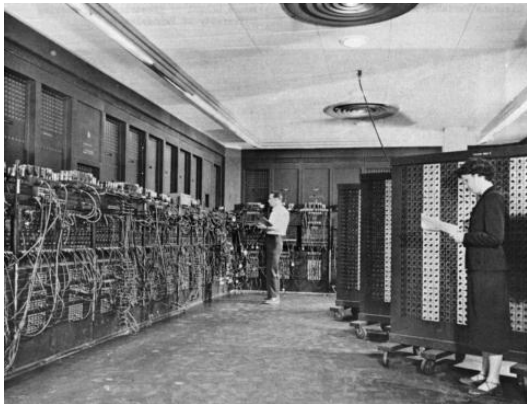


이사코드

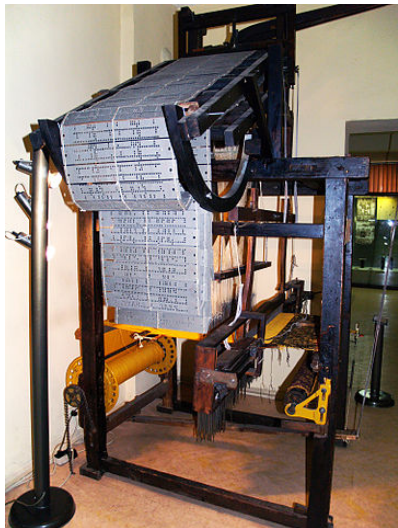
## 애니악 – 코드 연결을 통한 프로그램)

- n 애니악은 ‘최초의 프로그래밍이 가능한 컴퓨팅기계’라는 수식어가 항상 따라붙는다. © 위키미디어 퍼블릭도메인



# 자카드 직조기 (일련의 천공 나무막대)

- n 스톡 콘텐츠 - 자카드 직조기는 천공 된 카드를 사용하여 제직 용 패턴을 프로그래밍하는 기계 직기입니다



# 폰노이만 컴퓨터

CPU

ALU

register

(+)

이전 컴퓨터는 하드웨어의 결함으로  
1+1에 4를 하는 것으로 때론 바깥  
계산

but

(+, - ...)

폰노이만 구조는 소프트웨어가 메모리 안에  
내장되어 있음 → 계산이 필요할 때 마다  
메모리 안의 프로그램 데이터를 CPU에 전송

주기억장치



1010101111000010

1011101010100101

.....

0011001010000111

0000111101011111

0000111101011111

1111100000010100

.....

메모리를 읽고 쓰는 방식

## 2장: 문제 해결 방법의 발견과 해결

---

2.1 알고리즘 표현하기

2.2 알고리즘으로 하는 문제 해결 예

2.3 알고리즘의 분석과 효율성 분석

---

## 2.1 알고리즘 표현하기

# Computation not calculation

---

- n Calculation and computation
  - | We use a calculator to perform simple arithmetic operations, whereas a computer is typically used to perform complicated tasks, often involving **complex algorithms**. You may therefore use the words “calculate” and “calculation” to indicate simplicity and “compute” and “computation” to indicate complexity.
- n Calculation : 간단한 계산
- n Computation : complex algorithm - 복잡한 계산
  - | 일련의 “간단한 계산과 다음 계산 순서 지정 “

# 알고리즘 표현하기

---

- n 알고리즘을 어떻게 표현할 것인가는 매우 중요한 일
  - | 명확하게 필요한 동작과 다음 단계를 나타내도록 표현
  - | 표기법
    - 4 자연어로 표현
    - 4 Pseudo Code



# 자연어 표현의 문제는?

## n “구조화되어 있지 않다”

- | 알고리즘을 읽기 어렵고, 알고리즘 흐름을 파악하기도 어려움
- | 판별식 알고리즘을 적을 때

‘~~~결과 값이 ‘영’과 같으면~~’은,

앞부분의 ‘~~~결과 값이 ‘영’보다 크면~~’과 동시에 성립하는 것인지

- 4 0보다 크고, 그리고 동시에 0과 같다는 것( $\geq$ )인지
- 4 아니면 두 문장이 배타적인지
  - 0보다 크지 않고, 따로 0과 같다는 것인지) 명확히 알기 어려움
  - 들여쓰기, 순환 시작 등을 알리는 표시를 사용하지 않았기 때문
  - 명확성을 강조하기 위해 **동시적인지 배타적인지**를 알리기 위해, 특정 시작 부분을 안쪽으로 들여 쓰는 표현을 사용

# 라면끓이는 절차 : 자연어 알고리즘

**조 리 법**



1 끓는물 550 ml(종이컵 3컵)에  
면과 스프를 넣고  
4분간 더 끓입니다.



2 기호에 따라 김치, 계란, 마늘,  
파 등을 넣어 드시면 더욱  
맛이 좋습니다.



※니트통(식염) 섭취를 조절하기 위하여 기호에 따라 적정량의 스프를 첨가하여  
조리하십시오.  
※조리시 안전에 주의하세요.

