



## Day26


📅 Date	@04/08/2022
🕒 작성일시	@2022년 8월 4일 오전 9:31
🔍 강의 번호	BD101
☰ 유형	Algorithm Data
👤 강사명	AustinYoon
☑ 강의자료	☑
☑ 노션 복습	☑
☑ 코딩 복습	☐
☑ 주말속제(교제)	☐
☑ 정리	☑

Day26

Aug

AlgorithmLab/Day26/src at main · Kray273/AlgorithmLab  
Contribute to Kray273/AlgorithmLab development by creating an account on GitHub.

Kray273/  
AlgorithmLab



<https://github.com/Kray273/AlgorithmLab/tree/main/Day26/src>

1 Contributor  
0 Issues  
0 Stars  
0 Forks

<https://github.com/Kray273/SQLLab>

### ▼ 에라토스테네스의 체\_Sieve of Eratosthenes

- 소수 prime number를 찾아내는 알고리즘
  - 소수는 2 이상의 정수에서 1과 그 수 자체로만 나눌 수 있는 수
  - 소수는 나열되어 있는 구간이 불규칙하므로 임의로 찾기가 힘들다.

👉 소수는 2 이상의 정수 중에서 1과 그 수 자신 외에는 나눌수 없는 숫자.  
10이하에서는 2,3,5,7이 소수에 해당한다.  
소수 prime에서의 '소'는 합성되지 않은 소박한 숫자라는 뜻을 가진다.  
모든 수의 소(근본)을 의미하기도 한다.

소수인지 아닌지를 구분하는 것은 의외로 어렵다.

👉 소수의 내용만으로 보면 어렵지 않아보이지만 사실은 의외로 아주 어렵다. 무엇이 어려운지도 바로 찾아내기 어렵다. 예로 3의 배수는 3,6,9,12,15 처럼 3개의 간격으로 나열된다. 따라서 1에서 100까지의 사이에 있는 3의 배수를 찾는 일은 간단하다. 하지만 소수는 규칙성이 없기에(규칙이 있다고 생각하는게 리만가설) 간격이 불규칙하고 랜덤하다. 즉 소수를 한번에 열거하기가 어렵다.

- 소수를 찾아내는 방법
  - 고대 그리스의 과학자인 에라토스테네스는 모든 숫자로 나눠서 소수를 찾는 방법을 개선하여 소수를 효율적으로 발견하는 방법을 알아냈다. 그의 이름을 따서 '에라토스테네스의 체'라고 부른다.

## ▼ Sieve of Eratosthenes

- 어떤 수 이하의 범위에 존재하는 모든 소수를 찾고 싶은 경우

| '그 수의 제곱근보다 작은 소수의 배수만 없애면 남은 소가 소수다'

라는 생각을 바탕으로 소수를 찾는 방법이다.

	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	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

- 👉 예를 들어
1. 100 이하의 소수를 모두 찾아내려면 먼저 100의 제곱근 이하 소수를 선택한다.
  2. 루트 100의 제곱근은 10이다. 즉 제곱하면 100이 되는 수는 10이다.
  3. 10 이하에서 소수는 2,3,5,7, 네개이다.

👉 우선 2에서 100까지의 표에서 2로 나눌 수 있는 수를 2를 제외하고 나머지를 모두 삭제한다.

맨 처음에 소수인 2를 발견한 후 2의 배수를 모두 지운다.

	2	3		5		7		9	
11		13		15		17		19	
21		23		25		27		29	
31		33		35		37		39	
41		43		45		47		49	
51		53		55		57		59	
61		63		65		67		69	
71		73		75		77		79	
81		83		85		87		89	
91		93		95		97		99	

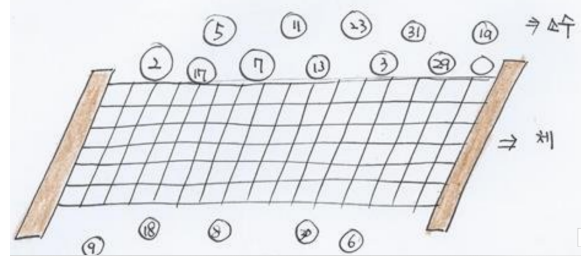
이런 방식으로 3보다 큰 수에 대해서도  $n*n$ 부터 지워주면 된다.

	2	3		5		7			
11		13				17		19	
		23		25				29	
31				35		37			
41		43				47		49	
		53		55				59	
61				65		67			
71		73				77		79	
		83		85				89	
91				95		97			

👉 같은 방식으로 5와 7을 모두 제거하면 다음과 같이 된다.

👉 이 방법을 사용하면 1에서 100까지의 모든 숫자를 해당 숫자보다 작은 수로 나눌 수 있는 지 하나하나 전부 순서대로 확인하는 것보다는 훨씬 더 빨리 소수를 찾을 수 있다.

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	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100



## ▼ Algorithm

- 에라토스테네스의 체는 크게 3개의 처리로 구성한다.
  - 어떤 수 이하의 모든 정수 데이터 준비
  - 어떤 수의 제곱근 보다 작은 소수의 배수들을 차례로 제거한다.
  - 마지막까지 남은 수들을 출력한다.



