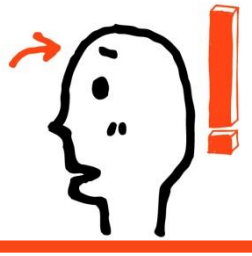
01

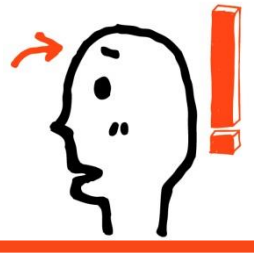
알고리즘이란?



생각하는 방법을 터득한 것은
미래의 문제를 미리 해결한 것이다.

– 제임스 왓슨

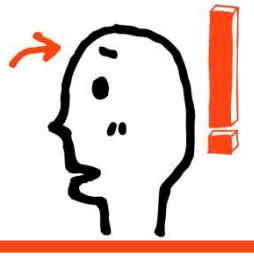
알고리즘이란?



알고리즘은 생각하는 방법을 훈련하는 것

- 문제 자체를 해결하는 알고리즘을 배운다
- 그 과정에 깃든 '생각하는 방법'을 배우는 것이 더 중요하다
- 미래에 다른 문제를 해결하는 생각의 빌딩블록을 제공한다

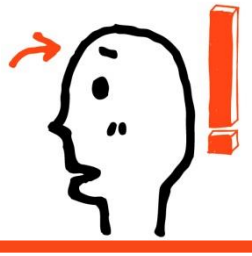
알고리즘이란?



알고리즘은 자료구조의 확장

- 선행 과목
  - 프로그래밍, 자료구조
- 자료구조
  - 건축의 건축 자재나 모듈 같은 것
  - 자동차 제작의 부품이나 모듈 같은 것

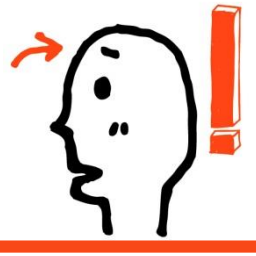
알고리즘이란?



알고리즘의 분석

- 크기가 작은 문제
  - 알고리즘의 효율성이 중요하지 않다
  - 비효율적인 알고리즘도 무방
- 크기가 충분히 큰 문제
  - 알고리즘의 효율성이 중요하다
  - 비효율적인 알고리즘은 치명적

세 값의 최대값



- 3개의 정수값 가운데 최대값을 구하는 프로그램
- 변수 a,b,c에 들어가는 값은 키보드에서 입력한 값, 그 3개의 값 중 최대값을 변수 max로 구현

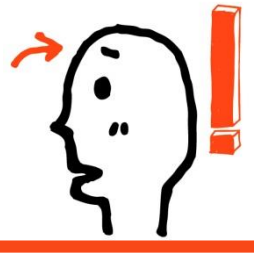
```
01  package chap01;
02  import java.util.Scanner;
03  // 3개의 정숫값을 입력하고 최댓값을 구합니다.
04
05  class Max3 {
06    public static void main(String[] args) {
07      Scanner stdIn = new Scanner(System.in);
08
09      System.out.println("세 정수의 최댓값을 구합니다.");
10      System.out.print("a의 값 : "); int a = stdIn.nextInt();
11      System.out.print("b의 값 : "); int b = stdIn.nextInt();
12      System.out.print("c의 값 : "); int c = stdIn.nextInt();
13      int max = a;
14      if (b > max) max = b;
15      if (c > max) max = c;
16
17      System.out.println("최댓값은 " + max + "입니다.");
18    }
19  }
```

a, b, c의 최댓값을 구하여 max에 대입

실행 결과

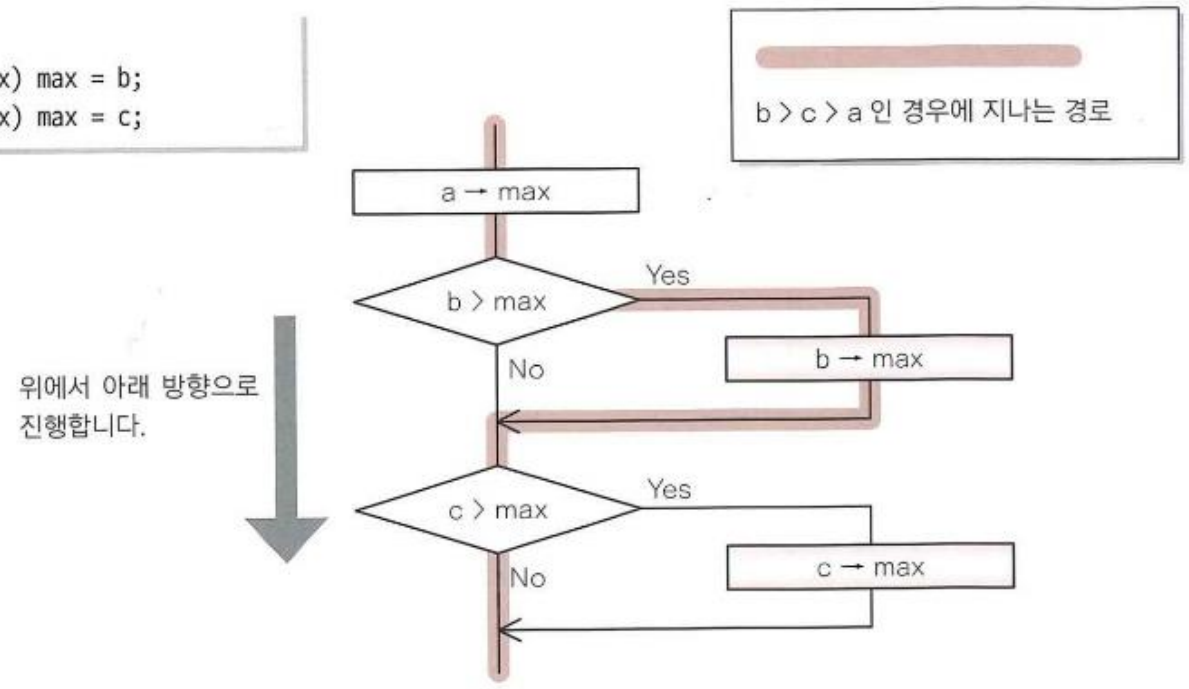
```
세 정수의 최댓값을 구합니다.
a의 값 : 1
b의 값 : 3
c의 값 : 2
최댓값은 3입니다.
```