• 고객만족팀 : 수신자요금 부담 080-940-3333 • 보관방법 : 직사광선을 피하고  
건조하고 서늘한 곳에 보관하십시오. ① 반드시 조리법에 따라 조리하여 드십시오. ② 유통기한이 지나 제품을 그대로 드시지 마세요. • 반죽 및 교반 : 본사, 각 공장체 삼익  
삼분지이 한길부다.

# 자연어로 알고리즘 기술

- n 자연어는 의미에서 여러 의미로 해석할 수 있다.
    - l 항상 같은 순서와 방법으로 실행하여 항상 같은 결과를 만들어야
    - l ‘~~판별식의 루트 값을~~’에서
      - 4 판별식을 계산한 결과를 루트로 계산한다는 것인지,
      - 4 아니면 다른 ‘루트’값이라는 것이 있는 것인지 명확하지 않다.
        - ‘아래 나열한 100개의 수중에서 가장 작은 수를 찾으시오’
          - » 가장 작다는 것이 **자연수**를 말하는 것인지, **정수**를 말하는 것인지 애매하다
  - l 애매모호함과 불 완전성 ambiguous and incomplete 존재
- n 자연어 표현 알고리즘
  - 4 라면 요리 설명서
  - 4 레시피 등

## 2.1.1 의사코드

---

- n 코드 비슷하다는 의미 (프로그램 코드와 비슷한 자연어)
- n 컴퓨터 안에서 실행하는 코드가 아님 )
- n 프로그래밍 언어의 문장 의미를 나타내려고 만든 영어 형식의 문장 구조
- n 구조화가 잘된 몇 개의 동작 구조로만 알고리즘을 작성
- n 여러 종류의 프로그래밍 언어와 구조나 문장에서 유사
  - | 알고리즘의 각 동작을 프로그램 문장으로 간단하게 대응
  - | 문제 해결 구상 단계에서 빠르게 알고리즘 구조를 잡을 때 유용

# 의사코드 예

이차방정식  $ax^2+bx+c=0$ 에서  $a, b, c$ 는 실수이고,  
 $a \neq 0$ 일 때 근을 구하는 동작으로의 표현

단계1:  $a, b, c$  세 개의 실수 값을 입력 받는다.

단계2: 세 개의 값으로  $D \leftarrow b^2 - 4ac$ 를 계산하라.

단계3: 만약  $D > 0$ 이면, 아래를 계산한다. // 방정식의 근  $x_1, x_2$  두 개

$$x_1 \leftarrow (-b + \sqrt{D}) / (2 \times a)$$

$$x_2 \leftarrow (-b - \sqrt{D}) / (2 \times a)$$

결과  $x_1, x_2$  값을 출력한다.

만약  $D = 0$ 이면, 아래를 계산한다. // 방정식의 근  $x$  한 개(중근)

$$x \leftarrow -b / (2 \times a)$$

결과  $x$  값을 출력한다.

만약  $D < 0$ 이면, 아래를 수행한다. // 방정식의 근은 허근

결과로 “허근을 가짐” 출력한다.

단계4: 종료한다.

n 교제 그림 1.1

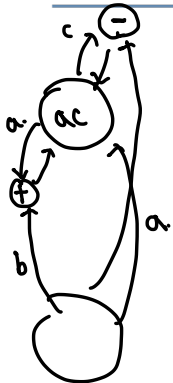
# 기계어(컴퓨터 이해)

주소	내장 프로그램(16-비트 이진수)
01100000	1010101111000010
01100010	1011101010100101
.....	.....
01110100	0011001010000111
01110110	0000111101011111
01111000	0000111101011111
.....	1111100000010100
	.....

Machine language

[그림 1.2] 컴퓨터 메모리 주소와 내장 프로그램

# Assembly Language



standby the  
for

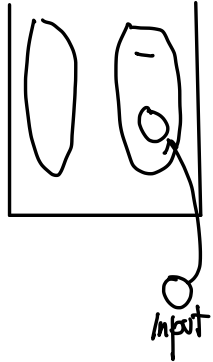
Read a  
Read b  
Read c  
Load b  
Mult b  
Store sb  
Load a  
Mult c  
Mult 4  
Store fac  
Load sb  
Minu fac  
JEQZero 1  
JGTZero 2  
JLTZero 3

1 실근 한 개를  
구하고 출력

2 실근 두 개를  
구하고 출력

3 Print " 답없음"  
Print

stop  
a  
B  
C  
Sb  
fac



# Program with goto(jump)-Fortran



Read a, b, c

$D = b*b - 4 * a * c$

if  $D = 0$  goto L1

if  $D > 0$  goto L2

if  $D < 0$  goto L3

L1  $x = -b / 2 * a$

print x

goto End

L2  $x = (-b + \text{rood}(D)) / 2 * a$

print x

$x = (-b - \text{rood}(D)) / 2 * a$

print x

goto End

L3 print "실근이 없음"

