전산 SMP 6주차

2014. 11. 02

김범수
bskim45@gmail.com

Special thanks to 박기석 (kisuk0521@gmail.com)

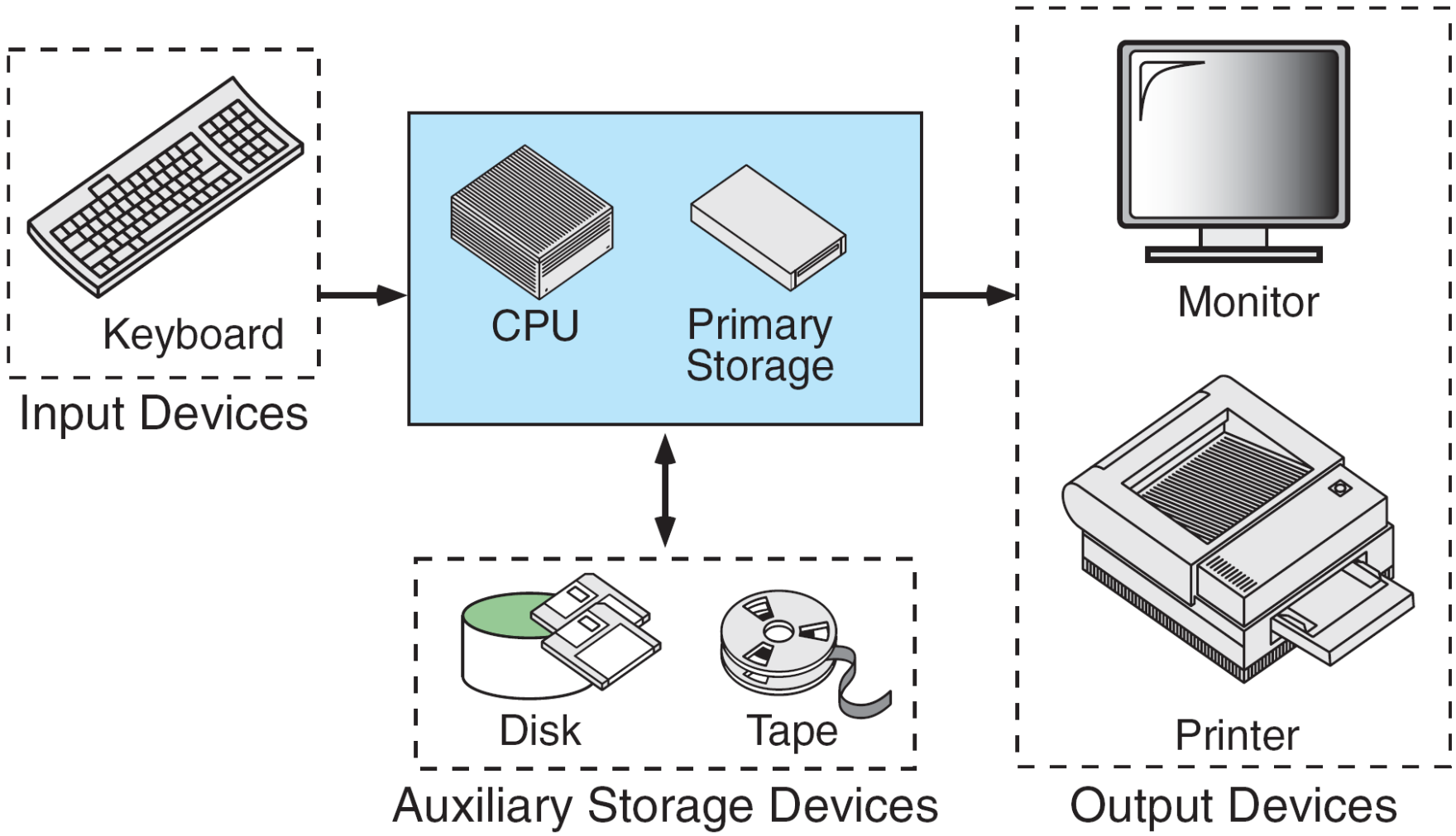
들어가기 전에

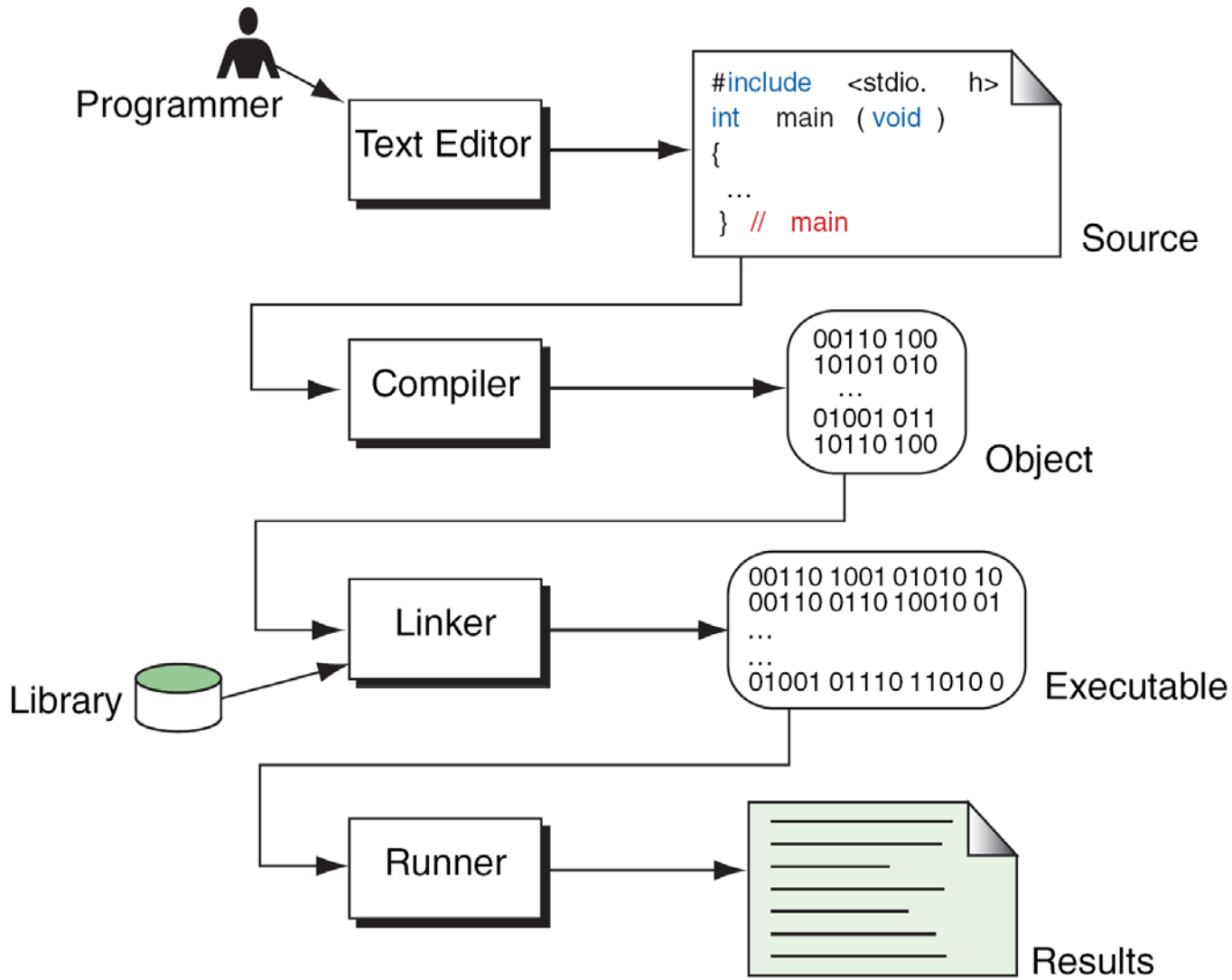
- 중간고사 어땠나요? 점수는?

- Welcome to Array & Pointer

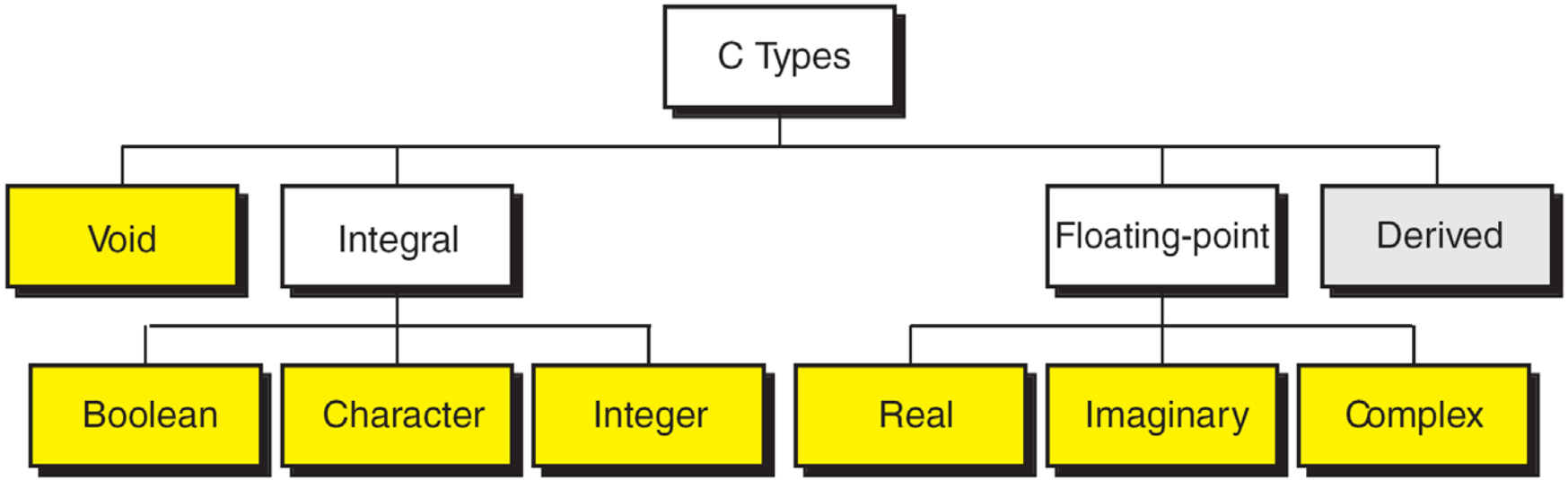
지난 내용 복습

~중간고사


Keyboard
Input Devices
CPU
Primary Storage
Disk
Tape
Auxiliary Storage Devices
Monitor
Printer
Output Devices


Programmer
Text Editor
#include <stdio. h>
int main (void)
{
…
} // main
Source
Compiler
00110 100
10101 010
…
01001 011
10110 100
Object
Linker
00110 1001 01010 10
00110 0110 10010 01
…
…
01001 01110 11010 0
Executable
Library
Runner
Results

자료형과 형변환


C Types
Void
Integral
Floating-point
Derived
Boolean
Character
Integer
Real
Imaginary
Complex

연산자, 우선순위, 결합순서

- 산술연산자 : +,−, *, /, %
- 대입연산자 : =, +=, −=, *=, /=, %=, 〈〈=, 〉〉=,
  &=, ^=, |=
- 관계(비교)연산자 : 〉, 〈, 〉=, 〈=, ,!=
- 논리연산자 : &&, ||, !
- 증가/감소 연산자 : ++, −−
- 비트연산자 : &, |, ^, ~
- 시프트 연산자 : 〈〈, 〉〉
- 주소 연산자 : &

조건문 (Selection Statement)

- 조건 (Condition)에 따라서 선택적으로 프로그램을 진행
- 프로그램의 Flow를 조종

2-Way Selection
  - if~else

Multi-Way Selection
  - if~else if~else, switch

- 조건문 안에 얼마든지 또 조건문을 넣을 수 있다. (nested)

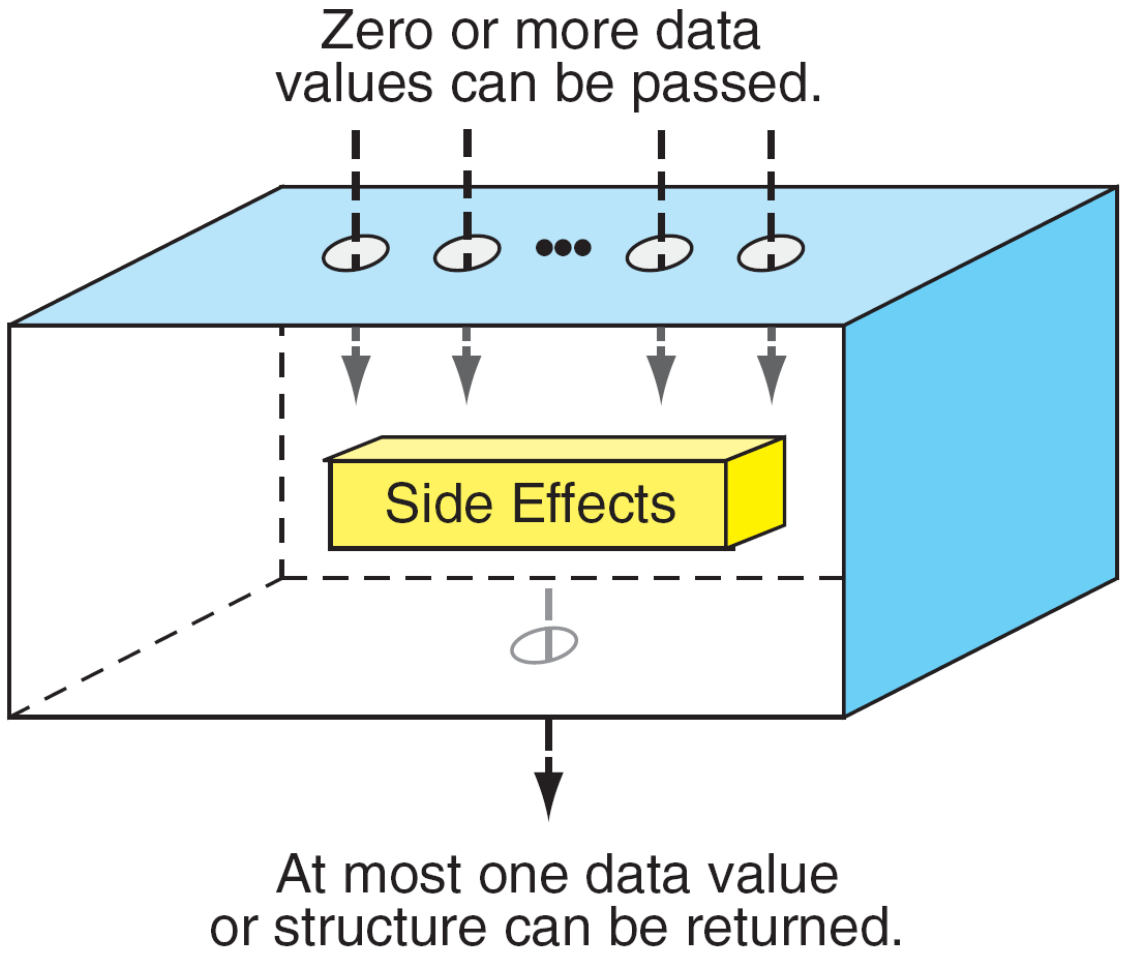
Loop (고리)

