

---

# **컴퓨터 프로그래밍 I**

## **(CSE2003-3)**

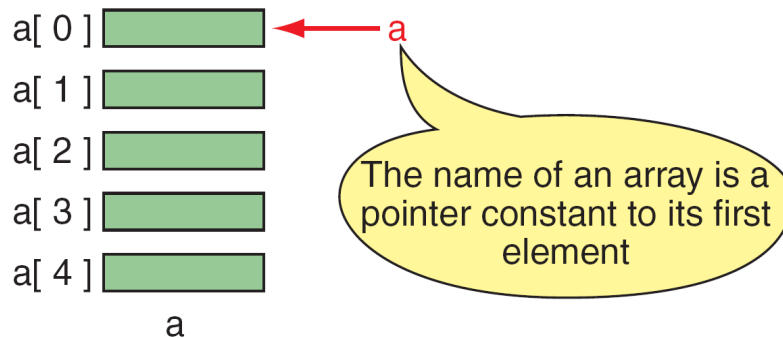
---

**Mon/Wed 16:30-17:45 pm**  
**Lecture 22**

Array and Pointers

# Arrays and pointers

- ◆ 배열 이름은 첫 번째 요소의 주소 값을 나타낸다.
- ◆ `int a[5];` 와 같이 선언되었을 때 메모리의 상황은 다음과 같다.



- 배열 이름인 `a`는 배열의 첫 번째 원소인 `a[0]`을 가리키고 있다.
- 즉, `a` 는 `&a[0]` 값을 가지고 있다.

`a` ← same → `&a[0]`  
`a` is a pointer only to the first element—not the whole array.

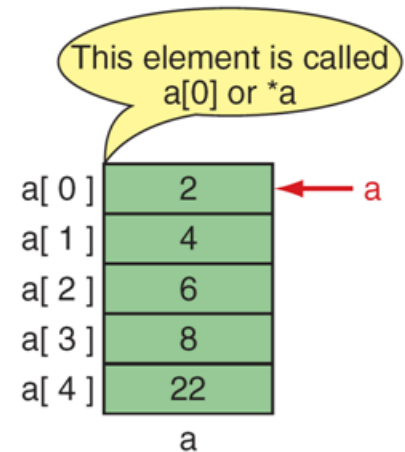
# Arrays and pointers

- ◆ 다음은 `a`와 `&a[0]`이 같은 값을 가짐을 보이는 예제이다.
- ◆ `a`가 `a[0]`의 주소값을 가지고 있기 때문에, 당연히 `a[0]`은 `*a`와 같은 표현으로도 접근할 수 있다.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a[5]={2, 4, 6, 8, 22};
6
7     printf("a : %d,      &a[0] : %d\n", a, &a[0]);
8     printf("*a : %d,      a[0] : %d\n", *a, a[0]);
9
10    return 0;
11 }
```

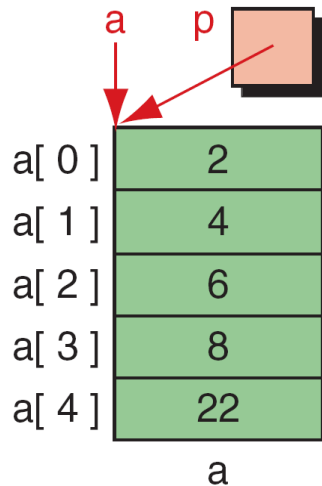
```
[root@mclab chap10]# ./chap10-1
a : -1076213348,      &a[0] : -1076213348
*a : 2,      a[0] : 2
[root@mclab chap10]#
```

`*a`와 `a[0]` 모두 배열 `a`의 첫번째 원소를 가리킨다.



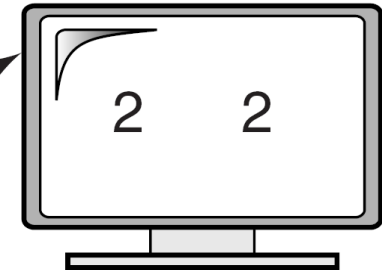
# Arrays and pointers

- ◆ 배열 이름 **a**는 주소값을 갖는 포인터이다.
  - 주소값을 다른 포인터 변수에 할당할 수도 있다.