End stop

Data a

Data b

Data c

Data x

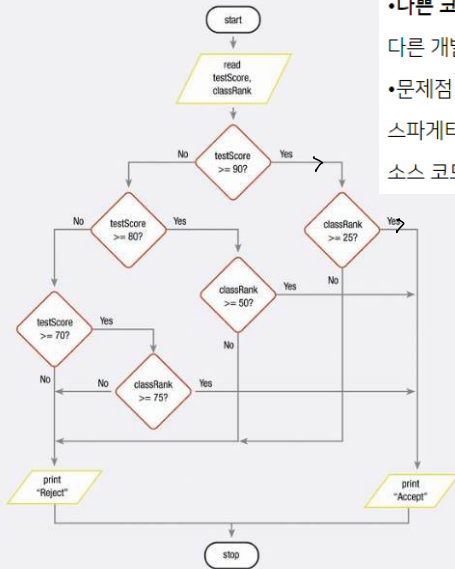
Data D



# 프로그래밍 언어의 한계

## Spaghetti code with goto

FIGURE 2-2: SPAGHETTI CODE EXAMPLE



[소프트웨어개발 프로그래밍] 스파게티 코드 예

### •나쁜 코드(Bad Code)

다른 개발자가 로직을 이해하기 어렵게 작성된 코드

### •문제점

스파게티 코드의 경우 찾은 오류가 발생할 가능성 있음

소스 코드 이해가 안 되어 계속 덧붙일 경우 코드 복잡도 증가

# SOFTWARE 위기

n F. L. 바우어가 처음 1968년 독일 가미시에서 열린 첫 번째 나토 소프트웨어 공학 학회

[1]

↓ 4년

n 에츠허르 데이크스트라의 1972년 ACM 튜링상 수상 연설[2]

| 소프트웨어 위기의 주요한 위기는 컴퓨터 성능이 몇수십배나 더 강력해졌기 때문입니다! 심하게 말하면: 컴퓨터가 없었을 때는 프로그래밍에는 전혀 문제가 없었습니다! 느린 컴퓨터 몇 개 뿐이었을 때는 프로그래밍이 조금 문제가 되었고, 이제는 거대한 컴퓨터에 프로그래밍도 비슷하게 거대한 문제가 되었습니다.

- 프로젝트 예산이 초과되었다.
- 프로젝트 일정이 지연되었다.
- 소프트웨어가 비효율적이었다.
- 소프트웨어 품질이 낮았다.
- 소프트웨어가 요구 사항을 만족시키지 못하는 일이 빈번히 일어났다.
- 프로젝트는 관리 불가능했고 코드 관리는 힘들었다.
- 소프트웨어가 고객의 손에 전달 되지 못했다.

전혀를 생각해 시작..

# 위기대응방안 - 위키피디아

---

- 대응 방안[[편집](#)]
- 여러 소프트웨어 공학 수단을 더 적극적으로 활용하는 것이 한가지 해결책이 될 수 있다.
- 객체 지향 프로그래밍, 캡슐화
- 구조적 프로그래밍
- 통합 개발 환경, 신속 응용 프로그램 개발 IDE
- 소프트웨어 컴포넌트화
- 소프트웨어 프로토타이핑
- 애자일 개발 프로세스
- 반복형 개발
- 버그 / 이슈 관리 시스템, 버전 관리 시스템
- 디자인 패턴
  - Model+View+Controller
- 가비지 콜렉션
- 멀티스레드 프로그래밍

# 구조적 프로그램

---

- n 스파게티코드를 근본적으로 개선할 수 있을까?
- n 문제점
  - l 나는 200K 라인의 스파게티 코드를 물려 받았다 – 지금 무엇? (gastack.kr)

# Structured programming

- n 1972, Dijkstra published the "Notes on structured programming" article
- n restricting program control to three structures: **sequence**, **selection**, and **repetition**. Dijkstra's work "triggered the structured programming movement
- n Structured Programming2.pdf (iicseonline.org)

goto 문 없애자!



# 컴퓨터의 기본 동작

---

- n 입력 (외부장치 → 기억장치)
- n 출력 (기억장치 → 외부장치)
- n 계산
  - | 자료저장 + × - ...
  - | 산술계산 등
- n 다음 동작 순서 지정
  
- n Remind
  - | Computation
  - | 일련의 "동작과 다음 순서"

## 2.1.2 순차동작

쉬운 코드 (goto x)

if (0)

n 산술 계산 문장과 결과를 저장하는 문장

n '변수'의 값에 '산술 수식' 결과를 넣어라.

변수  $\leftarrow$  산술 수식

n 변수는 값이 저장되는 위치를 나타내는 이름

'D'의 값을 ' $b \times b - 4 \times a \times c$ '로 넣어라.