- pre-test loop: 검사하고 실행하기
  while 문, for 문
- post-test loop: 실행하고 검사하기
  do~while 문

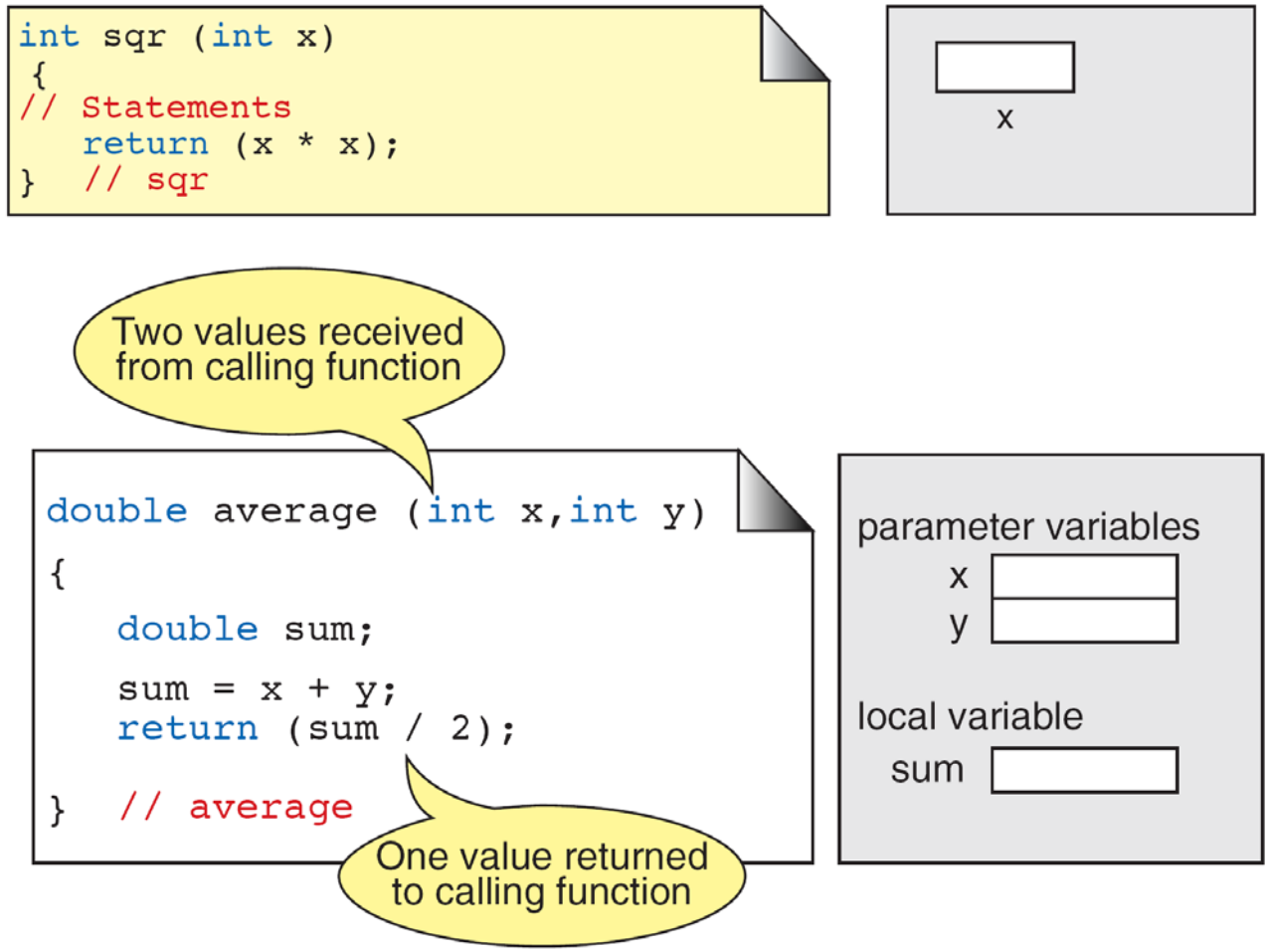
- Requirement
  - Initialization
  - Update
  - Condition Check

- 언제 무엇을 쓰나요 – “주로”
  for는 반복 횟수를 알 때, while, do~while은 모를 때

함수


Zero or more data
values can be passed.
Side Effects
At most one data value
or structure can be returned.

메모리 공간은 함수마다 따로따로

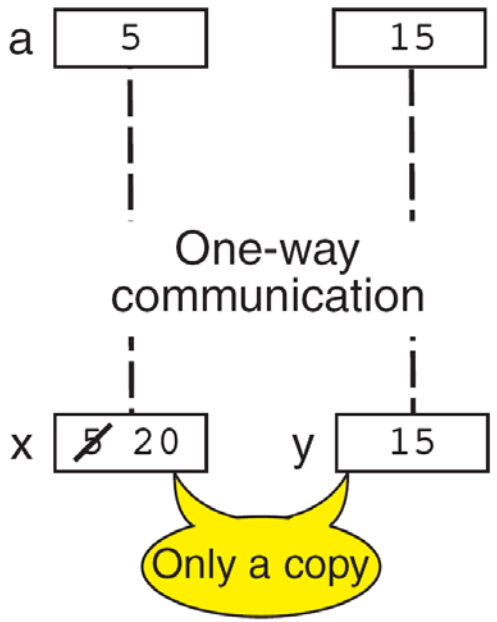


Call-by-Value

```c
// Function Declaration
void downFun (int x, int y);
int main (void)
{
// Local Definitions
   int a = 5;
// Statements
   downFun (a, 15);
   printf("%d\n", a);
   return 0;
}  // main
```

prints 5

```c
  void downFun (int x, int y)
  {
  // Statements
     x = x + y;
     return;
  }  // downFun
```



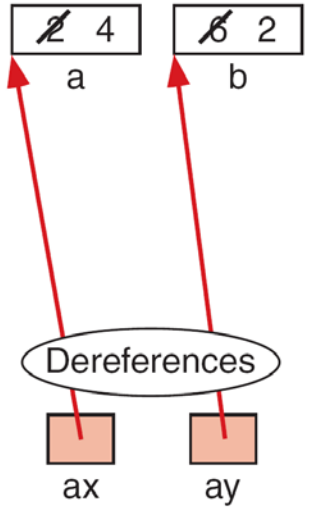
Call-by-Reference

```
// Function Declaration
void biFun (int* ax, int* ay);

int main (void)
{
// Local Definitions
   int  a = 2;
   int  b = 6;

// Statements
   …
   biFun (&a, &b);
   …
 return 0;
}  // main
```

```
void biFun (int* ax, int* ay)
{
   *ax = *ax + 2;
   *ay = *ay / *ax;
   return;
}  // biFun
```



SWAP

```
// Function Declarations
void exchange (int* num1, int* num2);

int main (void)
{
// Local Definitions
   int  a;
   int  b;

// Statements
   …
   exchange (&a, &b);
   …
 return 0;
}  // main
```

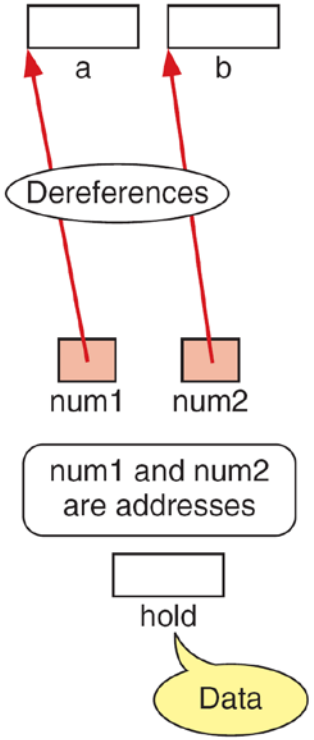
Note that the type includes an asterisk.

Address operators

```
void exchange (int* num1, int* num2)
{
// Local Definitions
   int  hold;

// Statements
   hold   =  *num1;
   *num1  =  *num2;
   *num2  =   hold;
   return;
}  // exchange
```

Note the indirection operator is used for dereferencing.



Recursion

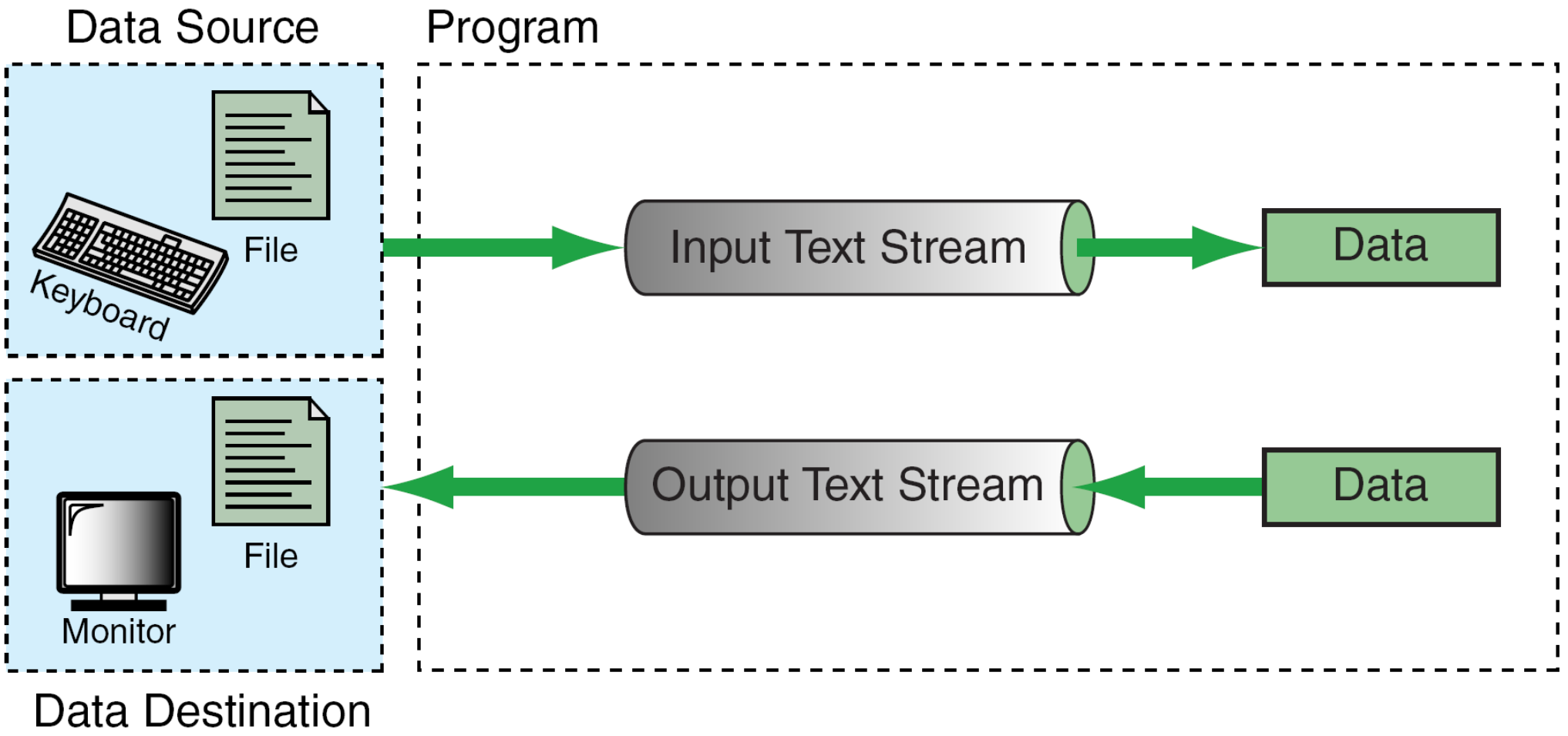
필수조건

- Base Case (n=1)
- n=k → n=k+1?

왜 쓰나요?