```
#include <stdio.h>
int main (void)
{
    int a[5] = {2, 4, 6, 8, 22};
    int* p = a;
    ...
    printf("%d %d\n", a[0], *p);
    ...
    return 0;
} // main
```

배열의 시작주소값을  
포인터 변수 **p**에 할당

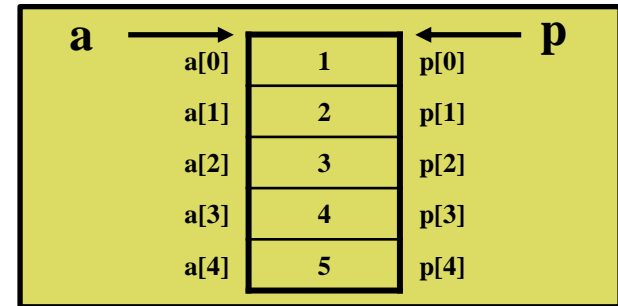


- ◆ 단, 배열 이름 **a**는 배열의 시작 주소를 갖도록 고정되어 있기 때문에 **a**의 값을 변경할 수는 없다.

# Arrays and pointers

- ▶ 포인터 변수가 배열을 참조하도록 함으로써 포인터 변수를 배열처럼 사용하는 것도 가능하다.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i;
6     int sum1=0;
7     int sum2=0;
8     int a[5]={1,2,3,4,5};
9     int *p=a;
10
11     for(i=0; i<5; i++) {
12         sum1 += a[i];
13         sum2 += p[i];
14     }
15
16     printf("%d == %d\n", sum1, sum2);
17
18     return 0;
19 }
```



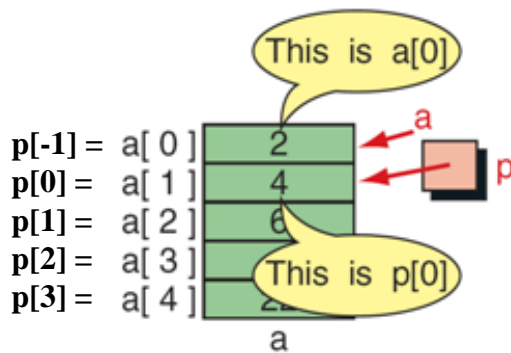
포인터변수 p도 a와 같은 값을 갖는다. 즉, 배열의 시작 주소를 갖는다.

a[i]와 p[i]는 같은 결과를 갖는다.  
결국 p를 이용해서도 배열에 접근할 수 있음을 알 수 있다.

```
[root@mclab chap10]# ./chap10-2
15 == 15
[root@mclab chap10]#
```

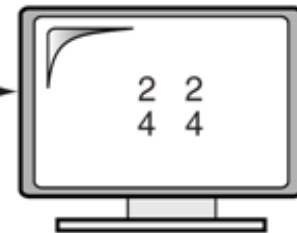
# Arrays and pointers

- ◆ 배열의 중간 부분의 주소를 pointer 변수에 할당할 수도 있다.
  - 이 경우 같은 메모리 공간을 access하기 위해서는 서로 다른 index를 사용해야 한다.



```
#include <stdio.h>
int main (void)
{
    int a[5] = {2, 4, 6, 8, 22};
    int* p;
    ...
    p = &a[1];

    printf("%d %d", a[0], p[-1]);
    printf("\n");
    printf("%d %d", a[1], p[0]);
    ...
} // main
```



# Pointer arithmetic and arrays

## ◆ Pointers and One-Dimensional Arrays

- $p$ 가 특정 데이터타입에 대한 포인터 변수일 때
  - ◆ 수식  $p + 1$ 은 해당 데이터타입의 다음 변수를 나타낸다.
- 포인터 연산  $p + 1$ 에서 더해지는 값 1은 현재 포인터의 data type의 size만큼을 더하는 것이다.

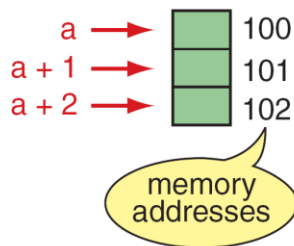
포인터 + 정수값



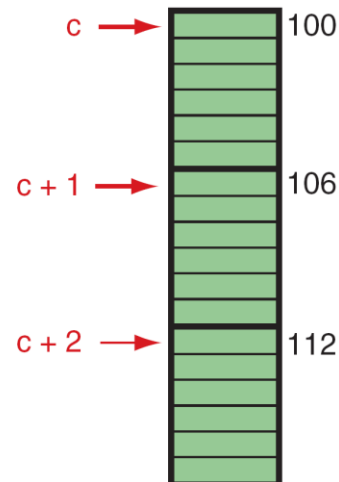
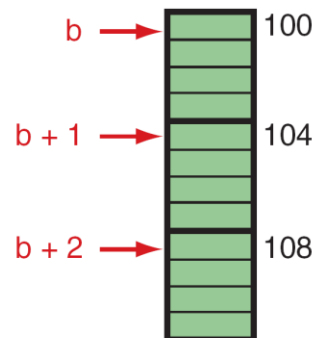
포인터 + (정수값 \* 포인터가 가리키는 자료형의 크기)