우선 10이하의 소수를 구하는 경우로 생각해보자.

- 10이하의 정수 데이터를 준비한다.
- 10의 제곱근 약 3.16보다 작은 소수의 배수를 차례대로 제거한다.
- 마지막까지 남은 수들을 출력한다.

### ▼ (1) 10 이하의 정수 데이터들을 준비한다.

- 먼저 체의 대상이 되는 10까지의 정수를 데이터로 준비하자. 여기에서는 11개의 요소를 가지는 정수형 배열을 준비한다. 배열의 이름은 arr로 정하고, 첨자는 0부터 시작하기 때문에 첨자를 10까지 동일하게 하기 위해 요소를 11개를 준비한다.

0	1	2	3	4	5	6	7	8	9	10

- 요소는 그 첨자가 소수인지의 여부를 판정하는 데이터를 넣는 것으로 사용하자. 즉, 소수가 아닌 것으로 판정된 첨자의 요소에 '소수가 아니다'라는 것을 나타내는 데이터를 넣자. 다시말해 **소수의 가능성이 있는 경우에는 1을 대입하고, 소수가 아닌 경우에는 0을 대입하자.**
  - 위의 개념 소개에서 사용한 소수가 아닌 수를 제거하는 방식을 0을 대입하는 방식으로 처리한다.
  - 따라서, 초기값을 1로 전부 대입해둔다. 그리고 소수가 아닌 것으로 판정된 수(첨자)의 요소에는 0을 대입한다.
  - 이러한 방식을 통해 소수의 배수를 모두 제거(0을 대입한 요소들)한 후 '1'이 남아 있는 요소들의 첨자는 모두 소수이고 0이 대입되어 있는 요소의 첨자는 '소수가 아니다'라고 구별할 수 있다.

1	1	1	1	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10

### ▼ (2) 10의 제곱근은 약 3.16이므로 제곱근의 이하의 소수 2와 3의 배수들을 모두 제거한다.

- 먼저 2의 배수들을 모두 지운다. 이때 지운다는 의미를 0을 대입한다는 표현으로 적용한다.

1	1	1	1	0	1	0	1	0	1	0
0	1	2	3	4	5	6	7	8	9	10

- 다음으로 3의 배수에 0을 대입한다.

1	1	1	1	0	1	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10

▼ (3) 마지막까지 남은 수들을 출력한다.

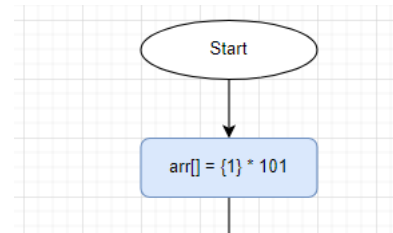
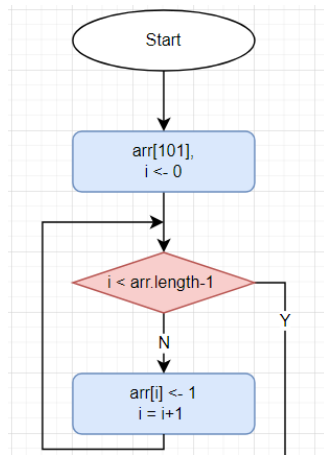
- 제곱근 이하의 소수들의 모든 배수를 제거(0을 대입)했다면 나머지는 모두 소수임으로 요소의 값이 '1'인 첨자는 소수이다.
- 첨자가 2이상이고 1이 들어 있는 요소들의 첨자들만 뽑아낸다.

1	1	1	1	0	1	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10

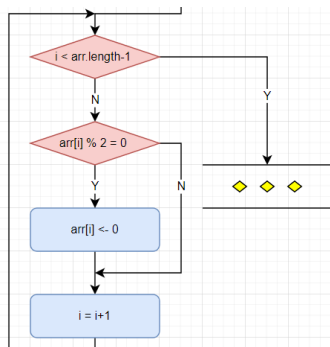
▼ Flow chart

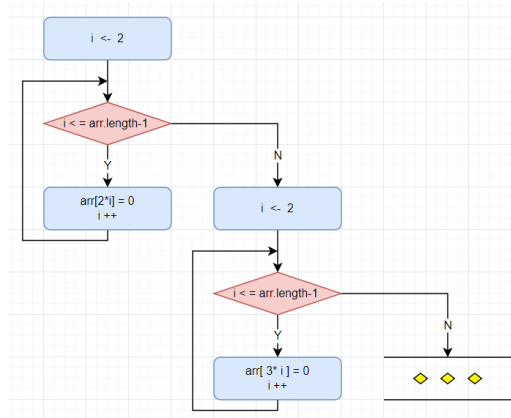
- 수를 늘려서 100 이하의 소수를 모두 구하는 경우로 생각해보자.

▼ (1) 100 이하의 정수 데이터들을 준비하고 1을 대입한다.