- 장점: 문제를 단순하게 풀 수 있다
- 단점: 메모리 소모가 많다

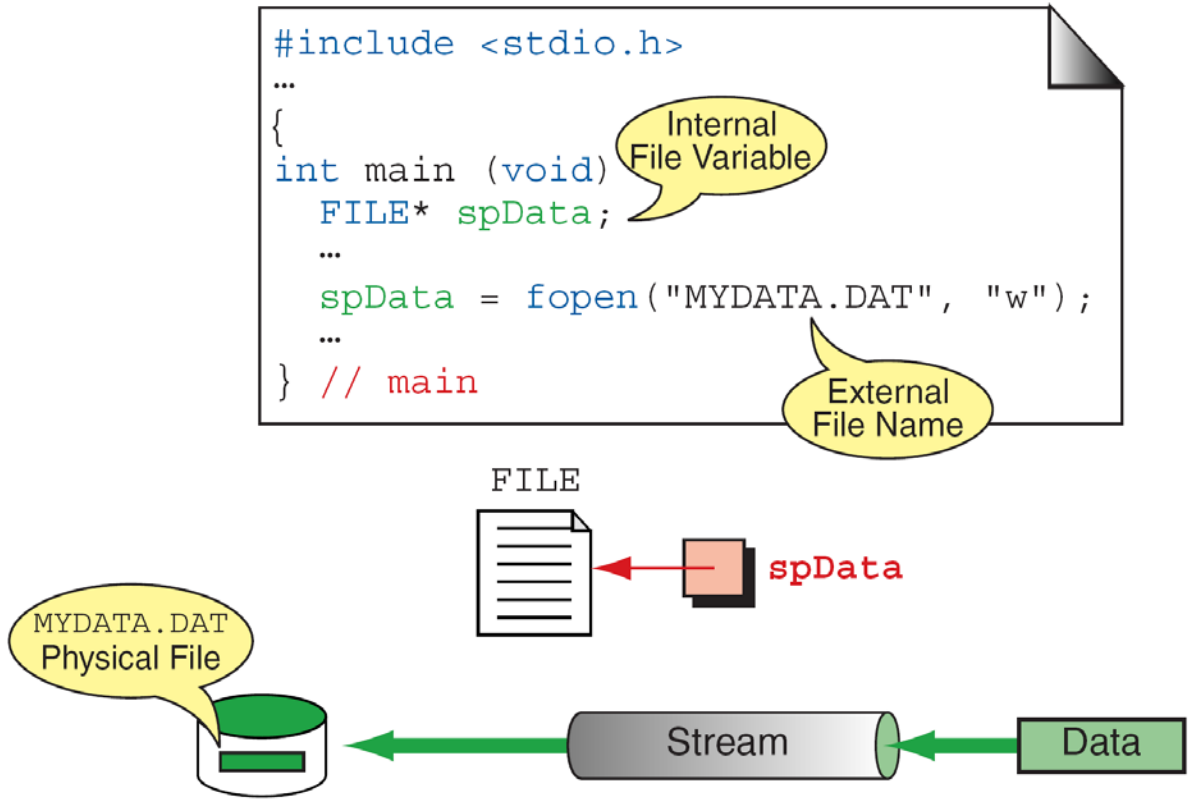
I/O


Data Source
Program
File
Keyboard
Input Text Stream
Data
Output Text Stream
Data
File
Monitor
Data Destination

FILE I/O

- 콘솔(console) 화면과의 소통이 아닌 파일(.txt .c etc) 과 소통하기 위한 Steam

```
FILE *infile;
infile = fopen(“anyfile.txt”, “w”);
~~
fclose(infile);
```



대표적 FILE I/O 관련 함수들

int fprintf(FILE * out, const char * format, …)

Filestream, out 으로 부터 format의 형태로 출력한다.

비교)int printf(const char * format, …)

int fscanf(FILE * in, const char * format, …)

File stream, in 으로 부터 format의 형태로 입력받는다.

비교)int scanf(const char * format, …)

참고 : http://weezzle.net/1611

FLUSH

- 스트림 버퍼를 비우는 역할을 한다.
- Scanf 등 입력받는 함수 사용 시 주의!! 할 점!!
  - 근접한 scanf 두번 사이에는… 뭔가… 언짢은 일들이…

Test)
Int a; char b;
Scanf(“%d”, &a);
Scanf(“%c”,&b);
Printf(“%d %c\n”, a, b);

fflush();
#define FLUSH while(getchar != ‘\n’)

자료구조

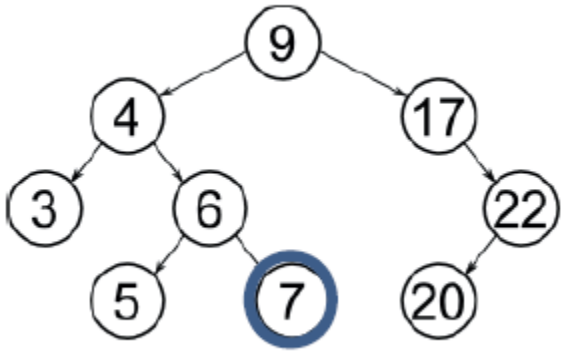
- 많은 데이터를 예쁘게 잘 저장해서
- 쉽고 빠르게 꺼내 쓰려고
- 데이터가 많아지면 찾는데도 오래 걸린다.

- Example: finding a number from a set of numbers
  - How many comparisons do we need to retrieve 7?

In linear array

8 comparisons

In binary search tree



4 comparisons

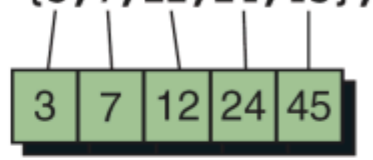
선언, 초기화, 접근

(a) Basic Initialization

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

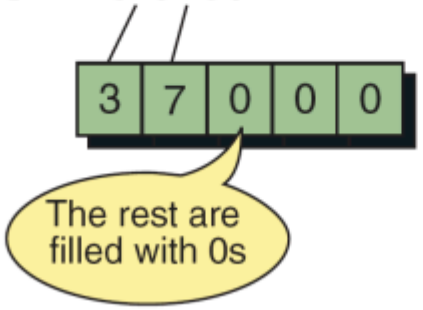
(b) Initialization without Size

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



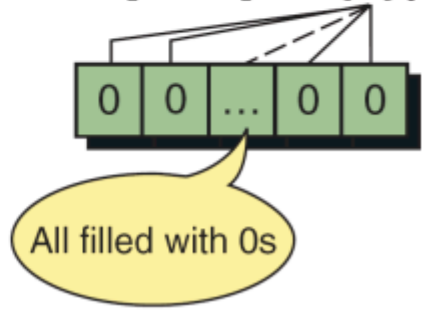
(c) Partial Initialization

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



(d) Initialization to All Zeros

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

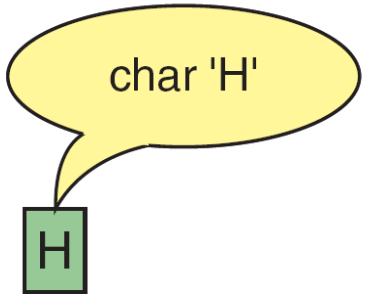


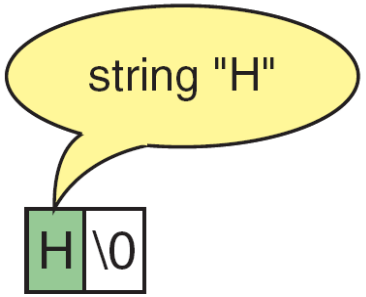
오늘 할 것

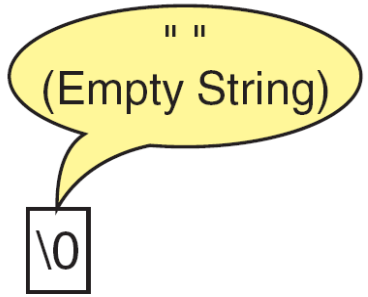
- String
- Multi-dimensional Array
- Sorting
- Search

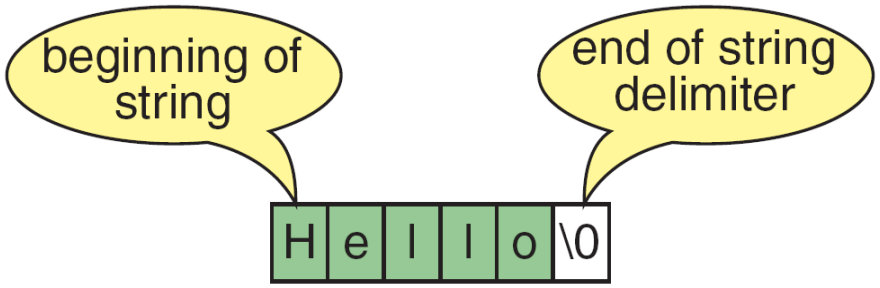
문자열(string)도 결국은 배열이다

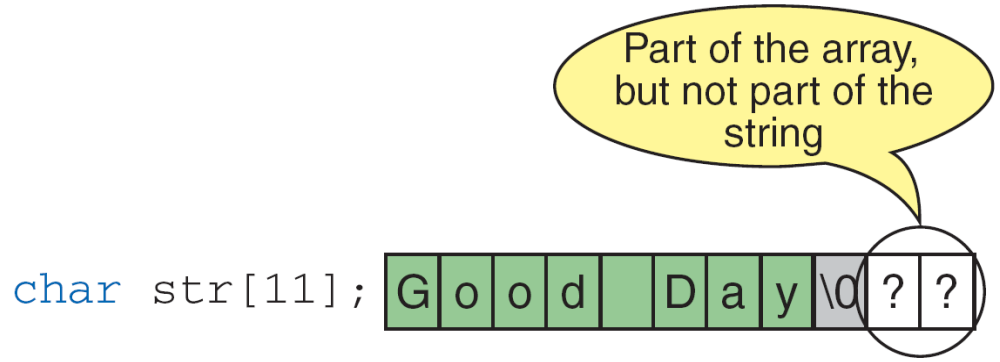
- char 형의 배열 & 끝에 널문자 '\0'
- 배열 & 포인터를 완전하게 배우고 string에 대해서 더 자세히 합니다.





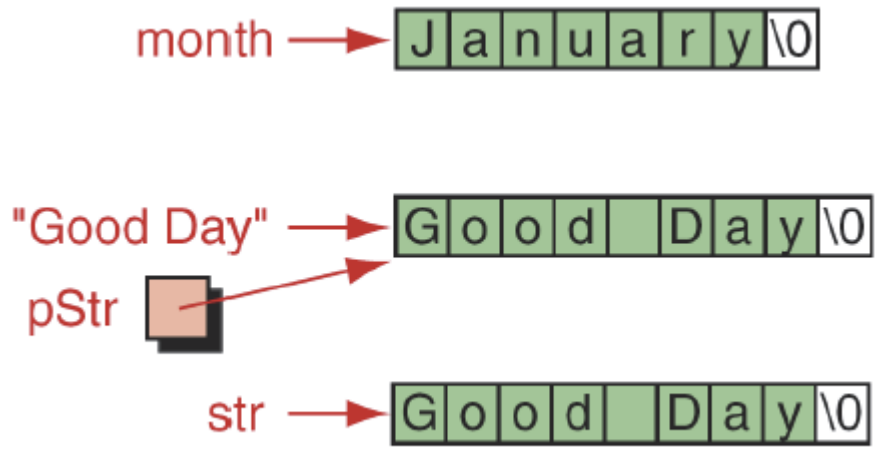






문자열의 선언

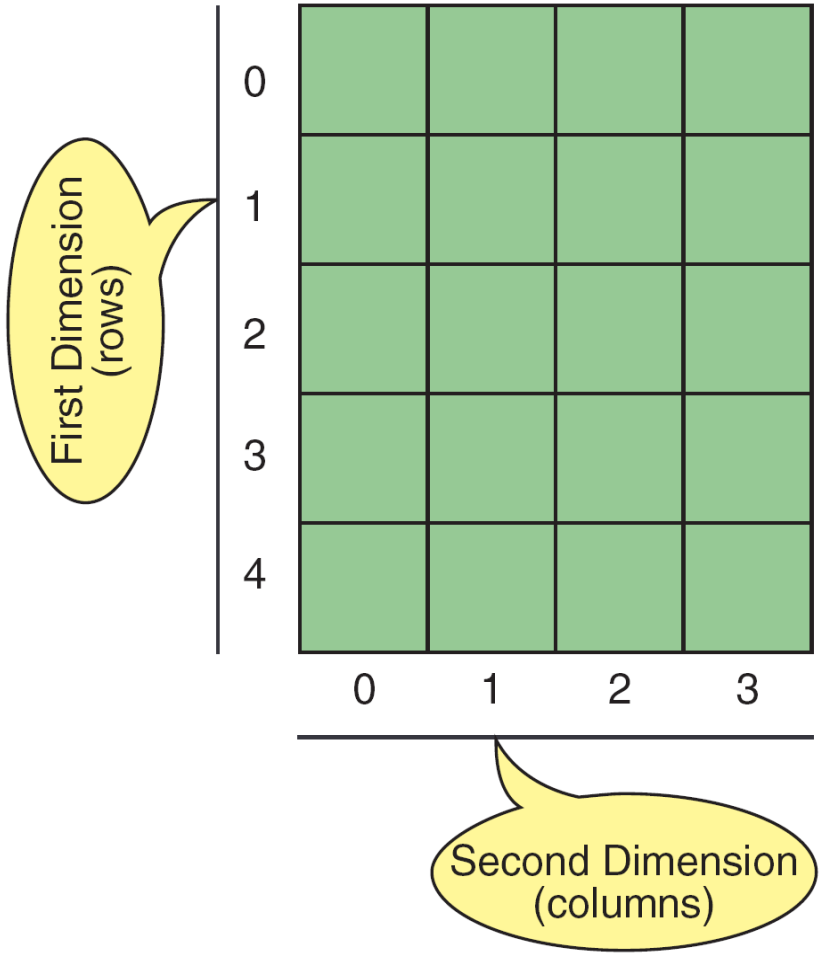
- char str[9] = “Good Day”;
- char month[] = “January”;
- char *pStr = “Good Day”;
- char str[9] = {‘G’, ‘o’, ‘o’, ‘d’, ‘ ‘, ‘D’, ‘a’, ‘y’, ‘\0’};