- 다른 타입을 갖는 배열에 대한 포인터 연산의 결과

*size of data type*

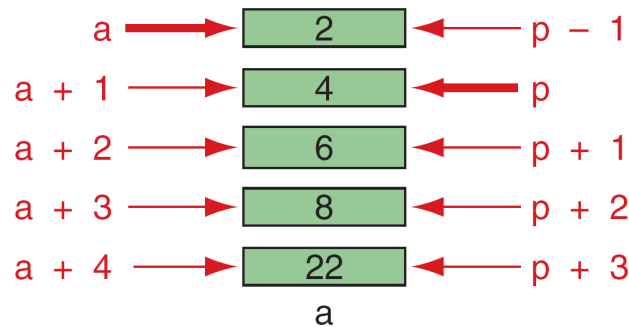


```
char a[3];  
int b[3];  
float c[3];
```

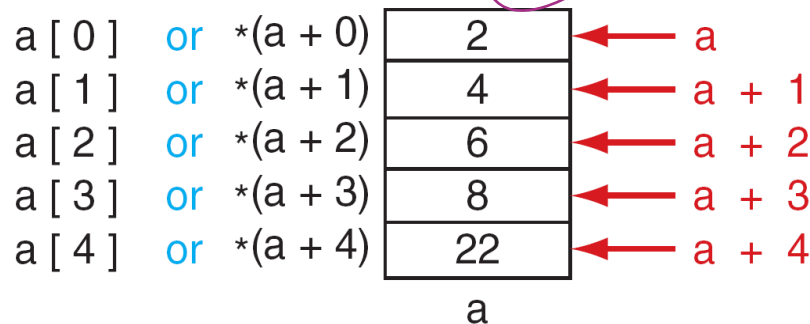


# Pointer arithmetic and arrays

- ◆ 포인터 변수  $p$ 가 배열의 두 번째 원소를 참조하도록 초기화 한 경우
  - $(-)$  연산은  $(+)$  연산과 마찬가지로 해당되는 type의 하나 이전의 element를 가리키게 된다.



- ◆ 역참조 연산자를 이용한 사용도 가능하다.
  - $a+1$ 은  $\&a[1]$ 와 같으므로  $*(a+1)$ 은  $a[1]$ 와 같다.

