두 정수 입력받아 비교하기3

두 정수(a, b)를 입력받아
b가 a보다 크거나 같으면 1을, 그렇지 않으면 0을 출력하는 프로그램을 작성해보자.

참고
어떤 값을 비교하기 위해 비교/관계연산자(comparison/relational)를 사용할 수 있다.

비교/관계연산자는 주어진 2개의 값을 비교하여
그 결과가 참인 경우 참(true)을 나타내는 정수값 1로 계산하고,
거짓인 경우 거짓(false)를 나타내는 정수값 0으로 계산한다.

비교/관계연산자도 일반적인 사칙연산자처럼 주어진 두 수를 이용해 계산을 수행하고,
그 결과를 1(참), 또는 0(거짓)으로 계산해 주는 연산자이다.

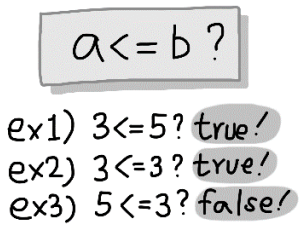
비교/관계연산자는 >, <, >=, <=, ==(같다), !=(다르다)의 6가지가 있다.

>=, <= 연산자는 같음(==)을 포함한다.
따라서 "작다/크다" 또는 "같다"는 의미를 가진다.

◎ 입력 형식
두 정수 a, b가 공백을 두고 입력된다.
-2147483647 <= a, b <= +2147483648

◎ 출력 형식
b가 a보다 같거나 큰 경우 1을, 그렇지 않은 경우 0을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2	입력 예시3	출력 예시3
0 -1	0	1 999	1	9 9	1



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

두 정수 입력받아 비교하기4

두 정수(a, b)를 입력받아
a와 b가 서로 다르면 1을, 그렇지 않으면 0을 출력하는 프로그램을 작성해보자.

참고
어떤 값을 비교하기 위해 비교/관계연산자(comparison/relational)를 사용할 수 있다.

비교/관계연산자는 주어진 2개의 값을 비교하여
그 결과가 참인 경우 참(true)을 나타내는 정수값 1로 계산하고,
거짓인 경우 거짓(false)를 나타내는 정수값 0으로 계산한다.

비교/관계연산자도 일반적인 사칙연산자처럼 주어진 두 수를 이용해 계산을 수행하고,
그 결과를 1(참), 또는 0(거짓)으로 계산해 주는 연산자이다.

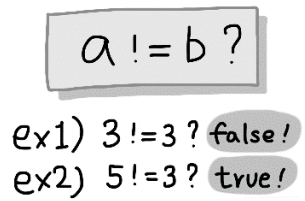
비교/관계연산자는 >, <, >=, <=, ==(같다), !=(다르다)의 6가지가 있다.

예시
```
printf("%d", 123!=123); //비교 연산자 !=의 계산 결과인 0(거짓)이 출력된다.
```

◎ 입력 형식
두 정수 a, b가 공백을 두고 입력된다.
-2147483647 <= a, b <= +2147483648

◎ 출력 형식
a와 b가 다른 경우 1을, 그렇지 않은 경우 0을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
0 1	1	9 9	0



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

참 거짓 바꾸기

1(true, 참) 또는 0(false, 거짓) 이 입력되었을 때
반대로 출력하는 프로그램을 작성해보자.

참고
C언어에서 비교/관계 연산(==, !=, >, <, >=, <=)이 수행될 때,
0은 거짓(false)으로 인식되고, 0이 아닌 모든 수는 참(true)으로 인식된다.

참 또는 거짓의 논리값을 역(반대)으로 바꾸기 위해서는
논리(logical)연산자 !를 사용할 수 있다.

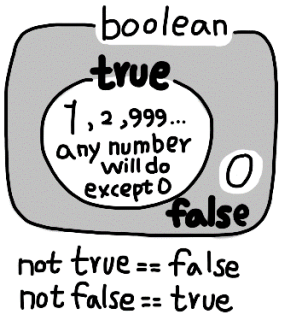
이러한 논리연산을 NOT 연산이라고도 부르고,
프라임(기호 오른쪽에 따옴표) 이나 바(기호 위에 가로 막대)로 표시하고,
집합 기호로는 c (여집합, complement)를 의미한다. 모두 같은 의미이다.

참, 거짓의 논리값(boolean value)인 불 값을 다루어주는 논리연산자는
!(not), &&(and), ¦¦(or) 이 있다.