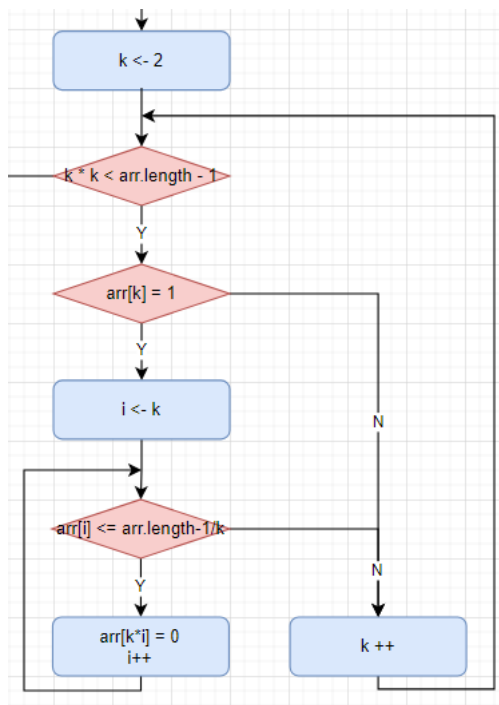
▼ (2) 100 제곱근인 10, 10 이하의 소수의 배수들을 모두 제거한다.





- K에 대입된 값이 소수인지의 여부에 따라 반복문으로 진입을 해야한다.
  - 따라서 arr[k] = 1이면 소수, arr[k]가 0이면 소수가 아니라는 것이다.

👉 변수 k는 하나씩 증가한다. 2인 경우에는 2의 배수들이 전부 0이되고, 3인 경우에는 3의 배수들을 전부 0이 되는 것을 반복한다. 그러나 4는 소수가 아니다.  
3 다음의 소수는 5이므로 5의 배수를 제거해야한다. 따라서 k값이 소수인지를 판단하는 처리를 추가해야한다. 허나. 2와 3의 배수를 제거하는 처리를 한 뒤에는 이미 2와 3의 배수에는 모두 요소가 0이 되어 있다. 즉. 이 시점에는 첨자가 소수가 아닌 경우 0이 대입되어 있다.



👉 arr[k] = 1 이 Yes인 경우 k는 소수이므로 k의 배수를 제거하는 반복처리가 실행된다. 이와 반대로 No의 경우는 k는 하나 늘려 k가 소수인지 판단하게 된다.

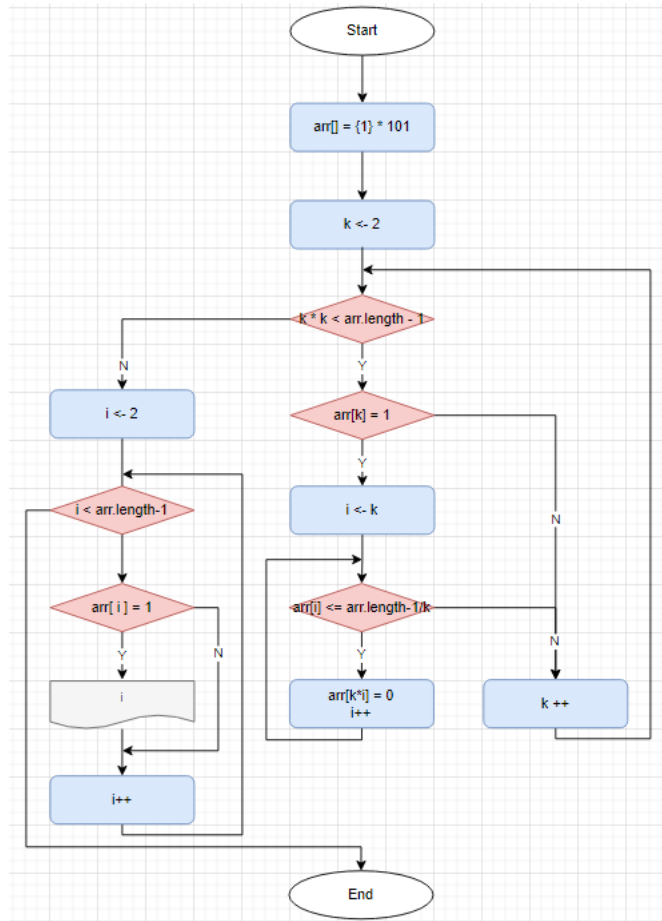
이를 과정을 통해 k=4인 경우 arr[4]는 이미 0이므로 배수를 제거하는 반복처리로 들어가지 않고 하나 증가 하게 된다.

허나... k는 무한히 증가하며 반복에 빠지게 된다.

- K의 무한루프 해결



에라토스테네스의 체는 '어떤 숫자의 제곱근보다 작은 소수의 배수를 제거하면 남은 소가 소수이다.' 라는 이론이다. 따라서 이번 예제에서는 그 어떤 수가 100이므로 k는 100의 제곱근 즉 10 이하가 된다. 따라서 'k가 100의 제곱근 이하인가?' 라는 판별식을 추가하여 반복을 마치고 소수를 출력하자.



