Fundamentals of Computer System
- chapter 10. Arrays and Pointers

민기복

Ki-Bok Min, PhD

서울대학교 에너지자원공학과 조교수

Assistant Professor, Energy Resources Engineering

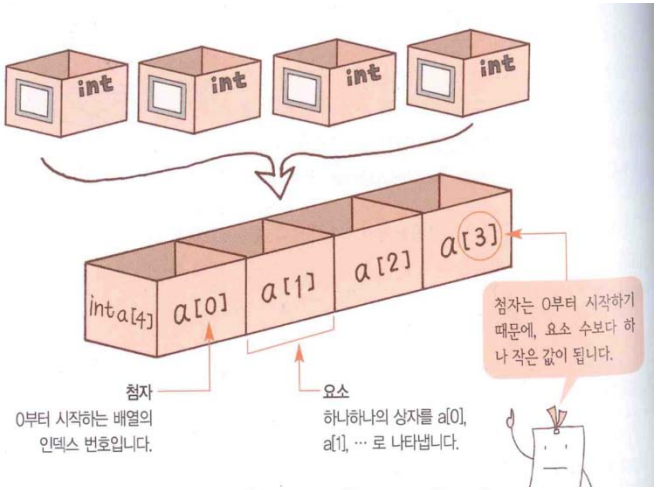
SEOUL NATIONAL UNIVERSITY

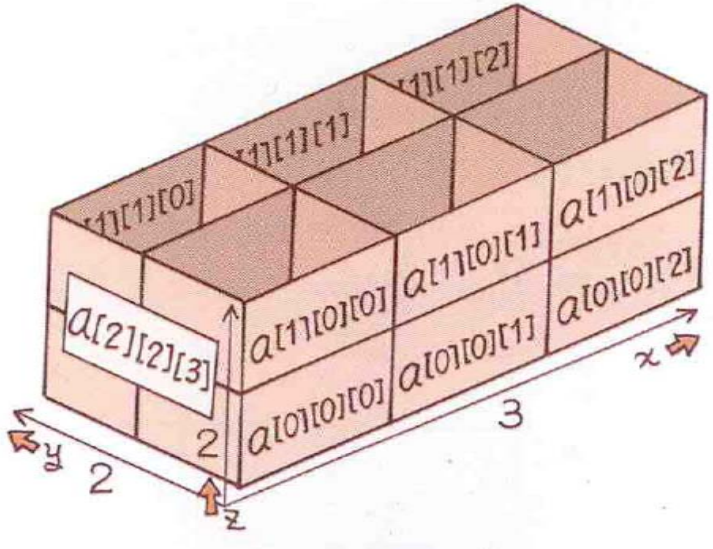
Last Week

Chapter 10. Arrays and Pointers (배열과 포인터)

SEOUL NATIONAL UNIVERSITY

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





Array (배열)
What is it?

