




동양미래대학교

고급 알고리즘

컴퓨터소프트웨어공학과 강은영



Bitwise 연산

1 Check if a Number is Odd or Even using Bitwise Operators 비트 연산을 통해 짝수, 홀수 판별하기

Input: N = 11
Output: Odd

2 Toggle case of a string using Bitwise Operators 비트 연산을 통해 문자의 대소문자 바꾸기

Input : "GeekSfOrgEEKs"
Output : "gEEKsFoRGeekS"

3 Set the rightmost unset bit 가장 오른쪽 0 값의 비트를 1로 바꾼 값 찾기

Number : 11
Binary representation: 1 0 1 1
Position of right most unset bit: 2

Bitwise 연산자 종류

구분	연산자	연산 기능
비트 논리 연산자	&	비트 단위 논리곱(AND) 연산자
		& 연산은 두 비트가 모두 1인 경우에만 1로 계산한다.
	^	비트 단위 배타적 논리합(XOR) 연산자
		^ 연산은 두 비트가 서로 다른 경우만 1로 계산한다.
		비트 단위 논리합(OR) 연산자
		연산은 두 비트 중에서 하나라도 참이면 1로 계산한다.
	~	비트 단위 부정(NOT) 연산자
		~ 연산은 1은 0으로 바꾸고 0은 1로 바꾼다.
비트 이동 연산자	<<	왼쪽 비트 이동 연산자
		<< 연산자는 왼쪽으로 이동시킨다.
	>>	오른쪽 비트 이동 연산자
		>> 연산자는 오른쪽으로 이동시킨다.

비트 연산을 통해 짝수, 홀수 판별하기 ①

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 짝수, 홀수 판별하기

Check if a Number is Odd or Even using Bitwise Operators

정수 값 N이 주어지면, bitwise 연산을 이용하여 홀수인지 짝수 인지 검사하시오.

Given a number N, the task is to check whether the number is even or odd using Bitwise Operators

입력	Input: N = 11	
출력	Output: Odd	
입출력 예	<u>입력</u> Input: N = 10	<u>출력</u> Output: Even

비트 연산을 통해 짝수, 홀수 판별하기 1

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 짝수, 홀수 판별하기

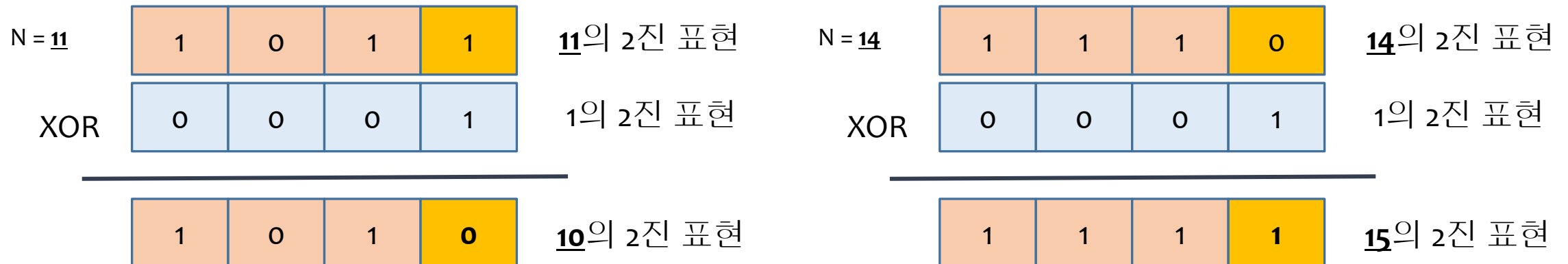
Check if a Number is Odd or Even using Bitwise Operators

정수 값 N이 주어지면, bitwise 연산을 이용하여 홀수인지 짝수 인지 검사하시오.

Given a number N, the task is to check whether the number is even or odd using Bitwise Operators

Think-1

XOR을 활용하여 숫자의 마지막 비트가 1인지 0인지 비교한다.