## ▼ Java Code

```

import java.util.*;

public class SieveofEratosthenes {

    public static void main(String[] args) {
        // 101개의 배열을 생성하고 0으로 시작
        int[] arr = new int[101];
        System.out.println(Arrays.toString(arr));

        // for ( int i = 1; i < arr.length; i++) {
        //     arr[i] = 1;
        // } //반복구조를 통해 arr를 1로 초기화

        // Arrays.fill(arr,1); //for문을 사용하지 않고 1로 초기화

        //소수를 제외하고 1을 대입
        //for문
        for (int k = 2; k*k <= arr.length-1; k++) {
            if(arr[k] == 0) {
                for (int i = k; i <= (arr.length-1)/k; i++) {
                    if(arr[i] <= (arr.length-1)/k) {
                        arr[k*i] = 1;
                    }
                }
            }
        }
    }
}

```

```
}  
}  
}  
  
//while문  
//    int k = 2;  
//    while(k*k < arr.length-1) {  
//        if(arr[k] == 0) {  
//            int i = k;  
//            while(i <= (arr.length-1) / k) {  
//                arr[k*i] = 1;  
//                i++;  
//            }  
//        }  
//        k++;  
//    }  
  
System.out.println(Arrays.toString(arr));  
  
//소수를 출력(0을 출력)  
for (int i = 2; i < arr.length-1; i++) {  
    if(arr[i] == 0) {  
        System.out.print(i+", ");  
    }  
}  
}  
}  
  
Outputs :  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
[0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0,  
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
```

## ▼ 유클리드 알고리즘\_Euclidean Algorithms

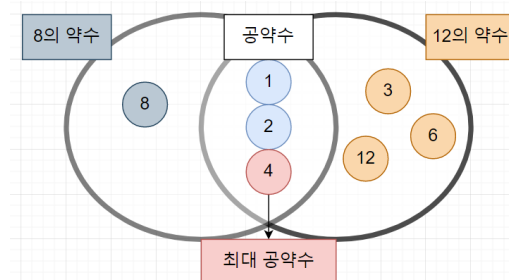
- 최대 공약수를 구하는 알고리즘
  - ▼ 최대공약수는 약수들 중에서 가장 큰수를 말한다.

👉 약수

- 3의 약수  $\Rightarrow 1, 3$
- 4의 약수  $\Rightarrow 1, 2, 4$
- 5의 약수  $\Rightarrow 1, 5$
- 6의 약수  $\Rightarrow 1, 2, 3, 6$
- 8의 약수  $\Rightarrow 1, 2, 4, 8$
- 12의 약수  $\Rightarrow 1, 2, 3, 4, 6, 12$

👉 공약수  
8과 12의 공통된 약수  $\Rightarrow 1, 2, 4$

최대 공약수  
: 두 수의 공약수 중 최대값  
8과 12의 최대 공약수  $\Rightarrow 4$



- 반복구조를 이용하는 중요한 알고리즘

### ▼ 최대공약수를 구하는 절차

1. 먼저 어떤 복수의 수를 소수의 곱셈 형태로 분해하자.  $\Rightarrow$  '소인수분해'



소인수분해

$$8 \Rightarrow 1 * 2 * 2 * 2$$

$$12 \Rightarrow 1 * 2 * 2 * 3$$

2. 이들 중에 공통되는 소수를 서로 곱한 수가 바로 두 수의 최대 공약수이다.



$$1 * 2 * 2 = 4$$

★ 그러나 이런 절차를 반복하는 것은 복잡하다.

어떤 수를 소인수 분해하려면 먼저 그 수의 이하의 소수들을 모두 구해야 한다.  
 그리고 그 소수 중 작은 숫자부터 순서대로 원래의 수를 나누고,  
 나누어지지 않는다면 그 다음 소수의 순서로 계속 계산을 반복해야 한다.  
 따라서 단순히 보이지만 절차는 상당히 복잡해진다.  
 이러한 복잡성에 비해 매우 간단한 방법으로 최대 공약수를 구하는 것이  
 바로 '유클리드 알고리즘'이다.

#### ▼ Algorithm



약 2300년 전의 고대 그리스의 수학자로 수많은 수학적 이론을 생각해냈다. 그 중 하나가 바로 유클리드 알고리즘이다. 간단하게 말하면 두 수의 나눗셈을 반복하여 최대공약수를 구하는 것이다.  
 그러면 어떻게 나눗셈을 반복할까?