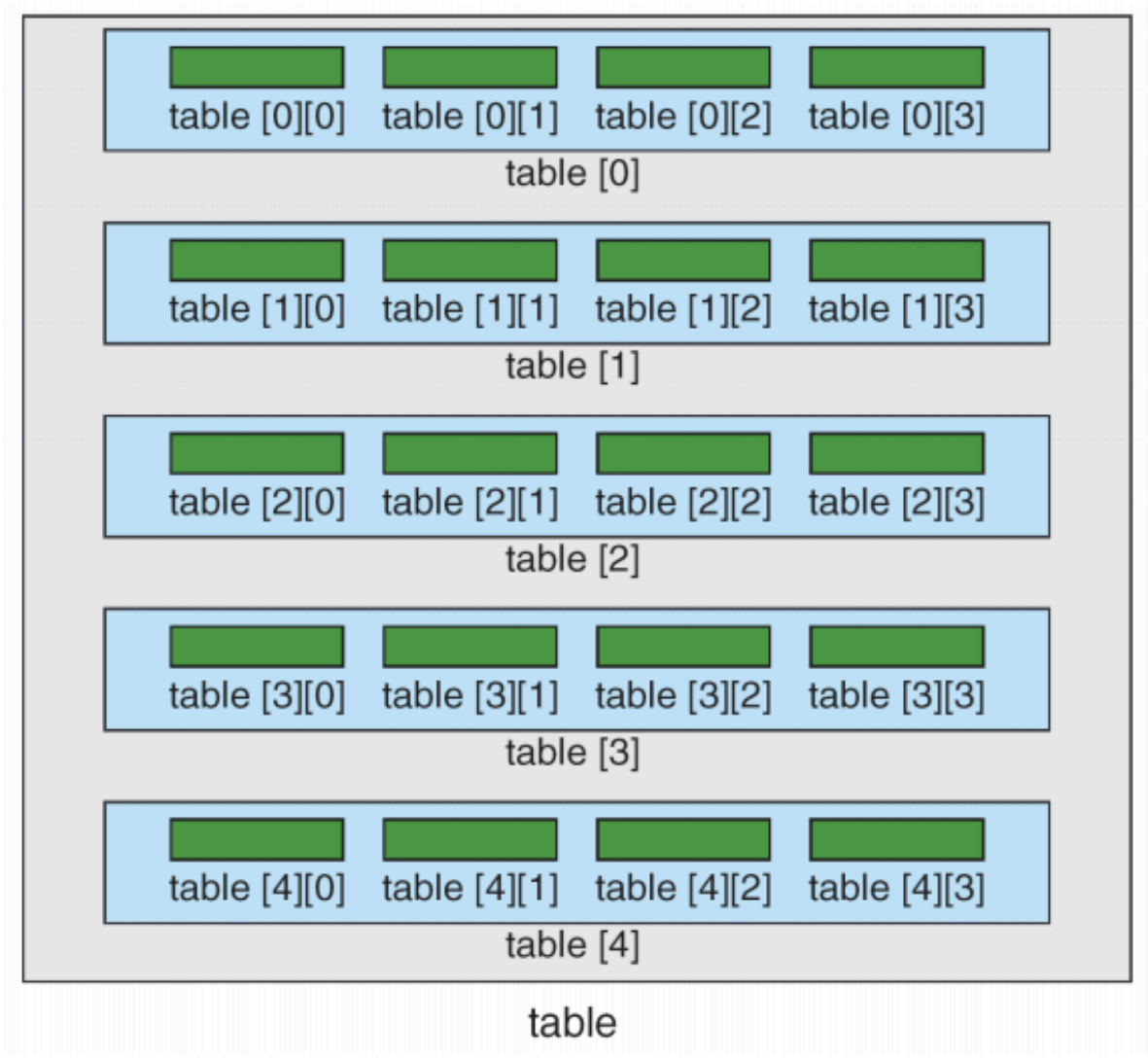


Usage

- char str[100];
- printf(“Input your povis id: “);
- scanf(“%s”, str);
- printf(“%s”, str);

Two-dimensional Array (Matrix)

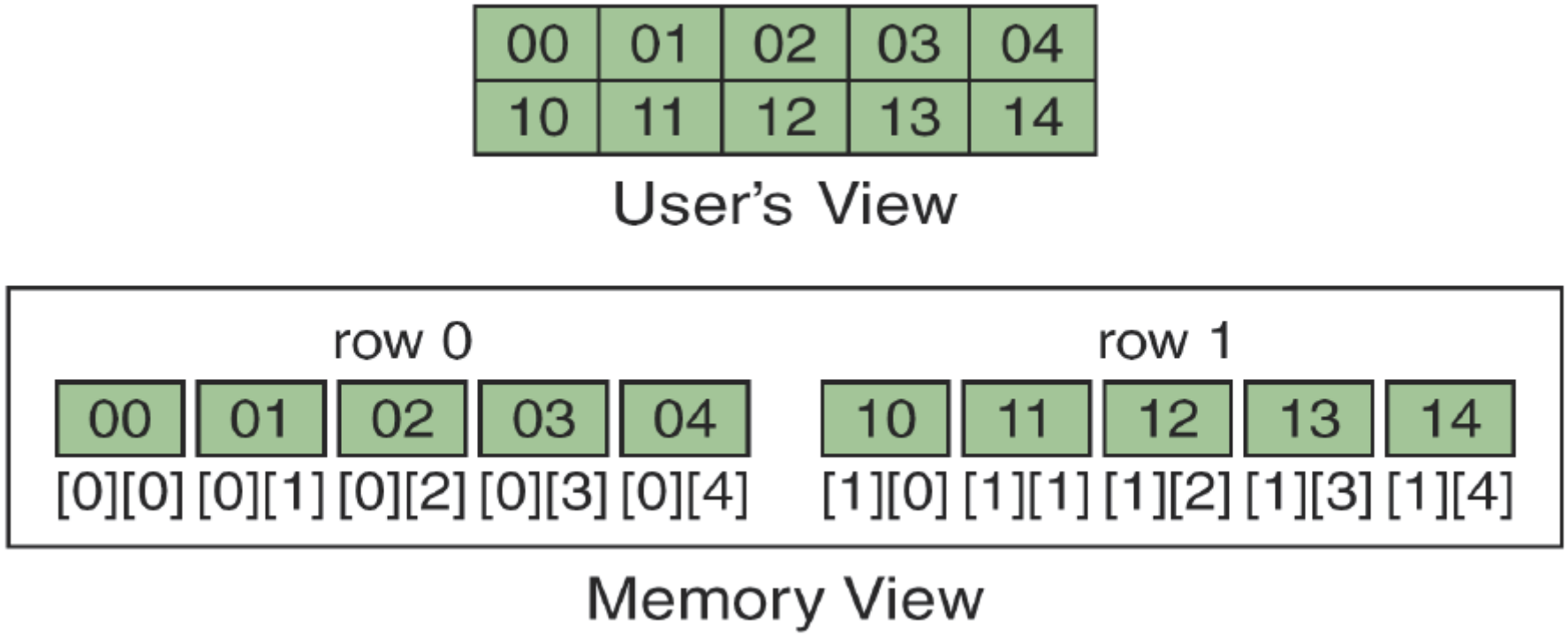




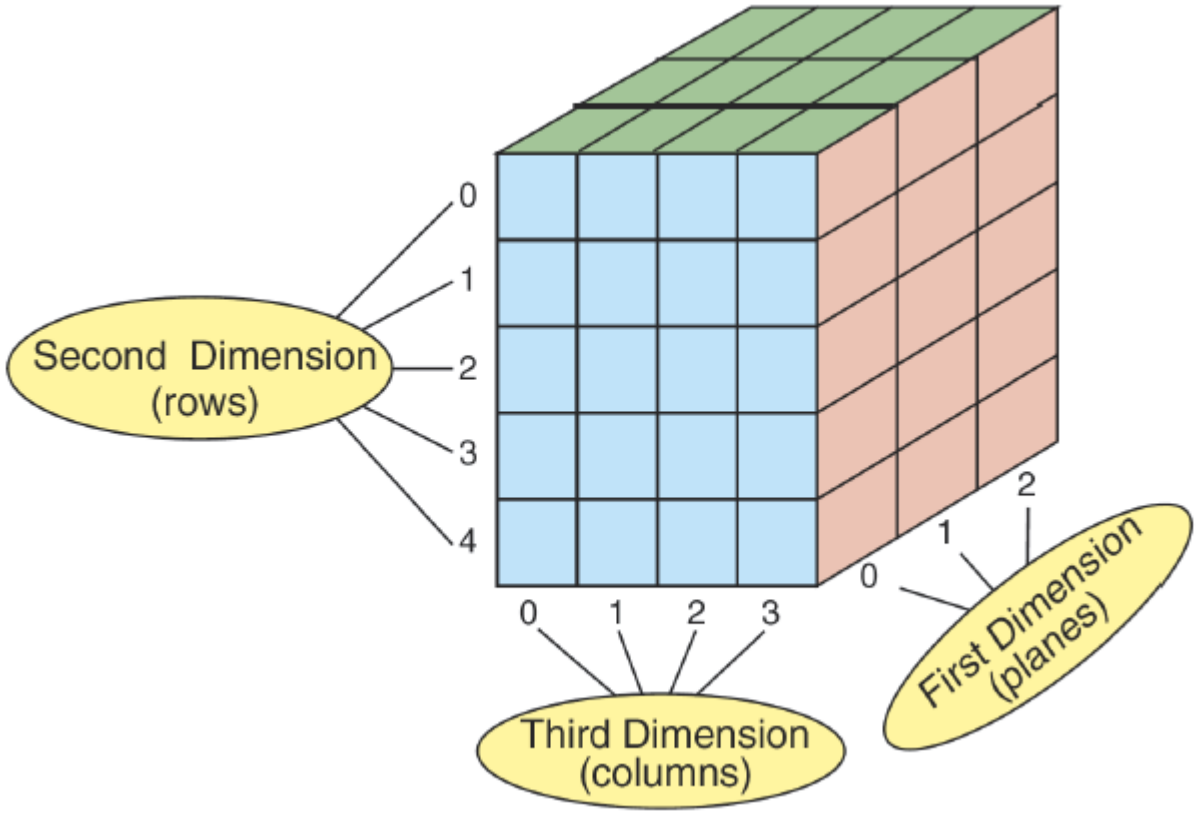
Declaration

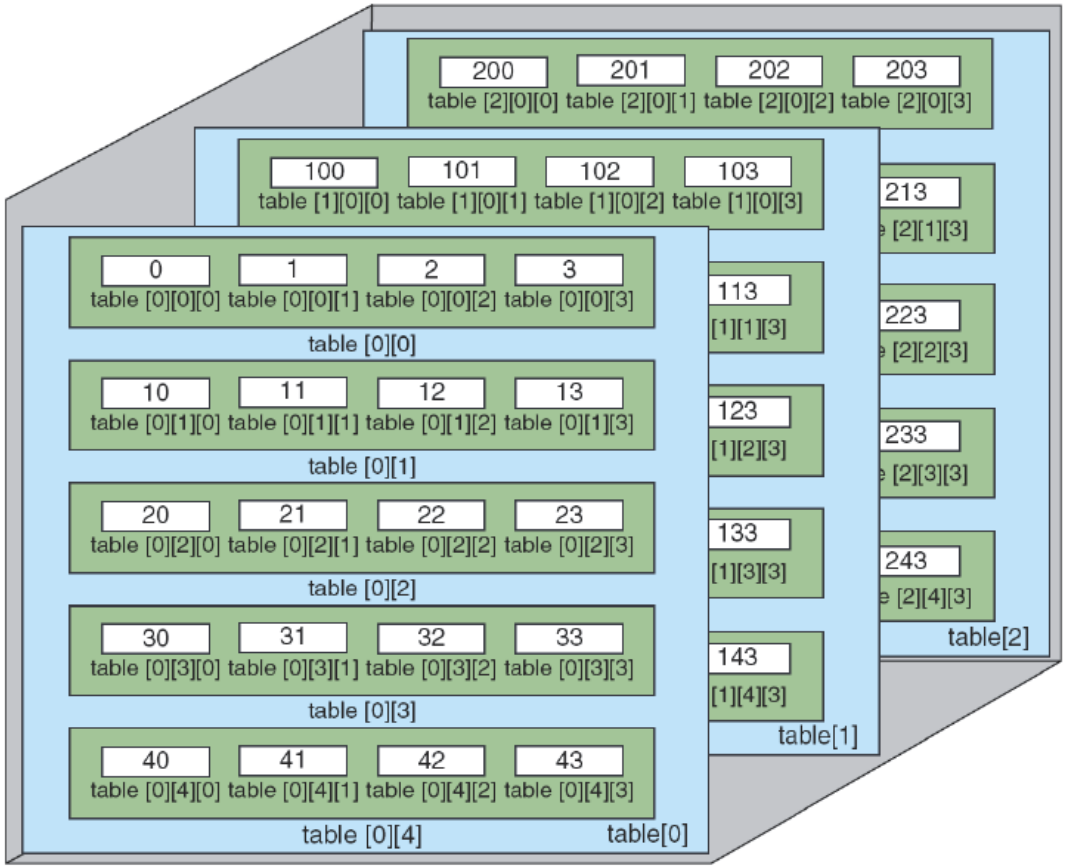
- int table[5][4];
- int table[3][2] = {0, 1, 2, 3, 4, 5};
- int table[3][2] = { {0, 1}, {2, 3}, {4, 5} };
- int table[][2] = {{0, 1}, {2, 3}, {4, 5} };
- int table[3][2] = {0};
- table[r][w]
- &table[r][w]

Memory Layout



Multi-dimension array





Declaration

```
int arr[2][3][4] = { { {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4} },
                     { {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4} } };

int array[][][]={ { {0,1,2}, {3,4,5}, {6,7,8} },
                  { {9,10,11}, {12,13,14}, {15,16,17} },
                  { {18,19,20}, {21,22,23}, {24,25,26} } };
```