SEOUL NATIONAL UNIVERSITY

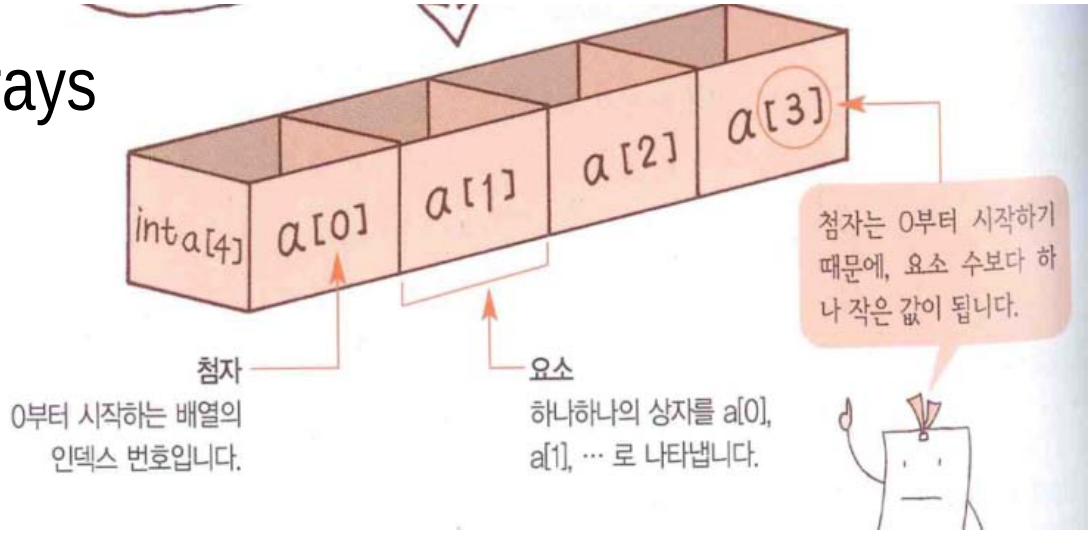
- Array (배열): a series of elements of one data type
  동일한 하나의 데이터형을 가진 연속된 원소
- Array declaration (배열선언): tells the compiler how many elemnts the array contains and what the type is.
  그 배열이 몇 개의 원소를 가지고 있으며, 원소들의 데이터형이 무엇인지 컴파일러에게 알려준다.
- Array elements can have the same types as ordinary variables.

Array (배열)
Declaration

SEOUL NATIONAL UNIVERSITY

```
int a[4];
    ——
    index
```

- [ ] 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]~~



Array (배열)
initialization (초기화)

SEOUL NATIONAL UNIVERSITY

```c
/* 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)
assigning (배열에 값 대입하기)

SEOUL NATIONAL UNIVERSITY

- 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)

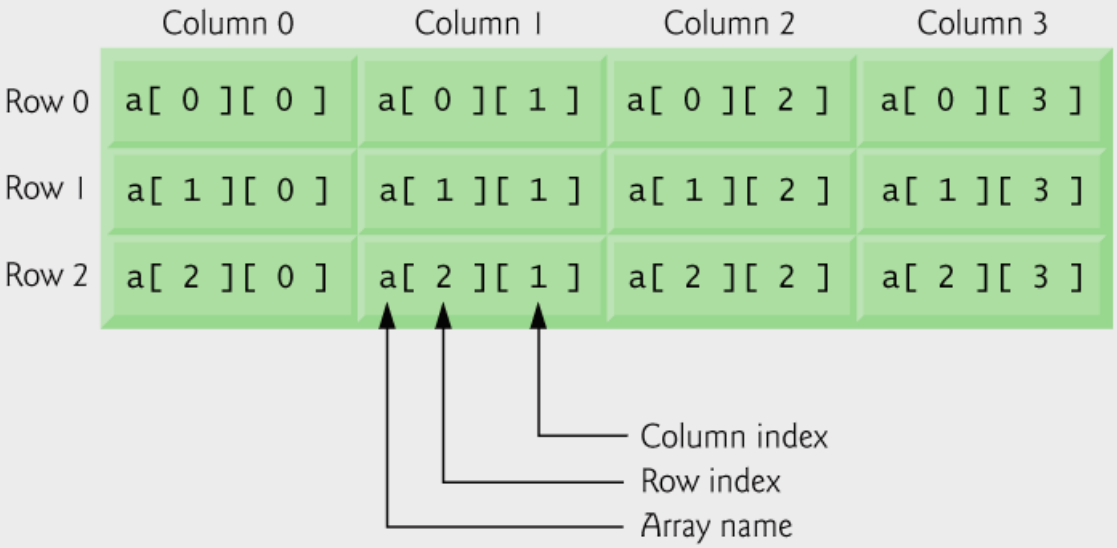
SEOUL NATIONAL UNIVERSITY

- 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)
multidimensional arrays

SEOUL NATIONAL UNIVERSITY



Array (배열)
multidimensional arrays - initialization

SEOUL NATIONAL UNIVERSITY

- 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}

Pointers: a first look
What is it?

SEOUL NATIONAL UNIVERSITY

- 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
the indirection operator(간접연산자)

SEOUL NATIONAL UNIVERSITY

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

↔