N을 1과 XOR한 값이, 홀수이면 1만큼 감소하고 짝수이면 1만큼 증가

비트 연산을 통해 짝수, 홀수 판별하기 (C언어예시)

1

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 짝수, 홀수 판별하기

Check if a Number is Odd or Even using Bitwise Operators

Think-1

XOR을 활용하여 숫자의 마지막 비트가 1인지 0인지 비교한다.

```
int isEven(int n) {  
    if ((n ^ 1) == (n + 1))  
        return 1;  
    else  
        return 0;  
}  
  
int main() {  
    int n = 17;  
  
    if(isEven(n) == 1)  
        printf("Even");  
    else  
        printf("Odd");  
  
    return 0;  
}
```

```
#include <stdio.h>  
#include <string.h>
```

if ((n ^ 1) == (n + 1))

Yes

Even

No

Odd

N을 1과 XOR한 값이,

홀수이면 1만큼 감소하고 짝수이면 1만큼 증가

비트 연산을 통해 짝수, 홀수 판별하기 ①

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 짝수, 홀수 판별하기

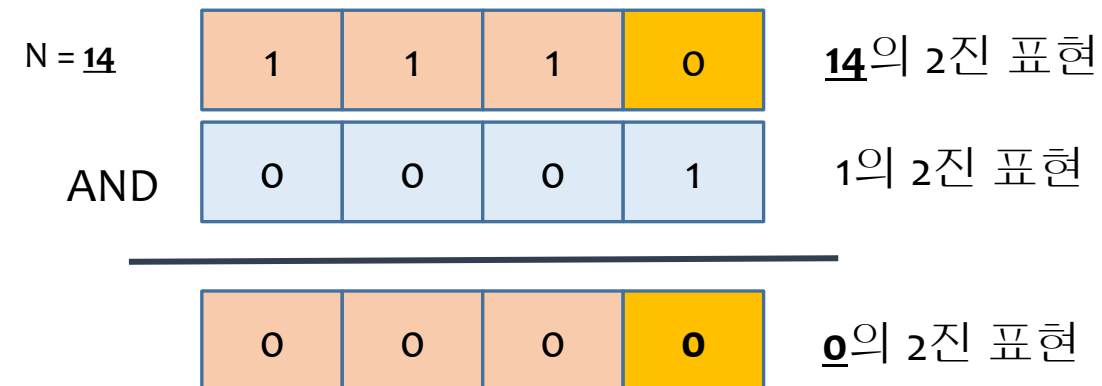
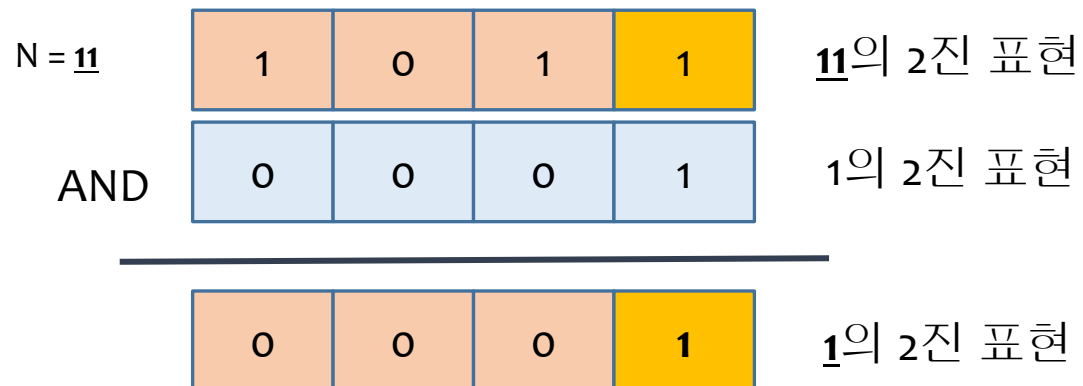
Check if a Number is Odd or Even using Bitwise Operators

정수 값 N이 주어지면, bitwise 연산을 이용하여 홀수인지 짝수 인지 검사하시오.

Given a number N, the task is to check whether the number is even or odd using Bitwise Operators

Think-2

AND 연산을 활용하여 숫자의 마지막 비트가 1인지 0인지 비교한다.