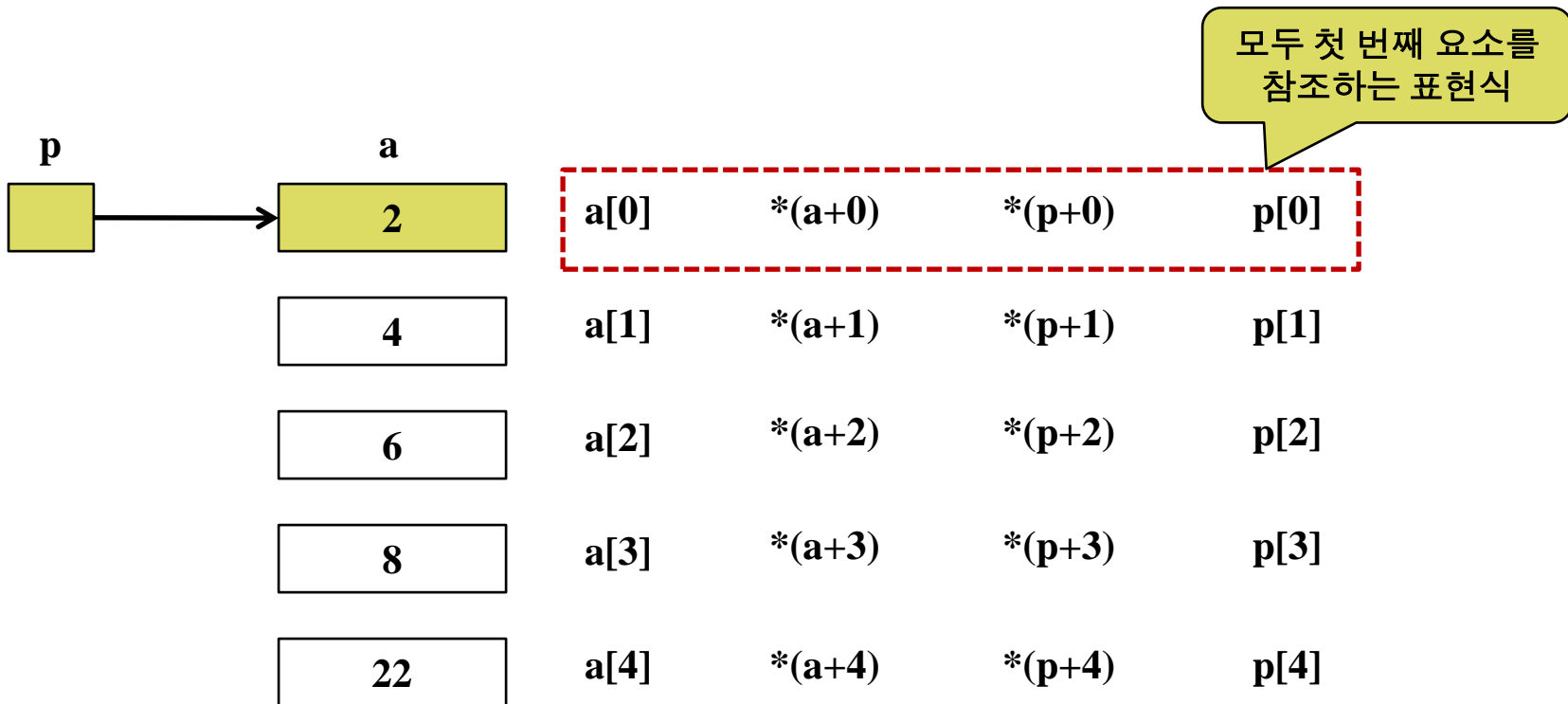


7을 40/21  
\*a+1은 (x a)+1이므로 3, ...  
값으로 주어야



# Pointer arithmetic and arrays

- 배열의 요소를 참조하는 여러가지 방법



# Pointer arithmetic and arrays

## ➤ Pointers And Other Operators

- 포인터 연산이란 포인터 값을 증가 혹은 감소시키는 연산을 말한다.
- 포인터 연산이 가능한 연산은 제한되어 있다.
  - 포인터 연산에 따른 실질적인 값의 변화는 포인터 타입에 따라 다르다.
  - 포인터를 나누거나 곱하는 연산은 불가능하다.
  - 예제 프로그램 – 포인터 연산을 이용한 값의 변화를 출력한다.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     char* ptr1=0;
6     int* ptr2=0;
7     double* ptr3=0;
8
9     printf("%d, %d, %d\n", ptr1++, ptr2++, ptr3++);
10    printf("%d, %d, %d\n", ptr1, ptr2, ptr3);
11
12    return 0;
13 }
```

```
[root@mclab chap10]# ./chap10-3
0, 0, 0
1, 4, 8
[root@mclab chap10]#
```

# Pointer arithmetic and arrays

- Pointers And Other Operators

- 관계 연산자(Relational operators)는 양쪽의 피 연산자가 모두 포인터인 경우에만 사용이 가능하다.

```
p1 >= p2    p1 != p2
```

- 포인터에서 관계연산자를 사용하는 가장 일반적인 경우는 포인터와 NULL 상수를 비교하는 경우이다.

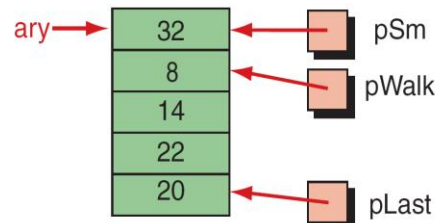
| Long Form        | Short Form |
|------------------|------------|
| if (ptr == NULL) | if (!ptr)  |
| if (ptr != NULL) | if (ptr)   |

*malloc  
↳ 할당 후 NULL로 초기화  
C 언어의 관례  
↳ NULL로 초기화  
NULL ⇒ 메모리 주소가 없음*

# Pointer arithmetic and arrays

- 포인터를 이용하여 배열의 원소 중 가장 작은 값을 가지는 원소를 찾는 프로그램의 일부이다. 그림은 프로그램이 실행되는 과정을 보인다.