```
b = a
```

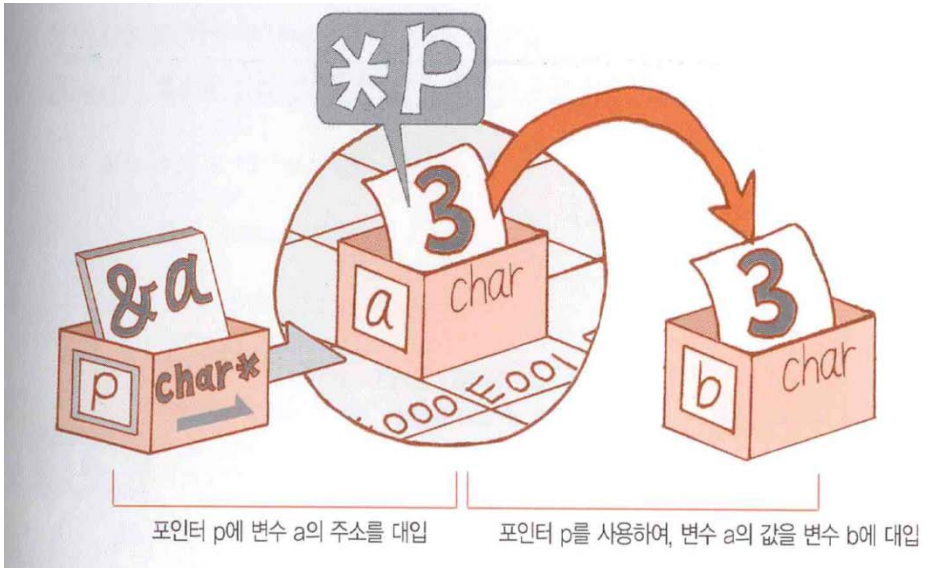
p가 가리키는 주소의 값

ex)

a = 3;

p = &a; //a를 가리키는 포인터

b = *p; //p이 가리키고 있는주소의 값을 b에 대입



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?)

SEOUL NATIONAL UNIVERSITY

```c
/* 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)

SEOUL NATIONAL UNIVERSITY

- 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.

```c
/* 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)

```c
/* 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 . . .
```

Pointers
Pointers and Arrays (포인터와 배열)

SEOUL NATIONAL UNIVERSITY

- Array is simply a disguised use of pointers. (배열표기는 실제로는 포인터의 변장된 사용에 불과하다)
- 배열명은 곧 그 배열의 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 (포인터와 배열)

SEOUL NATIONAL UNIVERSITY

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int ary[5] = {10, 20, 30, 40, 50};
5      int i;
6
7      for (i = 0; i < 5; i++) printf( "%6d", ary[i]);
8          printf("\n\n");
9      for (i = 0; i<5; i++) printf( "%6d", *(ary + i));
10         printf("\n\n");
11
12     return 0;
13 }
```

```
    10    20    30    40    50

    10    20    30    40    50
```

Pointers
Pointers and Arrays (포인터와 배열)

SEOUL NATIONAL UNIVERSITY

```c
#include <stdio.h>
int main(void)
{
    int ary[5] = {10, 20, 30, 40, 50};
    int *ptr, i;
    ptr = ary;

    for (i = 0; i < 5; i++) printf( "%6d", ary[i]);
        printf("\n\n");
    for (i = 0; i<5; i++) printf( "%6d", *(ptr + i));
        printf("\n\n");
    for (i = 0; i<5; i++) printf( "%6d", *(ary + i));
        printf("\n\n");
    for (i = 0; i<5; i++) printf( "%6d", ptr[i]);
        printf("\n\n");

    return 0;
}
```

```
    10    20    30    40    50

    10    20    30    40    50

    10    20    30    40    50

    10    20    30    40    50
```