N을 1과 AND한 값은, 홀수이면 1 짝수이면 0이다.

비트 연산을 통해 짝수, 홀수 판별하기(C언어예시)

1

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 짝수, 홀수 판별하기

Check if a Number is Odd or Even using Bitwise Operators

Think-2

AND 연산을 활용하여 숫자의 마지막 비트가 1인지 0인지 비교한다.

```
int isEven(int n) {  
    return (n & 1);  
}  
  
int main() {  
    int n = 16;  
  
    if(isEven(n) == 1)  
        printf("Odd");  
    else  
        printf("Even");  
  
    return 0;  
}
```

```
#include <stdio.h>  
#include <string.h>
```

if ((n & 1) == 1)

Yes

Odd

No

Even

N을 1과 AND한 값은, 홀수이면 1 짝수이면 0이다.

비트 연산을 통해 문자의 대소문자 바꾸기 2

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 문자의 대소문자 바꾸기

Toggle case of a string using Bitwise Operators

주어진 문자열에 대해 대문자는 소문자로, 소문자는 대문자로 변경하는 함수를 작성하시오.

Given a string, write a function that returns toggle case of a string using the bitwise operators in place.

In ASCII codes, character 'A' is integer 65 = (0100 0001)₂, while character 'a' is integer 97 = (0110 0001)₂.

Similarly, character 'D' is integer 68 = (0100 0100)₂, while character 'd' is integer 100 = (0110 0100)₂.

As we can see, only sixth least significant bit is different in ASCII code of 'A' and 'a'. Similar behavior can be seen in ASCII code of 'D' and 'd'.

Therefore, we need to toggle this bit for toggling case.

입력	Input: "GeekSfOrgEEKs"	
출력	Output: "gEEKsFoRGeekS"	
입출력 예	입력 Input: "StRinG"	출력 Output: "sTrINg"

비트 연산을 통해 문자의 대소문자 바꾸기 2

(출처)

<https://www.geeksforgeeks.org/>

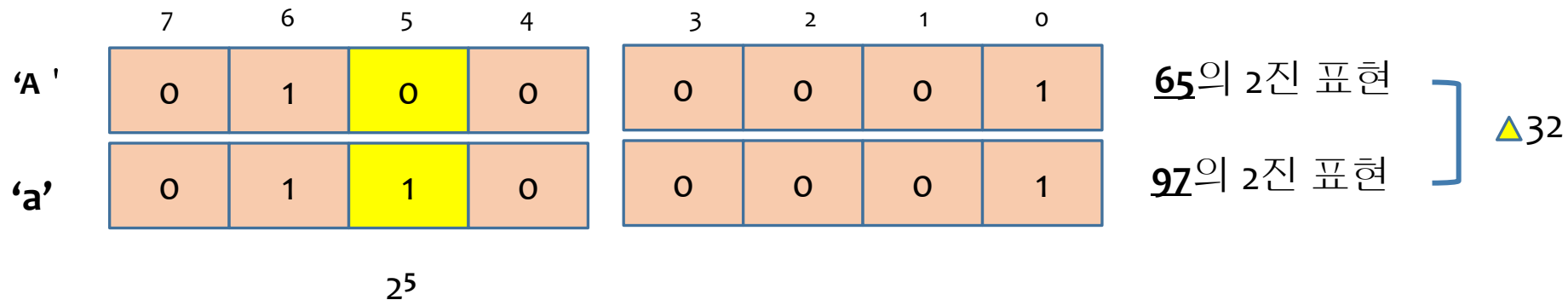
비트 연산을 통해 문자의 대소문자 바꾸기
Toggle case of a string using Bitwise Operators

주어진 문자열에 대해 대문자는 소문자로, 소문자는 대문자로 변경하는 함수를 작성하시오

Given a string, write a function that returns toggle case of a string using the bitwise operators in place.

Think-1

왼쪽 6번째 비트가 1인지 0인지 비교한다.



왼쪽 6번째 비트 값이 0이면 대문자, 1이면 소문자
왼쪽 6번째 비트 값이 0이면 1로, 1이면 0으로 바꾼다.

