열과 포인터)

- Arrays
  - initialization, assignment,
  - Multidimensional arrays
  - Pointers and arrays
- Pointers
  - Functions, Arrays, and Pointers
  - Pointer operations


int
int
int
int
int a[4]
a[0]
a[1]
a[2]
a[3]
첨자는 0부터 시작하기 때문에, 요소 수보다 하나 작은 값이 됩니다.
첨자
0부터 시작하는 배열의 인덱스 번호입니다.
요소
하나하나의 상자를 a[0], a[1], … 로 나타냅니다.



Next Week

Chapter 10. & Chapter 13 (File Input/Output)

- Using const for formal parameters
- Pointers and multidimensional array
- File I/O functions
- How to process files using C’s standard I/O family of functions
- Text/binary modes
- Text and binary formats