$D \leftarrow b \times b - 4 \times a \times c$

n 알고리즘 : 문제를 해결하는 절차

| Do; do; do; do; do; do ;

| Do  $\square$  statechanged  $\square$  do  $\square$  statechanged  $\square$  do  $\square$  state  
changed  $\square$  do  $\square$  state changed .....  $\square$  final state (end)

## 예

n D에 어떤 값 26이 저장한 상태라면

변수 ← D

26

n  $b \times b - 4 \times a \times c$ 를 계산한 새로운 결과 값

| a, b, c 값이 각각 2, 3, 1 □  $9 - 4 * 2 * 1$  □ 1

n '변수' D의 값에 저장된 결과

D

1

n 변수 D는 값이 저장되는 위치를 나타내는 이름



# 입력과 출력

- n 입력: **read** 변수, 변수, ...
- n 출력: **write** 변수, 변수, ...
- n 원의 반지름인 r 값을 읽어 들여라.

**read** r

**write** area

**write** "sorry, addition error."

**write** "Can not find desired Number."

**write** "홀수를 입력하세요."

의사코드

# 3개학급의 성적 하계 및 평균(의사코드)

n Total ← 0

n Read Score1 90

n Total = Total + Score1 90

n Read Score 2 100

n Total = Total + Score2 190

n Read Score 3 80

n Total = Total + Score3 270

n Average = Total / 3 90

n Write Total, Average

n Stop

n Total ← 0

n Read Score1

n Read Score2

n Read Score3

n Total = Score1 + Score2 \* Score3

n Average = Total / 3

n Write Total, Average

n Stop

## 2.1.3 조건동작

---

- n 실행 도중에 필요한 부분을 선택하거나, 반복하여야 하는 경우
- n **조건 동작**이나 **반복 동작**을 하도록 의사코드를 추가
- n 필요한 부분을 선택하거나 조건에 따라 특정 부분을 반복하여 실행 동작
  - | **제어 동작**

**if** '참/거짓에 대한 조건'**then**

동작 블록1

**else**

동작 블록2

# 조건 동작 순서도

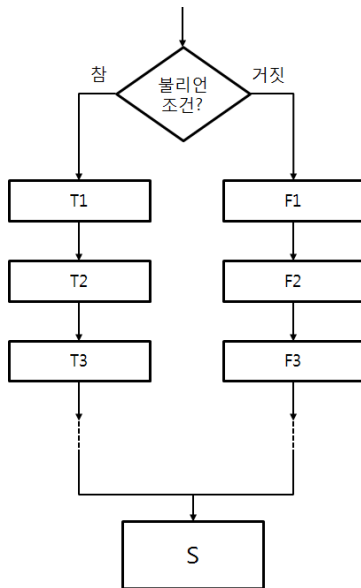


그림 2.3 **if..then..else**문장에 대한 순서도

---

.  
.  
단계3 **if**  $D > 0$  **then** // 방정식의 근  $x_1, x_2$ 는 각각

$x_1 \leftarrow (-b + ) / (2 \times a)$  //블록1

$x_2 \leftarrow (-b - ) / (2 \times a)$

**write**  $x_1, x_2$

**else if**  $D = 0$  **then** // 방정식의 근은 한 개의 실근(중근)을 가짐

$x \leftarrow -b / (2 \times a)$  //블록2

**write**  $x_1$

**else if**  $D < 0$  **then** // 방정식은 허근을 가짐

**write** “허근입니다” //블록3

.  
.

30일 동안 매일 입금한 금액을 합하고, 하루 평균 입금액을 구하는 알고리즘

(조건과 반복 동작 버전)

단계1 Money  $\leftarrow$  0

단계2 Day  $\leftarrow$  1

단계3 **read** Daymoney

단계4 Money  $\leftarrow$  Money + Daymoney

단계5 **if** Day < 30 **then**

Day  $\leftarrow$  Day + 1

**goto** 단계3

**else**

AveMoney  $\leftarrow$  Money  $\div$  30

**write** Money, AveMoney

단계6 **stop**

그림 2.4 30일 동안 저축 총액수와 하루 평균 입금액을 계산하는 알고리즘

## 2.1.4 반복 동작

---

- n 반복하여야 하는 경우
- n **반복 동작**을 하도록 의사코드를 추가
- n 순환 몸체

**while** ('참/거짓에 대한 조건') **do**

.

.

단계i: 동작

단계i+1: 동작

.

.