비트 연산을 통해 문자의 대소문자 바꾸기(C언어예시)

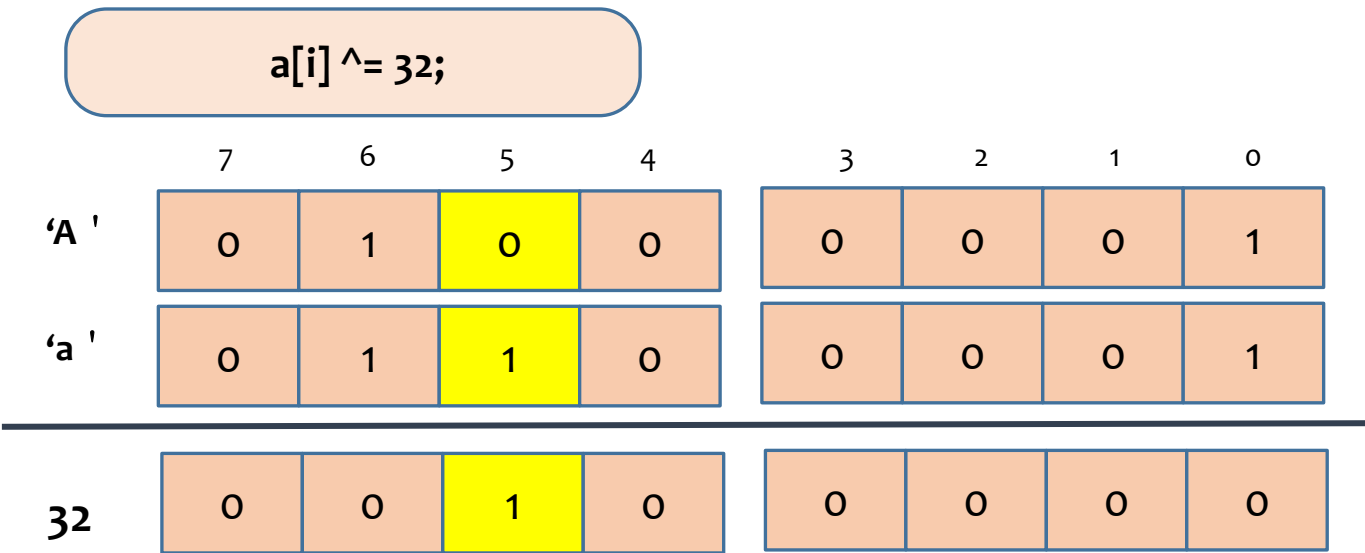
(출처)
<https://www.geeksforgeeks.org/>

비트 연산을 통해 문자의 대소문자 바꾸기
 Toggle case of a string using Bitwise Operators

```
char *toggleCase(char *a)
{
    for (int i=0; a[i]!='\0'; i++) {
        a[i] ^= 32;
    }
    return a;
}

int main()
{
    char str[] = "CheRrY";
    printf("Toggle case: %s\n", toggleCase(str));
    printf("Original string: %s", toggleCase(str));
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
```



가장 오른쪽 0값의 비트를 1로 바꾼 값 찾기

3

(출처)

<https://www.geeksforgeeks.org/>

가장 오른쪽 0 값의 비트를 1로 바꾼 값 찾기

Set the rightmost unset bit

양의 정수에 대해 가장 오른쪽 0 값의 비트를 1로 바꾼다. 만약 그런 0 값이 없으면 원래 정수를 반환한다.

Given a non-negative number n. The problem is to set the rightmost unset bit in the binary representation of n. If there are no unset bits, then just leave the number as it is.