큰수를 작은 수로 나눈다\_(%)

나머지를 확인

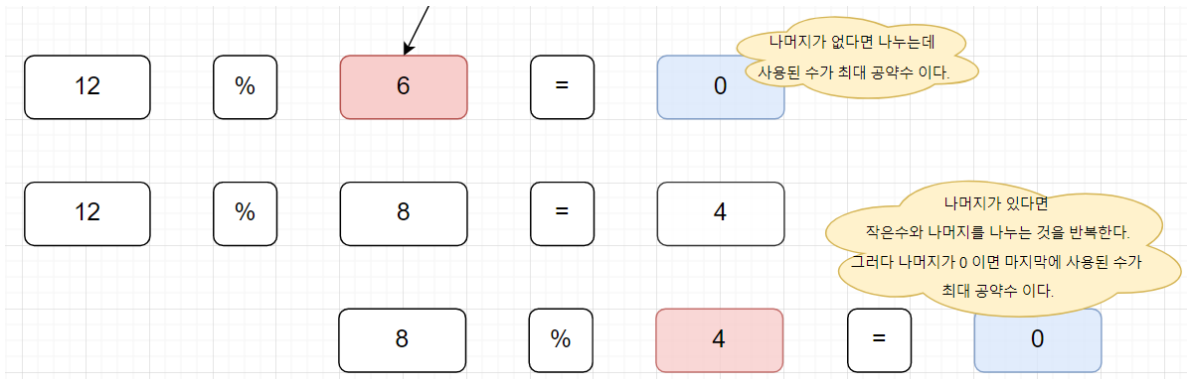
Y - 나머지가 있다.

| 나머지가 없을 때까지 작은수와 나머지를 나눈다\_(%)

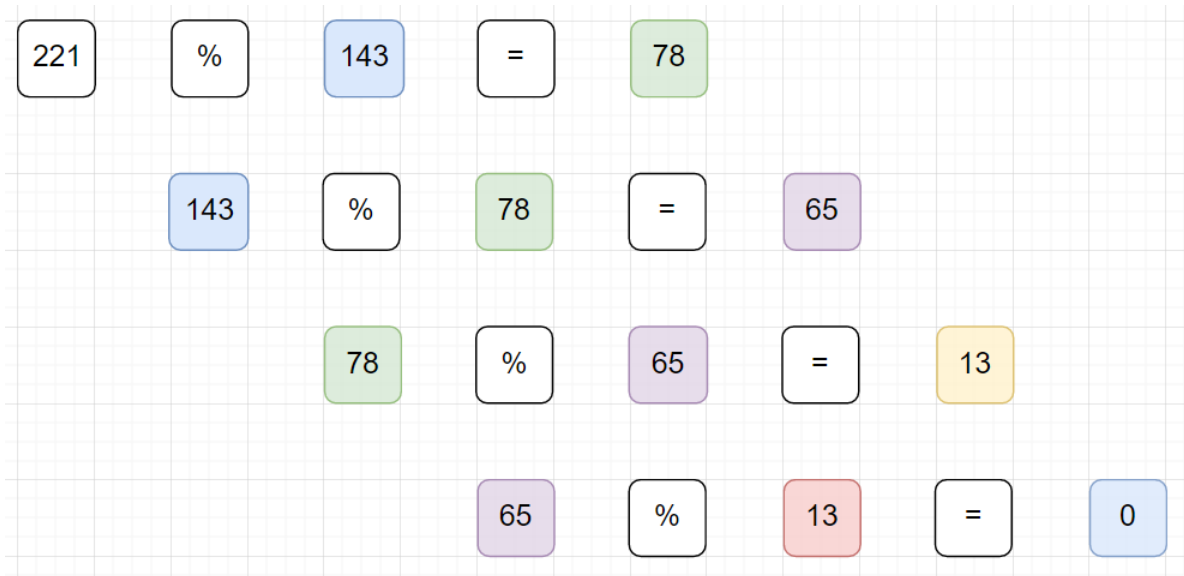
N - 나머지가 없다.