Sorting

왠 갑자기 Sorting

- Array는 자료구조
- 우리가 원하는 데이터를 Array에서 빠르게 찾으려면?
- 저장할 때 잘 저장해서 꺼내 쓸 때 편해야 한다

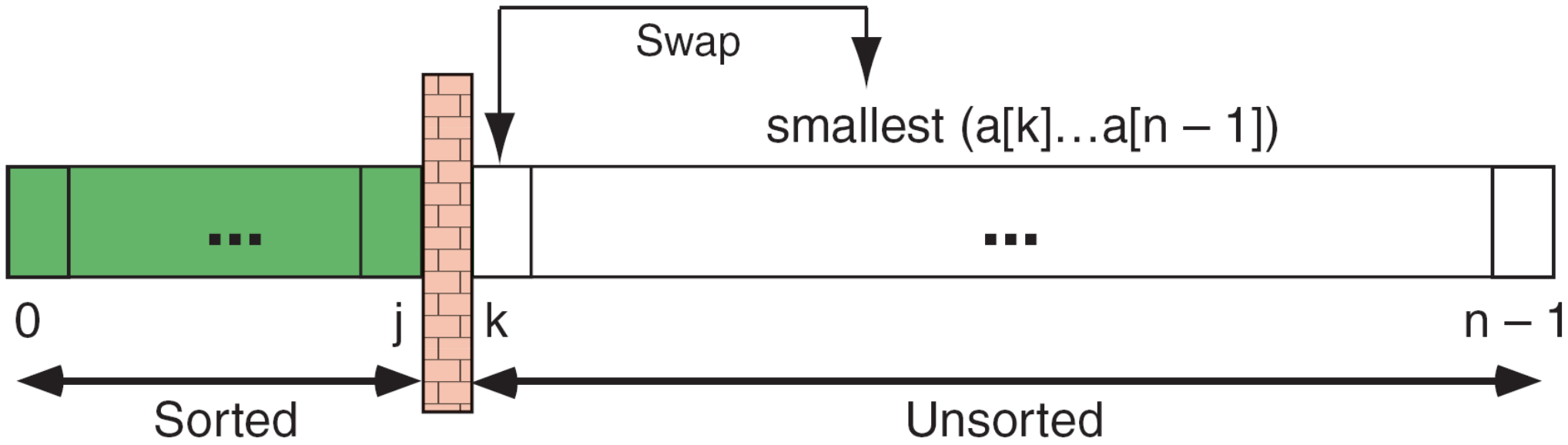
- 데이터를 효율적으로 관리하는 방법

  데이터를 저장하는 자료구조 + 데이터를 찾는 알고리즘

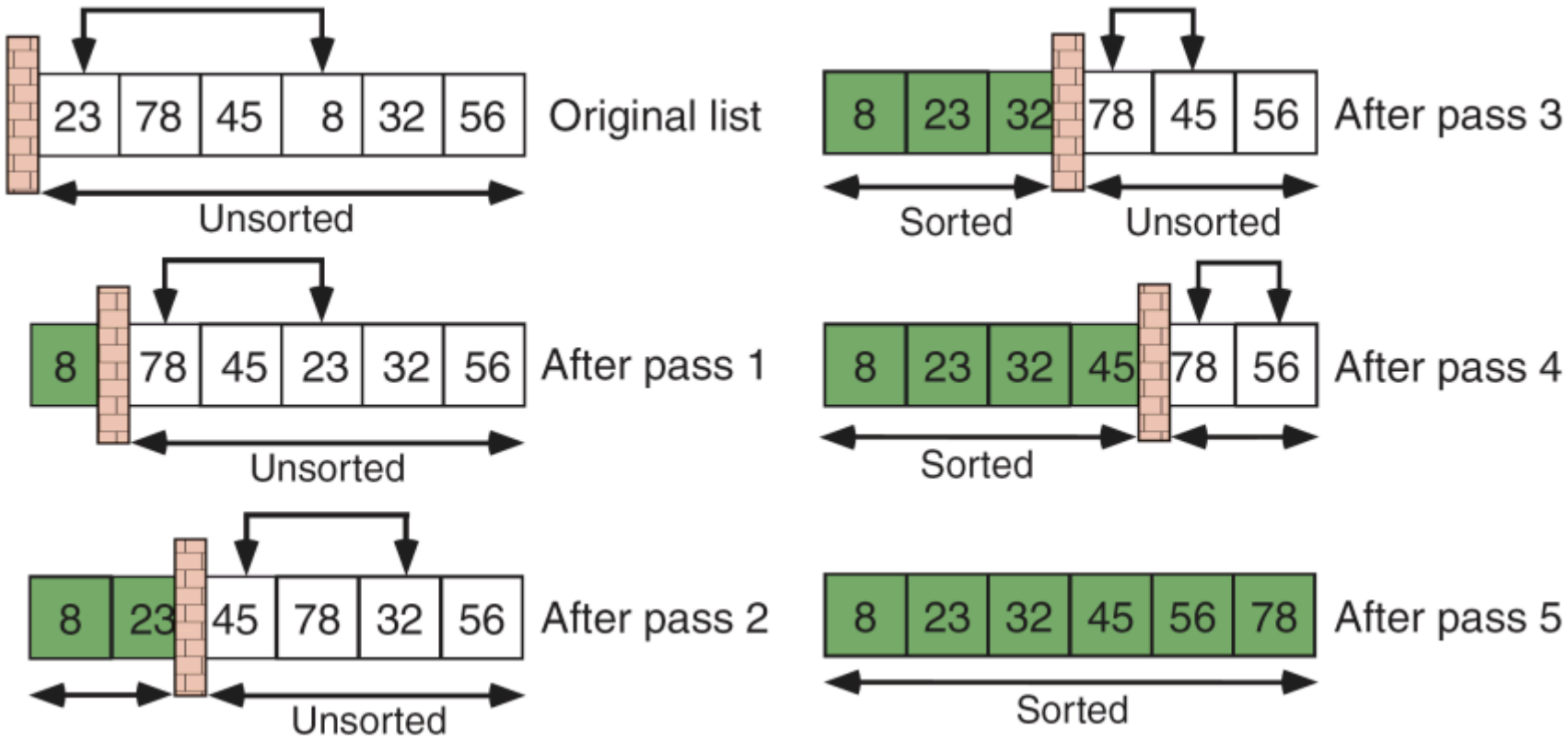
Array Sorting

- Selection Sort
- Bubble Sort
- Insertion Sort

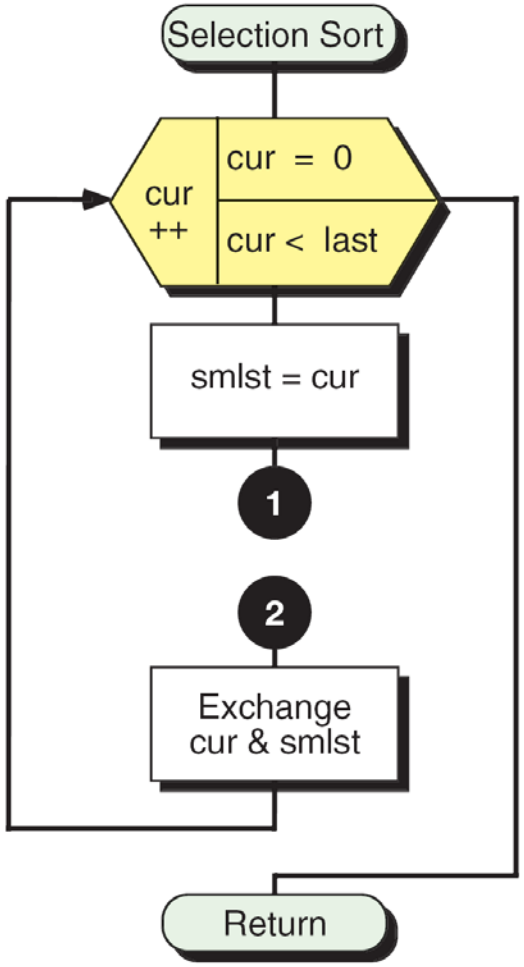
Selection Sort Concept

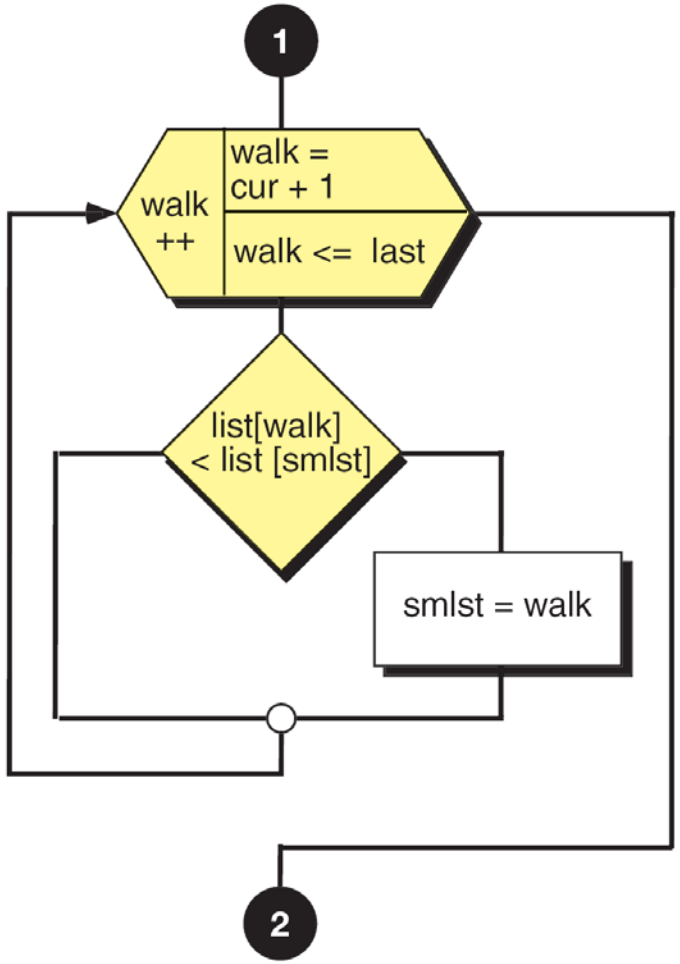


Selection Sort



Algorithm


Selection Sort
cur
++
cur = 0
cur < last
smlst = cur
1
2
Exchange
cur & smlst
Return


1
walk
++
walk =
cur + 1
walk <= last
list[walk]
< list [smlst]
smlst = walk
2

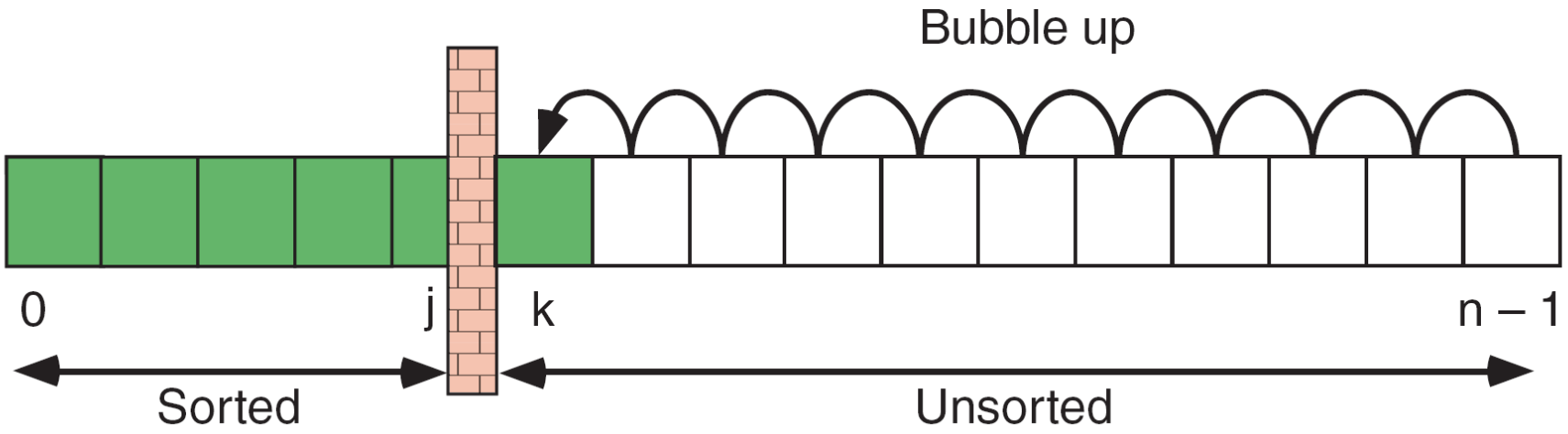
Code

```
 1  /* ================== selectionSort ====================
 2     Sorts by selecting smallest element in unsorted
 3     portion of array and exchanging it with element at
 4     the beginning of the unsorted list.
 5        Pre   list must contain at least one item
 6              last contains index to last element in list
 7        Post  list rearranged smallest to largest
 8  */
 9  void selectionSort (int list[], int last)
10  {
11  // Local Declarations
12     int smallest;
13     int tempData;
14
15  // Statements
16     // Outer Loop
17     for (int current = 0; current < last; current++)
18         {
19          smallest = current;
```

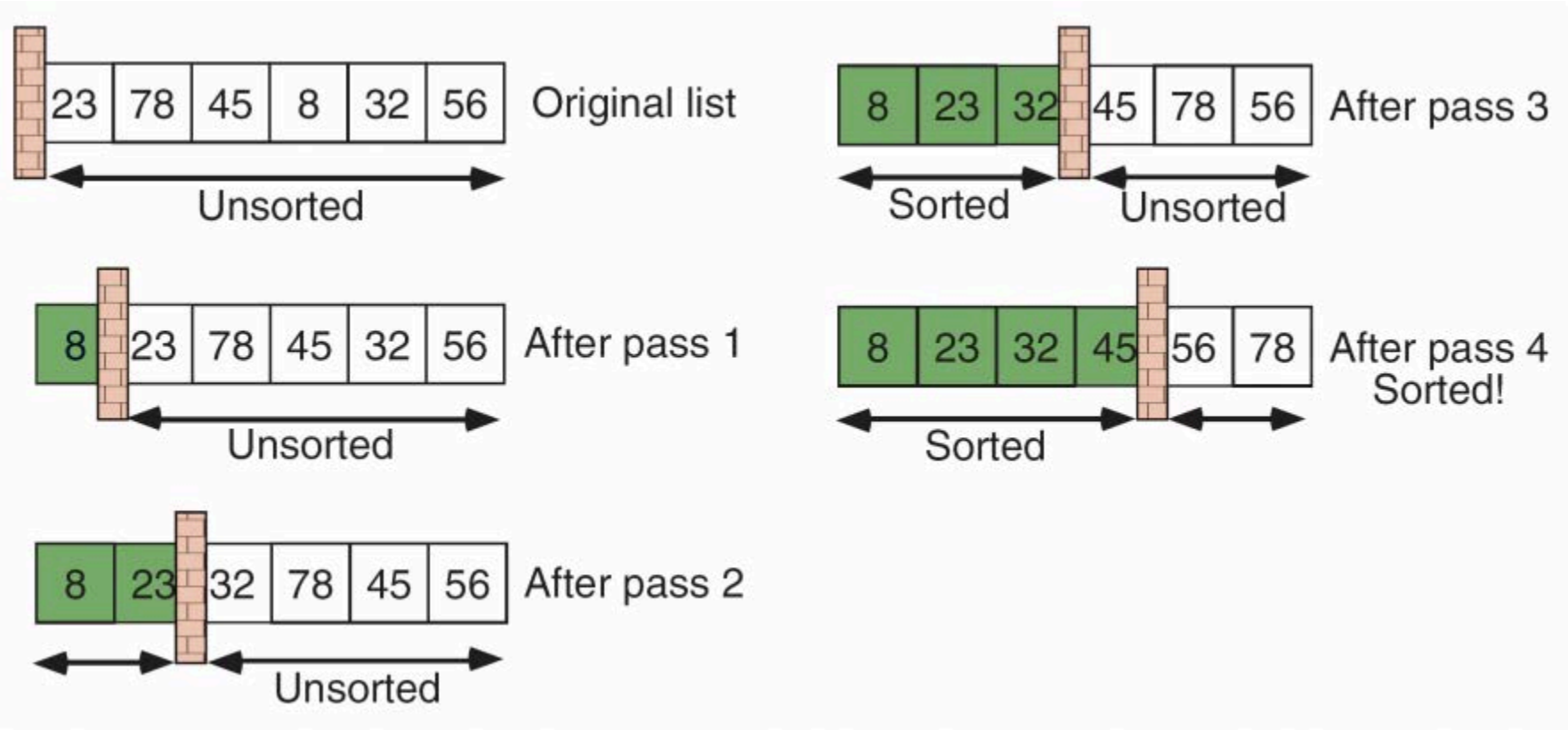
Code #2

```
20          // Inner Loop: One sort pass each loop
21          for (int walk = current + 1;
22                   walk <= last;
23                   walk++)
24              if (list[walk] < list[smallest])
25                  smallest = walk;
26          // Smallest selected: exchange with current
27          tempData        = list[current];
28          list[current]   = list[smallest];
29          list[smallest] = tempData;
30          } // for current
31      return;
32  }  // selectionSort
```