세 값의 최대값



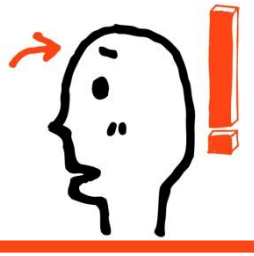
• 세 값의 최대값을 구하는 순서도

```
max = a;
if(b > max) max = b;
if(c > max) max = c;
```



[그림 1-1] 세 값의 최댓값을 구하는 알고리즘의 순서도

세 값의 최대값

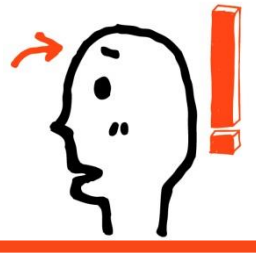


• 세 값의 최대값을 구하는 과정에서 변수 max의 변화

```
max = a;
if(b > max) max = b;
if(c > max) max = c;
```

a	b	c	d	e
a = 1 b = 3 c = 2	a = 1 b = 2 c = 3	a = 3 b = 2 c = 1	a = 5 b = 5 c = 5	a = 1 b = 3 c = 1
max	max	max	max	max
1	1	3	5	1
↓	↓	↓	↓	↓
3	2	3	5	3
↓	↓	↓	↓	↓
3	3	3	5	3

세 값의 최대값



• 프로그램 구현

실습 1-2

• 완성 파일 chap01/Max3m.java

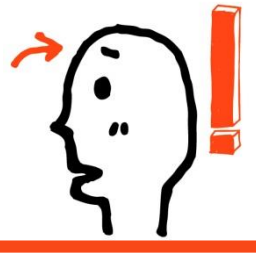
```
01  package chap01;
02  // 3개의 정숫값 가운데 최댓값을 구하여 출력합니다.
03
04  class Max3m {
05    // a, b, c의 최댓값을 구하여 반환합니다.
06    static int max3(int a, int b, int c) {
07      int max = a;                    // 최댓값
08      if (b > max)
09        max = b;
10      if (c > max)
11        max = c;
12
13      return max;
14    }
```

구한 최댓값을 호출한 곳으로 반환

실행 결과

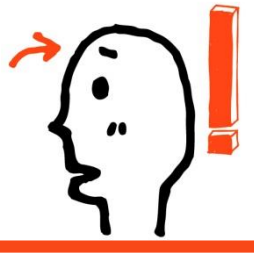
```
max3(3,2,1) = 3
max3(3,2,2) = 3
max3(3,1,2) = 3
max3(3,2,3) = 3
… 중략 …
max3(2,3,2) = 3
max3(1,3,2) = 3
max3(2,3,3) = 3
max3(1,2,3) = 3
```

세 값의 최대값

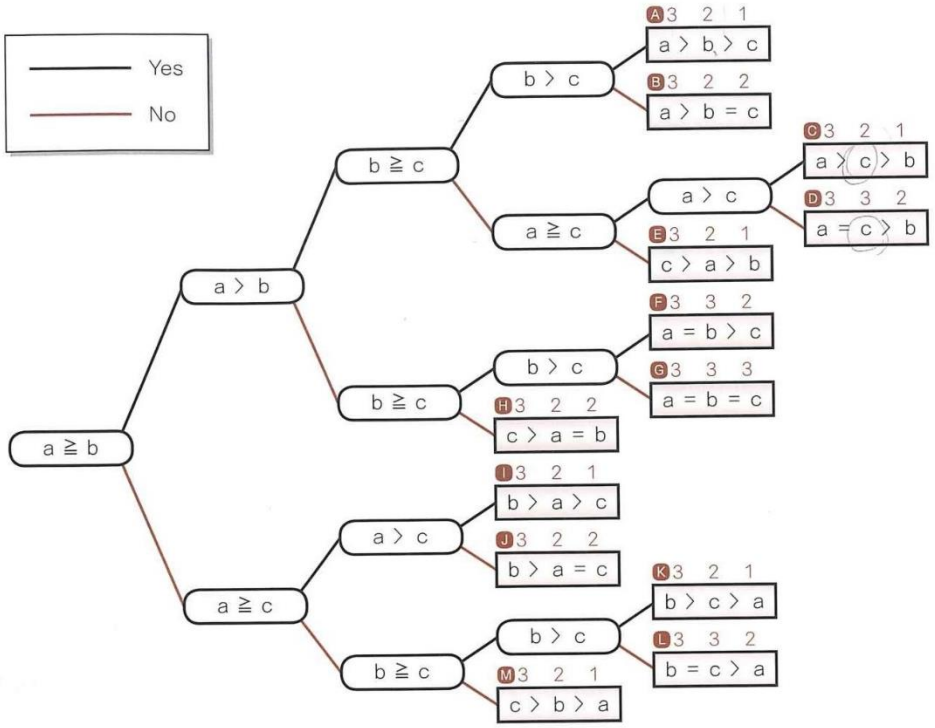


```
15
16      public static void main(String[] args) {
17          System.out.println("max3(3,2,1) = " + max3(3, 2, 1));  // [A] a> b> c
18          System.out.println("max3(3,2,2) = " + max3(3, 2, 2));  // [B] a> b=c
19          System.out.println("max3(3,1,2) = " + max3(3, 1, 2));  // [C] a> c> b
20          System.out.println("max3(3,2,3) = " + max3(3, 2, 3));  // [D] a=c> b
21          System.out.println("max3(2,1,3) = " + max3(2, 1, 3));  // [E] c> a> b
22          System.out.println("max3(3,3,2) = " + max3(3, 3, 2));  // [F] a=b> c
23          System.out.println("max3(3,3,3) = " + max3(3, 3, 3));  // [G] a=b=c
24          System.out.println("max3(2,2,3) = " + max3(2, 2, 3));  // [H] c> a=b
25          System.out.println("max3(2,3,1) = " + max3(2, 3, 1));  // [I] b> a> c
26          System.out.println("max3(2,3,2) = " + max3(2, 3, 2));  // [J] b> a=c
27          System.out.println("max3(1,3,2) = " + max3(1, 3, 2));  // [K] b> c> a
28          System.out.println("max3(2,3,3) = " + max3(2, 3, 3));  // [L] b=c> a
29          System.out.println("max3(1,2,3) = " + max3(1, 2, 3));  // [M] c> b> a
30      }
31  }
```