** 불 대수(boolean algebra)는 수학자 불이 만들어낸 것으로
   참/거짓만 가지는 논리값과 그 연산을 다룬다.

예시
```
printf("%d", !0); //거짓의 반대, 즉 참인 1로 계산됨
printf("%d", !1); //참의 반대, 즉 거짓인 0으로 계산됨
printf("%d", !999); //참의 반대, 즉 거짓인 0으로 계산됨
```

어떤 변수 a에, !a 와 같은 논리 연산이 가능하다.



◎ 입력 형식
정수 1개가 입력된다.(단, 0 또는 1 만 입력된다.)

◎ 출력 형식
입력된 값이 0이면 1, 1이면 0을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
1	0	0	1

Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

둘 다 참일 경우만 참 출력하기

두 개의 참(1) 또는 거짓(0)이 입력될 때,
모두 참일 때에만 참을 출력하는 프로그램을 작성해보자.

참고
논리연산자 && 는 주어진 2개의 논리값이 모두 참(1) 일 때에만 1(참)로 계산하고,
그 외의 경우에는 0(거짓) 으로 계산한다.

이러한 논리연산을 AND 연산이라고도 부르고, ·로 표시하거나 생략하며,
집합 기호로는 ∩(교집합, intersection)을 의미한다. 모두 같은 의미이다.

참, 거짓의 논리값(boolean value)인 불 값을 다루어주는 논리연산자는
!(not), &&(and), ¦¦(or) 이 있다.

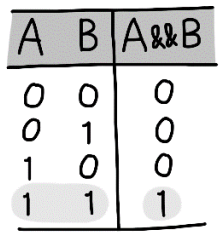
** 불 대수(boolean algebra)는 수학자 불이 만들어낸 것으로
참/거짓만 가지는 논리값과 그 연산을 다룬다.

예시
```
printf("%d", 1&&1); //둘 다 참을 의미하므로 1로 계산되어 출력된다.
```

◎ 입력 형식
1 또는 0의 값만 가지는 2개의 정수가 공백을 두고 입력된다.

◎ 출력 형식
둘 다 참(1)일 경우에만 1을 출력하고, 그 외의 경우에는 0을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
1 1	1	0 1	0



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

하나라도 참이면 참 출력하기

두 개의 참(1) 또는 거짓(0)이 입력될 때,
하나라도 참이면 참을 출력하는 프로그램을 작성해보자.

참고
논리연산자 ¦¦ 는 주어진 2개의 논리값 중에 하나라도 참(1) 이면 1(참)로 계산하고,
그 외의 경우에는 0(거짓) 으로 계산한다.

** ¦ 기호는 쉬프트를 누른 상태에서 백슬래시(\)를 누르면 나오는 기호로,
버티컬 바(vertical bar) 기호이다.

이러한 논리연산을 OR 연산이라고도 부르고,
+ 로 표시하며, 집합 기호로는 ∪(합집합, union)을 사용한다. 모두 같은 의미이다.

참, 거짓의 논리값(boolean value)인 불 값을 다루어주는 논리연산자는
!(not), &&(and), ¦¦(or) 이 있다.

** 불 대수(boolean algebra)는 수학자 불이 만들어낸 것으로
참/거짓만 가지는 논리값과 그 연산을 다룬다.

예시
printf("%d", 0¦¦0); //둘 다 거짓을 의미하므로 0이 계산되어 출력된다.

◎ 입력 형식
1 또는 0의 값만 가지는 2개의 정수가 공백을 두고 입력된다.

◎ 출력 형식
하나라도 참일 경우 1을 출력하고, 그 외의 경우에는 0을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
1 1	1	0 1	1

A	B	A ¦¦ B
0	0	0
0	1	1
1	0	1
1	1	1

Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

참/거짓이 서로 다를 때에만 참 출력하기

두 가지의 참(1) 또는 거짓(0)이 입력될 때,
참/거짓이 서로 다를 때에만 참을 출력하는 프로그램을 작성해보자.

참고
이러한 논리연산을 XOR(exclusive or, 배타적 논리합)연산이라고도 부른다.
집합의 의미로는 합집합에서 교집합을 뺀 것을 의미한다. 모두 같은 의미이다.

논리연산자는 사칙연산자와 마찬가지로 여러 번 중복해서 사용할 수 있는데,
연산의 순서를 만들어주기 위해 괄호 ( )를 사용해 묶어 주면 된다.