- Number : 11 Binary representation: 1 0 1 1 Position of rightmost unset bit: 2 15
- Number : 6 Binary representation: 0 1 1 0 Position of rightmost unset bit: 0 7
- Number : 13 Binary representation: 1 1 0 1 Position of rightmost unset bit: 1 15

입력	Input : 21	
출력	Output : 23 $(21)_{10} = (101\textcolor{red}{0}1)_2$ $(23)_{10} = (101\textcolor{red}{1}1)_2$ The bit at position 1 has been set.	
입출력 예	<u>입력</u> Input: 21	<u>출력</u> Output: 23

가장 오른쪽 0값의 비트를 1로 바꾼 값 찾기

3

(출처)

<https://www.geeksforgeeks.org/>

가장 오른쪽 0 값의 비트를 1로 바꾼 값 찾기

Set the rightmost unset bit

양의 정수에 대해 가장 오른쪽 0 값의 비트를 1로 바꾼다. 만약 그런 0 값이 없으면 원래 정수를 반환한다.

Given a non-negative number n . The problem is to set the rightmost unset bit in the binary representation of n . If there are no unset bits, then just leave the number as it is.

Think-1

오른쪽 0 값인 비트만 1로 만들고 나머지 비트는 0으로 만든다.

	7	6	5	4	3	2	1	0
$N = 11$	0	0	0	0	1	0	1	1
$N+1 = 12$	0	0	0	0	1	1	0	0
$\sim N$	1	1	1	1	0	1	0	0
$(N+1) \& \sim N$	0	0	0	0	0	1	0	0

11의 2진 표현

가장 오른쪽 0값 비트는 1, 이후 비트는 0

가장 오른쪽 0값 비트는 1,

이전 비트는 보수값, 이후 비트는 보수 값

오른쪽 0 값인 비트만 1로 만들고 나머지 비트는 0으로 만든다.

가장 오른쪽 0값의 비트를 1로 바꾼 값 찾기

3

(출처)

<https://www.geeksforgeeks.org/>

가장 오른쪽 0 값의 비트를 1로 바꾼 값 찾기

Set the rightmost unset bit

양의 정수에 대해 가장 오른쪽 0 값의 비트를 1로 바꾼다. 만약 그런 0 값이 없으면 원래 정수를 반환한다.

Given a non-negative number n. The problem is to set the rightmost unset bit in the binary representation of n. If there are no unset bits, then just leave the number as it is.

Think-1

오른쪽 0 값인 비트만 1로 만들고 나머지 비트는 0으로 만든다.

	7	6	5	4	3	2	1	0
N = 11	0	0	0	0	1	0	1	1
N+1 = 12	0	0	0	0	1	1	<u>0</u>	<u>0</u>
~N	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>0</u>	1	0	0
(N+1) & ~N	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	1	0	0

$\text{Log}_2((N+1) \& \sim N)$

4(2^2)의 2진 표현

2^2



$\text{Log}_2 2^2 = 2$

가장 오른쪽 0값의 비트를 1로 바꾼 값 찾기(C언어예시)

3

(출처)

<https://www.geeksforgeeks.org/>

가장 오른쪽 0 값의 비트를 1로 바꾼 값 찾기

Set the rightmost unset bit

Think-1

오른쪽 0 값인 비트만 1로 만들고 나머지 비트는 0으로 만든다.

```
int getPosOfRightmostSetBit(int n) {
    return log2(~n&(n+1));
}

int setRightmostUnsetBit(int n) {
    if (n == 0)
        return 1;
    if ((n & (n + 1)) == 0)
        return n;
    int pos = getPosOfRightmostSetBit(n);
    return ((1 << pos) | n);
}

int main() {
    int n = 21;
    printf("%d", setRightmostUnsetBit(n));
    return 0;
}
```

```
#include <stdio.h>
```

```
#include <math.h>
```

N = 11

7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1

(N+1) & ~N

<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>1</u>	<u>0</u>	<u>0</u>
----------	----------	----------	----------	----------	----------	----------	----------

$\text{Log}_2((N+1) \& \sim N)$

$\text{pos} = \text{Log}_2 2^2 = 2$

1<<pos

<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>1</u>	<u>0</u>	<u>0</u>
----------	----------	----------	----------	----------	----------	----------	----------

((1<<pos) | n)

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

비트 연산을 통해 짝수, 홀수 판별하기 (Java예시)