단계j: 조건을 변화시키는 동작

**endwhile**

# While ...반복동작 순서도

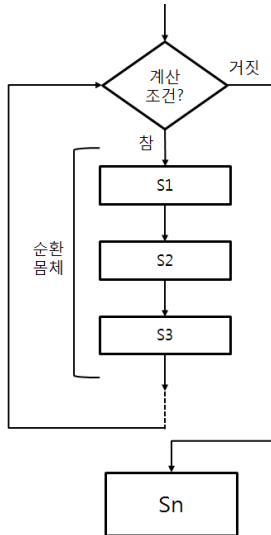


그림 2.5 **while** 순환 구조와 실행



# 반복 동작 예

---

n **while** 순환에 대한 간단한 예를 살펴보자.

```
단계1  Num  $\leftarrow$  1
단계2  Table  $\leftarrow$  1
단계3  while (Num  $\leq$  30) do
단계4      Table  $\leftarrow$  Table  $\times$  Num
단계5      write Table, Num
단계6      Num  $\leftarrow$  Num + 1
단계7  endwhile
```

## 반복 동작 다른 예-반복 계속 결정

---

- n 사용자 입력에 따라 순환을 진행하고, 입금한 총액을 출력하는 알고리즘  
(사용자 입력에 의한 반복 동작 버전)

repeat  $\leftarrow$  Yes

Money  $\leftarrow$  1

**while** (repeat = Yes) **do**

**read** Daymoney

    Money  $\leftarrow$  Money + Daymoney

**write** “다시 실행하려면 Yes를, 끝낼면 No를 입력하세요.”

**read** repeat

**endwhile**

**write** Money

**stop**

# Do ...반복 동작

---

## n 나중 검사

**do**

동작1

동작2

.

.

동작i: 조건을 변화시키는 동작

**enddo**(참 거짓 조건 검사)

## Do ...반복 동작 순서도

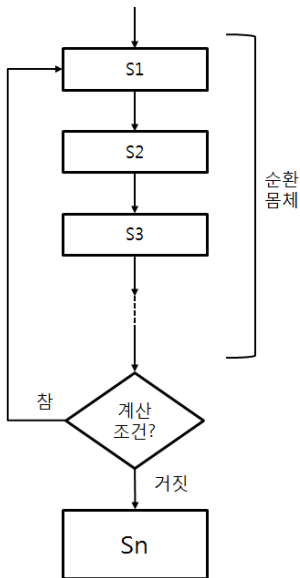


그림 2.7 나중 검사 **do..while** 순환 구조

# 정한 횟수 만큼 반복

**for** ( $i \leftarrow$  초기 값, 증가 값, 마지막 값) **do**

    단계1: 동작1

    단계2: 동작2

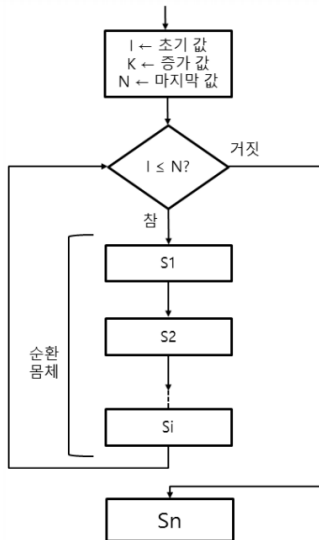
    .

    .

    .

    단계i: 동작i

**endfor**



---

## 2.2 알고리즘으로 하는 문제 해결의 예

## 2.2.1 구분구적법에 의한 적분하기

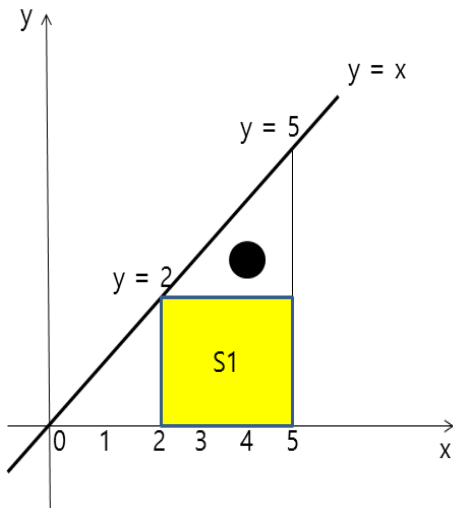
---

n 일차방정식  $y = x$ 의 적분 값  $y = \int_2^5 x dx$

n 구분구적법으로

n 그림 2.10(a) 같이  $(x, y)$  좌표가 각각  $(2, 0)$ ,  $(2, 2)$ ,  $(5, 0)$ ,  $(5, 5)$  인  $x = 2$ 에서  $x = 5$ 까지를 1등분한 사각형의 면적  $S_1 = (5 - 2) \times (2 - 0) = 6$ 를 구하면 된다.

n 이 때 사각형과  $y = x$  직선 사이의 오차 공간(● 표시)이 생기게 된다. 이렇게 구해진 사각형의 면적은 오차 공간의 크기가 너무 커서 정확한 적분 값으로 판단할 수 없다.