세 값의 최대값

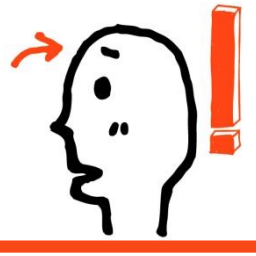


• 세 값의 대소 관계



[그림 1C-4] 세 값 a, b, c의 대소 관계를 나열한 결정 트리

순서도의 기호



프로그램 순서도

프로그램 순서도(program flowchart)에 사용되는 기호는 다음과 같습니다.

데이터

데이터(data)의 입력과 출력을 나타냅니다.

[그림 1-4] 데이터

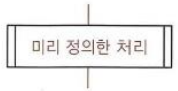
처리

처리(process)는 여러 종류의 처리 기능을 수행합니다. 즉, 정보의 값, 자료형, 위치를 바꾸도록 정의한 연산이나 연산집합의 실행 또는 연속적인 몇 가지 흐름 가운데 하나의 방향을 결정하는 연산집합이나 연산군의 실행을 나타냅니다.

[그림 1-5] 처리

미리 정의한 처리

미리 정의한 처리(predefined process)는 서브 루틴 및 모듈 등 다른 곳에서 이미 정의한 하나 이상의 연산 또는 명령어들로 이루어진 처리를 나타냅니다.



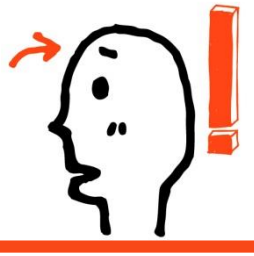
[그림 1-6] 미리 정의한 처리

판단

판단(decision)은 하나의 입구와 하나 이상을 선택할 수 있는 출구가 있고, 기호에서 정의한 조건을 평가하여 하나의 출구를 선택하는 판단 기능(스위치형 기능)을 나타냅니다. 주로 예상되는 평가 결과의 경로를 선 가까이에 씁니다.

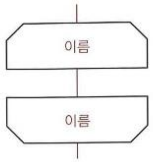
[그림 1-7] 판단

순서도의 기호

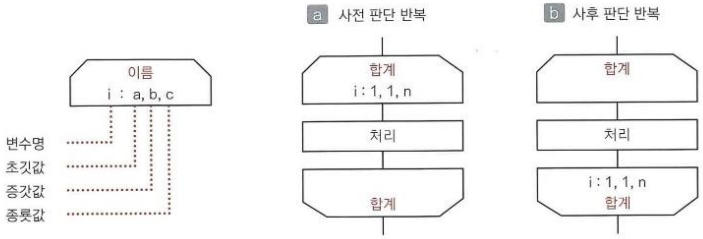


루프 범위

루프 범위(loop limit)는 그림 1-8과 같이 두 부분으로 구성되어 루프의 시작과 종료를 나타냅니다. 기호의 두 부분에는 같은 이름(루프에 대한 임의의 이름)을 사용합니다. 그림 1-9와 같이 루프의 시작 기호(반복 전에 판단하는 경우) 또는 종료 기호(반복 후에 판단하는 경우) 안에 초깃값(초기화), 증갓값, 종룟값(종료 조건)을 표기합니다.



[그림 1-8] 루프 범위



[그림 1-9] 루프의 시작과 종료 그리고 초깃값, 증갓값, 종룟값

위 그림에서 a와 b는 변수 i를 1부터 n까지 1씩 증가하면서 '처리'를 n번 반복하는 순서도입니다. '1, 1, n' 대신 '1, 2, …, n'을 사용하기도 합니다.

선

선(line)은 제어의 흐름을 나타냅니다. 주로 흐름의 방향을 분명히 나타내고자 할 때 화살표를 붙이는데, 순서도에 작성할 때도 보기 쉽게 화살표를 붙이기도 합니다.

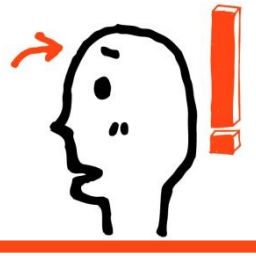
[그림 1-10] 선

단말

단말(terminator)은 외부 환경으로 나가거나 외부 환경에서 들어오는 것을 나타냅니다. 예를 들어 프로그램 흐름의 시작과 종료를 나타냅니다.