```
pLast = ary + arySize - 1;  
for (pSm = ary, pWalk = ary + 1;  
     pWalk <= pLast;  
     pWalk++)  
    if (*pWalk < *pSm)  
        pSm = pWalk;
```

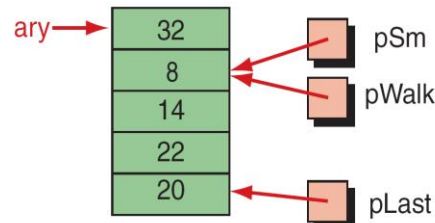


After initialization

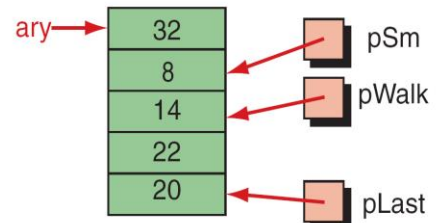
pSm is smallest.  
It tracks the  
smallest value.

pWalk is walker.  
It moves to  
find smallest.

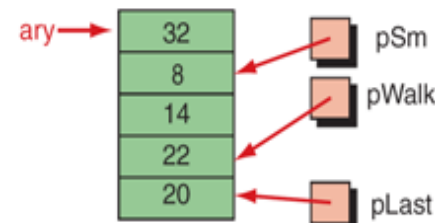
- pSm은 가장 작은 원소를 가리키기 위해 사용된다.
- pWalk은 가장 작은 원소를 찾기 위해 배열을 순서대로 탐색한다.
- pLast는 마지막 원소를 가리킨다.



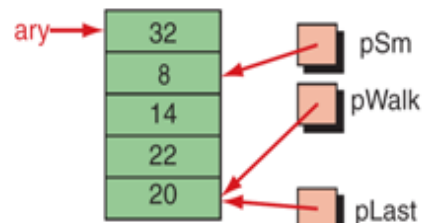
After first iteration



After second iteration



After third iteration



After fourth iteration

# Pointer arithmetic and arrays

## ◆ Pointer subtraction

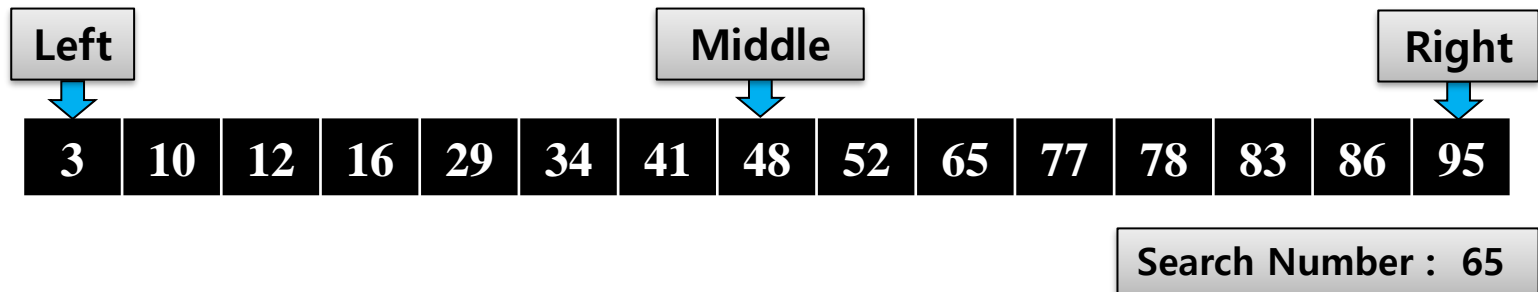
- 포인터 값끼리의 뺄셈은 두 개의 주소 값의 차를 자료형의 크기로 나누어 반환한다.
- 포인터 값끼리의 덧셈 연산은 불가능하다
- 포인터 값의 곱셈 및 나눗셈 연산은 불가능하다

| Type of Left Operand | Operator | Type of Right Operand | Type of result         |
|----------------------|----------|-----------------------|------------------------|
| Pointer              | +        | Pointer               | Error                  |
| Pointer              | -        | Pointer               | <u>Integer(offset)</u> |
| Pointer              | +        | Int                   | Address                |