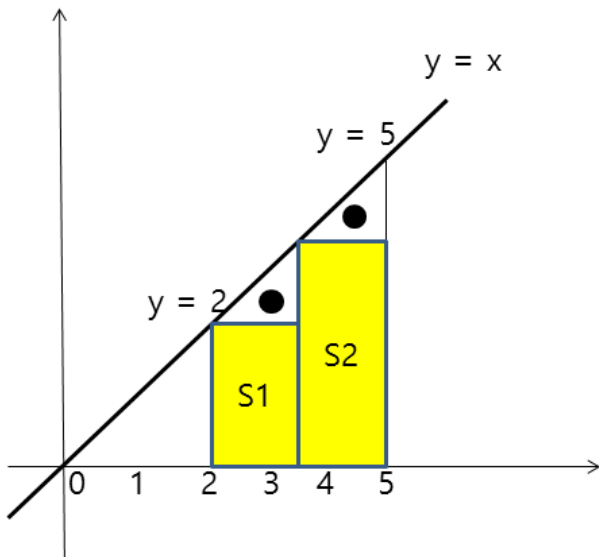




이 오차 공간을 줄이기 위해 그림 2.10(b)와 같이  $x = 2$ 에서  $x = 5$ 까지를 2등분하고, 두 개의 사각형 면적을 각각 구하여 합한다. 즉, 아래처럼 계산한 적분 값  $I = S1 + S2 = 3.0 + 5.25 = 8.25$ 이다.

$$S1 = (3.5 - 2) \times (2 - 0) = 1.5 \times 2 = 3.0$$

$$S2 = (5 - 3.5) \times (3.5 - 0) = 1.5 \times 3.5 = 5.25$$



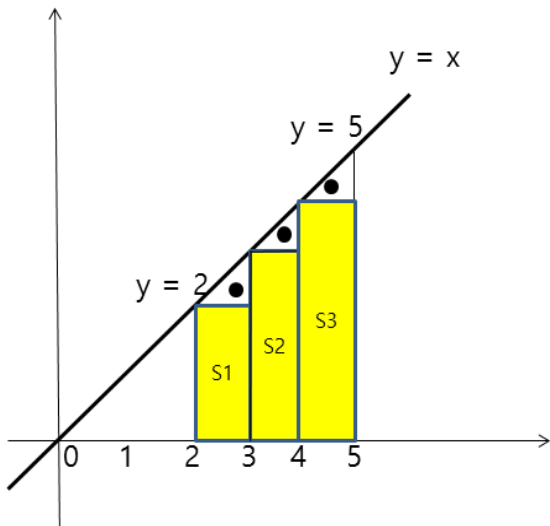
n 구간을 3등분하면

$$S1 + S2 + S2 = 2 \\ + 4 + 5 = 11 \text{이다.}$$

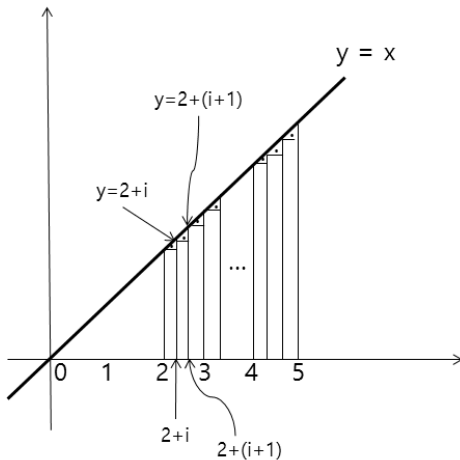
$$S1 = (3 - 2) \times (2 - 0) \\ = 1 \times 2 = 2$$

$$S2 = (4 - 3) \times (4 - 0) \\ = 1 \times 4 = 4$$

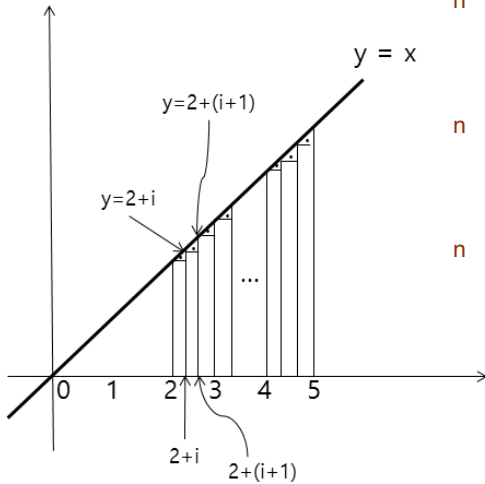
$$S3 = (5 - 4) \times (5 - 0) \\ = 1 \times 5 = 5$$



- n 그러면 사각형을 몇 등분으로 나누고 각 사각형의 합을 구해야 될까? 또 몇 등분으로 나누면 된다는 명확한 답은 있을까? 몇 등분하면 좋다는 명확한 답을 알 수 없다.