그 때의 나누는데 사용된 작은 수가 바로 '최대 공약수'이다

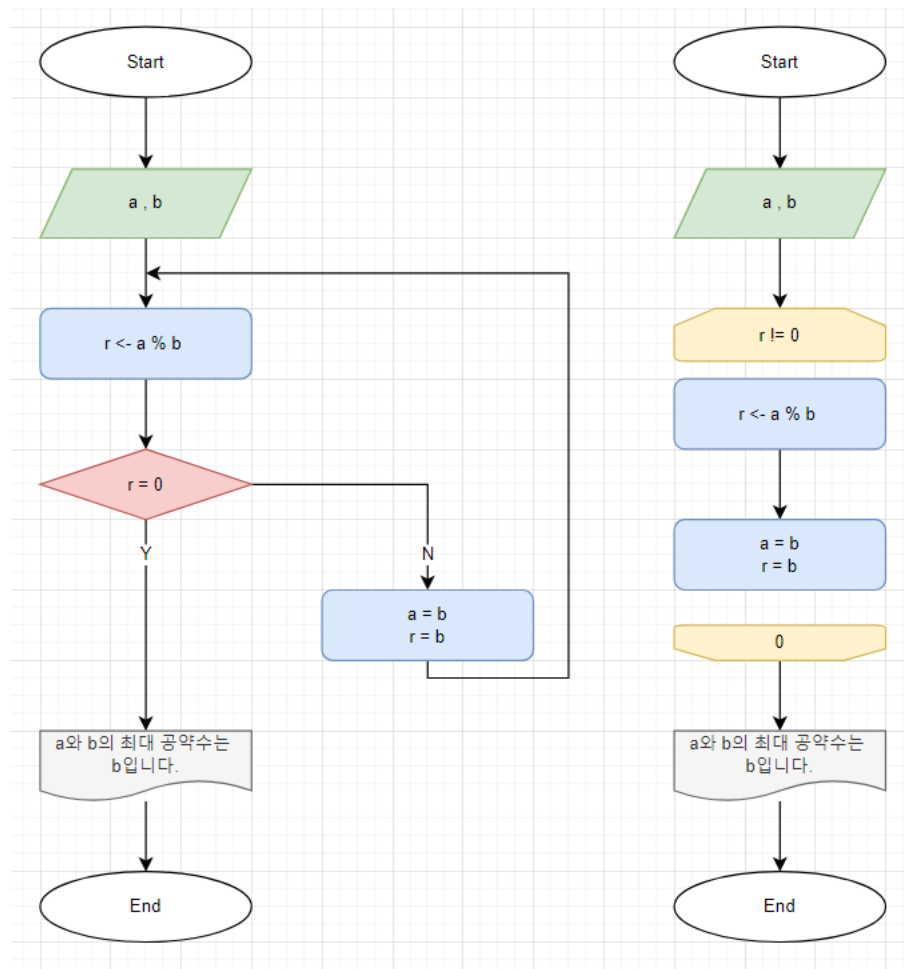
ex) 12로 6을 나누면 나머지는 0, 따라서 최대 공약수 6







▼ Flow chart



▼ Java Code

```
import java.util.*;
public class EuclideanAlgorithms {

    public static void main(String[] args) {
        System.out.print("a를 입력하세요: ");
        Scanner no = new Scanner(System.in);
        int a = no.nextInt();
        System.out.print("b를 입력하세요: ");
        int b = no.nextInt();
        int r = 0;

        do{
            r = a % b;
            a = b;
            b = r;
        } while (r !=0);
        System.out.printf("a와 b의 최대공약수는 %d입니다.", a);
    }
}
Outputs : ex)a: 221, b: 143
a를 입력하세요: 221
b를 입력하세요: 143
a와 b의 최대공약수는 13입니다.
```

## DB

```
-- 테이블 생성
CREATE TABLE string_tbl
(char_fld CHAR(30),
vchar_fld VARCHAR(30),
text_fld TEXT);
```

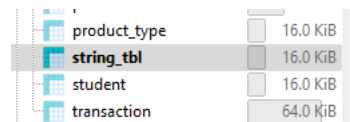


Table Name	Size
product_type	16.0 KiB
<b>string_tbl</b>	16.0 KiB
student	16.0 KiB
transaction	64.0 KiB

```
-- 데이터 삽입
INSERT INTO string_tbl(char_fld, vchar_fld, text_fld)
VALUES ('This is char data',
'This is varchar data',
'This text dara');
```

bank.string\_tbl 4 rows total (approximately)

char_fld	vchar_fld	text_fld
This is char data	This is varchar data	This text dara
This is char data	This is varchar data	This text dara
This is char data	This is varchar data	This text dara
This is char data	This is varchar data	This text dara

```
-- '사용 에러
UPDATE string_tbl
SET text_fld = 'This string doesn't work';
```

kray: Error

SQL Error (1064): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 't work' at line 2

```
-- ' 사용
UPDATE string_tbl
SET text_fld = 'This string doesn't work, but it does now';
```

char_fld	vchar_fld	text_fld
This is char data	This is varchar data	This string doesn't work, but it does now
This is char data	This is varchar data	This string doesn't work, but it does now
This is char data	This is varchar data	This string doesn't work, but it does now
This is char data	This is varchar data	This string doesn't work, but it does now

```
-- 특수문자
SELECT CONCAT('danke sch', CHAR(148), 'n');
```

Result #1 (1r x 1c)

CONCAT('danke sch', CHAR(148), 'n')

danke sch"n

```
-- 테이블 데이터 삭제
```

```
DELETE FROM string_tbl;
```

```
-- 삽입
INSERT INTO string_tbl(char_fld, vchar_fld, text_fld)
VALUES ('This streing is 28 characters',
       'This streing is 28 characters',
       'This streing is 28 characters');
```

```
-- 숫자를 반환하는 문자열 함수
SELECT LENGTH(char_fld) CHAR_LENGTH,
       LENGTH(vchar_fld) varchar_LENGTH,
       LENGTH(text_fld) text_LENGTH
FROM string_tbl;
```

```
-- 글자가 나타나는 위치 반환, 없을 경우 0을 반환
SELECT POSITION('charecters' IN vchar_fld)
FROM string_tbl;
```

```
-- 5번째 문자부터 시작하는 문자열 is의 위치를 반환.
-- 오라클에서는 position() and locate()를 사용할 수 없다.
SELECT LOCATE('is', vchar_fld,5)
FROM string_tbl;
```

```
-- 테이블 데이터 삭제후 삽입
INSERT INTO string_tbl(vchar_fld) VALUES ('abcd');
INSERT INTO string_tbl(vchar_fld) VALUES ('xyz');
INSERT INTO string_tbl(vchar_fld) VALUES ('QRSTUV');
INSERT INTO string_tbl(vchar_fld) VALUES ('qrstuv');
INSERT INTO string_tbl(vchar_fld) VALUES ('12345');
```

```
-- 순서대로 출력
SELECT vchar_fld
FROM string_tbl
ORDER BY vchar_fld;
```

```
-- 문자, 숫자 비교(대소문자를 제외하고 비교)
SELECT STRCMP('12345','12345') 12345_12345,
       STRCMP('abcd','xyz') abcd_xyz,
       STRCMP('abcd','QRSTUV') abcd_QRSTUV,
       STRCMP('qrstuv','QRSTUV') qrstuv_QRSTUV,
       STRCMP('12345','xyz') 12345_xyz,
       STRCMP('xyz','qrstuv') xyz_qrstuv;
```

```
-- ns로 끝나면 1을 반환하고 그렇지 않으면 0을 반환
SELECT NAME, NAME LIKE '%ns' ENDS_in_ns
FROM department;
```