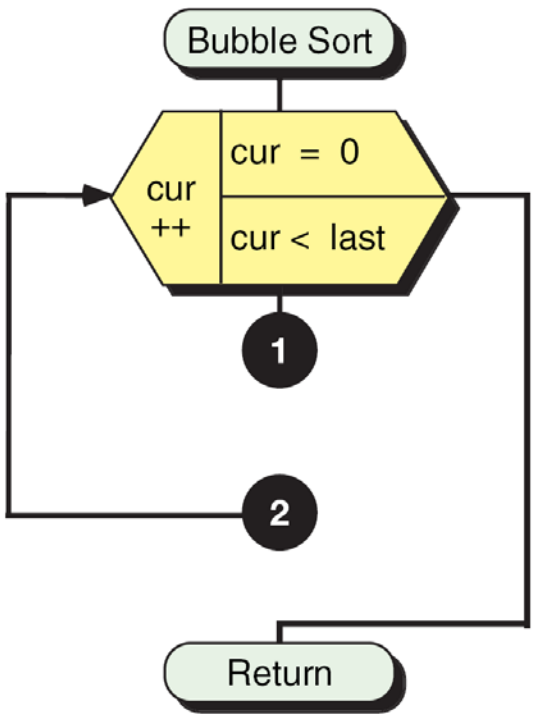
Bubble Sort Concept

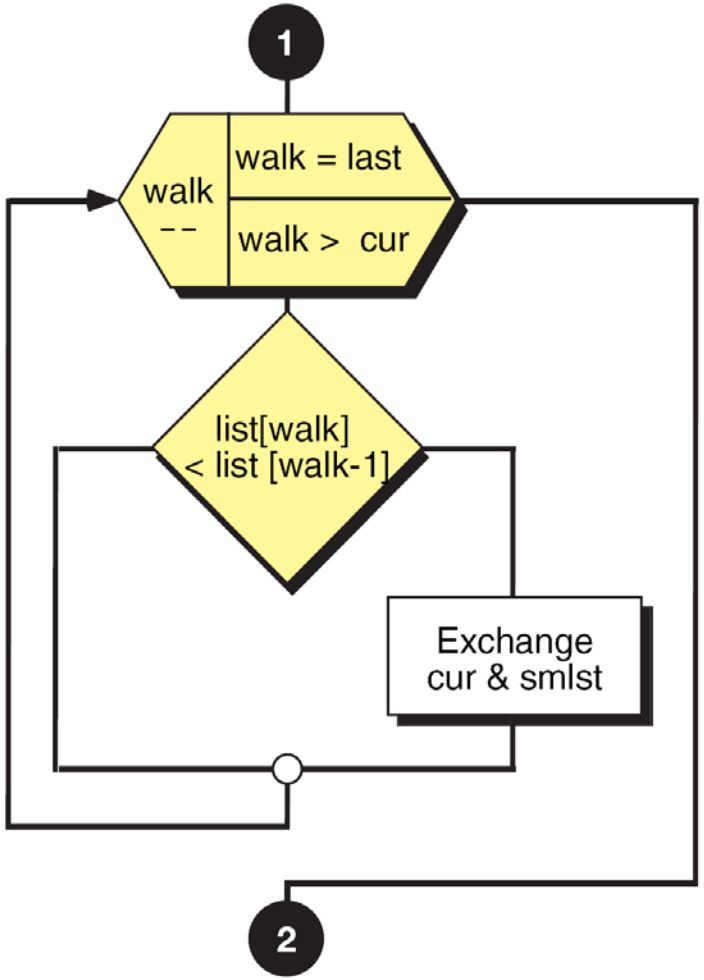


Bubble Sort Example



Bubble Sort Design





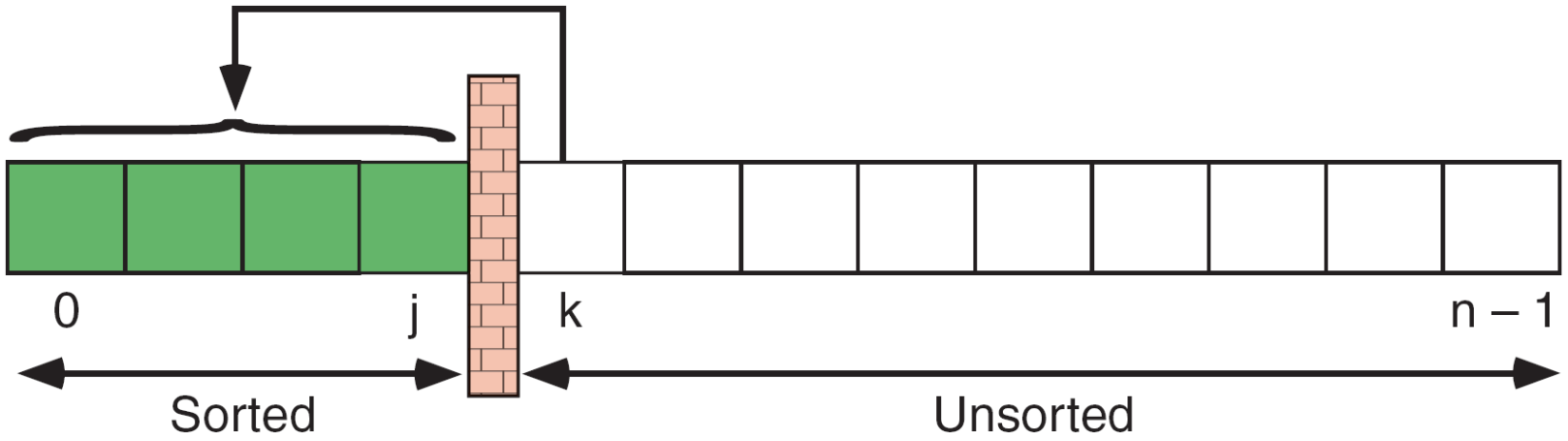
Code

```
1  /* ==================== bubbleSort ====================
2     Sort list using bubble sort. Adjacent elements are
3     compared and exchanged until list is ordered.
4        Pre  the list must contain at least one item
5             last contains index to last element in list
6        Post list rearranged in sequence low to high
7  */
8  void bubbleSort (int list [], int last)
9  {
10 // Local Declarations
11    int temp;
12
13 // Statements
14    // Outer loop
15    for(int current = 0; current < last; current++)
16       {
17        // Inner loop: Bubble up one element each pass
```

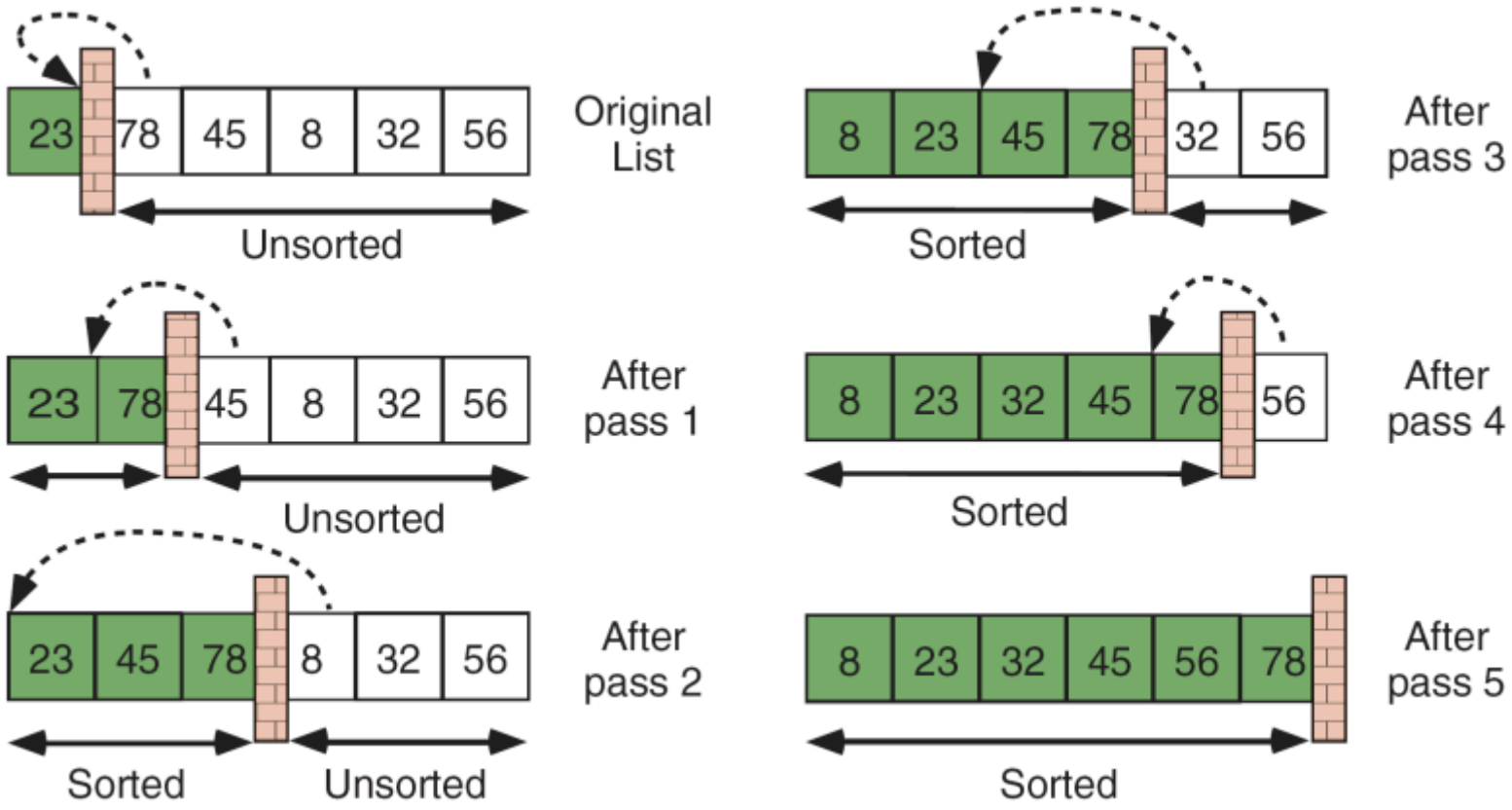
Code #2

```
18          for (int walker = last;
19                   walker > current;
20                   walker--)
21              if (list[walker]  < list[walker - 1])
22                 {
23                  temp               = list[walker];
24                  list[walker]       = list[walker - 1];
25                  list[walker - 1] = temp;
26                 } // if
27          } // for current
28      return;
29  }  // bubbleSort
```