(d)  $n$ 등분 하였을 때



$n$   $x_0 = 2, x_n = 5$

$n$  막대기 개수  $n$

$n$  막대기 넓이

| 넓이 :  $(5-2)/n$

| Width:  $(x_n - x_0)/n$

$n$   $i$  번째 막대기 높이

|  $2 + i$

|  $x_0 + i$

$n$   $i$  번째 막대기 넓이

|  $(2 + i * (5-2)/n) * (5-2)/n$

|  $(x_0 + i * \text{width}) * \text{width}$

For( $i=0, i < n, i++$ ) {

Area +=  $(x_0 + i * \text{width}) * \text{width}$

}

## 함수 area(x0, xn, n)

n x0-xn 구간을 n개 구간으로 나누어 n개의 면적을 더한다.

```
n int area( x0, xn, n ) {  
    | sum = 0, width = (xn - x0)/n  
    | for(i=0;i<n; i++)  
        4 Sum = sum + (x0+i*width) * width  
    | return sum;  
n }
```

int area(x0, xn, n) {

sum = 0, width = (xn - x0) / n

for (i = 0, i < n, i++) {

Sum = sum + (x0 + i \* width) \* width

}

return sum;

## 수렴 오차

- n 사각형 면적의 합이 일정 값에 수렴한다는 것을 알기 위해서는  $N-1$ 등분하여 구한 적분 값  $I(N-1)$ 과  $N$ 등분하여 구한 적분 값  $I(N)$ 과의 차이를 비교한다.
- n 만약 차이가 거의 없으면 더 이상 사각형으로 쪼갤 필요가 없다.
- n 그러므로 이때까지 수렴한 값인  $I(N-1)$ 나  $I(N)$  중에서 한 값을 적분 값으로 결정한다. 수렴 값의 차이가 없다는 것은 정해진 **일정한 값보다 적다**는 의미이다.
- n 컴퓨터에서 실수 값 연산은 유효자리 수 때문에 정확하게 제로(0)를 만들지 않는 경우가 발생할 수도 있다.
- n 그러므로 수렴 값의 차이를 비교하려면 아래처럼 일정한 값과 비교하는데, 오차 범위 안에 있으면 두 값  $I(N-1)$ 와  $I(N)$ 은 같은 값으로 판단한다.

$$\left( -\text{오차 값} \leq I(N-1) - I(N) \leq +\text{오차 값} \right)$$

## 알고리즘(교재 반영 수정필요)

$$u = \int_2^5 x dx$$

error=0

area0, area1=0

read x0, xn

n=1

Do {

area 0 = area1

area1 = area(x0, xn, n)

n = n+1

} While( 절대값(area1-area0) > 0.00001 )

Write “적분값:“

Write area1.

stop

교재에든 여치에 for loop를  
넣었음

## 2.2.2 연락처에서 전화번호 찾기

- n 스마트폰에 들어있는 연락처에서 찾으려는 사람의 **이름**을 입력하고 이름에 해당하는 전화번호를 찾는다
- n **이름이 있는 위치**를 알아내는 것이다.
- n 스마트폰에 1,000명의 이름과 전화번호가 있다
- n 이름은  $N_1, N_2, N_3, \dots, N_{1000}$
- n 각 이름에 대응하는 전화번호는  $T_1, T_2, T_3, \dots, T_{1000}$

$N_1$	$\leftarrow$	$T_1$
$N_2$		$T_2$
$N_3$		$T_3$
.		.
.		.
.		.
$N_{1000}$		$T_{1000}$



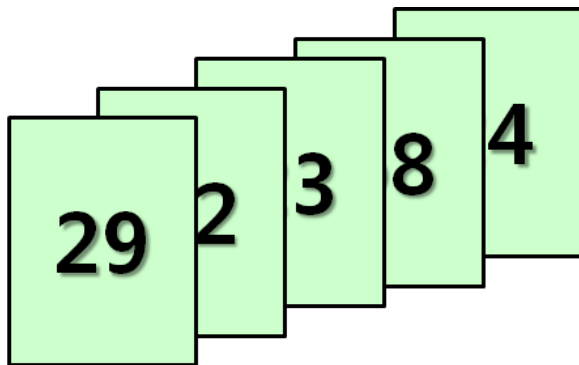
# 알고리즘

```
(1)    read N1, ..., N1000 // 목록에 이름 입력
(2)    read T1, ..., T1000 // 목록에 전화번호 입력
(3)    read NAME // 찾으려는 이름 입력
(4)    i ← 1
(5)    Found ← NO
(6)    while (i ≤ 1000)
(7)        if NAME = N[i] then // 찾으려는 이름과 목록 이름 비교
            write “찾는 전화 번호 입니다”, T[i]
(8)            Found ← YES
        endif
        i = i + 1
    endwhile
(9)    if Found = NO then
        write “찾는 이름이 없습니다”
11 Stop
```

## 2.2.3 나열한 수에서 가장 큰 수 찾기

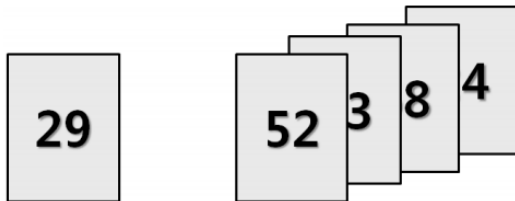
- n 알고리즘인 FindMax 알고리즘은 순서가 없이 나열된 수를 올림차순으로 정렬하는 것이다.
- n 올림차순 정렬은 나열한 수에서 가장 큰 수를 찾아서 제거한다. 그리고 나열한 나머지 수에서 다시 가장 큰 수를 찾아서 제거한다.