[그림 1-11] 단말

반복(while문 반복)



• 1부터 n까지의 정수 합 구하기

실습 1-4

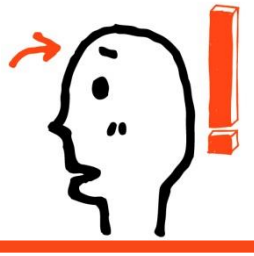
• 완성 파일 chap01/SumWhile.java

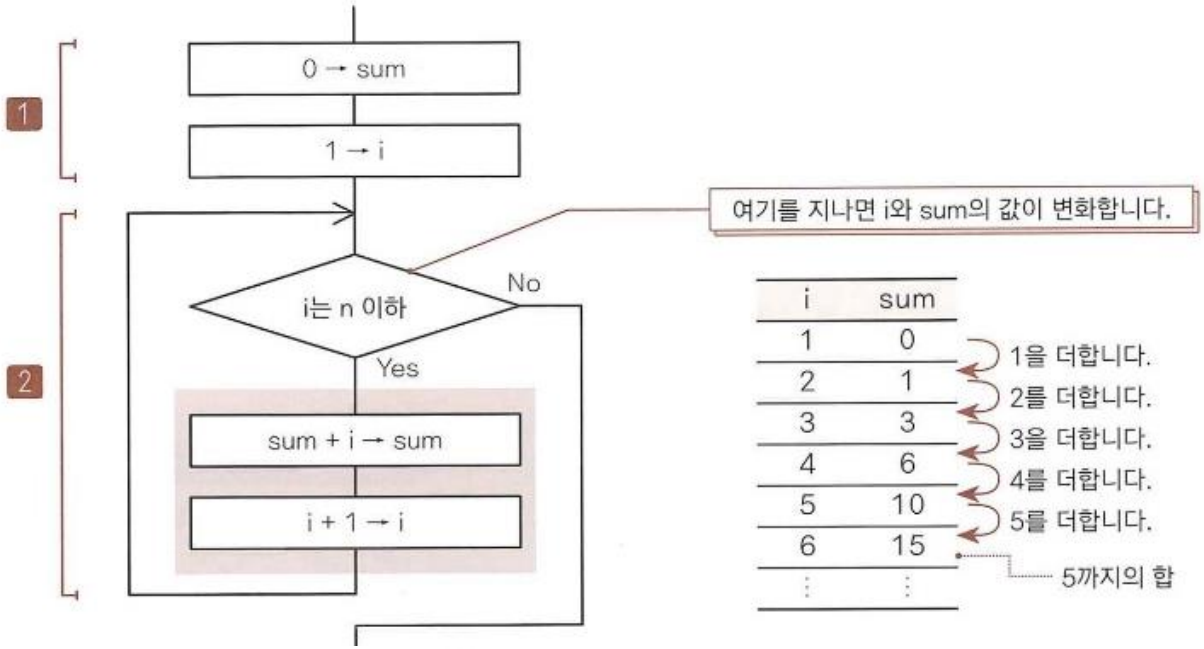
```
01  package chap01;
02  import java.util.Scanner;
03  // 1, 2, …, n의 합을 구합니다.
04
05  class SumWhile {
06    public static void main(String[] args) {
07      Scanner stdIn = new Scanner(System.in);
08
09      System.out.println("1부터 n까지의 합을 구합니다.");
10      System.out.print("n의 값 : ");
11      int n = stdIn.nextInt();
12
13      int sum = 0;          // 합                    ①
14      int i = 1;
15
16      while (i <= n) {      // i가 n 이하면 반복합니다.
17        sum += i;           // sum에 i를 더합니다.    ②
18        i++;                // i 값을 1만큼 증가시킵니다.
19      }
20      System.out.println("1부터 " + n + "까지의 합은 " + sum + "입니다.");
21    }
22  }
```

실행 결과

```
1부터 n까지의 합을 구합니다.
n의 값 : 5
1부터 5까지의 합은 15입니다.
```

반복



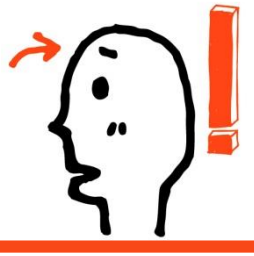


1 합을 구하기 위한 준비입니다. 합을 저장하는 변수 sum을 0, 반복을 제어하기 위한 변수 i를 1로 초기화합니다.

2 변수 i의 값이 n 이하인 동안 i의 값을 1씩 증가하면서 루프 본문을 n회 반복 실행합니다.

2항인 복합 대입 연산자 '+='은 우변의 값을 좌변에 더합니다. 또한 단항인 증가 연산자 '++'는 피연산자의 값을 1 증가(increment)시킵니다.

반복(for문 반복)



실습 1-5

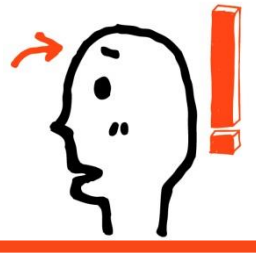
• 완성 파일 chap01/SumFor.java

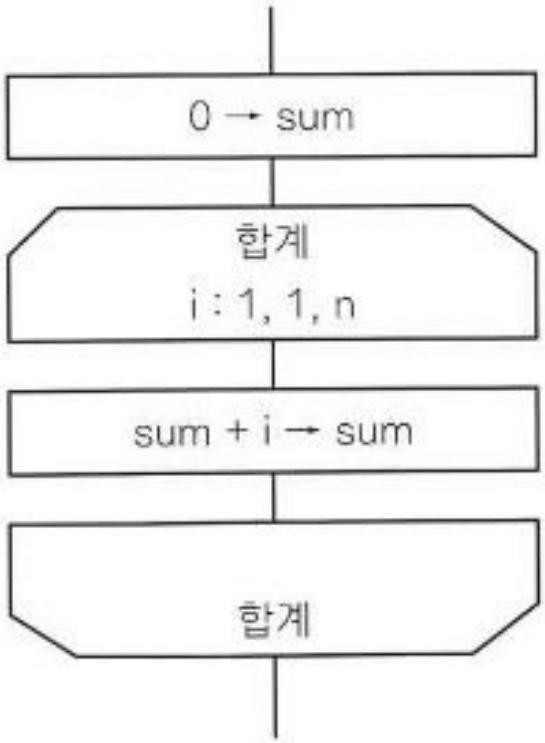
```
01  package chap01;
02  import java.util.Scanner;
03  // 1, 2, …, n의 합을 구합니다.
04
05  class SumFor {
06    public static void main(String[] args) {
07      Scanner stdIn = new Scanner(System.in);
08
09      System.out.println("1부터 n까지의 합을 구합니다.");
10      System.out.print("n의 값 : ");
11      int n = stdIn.nextInt();
12
13      int sum = 0;              // 합
14
15      for (int i = 1; i <= n; i++)
16        sum += i;               // sum에 i를 더합니다.
17
18      System.out.println("1부터 " + n + "까지의 합은 " + sum + "입니다.");
19    }
20  }
```