Pointers
pointers and arrays

SEOUL NATIONAL UNIVERSITY

```c
// 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

SEOUL NATIONAL UNIVERSITY

- 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

```c
#include <stdio.h>
int main(void)
{
    int ary[5] = {10, 20, 30, 40, 50};
    int *ptr, i;
    ptr = &ary[2];

    for (i = 0; i < 5; i++) printf("%6d", ary[i]);
        printf("\n\n");
    for (i = -2; i < 3; i++) printf("%6d", *(ptr + i));
        printf("\n\n");

    return 0;
}
```

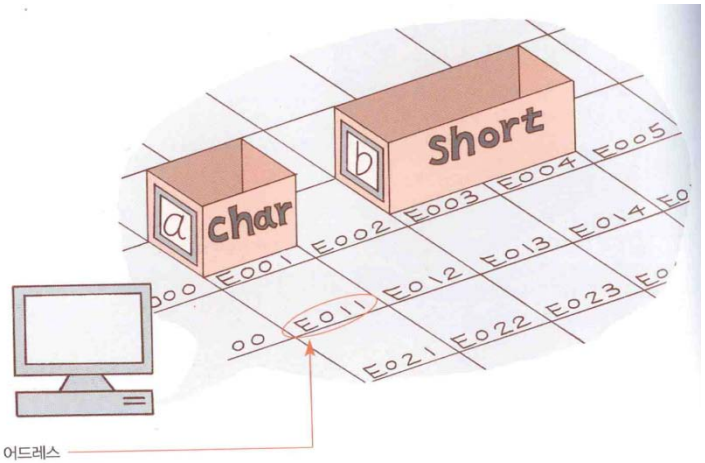
```
    10    20    30    40    50

    10    20    30    40    50

계속하려면 아무 키나 누르십시오 .
```

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.

p가 포인터 변수일때 p+i가 의미하는 값은
p + i* sizeof(대상체) 이다.



Pointers
pointers and arrays

SEOUL NATIONAL UNIVERSITY

- 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

SEOUL NATIONAL UNIVERSITY

```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

SEOUL NATIONAL UNIVERSITY

- 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 포인터**여야 한다.

Pointers
Functions that operates on an array

SEOUL NATIONAL UNIVERSITY

- 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

SEOUL NATIONAL UNIVERSITY

- 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

SEOUL NATIONAL UNIVERSITY

```
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

```c
// 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

ar 은 marble의 첫번째 원소를 가리키는 포인터

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

SEOUL NATIONAL UNIVERSITY

```c
/* 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

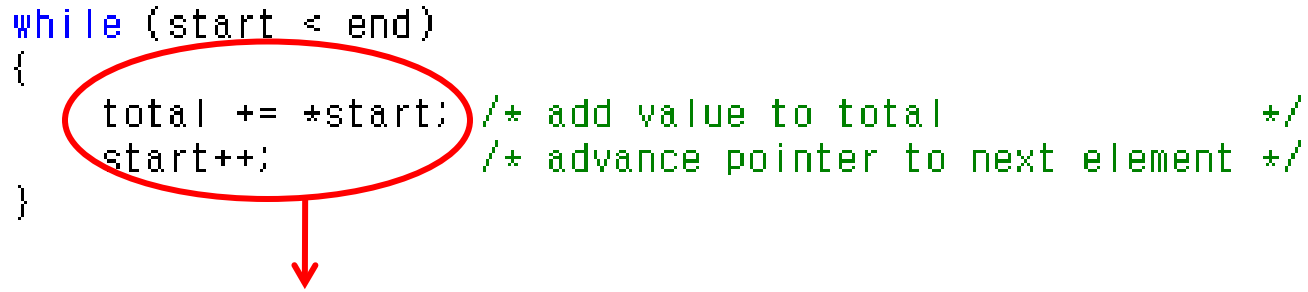
SEOUL NATIONAL UNIVERSITY