나열한 수	29	52	23	68	24
	↑	↑	↑	↑	↑
위치	1	2	3	4	5



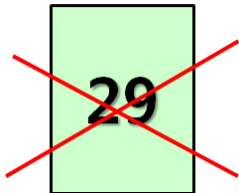
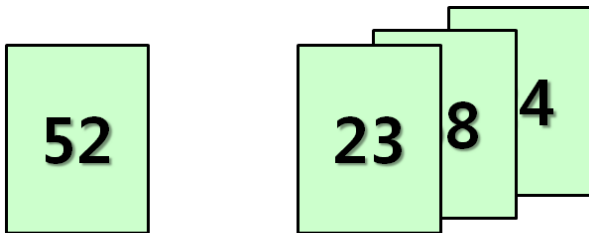
## 일단 첫 장을 제일 큰수로 가정

---



비교후 큰수를 남긴다.

---



---

52

68 4

~~23~~

# 알고리즘

## 가장 큰 수를 찾는 알고리즘 FindMax

```
read N                      // 나열한 수의 길이
read A1, A2, . . . , An    // 나열한 수
Max ← A[1]
position ← 1
k ← 2
while (k ≤ N) do           // 비교
    if A[k] > Max then
        Max ← A[k]
        position ← k
        k ← k + 1
    else
        k ← k + 1
endwhile
write Max, position
stop
```

## 2.2.4 순서가 있는 자료에서 원하는 데이터 찾기

n 순서가 중요

n 이진탐색

n 순서화된 n개의 자료에서 가운데( $n/2$ )를 지정

0	1	2	3	4	5	6	7	8
first				Mid				last
8	15	20	26	41	47	80	87	93



## 방법

---

- (1) 우선 자료는 올림차순으로 정렬되어 있다고 하자(입력).
- (2) 처음 자료배열의 중간 값을 선택하여, 그 값을 찾고자 하는 값과 비교한다.
- (3) 선택한 중앙값이 찾는 값이면 탐색에 성공한 것이다.
- (4) 처음 선택한 중앙값이 찾는 값보다 크면, 찾고자 하는 데이터는 중앙값 앞쪽에 있음을 알 수 있다. 또는 중앙값이 찾는 값보다 작으면 찾는 데이터는 중앙값 뒤에 있다.
- (5) 이런 방식으로 검색 영역을 줄이면 검색이 반복될 때마다 찾는다.

$$\text{Mid} = \lceil (\text{first} + \text{last}) / 2 \rceil$$

- n 여기에서 다시 first를 Mid 위치부터 마지막 까지 탐색해야 하므로,  $\text{first} \leftarrow \text{Mid}$ 로 대치한다. 그러면 아래와 같다.

0	1	2	3	4	5	6	7	8
무시	무시	무시	무시	first				last
8	15	20	26	41	47	80	87	93

- n 여기에서 찾으려는 데이터가 47이므로  $\text{Mid} = (\text{first} + \text{last}) / 2 = (4 + 8) / 2 = 6$ 로  $A[6]$ 을 비교한다. 즉,  $A[\text{Mid}]$ 는  $A[6] = 80$ 으로 찾는 데이터가 아니다.
- n  $A[\text{Mid}]$  값이 47보다 크다. 그러므로 새로운 탐색 영역은 Mid 앞 부분인 인덱스 4와 인덱스 5 사이이다. 다시 last를  $\text{last} \leftarrow \text{Mid}$ 로 대치한다. 그러면  $\text{Mid} = (\text{first} + \text{last}) / 2 = (4 + 6) / 2 = 5$ 로  $A[5]$ 을 비교한다. 즉,  $A[\text{Mid}]$ 는  $A[5] = 47$ 로 찾는 데이터이다.
- n 작은 부분이나 큰 부분을 무시하였으므로 전체적으로 탐색 속도는 순차 탐색보다 빨라진다.

---

## 2.3 알고리즘의 분석과 효율성 분석 예

```
(1)      read N1, ..., N1000 // 목록에 이름 입력
(2)      read T1, ..., T1000 // 목록에 전화번호 입력
(3)      read NAME // 찾으려는 이름 입력
(4)      i ← 1
(5)      Found ← NO
(6)      while (i ≤ 1000)
(7)          if NAME = N[i] then // 찾으려는 이름과 목록 이름 비교
              write “찾는 전화 번호 입니다”, T[i]
(8)              Found ← YES
(9)              break;
          endif
          i = i + 1
      endwhile
(10)     if Found = NO then
          write “찾는 이름이 없습니다”
11 Stop
```

## 예

$n$  이름이  $n$ 개 있는 연락처를 순차 탐색하는 알고리즘을 수행하면 비교회수는 아래와 같다.

최선일 때

1

$O(n)$

평균일 때

$n/2$

최악일 때

$n$

$N=50,000,000$ , 비교에 1초 필요하다면, 50,000,00 만초 13889시간 소요