실행 결과

```
1부터 n까지의 합을 구합니다.
n의 값 : 5
1부터 5까지의 합은 15입니다.
```

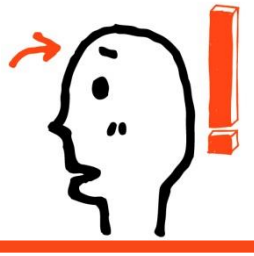
반복(for문 반복)





i 값을 1부터 시작하여 n이 될 때까지 1씩 증가하면서 반복합니다.

다중 루프



• 곱셈표

실습 1-7

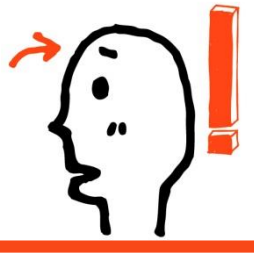
• 완성 파일 chap01/Multi99Table.java

```
01  package chap01;
02  // 곱셈표를 출력합니다.
03
04  public class Multi99Table {
05     public static void main(String[] args) {
06        System.out.println("----- 곱셈표 -----");
07
08        for (int i = 1; i <= 9; i++) {
09           for (int j = 1; j <= 9; j++)
10              System.out.printf("%3d", i * j);
11           System.out.println();
12        }
13     }
14  }
```

실행 결과

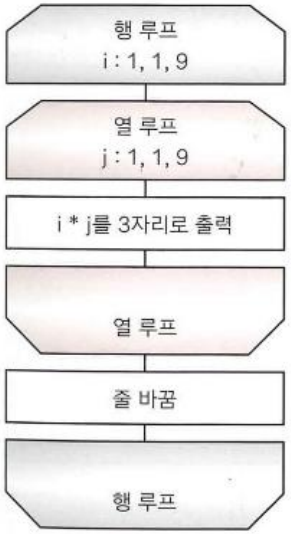
```
----- 곱셈표 -----
  1  2  3  4  5  6  7  8  9
  2  4  6  8 10 12 14 16 18
  3  6  9 12 15 18 21 24 27
  4  8 12 16 20 24 28 32 36
  5 10 15 20 25 30 35 40 45
  6 12 18 24 30 36 42 48 54
  7 14 21 28 35 42 49 56 63
  8 16 24 32 40 48 56 64 72
  9 18 27 36 45 54 63 72 81
```

다중 루프

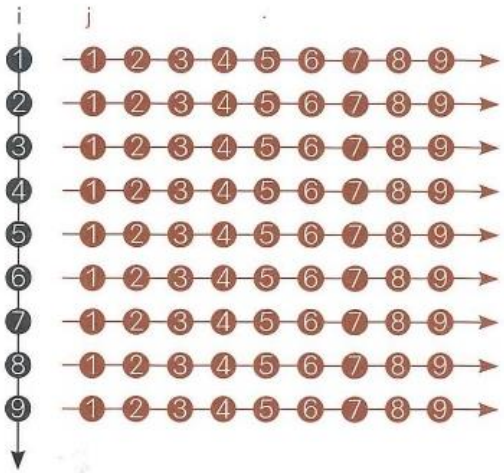


따라서 이 이중 루프는 다음과 같이 처리됩니다.

i가 1일 때 : j를 1 ⇨ 9로 증가시키면서 1 * j를 출력합니다. 그리고 줄 바꿈합니다.
i가 2일 때 : j를 1 ⇨ 9로 증가시키면서 2 * j를 출력합니다. 그리고 줄 바꿈합니다.
i가 3일 때 : j를 1 ⇨ 9로 증가시키면서 3 * j를 출력합니다. 그리고 줄 바꿈합니다.
… 중략 …
i가 9일 때 : j를 1 ⇨ 9로 증가시키면서 9 * j를 출력합니다. 그리고 줄 바꿈합니다.



변수 i와 j의 변화

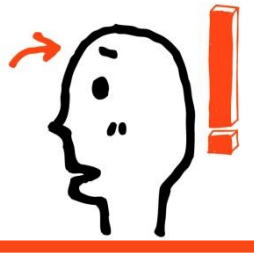


Q12 오른쪽과 같이 위쪽과 왼쪽에 곱하는 수가 있는 곱셈표를 출력하는 프로그램을 작성하세요.