```
/* 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



- 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

SEOUL NATIONAL UNIVERSITY

- 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

SEOUL NATIONAL UNIVERSITY

- Differencing
- Comparisons

```c
// 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 va ue
    ++ptr2;              // restore to original va ue
    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);
```

```
C:\Windows\system32\cmd.exe
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
Pointers and multidimensional arrays

SEOUL NATIONAL UNIVERSITY

- int zippo[4][2]
- zippo는 그 배열의 첫번째 원소의 주소
  → zippo ==&zippo[0]
  → zippo[0] == &zippo[0][0]
  →*(zippo[0]) ==zippo[0][0]
  →*(zippo) == zippo[0]

- **zippo = zippo[0][0]

Pointers
Pointers and multidimensional arrays

SEOUL NATIONAL UNIVERSITY

- 값 표현

  zippo[m][n] == *(*(zippo+ m) + n)

- 주소 표현

  &zippo[m][n] == (*(zippo+m) + n)

Compiler use these form → faster computation.

- 2차원 배열을 나타내는 포인터 변수에는 *를 두 개를 써야 비로소 값을 나타낸다. *를 하나 썼을 때 여전히 주소를 나타냄 → 2차원 포인터.
- 1차원 포인터 → * zippo → 값
- 2차원 포인터 → ** zippo → 값

Pointers
Pointers and multidimensional arrays

SEOUL NATIONAL UNIVERSITY

```c
#include <stdio.h>
int main(void)
{
    int ary[3][2] = {{1,2},{3,4},{5,6}};
    int i,j;

    for(i = 0; i < 3; i++) {
        printf("\n *(ary+%d) : %p\t", i, *(ary+i));
        for (j = 0; j < 2; j++)
            printf("%5d", *(*(ary+i)+j));
    }
    printf("\n");

    return 0;
}
```

```
*(ary+0) : 0017FF14        1    2
*(ary+1) : 0017FF1C        3    4
*(ary+2) : 0017FF24        5    6
계속하려면 아무 키나 누르십시오 . . .
```

Pointers
Pointers and multidimensional arrays

SEOUL NATIONAL UNIVERSITY

```c
/* zippo1.c --  zippo info */
#include <stdio.h>
int main(void)
{
    int zippo[4][2] = { {2,4}, {6,8}, {1,3}, {5, 7} };

    printf("   zippo = %p,    zippo + 1 = %p₩n",
           zippo,          zippo + 1);
    printf("zippo[0] = %p, zippo[0] + 1 = %p₩n",
           zippo[0],       zippo[0] + 1);
    printf("  *zippo = %p,   *zippo + 1 = %p₩n",
           *zippo,         *zippo + 1);
    printf("zippo[0][0] = %d₩n", zippo[0][0]);
    printf("  *zippo[0] = %d₩n", *zippo[0]);
    printf("    **zippo = %d₩n", **zippo);
    printf("      zippo[2][1] = %d₩n", zippo[2][1]);
    printf("*(*(zippo+2) + 1) = %d₩n", *(*(zippo+2) + 1));

    return 0;
}
```

```
   zippo = 0017FF0C,    zippo + 1 = 0017FF14
zippo[0] = 0017FF0C, zippo[0] + 1 = 0017FF10
  *zippo = 0017FF0C,   *zippo + 1 = 0017FF10
zippo[0][0] = 2
  *zippo[0] = 2
    **zippo = 2
      zippo[2][1] = 3
*(*(zippo+2) + 1) = 3
계속하려면 아무 키나 누르십시오
```

SEOUL NATIONAL UNIVERSITY