1

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 짝수, 홀수 판별하기

Check if a Number is Odd or Even using Bitwise Operators

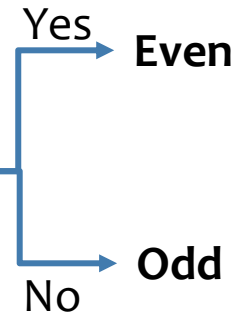
Think-1

XOR을 활용하여 숫자의 마지막 비트가 1인지 0인지 비교한다.

```
class GFG
{
    static boolean isEven(int n) {
        if ((n ^ 1) == n + 1)
            return true;
        else
            return false;
    }

    public static void main(String[] args) {
        int n = 100;
        System.out.print(isEven(n) == true ? "Even" : "Odd");
    }
}
```

if ((n ^ 1) == (n + 1))



N을 1과 XOR한 값이,

홀수이면 1만큼 감소하고 짝수이면 1만큼 증가

비트 연산을 통해 짝수, 홀수 판별하기(Java 예시)

1

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 짝수, 홀수 판별하기

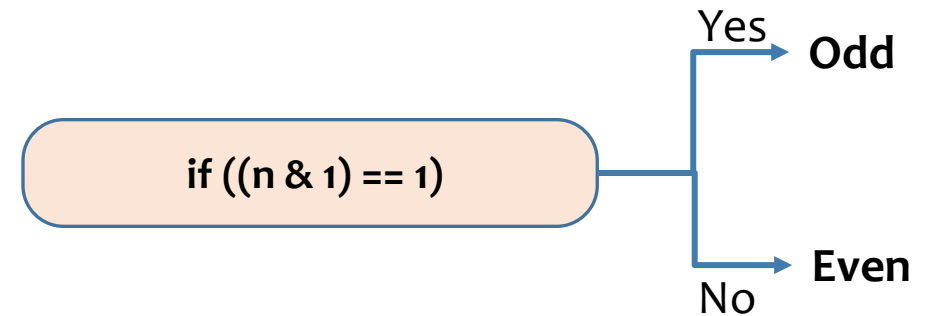
Check if a Number is Odd or Even using Bitwise Operators

Think-2

AND 연산을 활용하여 숫자의 마지막 비트가 1인지 0인지 비교한다.

```
class GFG
{
    static boolean isEven(int n)
    {
        return ((n & 1) != 1);
    }

    public static void main(String[] args)
    {
        int n = 101;
        System.out.print(isEven(n) == true ? "Even" : "Odd");
    }
}
```



N을 1과 AND한 값은, 홀수이면 1 짝수이면 0이다.

비트 연산을 통해 문자의 대소문자 바꾸기(Java 예시)

2

(출처)

<https://www.geeksforgeeks.org/>

비트 연산을 통해 문자의 대소문자 바꾸기

Toggle case of a string using Bitwise Operators

Think-1

왼쪽 6번째 비트가 1인지 0인지 비교한다.

$a[i] \wedge 32;$

	7	6	5	4	3	2	1	0
'A '	0	1	0	0	0	0	0	1
'a '	0	1	1	0	0	0	0	1
32	0	0	1	0	0	0	0	0

```
public class Test {  
    static int x=32;  
    static String toggleCase(char[] a) {  
        for (int i=0; i<a.length; i++) {  
            a[i]^=32;  
        }  
        return new String(a);  
    }  
    public static void main(String[] args) {  
        String str = "CheRrY";  
        System.out.print("Toggle case: ");  
        str = toggleCase(str.toCharArray());  
  
        System.out.println(str);  
        System.out.print("Original string: ");  
        str = toggleCase(str.toCharArray());  
        System.out.println(str);  
    }  
}
```

가장 오른쪽 0값의 비트를 1로 바꾼 값 찾기

3

(출처)

<https://www.geeksforgeeks.org/>

```
public class Test {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        int n = 11;  
        System.out.println("result = " + setBit(n));  
    }  
    public static int getBit(int n) {  
        return (int)( (Math.log(~n&(n+1)) / Math.log(2)) );  
    }  
    public static int setBit(int n) {  
        if (n == 0)  
            return 1;  
        if ( (n & (n+1)) == 0 )  
            return n;  
        int pos = getBit(n);  
        return ( (1 << (pos)) | n);  
    }  
}
```