구분선은 수직선 기호(|), 마이너스 기호(-), 플러스 기호(+)를 사용하세요.

```
  | 1  2  3  4  5  6  7  8  9
--+------------------------------
1 | 1  2  3  4  5  6  7  8  9
2 | 2  4  6  8 10 12 14 16 18
3 | 3  6  9 12 15 18 21 24 27
4 | 4  8 12 16 20 24 28 32 36
5 | 5 10 15 20 25 30 35 40 45
6 | 6 12 18 24 30 36 42 48 54
7 | 7 14 21 28 35 42 49 56 63
8 | 8 16 24 32 40 48 56 64 72
9 | 9 18 27 36 45 54 63 72 81
```

직각 이등변 삼각형 출력



실습 1-8

• 완성 파일 chap01/TriangleLB.javac

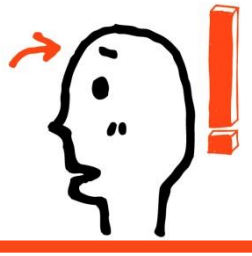
```
01  package chap01;
02  import java.util.Scanner;
03  // 왼쪽 아래가 직각인 이등변 삼각형을 출력합니다.
04
05  public class TriangleLB {
06    public static void main(String[] args) {
07      Scanner stdIn = new Scanner(System.in);
08      int n;
09
10      System.out.println("왼쪽 아래가 직각인 이등변 삼각형을 출력합니다.");
11
12      do {
13        System.out.print("몇 단 삼각형입니까? : ");
14        n = stdIn.nextInt();
15      } while (n <= 0);
16
17      for (int i = 1; i <= n; i++) {
18        for (int j = 1; j <= i; j++)
19          System.out.print('*');
20        System.out.println();
21      }
22    }
23  }
```

이등변 삼각형의 단 수를 입력합니다.

실행 결과

```
왼쪽 아래가 직각인 이등변 삼각형
을 출력합니다.
몇 단 삼각형입니까? : 5
*
**
***
****
*****
```

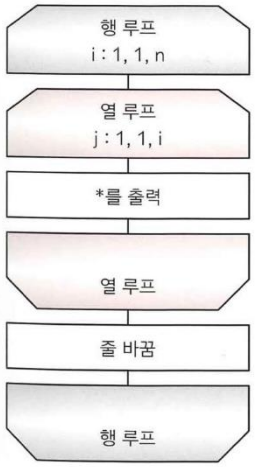
직각 이등변 삼각형 출력



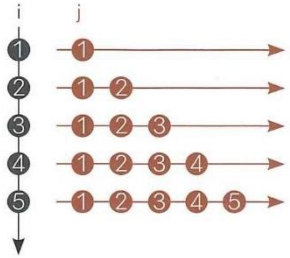
따라서 이 이중 루프는 아래처럼 처리됩니다.

i가 1 일 때 : j를 1 ⇨ 1로 증가시키면서 *를 출력합니다. 그리고 줄 바꿈합니다. (*)
i가 2 일 때 : j를 1 ⇨ 2로 증가시키면서 *를 출력합니다. 그리고 줄 바꿈합니다. (**)
i가 3 일 때 : j를 1 ⇨ 3으로 증가시키면서 *를 출력합니다. 그리고 줄 바꿈합니다. (***)
i가 4 일 때 : j를 1 ⇨ 4로 증가시키면서 *를 출력합니다. 그리고 줄 바꿈합니다. (****)
i가 5 일 때 : j를 1 ⇨ 5로 증가시키면서 *를 출력합니다. 그리고 줄 바꿈합니다. (*****)

이 삼각형을 위부터 1행 ~ n행이라고 하면 i행에 i개의 기호 문자 *를 출력하고 마지막 n행에 n개의 기호 문자 *를 출력합니다.



변수 i와 j의 변화(n은 5로 합니다.)



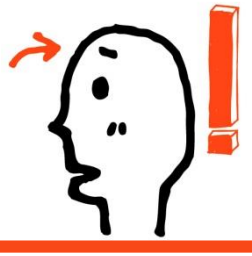
Q15 직각 이등변 삼각형을 출력하는 부분을 아래와 같은 형식의 메서드로 작성하세요.

```
static void triangleLB(int n)        // 왼쪽 아래가 직각인 이등변 삼각형을 출력
```

또 왼쪽 위, 오른쪽 위, 오른쪽 아래가 직각인 이등변 삼각형을 출력하는 메서드를 작성하세요.

```
static void triangleLU(int n)        // 왼쪽 위가 직각인 이등변 삼각형을 출력
static void triangleRU(int n)        // 오른쪽 위가 직각인 이등변 삼각형을 출력
static void triangleRB(int n)        // 오른쪽 아래가 직각인 이등변 삼각형을 출력
```

검색 알고리즘



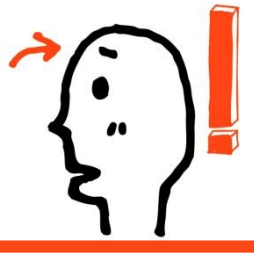
검색과 키

주소록을 검색한다고 가정해 보겠습니다. 검색(searching)은 다음과 같은 과정으로 이루어집니다.

1. 국적이 한국인 사람을 찾는다.
2. 나이가 21세 이상 27세 미만인 사람을 찾는다.
3. 찾으려는 이름과 가장 비슷한 이름의 사람을 찾는다.

1. 키 값과 일치하도록 지정합니다(한국).
2. 키 값의 구간을 지정합니다(21세 이상 27세 미만).
3. 키 값과 비슷하도록 지정합니다(발음이 가장 비슷한 이름).

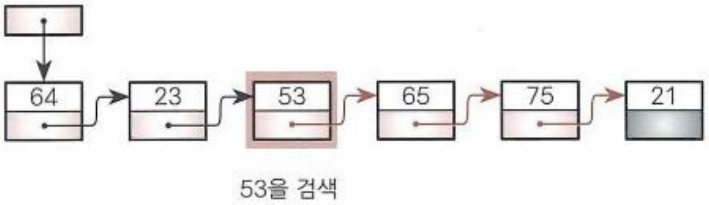
검색 알고리즘



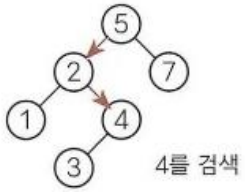
• 검색의 예

a 배열 검색

b 선형 리스트 검색

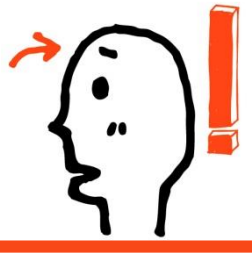


c 이진검색트리 검색



[그림 3-1] 검색 예

검색 알고리즘



선형 검색

요소가 직선 모양으로 늘어선 배열에서의 검색은 원하는 키 값을 갖는 요소를 만날 때까지 맨 앞부터 순서대로 요소를 검색하면 되는데, 이것이 선형 검색(linear search) 또는 순차 검색(sequential search)이라는 알고리즘입니다. 구체적인 과정을 아래의 데이터 나열을 예로 들어 살펴보겠습니다.