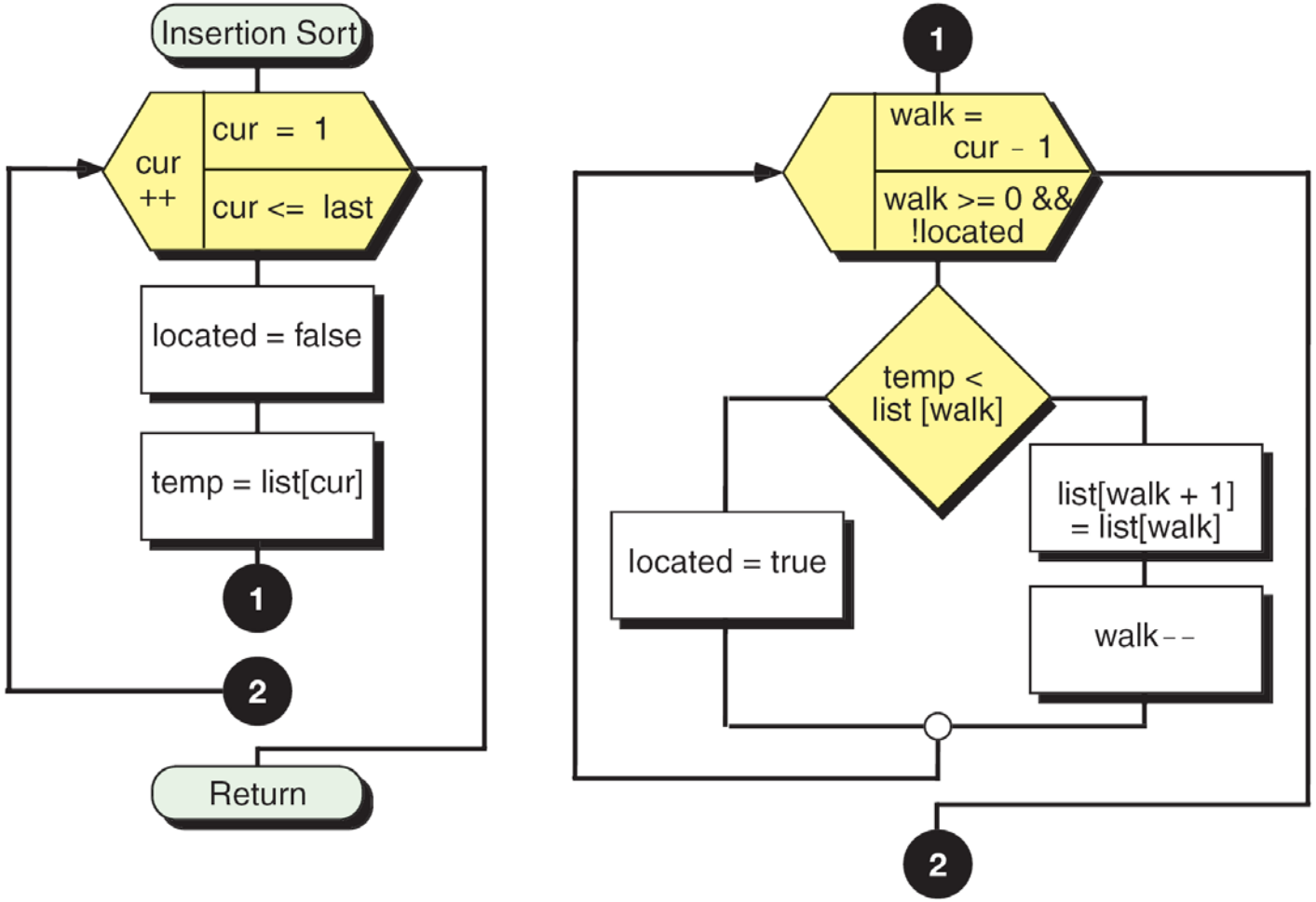
Insertion Sort Concept



Insertion Sort Example



Insertion Sort Design



Code

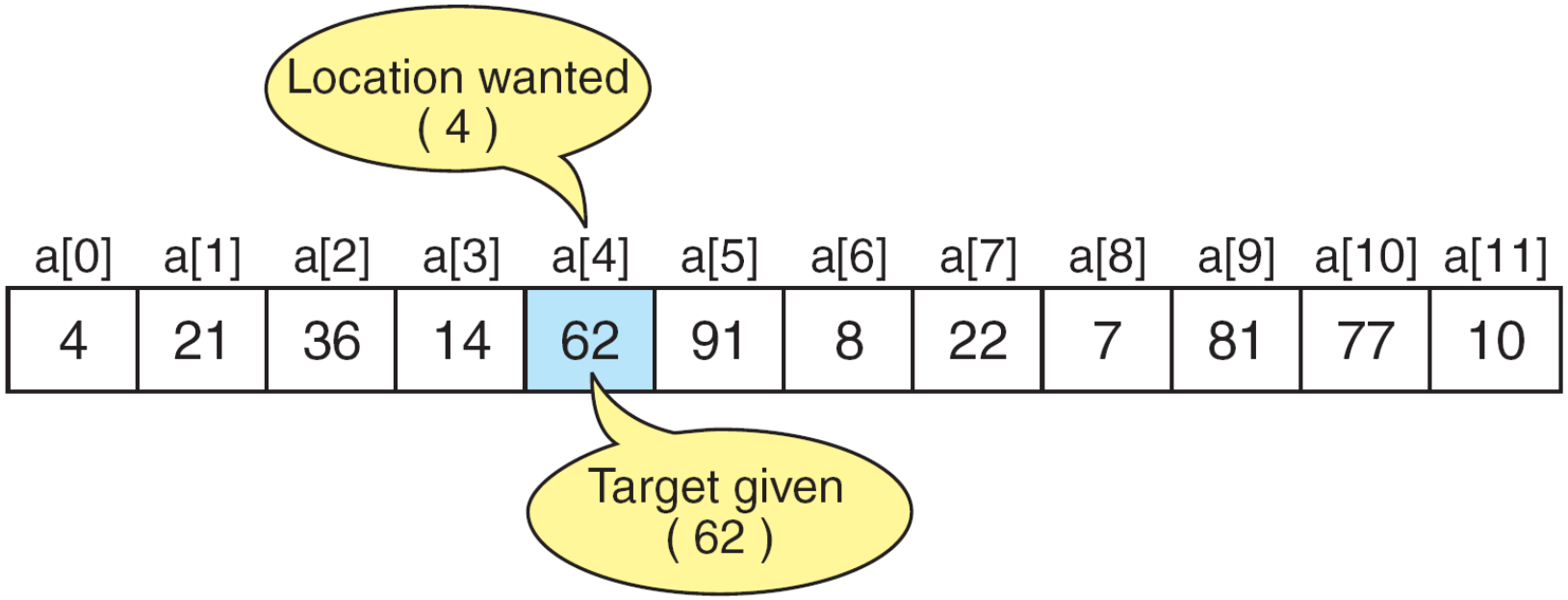
```
 1  /* ================== insertionSort ==================
 2     Sort list using Insertion Sort. The list is divided
 3     into sorted and unsorted lists. With each pass, first
 4     element in unsorted list is inserted into sorted list.
 5        Pre  list must contain at least one element
 6             last contains index to last element in list
 7        Post list has been rearranged
 8  */
 9  void insertionSort (int list[], int last)
10  {
11  // Local Declarations
12     int  walk;
13     int  temp;
14     bool located;
15
16  // Statements
17     // Outer loop
18     for (int current = 1; current <= last; current++)
19        {
```

Code #2

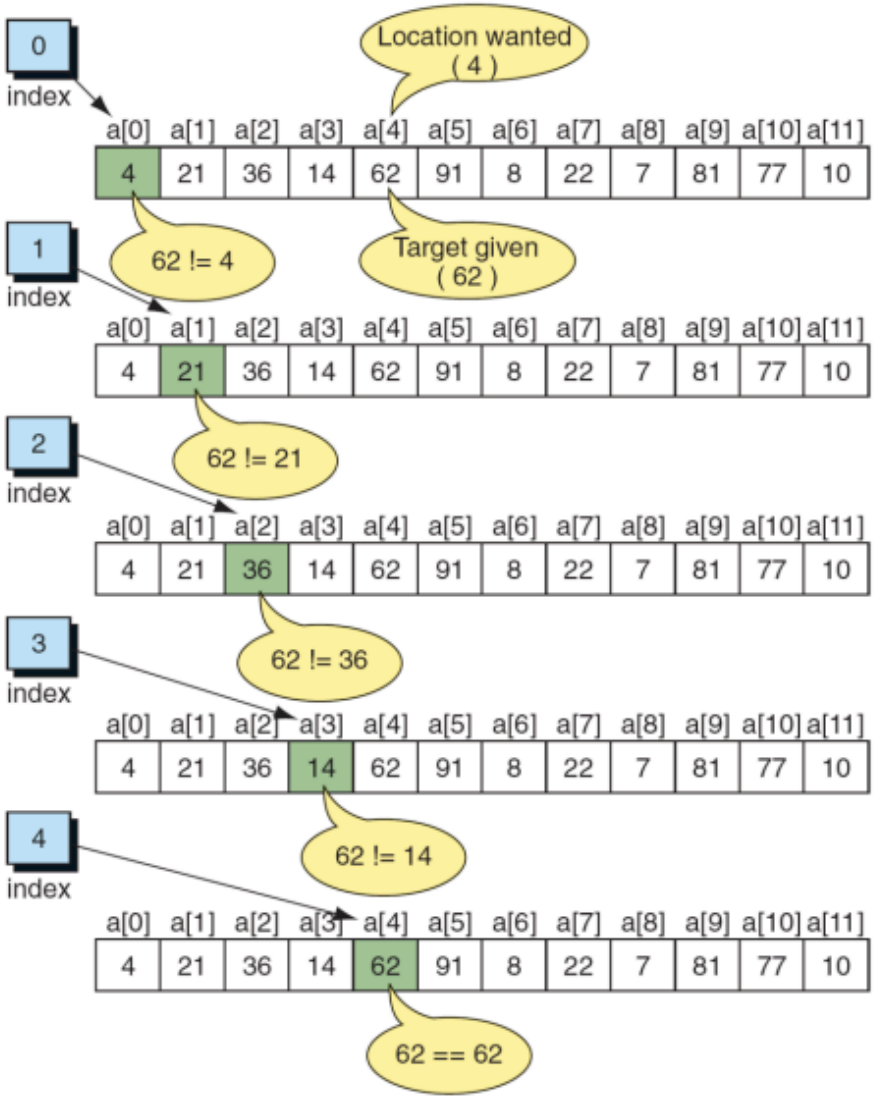
```
20          // Inner loop: Select and move one element
21          located = false;
22          temp = list[current];
23          for (walk = current - 1; walk >= 0 && !located;)
24             if (temp < list[walk])
25                {
26                 list[walk + 1] = list[walk];
27                 walk--;
28                } // if
29             else
30                 located = true;
31          list [walk + 1] = temp;
32       } // for
33    return;
34 }  // insertionSort
```

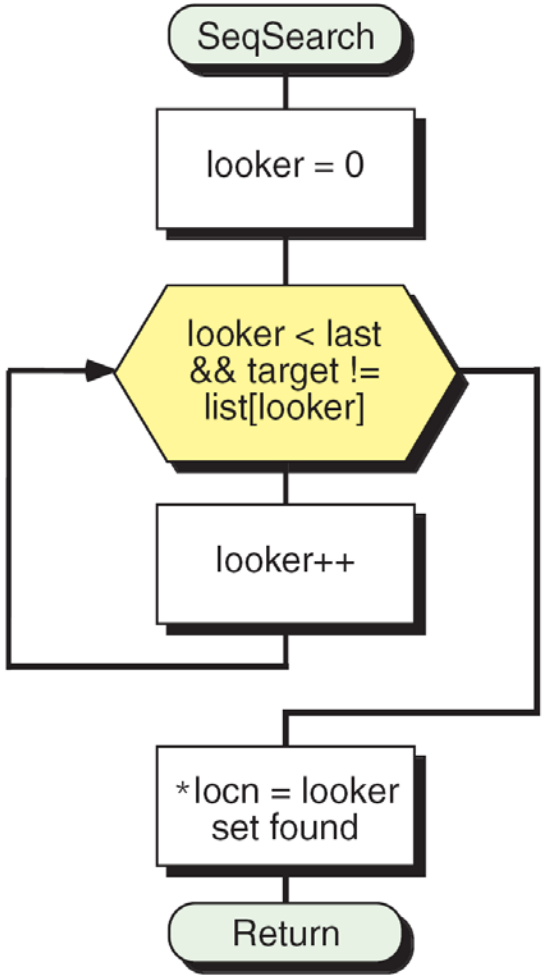
Search

Searching on Array



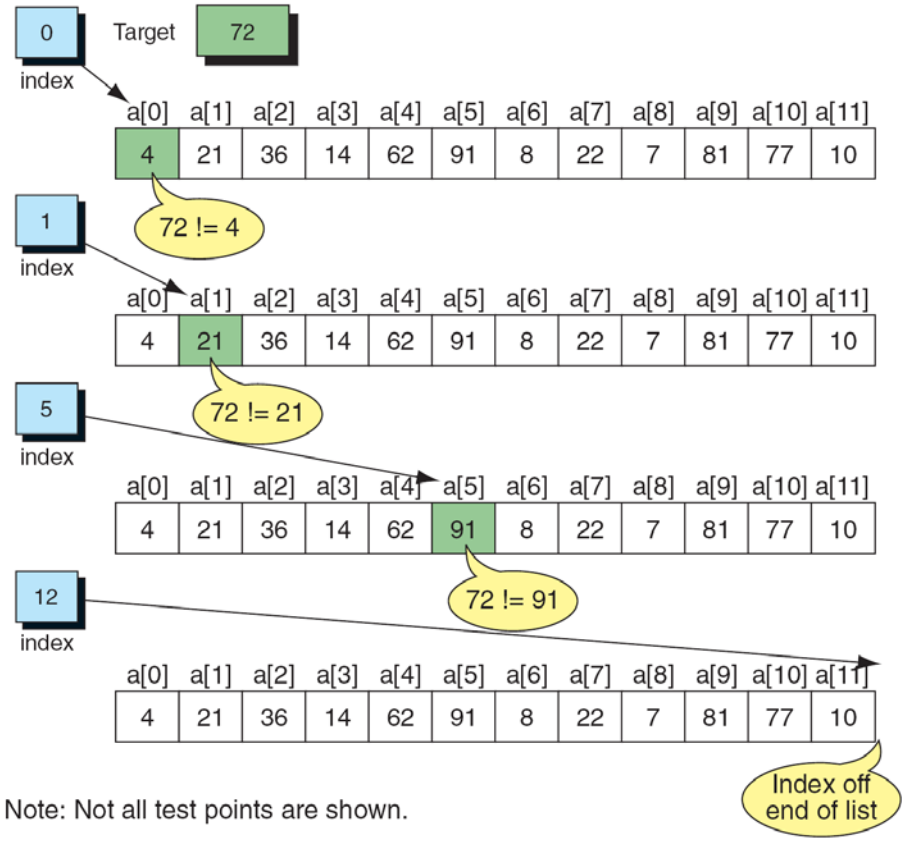
Sequential Search Design





Sequential Search 특징

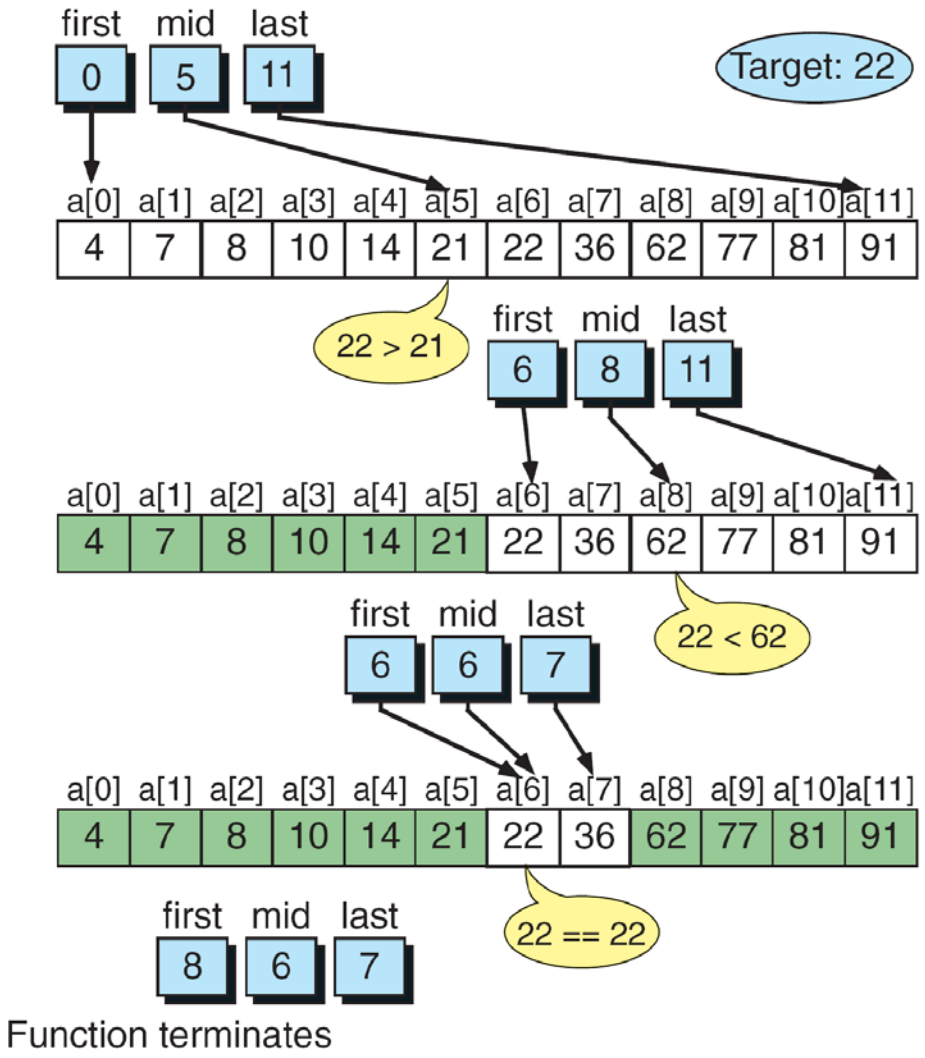
- Unsorted 배열에 대해서는 처음부터 쭉 봐야하기 때문에 비효율적이다.
- Worst Case: O(n)