bank.string\_tbl: 0 rows total (approximately)

char_fld	vchar_fld	text_fld
----------	-----------	----------

bank.string\_tbl: 1 rows total (approximately)

char_fld	vchar_fld	text_fld
This streing is 28 characters	This streing is 28 characters	This streing is 28 characters

string\_tbl (1r × 3c)

CHAR_LENGTH	varchar_LENGTH	text_LENGTH
29	29	29

string\_tbl (1r × 1c)

POSITION('charecters' IN vchar_fld)
20

string\_tbl (1r × 1c)

LOCATE('is', vchar_fld,5)
14

bank.string\_tbl: 5 rows total (approximately)

char_fld	vchar_fld	text_fld
(NULL)	abcd	(NULL)
(NULL)	xyz	(NULL)
(NULL)	QRSTUV	(NULL)
(NULL)	qrstuv	(NULL)
(NULL)	12345	(NULL)

string\_tbl (5r × 1c)

vchar_fld
12345
abcd
QRSTUV
qrstuv
xyz

Result #1 (1r × 6c)

12345_12345	abcd_xyz	abcd_QRSTUV	qrstuv_QRSTUV	12345_xyz	xyz_qrstuv
0	-1	-1	0	-1	1

department (3r × 2c)	
NAME	ENDS_in_ns
Operations	1
Loans	1
Administration	0

```
-- 데이터 삭제후
DELETE FROM string_tbl;
--데이터 입력
INSERT INTO string_tbl (text_fld)
VALUES ('This STRING was 29 characters');
-- 데이터 수정
UPDATE string_tbl
SET TEXT_fld = CONCAT(text_fld, ', but now it is longer');
```

bank.string\_tbl: 1 rows total (approximately)

char_fld	vchar_fld	text_fld
(NULL)	(NULL)	This STRING was 29 characters, but now it is longer

```
-- 데이터 출력
SELECT CONCAT(fname, ' ', lname, ' has been a ',
title, ' since ', start_date) emp_narrative
FROM employee
WHERE title = 'Teller' OR title = 'Head Teller';
/* 문자열을 반환하는 concat() 함수는
문자열이 저장된 데이터를 바꾸기 위한 용도로
사용될 수 있고 update() 함수는 데이터 자체를 바꾸지만,
select은 보여주는 것만 바꿀뿐이다.
```

employee (13r × 1c)	
emp_narrative	
Helen Fleming has been aHead Teller since 2004-03-...	
Chris Tucker has been aTeller since 2004-09-15	
Sarah Parker has been aTeller since 2002-12-02	
Jane Grossman has been aTeller since 2002-05-03	
Paula Roberts has been aHead Teller since 2002-07-27	
Thomas Ziegler has been aTeller since 2000-10-23	
Samantha Jameson has been aTeller since 2003-01-08	
John Blake has been aHead Teller since 2000-05-11	
Cindy Mason has been aTeller since 2002-08-09	

```
-- 숫자 데이터로 작업하기
-- 산술함수 수행
SELECT MOD(10,4);
-- % = mod()
SELECT MOD(22.75,5);
```

Result #1 (1r × 1c)	
MOD(10,4)	
2	

Result #1 (1r × 1c)	
MOD(22.75,5)	
2.75	

```
-- 제곱근
SELECT POW(2,8);
```

Result #1 (1r × 1c)	
POW(2,8)	
256	

```
-- 숫자 자리수 제어 (올림, 버림)
SELECT CEIL(72.445), FLOOR(72.445);
```

Result #1 (1r × 2c)	
CEIL(72.445)	FLOOR(72.445)
73	72

```
-- 반올림,
SELECT ROUND(72.49999), ROUND(72.5), ROUND(72.500001);
/* ROUND() 함수에서 두번째 인수 즉 반올림의 위치값을 생략하면
정수값을 기준으로 자동 반올림 처리한다. */
SELECT ROUND(72.49999,1), ROUND(72.0909,2), ROUND(72.500001,-1);
/* 인수의 위치값이 양수 일때는 소수점 아래 ~로 반올림하고
인수의 위치값이 음수일 때는 소수점 위의~에서 반올림한다. */
SELECT ROUND(17,-1), TRUNCATE(17,-1) ;
```