| Pointer              | -        | Int                   | Address                |

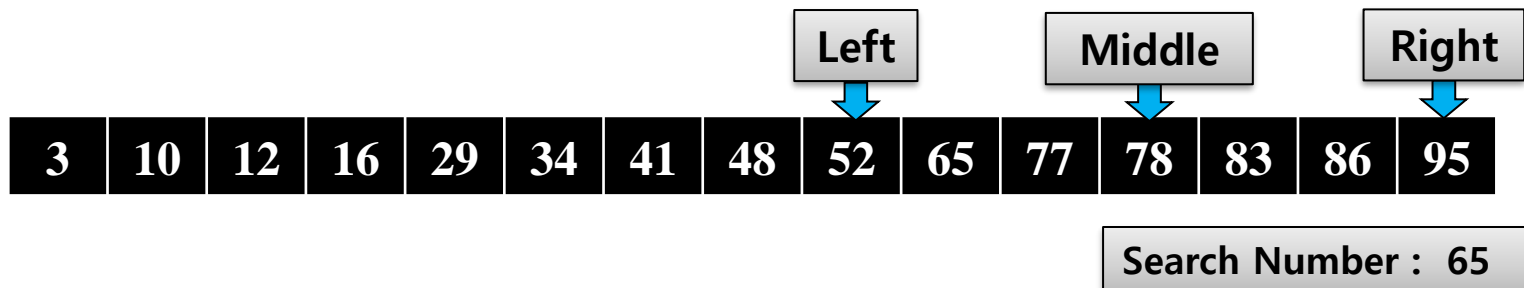
# Pointer arithmetic and arrays

## ◆ Binary Search

- 오름차순으로 정렬된 배열에서 원하는 숫자의 위치를 탐색하는 방법
- Left는 배열의 첫 번째 자리, Right는 마지막 자리, Middle은 중간 위치를 나타낸다.

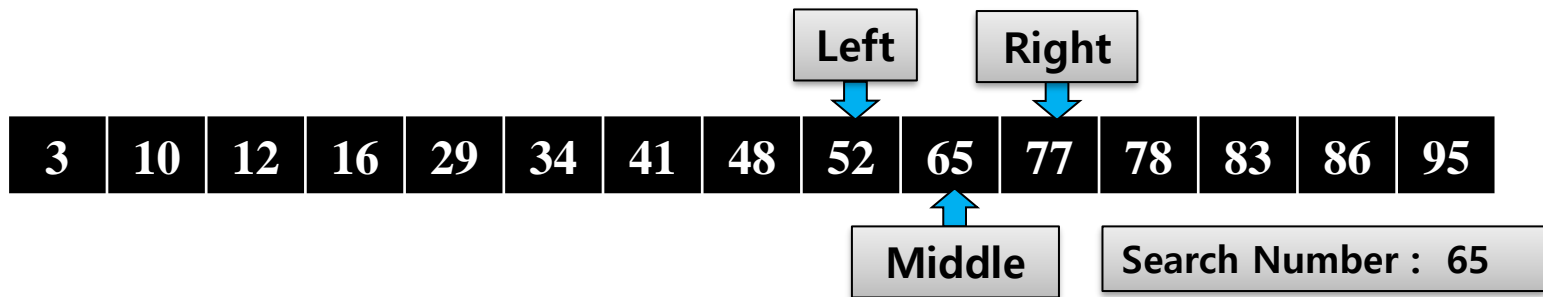


- 가운데 위치의 값과 Search Number를 비교하여 작을 경우에는 Left의 값을 가운데 위치보다 1 크게 잡는다.

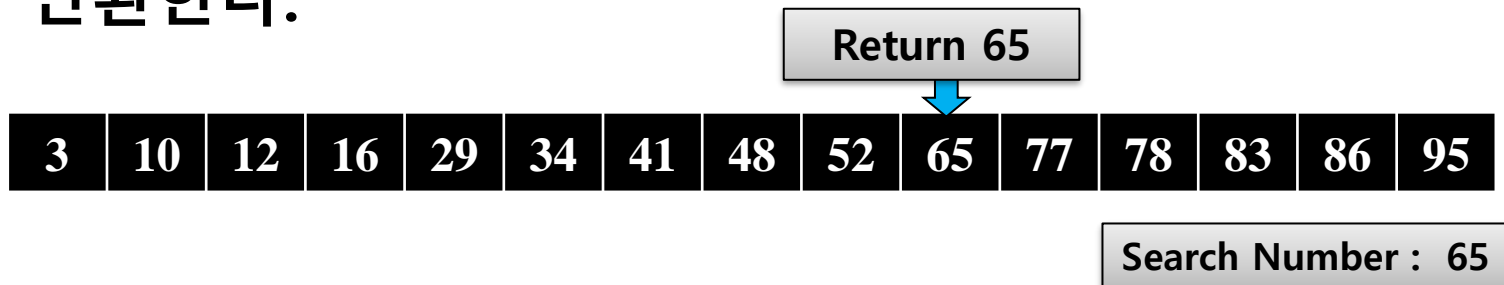


# Pointer arithmetic and arrays

- ◆ Search Number가 Middle에 위치한 값 보다 <sup>작다</sup> 경우에는 Right의 값을 가운데 위치보다 1 작게 잡는다.



- ◆ Middle에 위치한 값과 Search Number의 값이 같으면 Middle을 반환한다.



# Pointer arithmetic and arrays

- ◆ 배열과 index syntax를 사용할 경우, Binary search의 middle index의 계산은 다음과 같이 수행할 수 있다.

- $\text{Middle} = (\text{Left} + \text{Right}) / 2;$

- ◆ 그러나 middle, left, right가 배열이 가진 특정 원소의 주소 값이라면, 두 포인터의 덧셈과 나눗셈 연산이 불가능하기 때문에 다음과 같은 주소 값 계산 방식이 필요하다.

- $\text{midPtr} = \text{firstPtr} + (\text{LastPtr} - \text{FirstPtr}) / 2;$

*offset*

- ◆ 위 표현식에서 피연산자들의 자료형은 다음과 같은 순서로 계산된다

$$\text{Address} = \text{Address} + (\text{offset}) / 2$$

$$= \text{Address} + (\text{int}) / \text{int}$$

$$= \text{Address} + \text{int}$$



# Pointer arithmetic and arrays

## ➤ Using Pointer Arithmetic

- 다음은 포인터를 이용한 binary search 함수이다.

```
1  /* =====binary Search=====
2  Search an ordered list using Binary Search
3  Pre   list must contain at least one element
4        endPtr is pointer to largest element in list
5        target is value of element being sought
6  Post  FOUND: locnPtr pointer to target element
7        return 1 (found)
8        !FOUND: locnPtr = element below or above target
9        return 0 (not found)
10 */
11 int binarySearch (int list[], int* endPtr,
12                  int target, int** locnPtr)
13 {
14     // Local Declarations
15     int* firstPtr;
16     int* midPtr;
17     int* lastPtr;
18 }
```

- list[ ]는 정렬된 data set
- endPtr은 현재 list배열의 마지막 원소를 참조
- target은 list에서 찾아야 할 값
- locnPtr은 찾은 원소를 참조할 포인터 변수

# Pointer arithmetic and arrays

```
19 // Statements
20 firstPtr = list;
21 lastPtr = endPtr;
22 while (firstPtr <= lastPtr)
23 {
24     midPtr = firstPtr + (lastPtr - firstPtr) / 2;
25     if (target > *midPtr)
26         // look in upper half
27         firstPtr = midPtr + 1;
28     else if (target < *midPtr)
29         // look in lower half
30         lastPtr = midPtr - 1;
31     else
32         // found equal: force exit
33         firstPtr = lastPtr + 1;
34 } // end while
35 *locnPtr = midPtr;
36 return (target == *midPtr);
37 } // binarySearch
```

list의 중간 원소를 지정한다

list의 중간 이후  
부분일 경우

list의 중간 이전  
부분일 경우

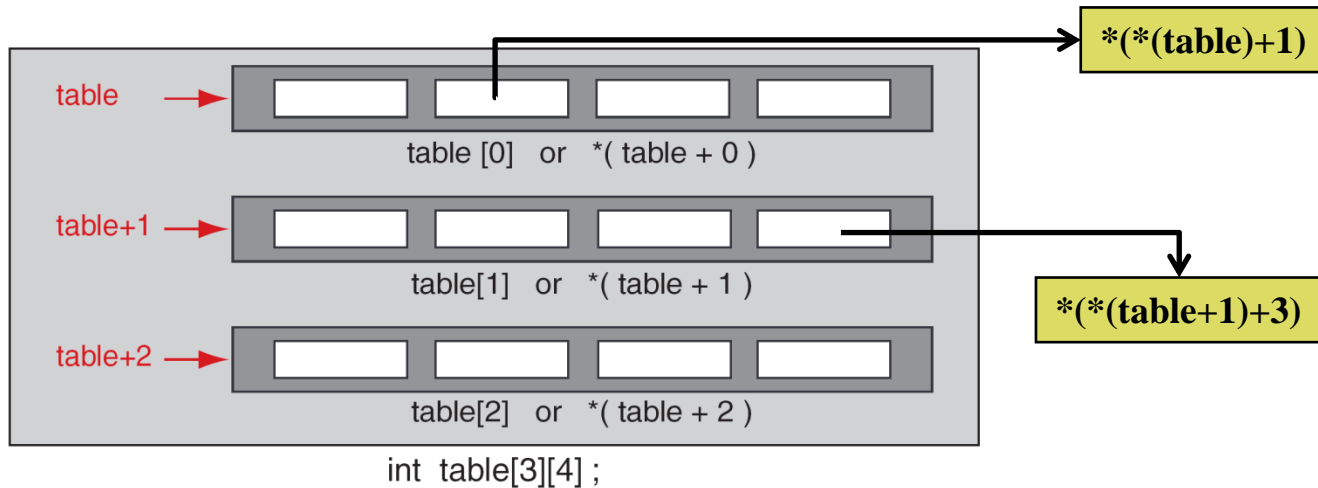
target == \*midPtr일 경우

찾았는지 여부를 리턴  
(찾았다면 target과 \*midPtr 이  
같으므로 1을 리턴)

# Pointer arithmetic and arrays

## ➤ Pointers And Two-Dimensional Arrays

- 이차원 배열 `table`의 원소 `table[1][3]`을 포인터를 사용하여 접근한다면 다음과 같은 표현이 가능하다 ➔ `*(*(table + i) + j)`



```
for (i = 0; i < 3; i++)  
{  
    for (j = 0; j < 4; j++)  
        printf("%6d", *(*(table + i) + j));  
    printf( "\n" );  
} // for i
```

Print Table

# Passing an array to a function

- ◆ 배열을 인자로 하는 함수의 일반적인 호출 형태는 다음과 같다.
  - 배열의 시작 주소값을 인자로 전달한다.

```
doIt (aryName);
```

- ◆ 따라서 배열을 인자로 받는 함수의 헤더는 다음과 같을 것이다.

```
int doIt (int ary[])
```

- ◆ 또한 1차원 배열을 인자로 하는 경우 포인터를 이용한 경우도 가능하다.

```
int doIt (int *arySalary)
```

# Passing an array to a function

- ◆ 함수의 인자로 배열명을 전달하면, 배열의 시작 주소가 전달된다.

```
doIt (aryName);
```

배열명은 배열의 시작 주소값을 가지므로 주소가 전달된다.

- ◆ 호출되는 함수의 헤더에서는 형식 매개변수(formal parameter)로 포인터 변수를 사용하여야 한다.

```
int doIt (int *arySalary)
```

- ◆ 표기상의 편리성 때문에 포인터를 매개변수로 선언할 때 배열의 각괄호 표기법을 사용할 수 있다.
  - 이 때 배열로 선언된 형식 매개변수는 실질적으로는 포인터이다.
  - 배열 원소 자체는 복사되지 않는다.

```
int doIt (int ary[])
```

사실상 포인터!

# Passing an array to a function

## ◆ 예제 프로그램: `a[]`와 `*a`가 같음을 보여주는 예제

```
1 #include <stdio.h>
2
3 double sum1(double a[], int n);
4 double sum2(double *a, int n);
5
6 int main(void)
7 {
8     double a[6]={1, 2, 3, 4, 5, 6};
9
10    printf("sum1 = %f\n", sum1(a,6));
11    printf("sum2 = %f\n", sum2(a,6));
12
13    return 0;
14 }
15
16 double sum1(double a[], int n)
17 {
18     int i;
19     double sum=0.0;
20     for(i=0; i<n; ++i)
21         sum+=a[i];
22     return sum;
23 }
24
25
```

배열의 주소값과 배열의  
길이를 인자로 전달

```
[root@mclab chap10]# ./chap10-4
sum1 = 21.000000
sum2 = 21.000000
[root@mclab chap10]#
```

두 함수의 수행 결과는 서로 같다.

형식 파라미터가 서로 다르다.

```
26 double sum2(double *a, int n)
27 {
28     int i;
29     double sum = 0.0;
30     for(i=0; i<n; i++)
31         sum+=a[i];
32     return sum;
33 }
34
```

# Passing an array to a function

- ◆ 배열을 인자로 하여 앞의 `sum()` 함수를 호출했을 때 계산되는 값은 다음 표와 같다. (`v`는 `main` 함수에서 선언된 배열이라 가정한다.)

| 호출                                 | 계산 및 리턴되는 값                                   |
|------------------------------------|---|
| <code>sum(v, 100)</code>           | <code>v[0] + v[1] + ... + v[99]</code>        |
| <code>sum(v, 88)</code>            | <code>v[0] + v[1] + ... + v[87]</code>        |
| <code>sum(&amp;v[7], k - 7)</code> | <code>v[7] + v[8] + ... + v[k - 1]</code>     |
| <code>sum(v + 7, 2 * k)</code>     | <code>v[7] + v[8] + ... + v[2 * k + 6]</code> |

# Passing an array to a function

- ◆ 2차원 이상의 배열을 형식 매개변수로 갖는 함수는 (첫 번째를 제외한 나머지 차원의) 배열 크기를 알고 있어야 한다.

- 예) 다음은 3차원 배열을 형식 매개변수로 갖는 함수의 헤더이다.

```
float doIt (int bigAry[][12][5])
```

- 예) 다음은 4차원 배열을 형식 매개변수로 갖는 함수의 헤더이다.

- ◆ void func1(int a[][3][4][8]);

- 예) 다음의 함수 헤더는 컴파일 에러이다.

- ◆ int func2(int a[][][]);

이 곳에 배열 크기를 써줘야 한다.



# Passing an array to a function

- ◆ 다음은 배열 원소의 값을 2배 하여 저장하는 multiply()함수를 사용한 프로그램이다.

```
1  /* Read from keyboard & print integers multiplied by 2.
2      Written by:
3      Date:
4  */
5  #include <stdio.h>
6  #define SIZE  5
7
8  // Function Declarations
9  void multiply (int* pAry, int size);
10
11 int main (void)
12 {
13     // Local Declarations
14     int  ary [SIZE];
15     int* pLast;
16     int* pWalk;
17
18     // Statements
```

multiply()함수의 선언부  
→ 배열을 인자로 받는다.

# Passing an array to a function

```
19  pLast = ary + SIZE - 1;
20  for (pWalk = ary; pWalk <= pLast; pWalk++)
21  {
22      printf("Please enter an integer: ");
23      scanf ("%d", pWalk);
24  } // for
25
26  multiply (ary, SIZE);
27
28  printf ("Doubled value is: \n");
29  for (pWalk = ary; pWalk <= pLast; pWalk++)
30      printf (" %3d", *pWalk);
31
32  return 0;
33 } // main
34
```

pLast → 배열의 마지막 요소를 참조한다.

배열 초기화 → scanf()를 통해 입력 받는다.

multiply() 호출

출력

# Passing an array to a function

```
35  /* ===== multiply =====
36      Multiply elements in an array by 2
37      Pre   array has been filled
38      size indicates number of elements in array
39      Post  values in array doubled
40  */
41  void multiply (int* pAry, int size)
42  {
43      // Local Declarations
44      int* pWalk;
45      int* pLast;
46
47      // Statements
48      pLast = pAry + size - 1;
49      for (pWalk = pAry; pWalk <= pLast; pWalk++)
50          *pWalk = *pWalk * 2;
51      return;
52  } // multiply
53  // ===== End of Program =====
```

배열과 배열크기를 인수로 받는다.

배열 원소 값 \* 2

# Passing an array to a function

- ◆ 다음은 multiply() 함수를 사용한 프로그램의 구조이다.