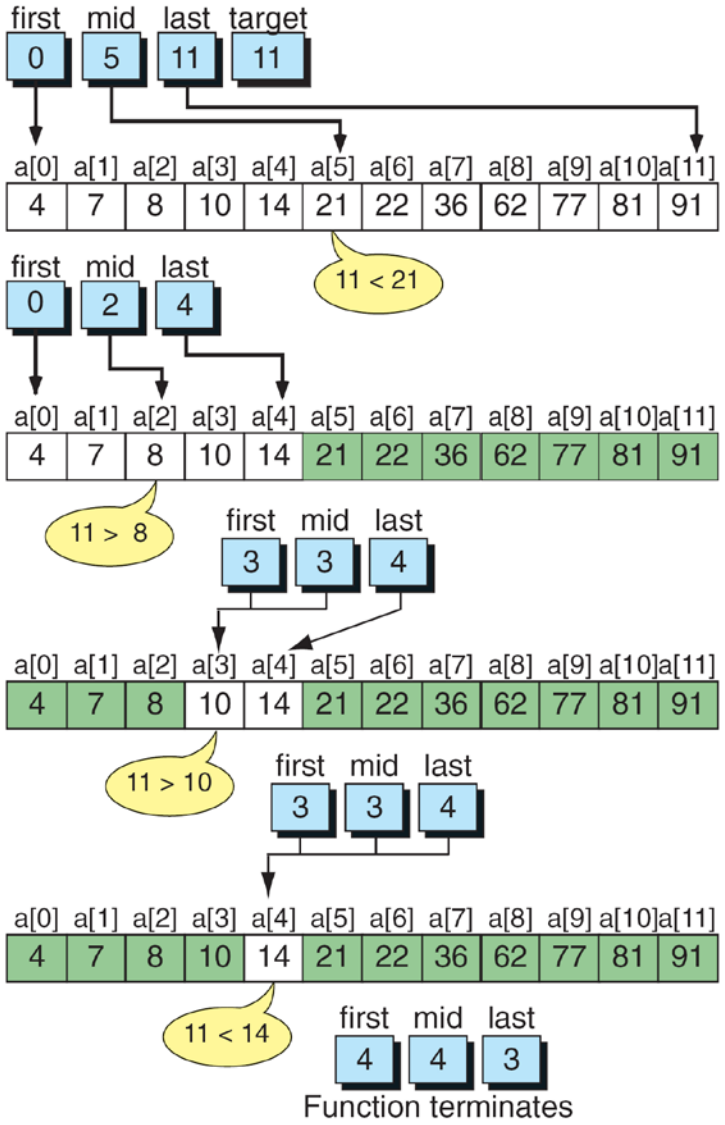
Code

```
1  /* ==================== seqSearch ====================
2     Locate target in an unordered list of size elements.
3        Pre   list must contain at least one item
4              last is index to last element in list
5              target contains the data to be located
6              locn is address for located target index
7        Post  Found: matching index stored in locn
8                     return true (found)
9              Not Found: last stored in locn
10                       return false (not found)
11 */
12 bool seqSearch (int list[], int  last,
13                  int target, int* locn)
14 {
15 // Local Declarations
16    int  looker;
17    bool found;
18
19 // Statements
20    looker = 0;
21    while (looker < last && target != list[looker])
22       looker++;
23
24    *locn = looker;
25    found = (target == list[looker]);
26    return found;
27 }  // seqSearch
```

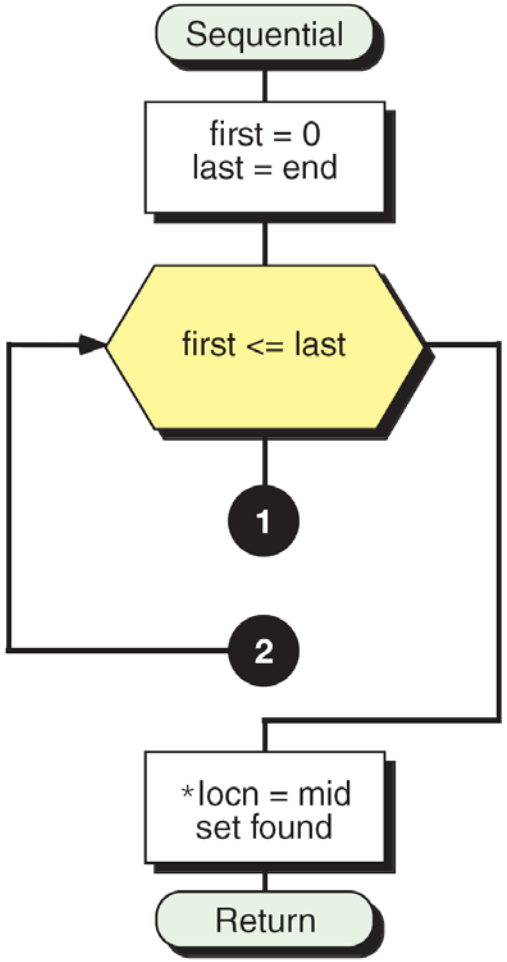
Binary Search

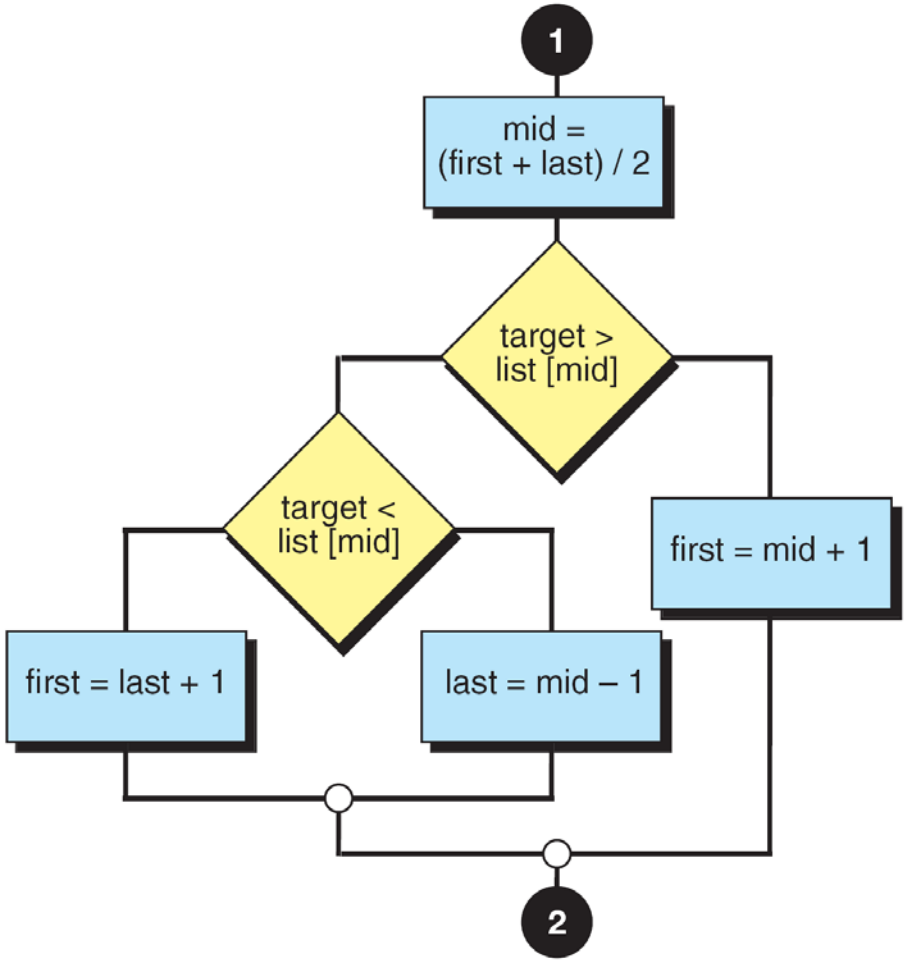


Unsuccessful Binary Search Example



Binary Search Design





Binary Search 특징

- 검색 대상 배열은 sorted 인 상태여야 한다.
  - “정렬된 데이터에 대해서는 이진검색이 빠르다”
- 매 iteration에서 반틈(1/2)을 서치 후보에서 제외시킨다.(제외된 거는 볼 필요 없음)
- O(logn)

Code

```
1  /* ================== binarySearch ==================
2     Search an ordered list using Binary Search
3        Pre   list must contain at least one element
4              end is index to the largest element in list
5              target is the value of element being sought
6              locn is address for located target index
7        Post  Found: locn = index to target element
8                     return 1 (found)
9              Not Found: locn = element below or above target
10                    return 0 (not found)
11 */
12 bool binarySearch (int list[], int  end,
13                    int target, int* locn)
14 {
15 // Local Declarations
16    int first;
17    int mid;
18    int last;
19
20 // Statements
21    first = 0;
22    last = end;
23    while (first <= last)
24       {
25        mid = (first + last) / 2;
26        if (target > list[mid])
27           // look in upper half
28           first = mid + 1;
29        else if (target < list[mid])
30           // look in lower half
31           last = mid - 1;
32        else
33           // found equal: force exit
34           first = last + 1;
35       } // end while
36    *locn = mid;
37    return target == list [mid];
38 }  // binarySearch
```

동영상 하나 보고 갑시다

- http://youtu.be/kPRA0W1kECg

Efficiency

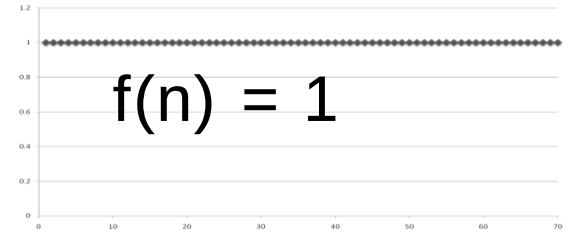
	Best	Average	Worst
Selection	n^2	n^2	n^2
Bubble	n	n^2	n^2
Insertion	n	n^2	n^2

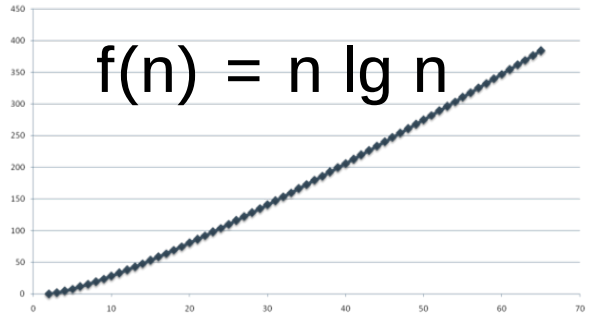
Search Efficiency

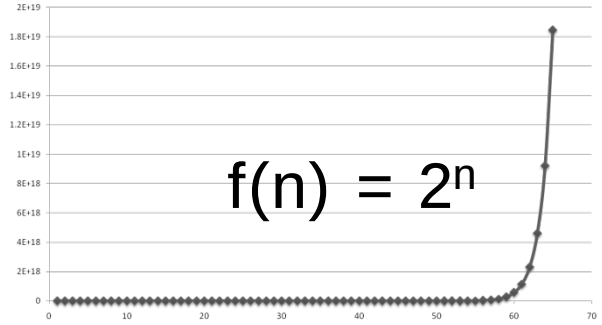
- Sequential O(n) vs Binary O(logn)

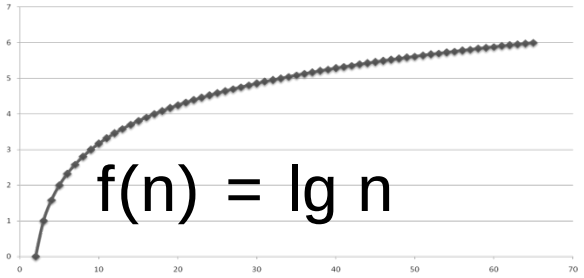
Size	Binary	Sequential (Average)	Sequential (Worst Case)
16	4	8	16
50	6	25	50
256	8	128	256
1,000	10	500	1,000
10,000	14	5,000	10,000
100,000	17	50,000	100,000
1,000,000	20	500,000	1,000,000

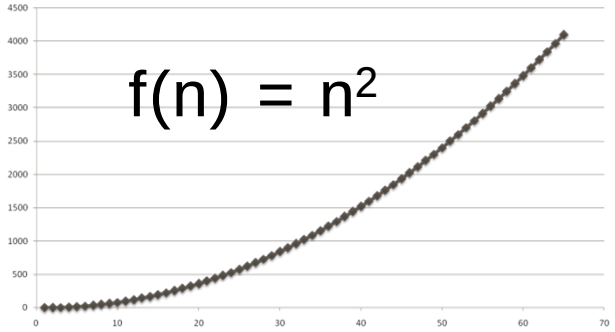
Functions Graphed Using “Normal” Scale

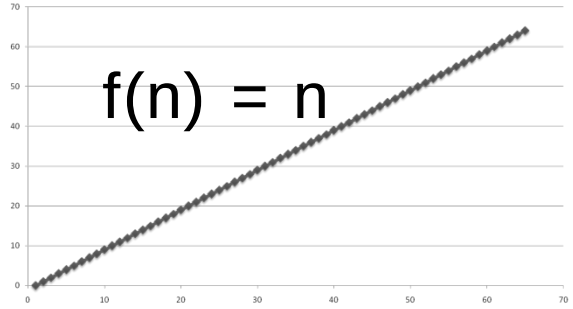


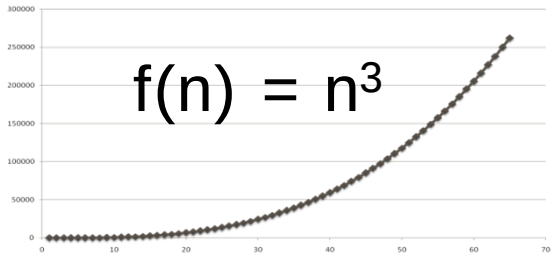












다음시간

- Pointer, 그리고 Array와의 관계
- Pointer Application
- Dynamic Memory Allocation