Result #1 (1r × 3c)		
ROUND(72.49999)	ROUND(73.5)	ROUND(72.500001)
72	74	73

Result #1 (1r × 3c)		
ROUND(72.49999,1)	ROUND(72.0909,2)	ROUND(72.500001,-1)
72.5	72.09	70

```
-- 시간데이터 작업
-- 문자를 날짜데이터로 변환
SELECT CAST('2008-09-17 15:30:00' AS DATETIME);
```

Result #1 (1r × 2c)	
ROUND(17,-1)	TRUNCATE(17,-1)
20	10

```
-- 시간데이터 생성 관련 함수
UPDATE individual
SET birth_date = STR_TO_DATE('September 17, 2007', '%M %d, %Y')
where cust_id = 9999;
-- 서버의 시간
SELECT CURRENT_DATE(), CURRENT_TIME(), CURRENT_TIMESTAMP();
```

Result #1 (1r × 1c)
CAST('2008-09-17 15:30:00' AS DATETIME)
2008-09-17 15:30:00

```
-- 시간 데이터 조작 함수
-- 날짜를 반환하는 함수
SELECT DATE_ADD(CURRENT_DATE(), INTERVAL 5 day);
```

Result #1 (1r × 3c)		
CURRENT_DATE()	CURRENT_TIME()	CURRENT_TIMESTAMP()
2022-08-04	16:40:10	2022-08-04 16:40:10

Result #1 (1r × 1c)
DATE_ADD(CURRENT_DATE(), INTERVAL 5 day)
2022-08-09

```
-- 해당달의 마지막 날 구하기
SELECT LAST_DAY('2008-09-17');
```

Result #1 (1r × 1c)
LAST_DAY('2008-09-17')
2008-09-30

```
-- 문자열로 데이 반환
SELECT DAYNAME('1994-05-05');
```

Result #1 (1r × 1c)
DAYNAME('1994-05-05')
Thursday

```
-- 원하는 년 반환
SELECT EXTRACT(YEAR FROM '2008-09-18 22:19:15');
```

2008-09-18 22:19:15 (1r × 1c)
EXTRACT(YEAR FROM '2008-09-18 22:19:15')
2,008

```
-- 숫자를 반환하는 시간 함수, 두 날짜 사이에 날짜 계산
SELECT DATEDIFF('2009-09-03', '2009-06-24');
```

Result #1 (1r × 1c)
DATEDIFF('2009-09-03', '2009-06-24')
71

```
-- 변환함수
SELECT CAST('1456328' AS SIGNED INTEGER);
/* cast()를 사용하려면 값또는 표현식 as키워드 변환할
값의 자료형을 제공해야 한다.*/
/* 문자열을 숫자로 변환할 때는 cast() 함수는
전체 문자열을 왼쪽부터 변환을 시도한다.
만약 숫자형 데이터가 아닌 문자가 있으면 에러를 발생시키지는 않지만
변환을 중지한다.
'999abc111'의 경우 첫 3글자만 변환되고 나머지는 버려진다.
```

Result #1 (1r × 1c)
CAST('1456328' AS SIGNED INTEGER)
1,456,328

```
show warnings; 를 출력하면
모든 문자열이 변환되지 않았다는 경고를 발생시킨다. */
```

```
-- 그룹화의 개념
SELECT open_emp_id
FROM account;

-- 중복을 제거하고 그룹화
SELECT open_emp_id
FROM account
GROUP BY OPEN_emp_id;
```

account (24 × 1c)
open_emp_id
1
1
1
1
1

account (4r × 1c)
open_emp_id
1
10
13
16

```
-- 별의 갯수를 확인
SELECT open_emp_id, COUNT(*) how_many
FROM account
GROUP BY open_emp_id;
```

account (4r × 2c)	
open_emp_id	how_many
1	8
10	7
13	3
16	6

```
-- 예러_ where절이 적용 될때 그룹이 아직 생성되지 않아서
-- where의 위치는 그룹 앞에 맞춤.
where절에서 함수를 못사용.
SELECT open_emp_id, C
FROM account
WHERE COUNT(*) > 4
GROUP BY open_emp_id;
```

kray: Error

SQL Error (1111): Invalid use of group function

```
-- having을 사용하여 해결_ 조건을 그룹 다음에 걸기위해 having이 있음
SELECT open_emp_id, COUNT(*) how_many
FROM account
GROUP BY open_emp_id
HAVING COUNT(*) > 4;
```

account (3r x 2c)	
open_emp_id	how_many
1	8
10	7
16	6

```
--
SELECT MAX(avail_balance) max_balance,
MIN(avail_balance) MIN_balance,
AVG(avail_balance) avg_balance,
SUM(avail_balance) sum_balance,
COUNT(*) num_accounts
FROM account
WHERE product_cd = 'CHK';
```

account (1r × 5c)				
max_balance	MIN_balance	avg_balance	sum_balance	num_accounts
38,552.05	122.37	7,300.800985	73,008.01	10