0	1	2	3	4	5	6
6	4	3	2	1	3	8

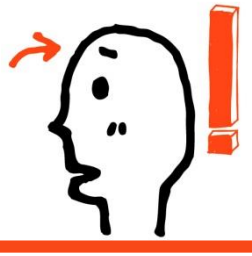
이 배열에서 값 2의 요소를 선형 검색하는 모습이 그림 3-2입니다.

배열의 요소를 맨 앞부터 순서대로 검색합니다.

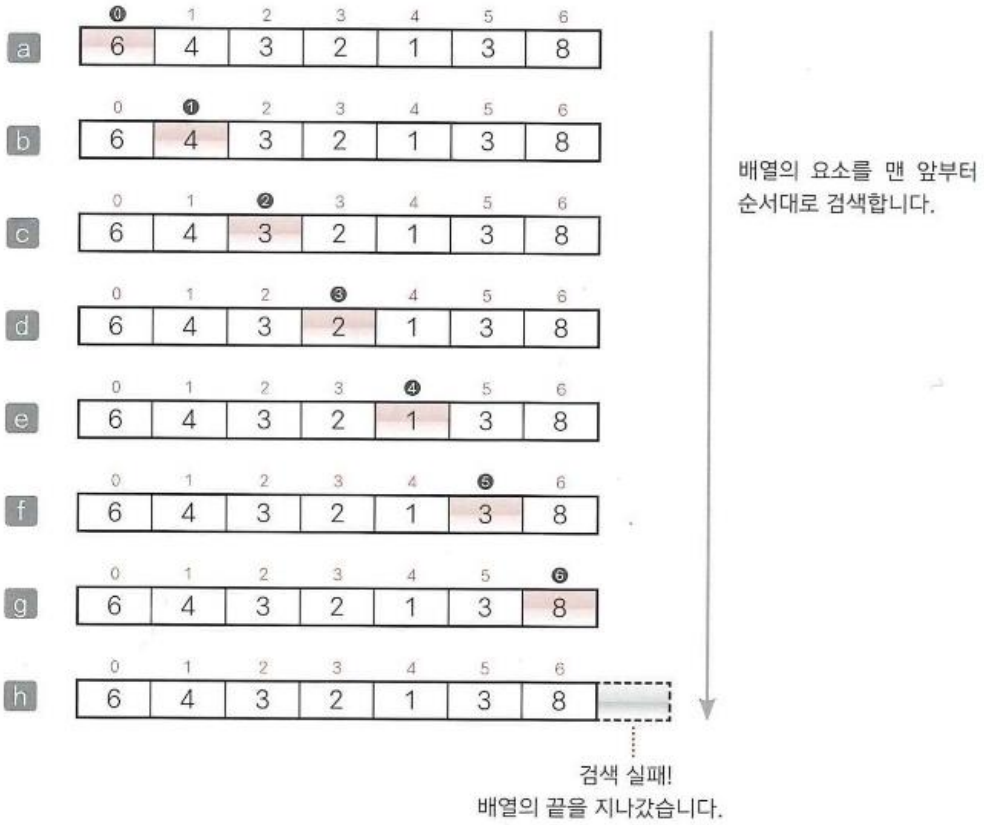
a 첫 번째 요소 6을 선택합니다. 원하는 값이 없습니다.
b 두 번째 요소 4를 선택합니다. 원하는 값이 없습니다.
c 세 번째 요소 3을 선택합니다. 원하는 값이 없습니다.
d 네 번째 요소 2를 선택합니다. 원하는 값입니다. 검색 성공!

[그림 3-2] 선형 검색의 예(2를 검색 : 검색 성공)

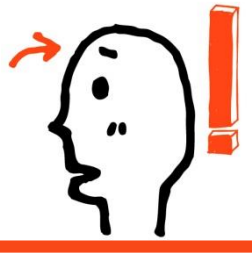
검색 알고리즘



• 선형 검색의 예



검색 알고리즘



실습 3-1

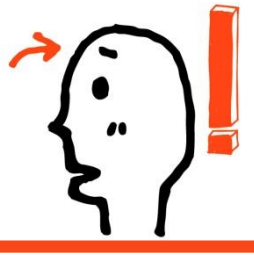
• 완성 파일 chap03/SeqSearch.java

```
01  package chap03;
02  import java.util.Scanner;
03  // 선형 검색
04
05  class SeqSearch {
06    // 요솟수가 n인 배열 a에서 key와 같은 요소를 선형 검색합니다.
07    static int seqSearch(int[] a, int n, int key) {
08      int i = 0;
09
10      while (true) {
11        if (i == n)
12          return -1;      // 검색 실패!(-1을 반환)
13        if (a[i] == key)
14          return i;       // 검색 성공!(인덱스를 반환)
15        i++;
16      }
17    }
18
19    public static void main(String[] args) {
20      Scanner stdIn = new Scanner(System.in);
21
22      System.out.print("요솟수 : ");
23      int num = stdIn.nextInt( );
24      int[] x = new int[num];             // 요솟수가 num인 배열
25
26      for (int i = 0; i < num; i++) {
27        System.out.print("x[" + i + "] : ");
28        x[i] = stdIn.nextInt( );
29      }
30
31      System.out.print("검색할 값 : ");      // 키 값을 입력
32      int ky = stdIn.nextInt( );
33
34      int idx = seqSearch(x, num, ky);    // 배열 x에서 키 값이 ky인 요소를 검색
35
36      if (idx == -1)
37        System.out.println("그 값의 요소가 없습니다.");
38      else
39        System.out.println(ky+"은(는) x[" + idx + "]에 있습니다.");
40    }
41  }
```