```c
/* zippo2.c --  zippo info via a pointer variable */
#include <stdio.h>
int main(void)
{
    int zippo[4][2] = { {2,4}, {6,8}, {1,3}, {5, 7} };
    int (*pz)[2];
    pz = zippo;

    printf("   pz = %p,    pz + 1 = %p\n",
               pz,          pz + 1);
    printf("pz[0] = %p, pz[0] + 1 = %p\n",
            pz[0],       pz[0] + 1);
    printf("  *pz = %p,   *pz + 1 = %p\n",
              *pz,         *pz + 1);
    printf("pz[0][0] = %d\n", pz[0][0]);
    printf("  *pz[0] = %d\n", *pz[0]);
    printf("    **pz = %d\n", **pz);
    printf("      pz[2][1] = %d\n", pz[2][1]);
    printf("*(*(pz+2) + 1) = %d\n", *(*(pz+2) + 1));

    return 0;
}
```

2차원 배열을 가리키는 포인터 변수의 선언

```
   pz = 0017FF0C,    pz + 1 = 0017FF14
pz[0] = 0017FF0C, pz[0] + 1 = 0017FF10
  *pz = 0017FF0C,   *pz + 1 = 0017FF10
pz[0][0] = 2
  *pz[0] = 2
    **pz = 2
      pz[2][1] = 3
*(*(pz+2) + 1) = 3
계속하려면 아무 키나 누르십시오 . . .
```

Pointers
Functions and multidimensional arrays

SEOUL NATIONAL UNIVERSITY

- 2차원 배열을 전달인자로 하는 함수의 예 (pt가 형식매개변수일때)

- void somefunction (int (* pt)[4]); or
- void somefunction(int pt[ ][4]);

Pointers
Functions and multidimensional arrays

SEOUL NATIONAL UNIVERSITY

ay2d.c 지속 페이지

범위)

```c
// array2d.c -- functions for 2d arrays
#include <stdio.h>
#define ROWS 3
#define COLS 4
void sum_rows(int ar[][COLS], int rows);
void sum_cols(int [][COLS], int );    // ok to omit names
int sum2d(int (*ar)[COLS], int rows); // another syntax
int main(void)
{
     int junk[ROWS][COLS] = {
            {2,4,6,8},
            {3,5,7,9},
            {12,10,8,6}
     };

     sum_rows(junk, ROWS);
     sum_cols(junk, ROWS);
     printf("Sum of all elements = %d₩n", sum2d(junk, ROWS));

     return 0;
}

void sum_rows(int ar[][COLS], int rows)
{
    int r;
    int c;
    int tot;

    for (r = 0; r < rows; r++)
    {
        tot = 0;
```

```
row 0: sum = 20
row 1: sum = 24
row 2: sum = 36
col 0: sum = 17
col 1: sum = 19
col 2: sum = 21
col 3: sum = 23
Sum of all elements = 80
계속하려면 아무 키나 누르십시
```

Pointers
uninitialized pointer

SEOUL NATIONAL UNIVERSITY

- 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)

SEOUL NATIONAL UNIVERSITY

- Pointers and arrays
- Functions, Arrays, and Pointers
- Pointer operations
- Pointers and multidimensional (2D) arrays
- Functions and multidimensional (2D) arrays
- Try to understand the following files which are available in the eTL
  - pointer1.c, pointer2.c, pointer3.c, poiner4.c, order.c, zippo1.c, zippo2.c, array2d.c

Exercise
Preview of Homework 6.

SEOUL NATIONAL UNIVERSITY

- Homework 6.1.
  - Rewrite the program for homework 5.1 so that the main tasks are performed by functions instead of in main(). Please refer to array2d.c in the textbook.

1. Monthly temperature data of the past three years are given in the below. Write a program that finds the yearly average temperature and monthly average temperature for the past three years.

Temperature data (in °C):

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2006	-4.2	-1.4	4.5	10.8	16.9	21.8	24.3	25.2	19.5	13.0	5.8	-0.8
2007	-4.0	-2.0	4.4	11.0	15.9	22.0	25.4	24.9	20.0	13.9	6.0	0.0
2008	-4.1	0.1	5.6	12.0	17.1	22.0	25.9	27.0	20.3	13.7	6.1	-0.5

The program may produce results similar to the followings.

The yearly average temperature during 2006 - 2008:

2006	2007	2008
xxx	xxx	xxx

The monthly average temperature during 2006-2008:

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx