C언어 (CSE2035)
(Chap9. Pointer Applications) (1-2)

Sungwon Jung, Ph.D.

Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

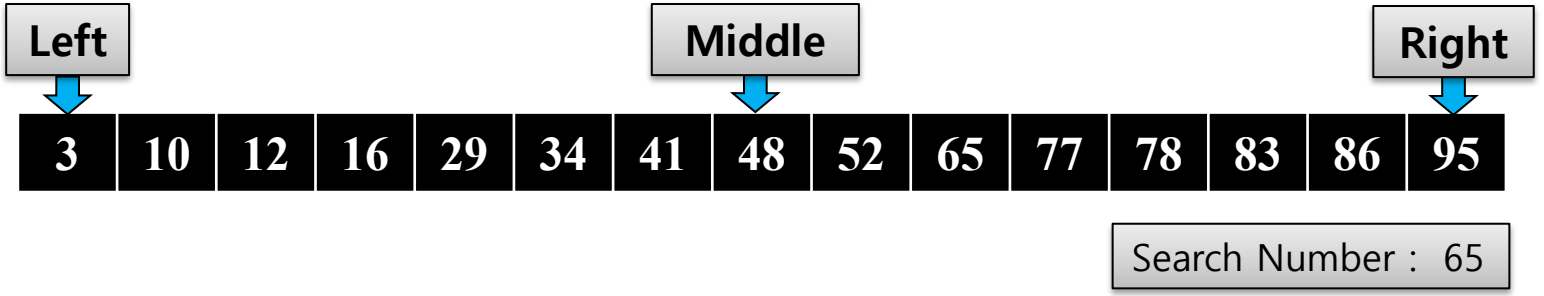
Pointer arithmetic and arrays

- **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

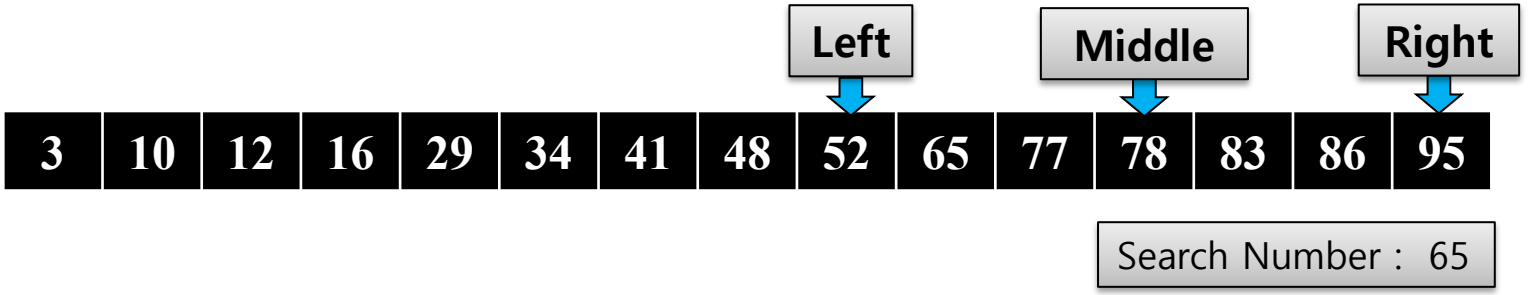
Type of Left Operand	Operator	Type of Right Operand	Type of result
Pointer	+	Pointer	Error
Pointer	-	Pointer	Integer(offset)
Pointer	+	Int	Address
Pointer	-	Int	Address

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.

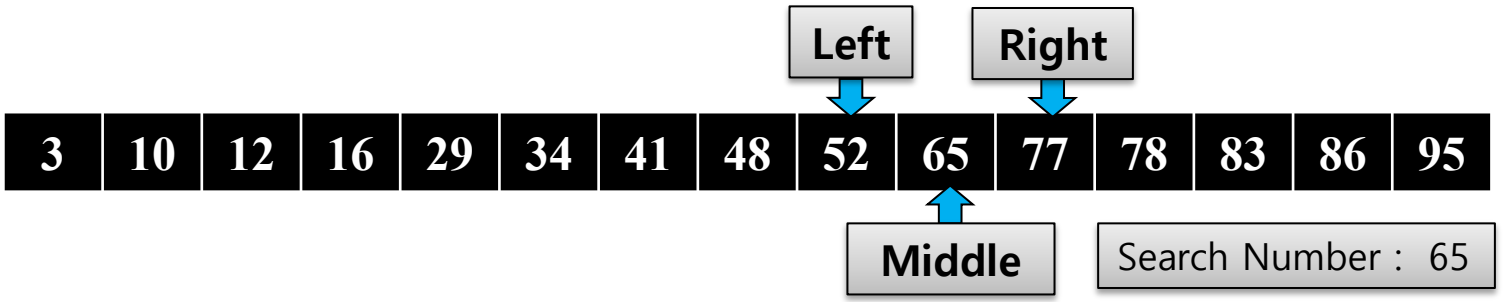


  - Middle 위치의 값과 Search Number를 비교하여 Middle 위치의 값이 작을 경우에는 Left의 위치를 Middle 위치보다 1 크게 잡는다.

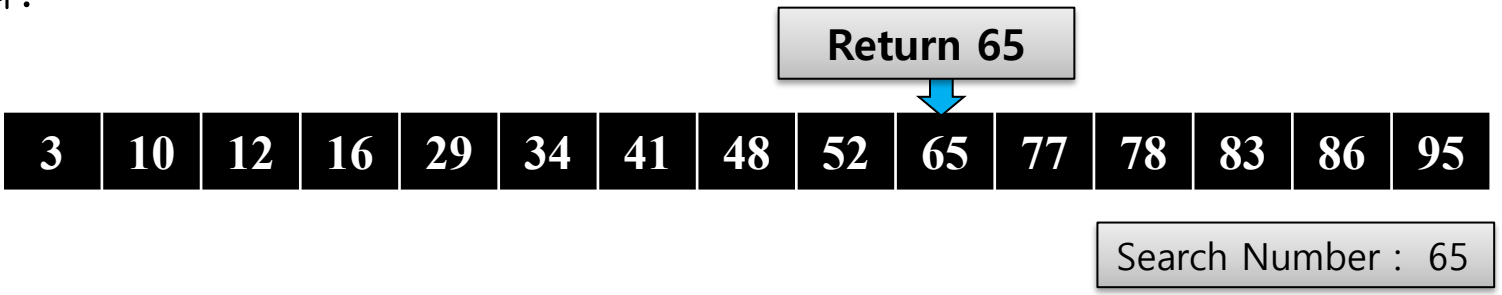


Pointer arithmetic and arrays

- Search Number가 Middle에 위치한 값 보다 작을 경우에는 Right의 위치를 Middle 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.
  - Middle = (Left + 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.
  - midPtr = firstPtr + (LastPtr – FirstPtr) / 2;

- 위 표현식에서 피연산자들의 자료형은 다음과 같은 순서로 계산된다

Address = Address + (offset) / 2

= Address+ (int) / int

= Address + 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
- **endPtr**은 현재 list배열의 마지막 원소를 참조
- **target**은 list에서 찾아야 할 값
- **locnPtr**은 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

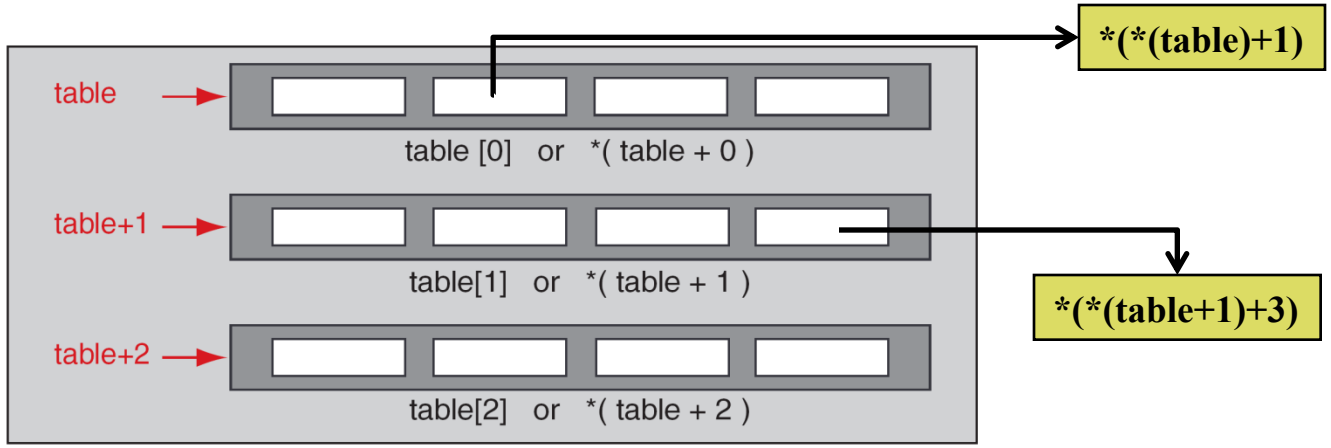
list의 중간 이전 부분일 경우

target == *midPtr일 경우

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

Pointer arithmetic and arrays

- **Pointers And Two-Dimensional Arrays**
  - 이차원 배열 table의 원소 table[1][3]을 포인터를 사용하여 접근한다면 다음과 같은 표현이 가능하다 ➔ *(*(table +i) + j)



int table[3][4] ;

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

Print Table