실행 결과

```
요솟수 : 7
x[0] : 22
x[1] : 8
x[2] : 55
x[3] : 32
x[4] : 120
x[5] : 55
x[6] : 70
검색할 값 : 55
55은(는) x[2]에 있습니다.
```

검색 알고리즘



• 종료 조건

1. i == n이 성립하는 경우 (종료 조건 ① : 검색 실패이므로 -1을 반환)
2. a[i] == key가 성립하는 경우 (종료 조건 ② : 검색 성공이므로 i를 반환)

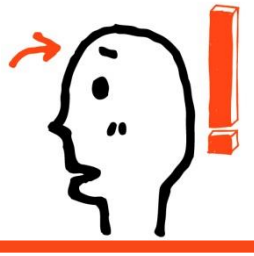
• FOR문으로 수정

실습 3-2

• 완성 파일 chap03/SeqSearchFor.java

```
06  // 배열 a의 앞쪽 n개의 요소에서 key와 같은 요소를 선형 검색
07  static int seqSearch(int[] a, int n, int key) {
08    for (int i = 0; i < n; i++)
09      if (a[i] == key)
10        return i;          // 검색 성공!(인덱스를 반환)
11    return -1;             // 검색 실패!(-1을 반환)
12  }
```

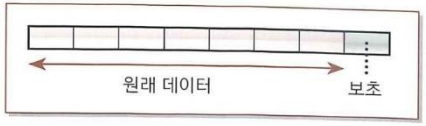
검색 알고리즘



보초법

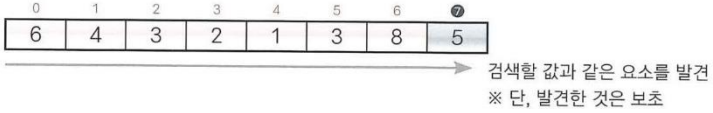
선형 검색은 반복할 때마다 다음의 종료 조건 ①과 ②를 모두 판단합니다. 단순한 판단이라고 생각할 수 있지만 '티끌 모아 태산'이라는 말이 있듯이 종료 조건을 검사하는 비용은 결코 무시할 수 없습니다.

> 종료 조건 ① 검색할 값을 발견하지 못 하고 배열의 끝을 지나간 경우
> 종료 조건 ② 검색할 값과 같은 요소를 발견한 경우



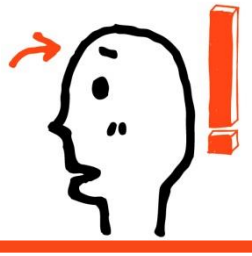
a 2를 검색(검색 성공)

b 5를 검색(검색 실패)



[그림 3-4] 보초법을 이용한 선형 검색

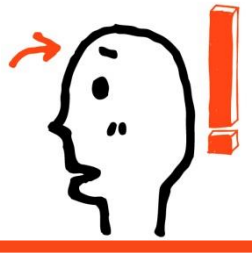
검색 알고리즘



이 그림에서 배열의 요소 a[0] ~ a[6]은 초기에 준비해 놓은 데이터입니다. 검색하기 전에 검색하고자 하는 키 값을 맨 끝 요소 a[7]에 저장합니다. 이때 저장하는 값을 보초(sentinel)라고 합니다.

a : 2를 검색하기 위해 보초로 a[7]에 2를 저장합니다.
b : 5를 검색하기 위해 보초로 a[7]에 5를 저장합니다.

검색 알고리즘

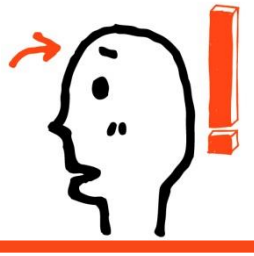


```
06    // 요솟수가 n인 배열 a에서 key와 같은 요소를 보초법으로 선형 검색합니다.
07    static int seqSearchSen(int[] a, int n, int key) {
08      int i = 0;
09
10      a[n] = key;                    // 보초를 추가          ← 1
11
12      while (true) {
13        if (a[i] == key)             // 검색 성공!
14          break;                                            ← 2
15        i++;
16      }
17      return i == n ? -1 : i;                               ← 3
18    }
```

실행 결과

```
요솟수 : 7
x[0] : 22
x[1] : 8
x[2] : 55
x[3] : 32
x[4] : 120
x[5] : 55
x[6] : 70
검색할 값 : 120
120은(는) x[4]에 있습니다.
```

검색 알고리즘



```
20    public static void main(String[] args) {
21      Scanner stdIn = new Scanner(System.in);
22
23      System.out.print("요솟수 : ");
24      int num = stdIn.nextInt( );
25      int[] x = new int[num + 1];          // 요솟수 num + 1
26
27      for (int i = 0; i < num; i++) {
28        System.out.print("x[" + i + "] : ");
29        x[i] = stdIn.nextInt( );
30      }
31
32      System.out.print("검색할 값 : ");       // 키 값을 입력
33      int ky = stdIn.nextInt( );
34
35      int idx = seqSearchSen(x, num, ky);   // 배열 x에서 값이 ky인 요소를 검색
36
37      if (idx == -1)
38        System.out.println("그 값의 요소가 없습니다.");
39      else
40        System.out.println(ky+"은(는) x[" + idx + "]에 있습니다.");
41    }
42  }
```