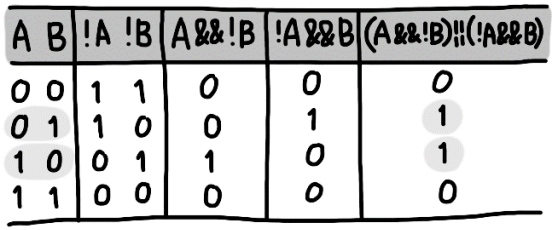
수학에서는 괄호, 중괄호, 대괄호를 사용하지만 C언어에서는 소괄호 ( )만을 사용한다.

예시
```
printf("%d", (a&&!b)||(!a&&b)); //참/거짓이 서로 다를 때에만 1로 계산
```

◎ 입력 형식
1 또는 0의 값만 가지는 2개의 정수가 공백을 두고 입력된다.

◎ 출력 형식
참/거짓이 서로 다를 때에만 1을 출력하고, 그 외의 경우에는 0을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
1 1	0	0 1	1



참/거짓이 서로 같을 때에만 참 출력하기

두 개의 참(1) 또는 거짓(0)이 입력될 때,
참/거짓이 서로 같을 때에만 참이 계산되는 프로그램을 작성해보자.

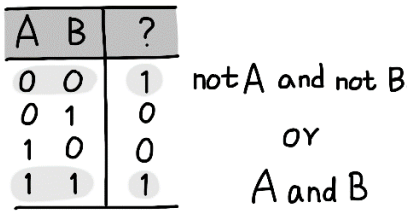
◎ 입력 형식

1 또는 0의 값만 가지는 2개의 정수가 공백을 두고 입력된다.

◎ 출력 형식

참/거짓이 서로 같을 때에만 1을 출력하고, 그 외의 경우에는 0을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
0 0	1	0 1	0



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

둘 다 거짓일 경우만 참 출력하기

두 개의 참(1) 또는 거짓(0)이 입력될 때,
모두 거짓일 때에만 참이 계산되는 프로그램을 작성해보자.

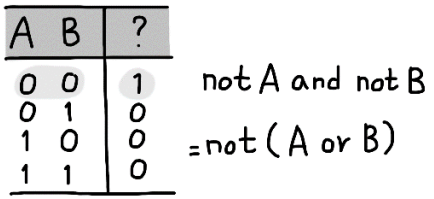
◎ 입력 형식

1 또는 0의 값만 가지는 2개의 정수가 공백을 두고 입력된다.

◎ 출력 형식

둘 다 거짓일 경우에만 1을 출력하고, 그 외의 경우에는 0을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
0 1	0	0 0	1



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

비트단위로 바꿔 출력하기

입력 된 정수를 비트단위로 참/거짓을 바꾼 후 정수로 출력해보자.
비트단위(bitwise)연산자 ~ 를 붙이면 된다.(~ : tilde, 틸드라고 읽는다.)

** 비트단위(bitwise) 연산자는,
~(bitwise not), &(bitwise and), ¦(bitwise or), ^(bitwise xor),
<<(bitwise left shift), >>(bitwise right shift)
가 있다.

예를 들어 1이 입력되었을 때 저장되는 1을 32비트 2진수로 표현하면
00000000 00000000 00000000 00000001 이고,
~1은 11111111 11111111 11111111 11111110 가 되는데 이는 -2를 의미한다.

예시
```
int a=1;
printf("%d", ~a); //-2가 출력된다.
```

참고
컴퓨터에 저장되는 모든 데이터들은 2진수 형태로 바뀌어 저장된다.
0과 1로만 구성되는 비트단위들로 변환되어 저장되는데,
양의 정수는 2진수 형태로 바뀌어 저장되고,
음의 정수는 "2의 보수 표현"방법으로 저장된다.

예를 들어 int형(4바이트(byte), 32비트)으로 선언된 변수에 양의 정수 5를 저장하면
5의 2진수 형태인 101이 32비트로 만들어져
00000000 00000000 00000000 00000101
로 저장된다.(공백은 보기 편하도록 임의로 분리)

int 형의 정수 0은
00000000 00000000 00000000 00000000

그리고 -1은 0에서 1을 더 빼고 32비트만 표시하는 형태로
11111111 11111111 11111111 11111111 로 저장된다.

-2는 -1에서 1을 더 빼면 된다.
11111111 11111111 11111111 11111110 로 저장된다.

그래서 int 형으로 선언된 변수에는 최소 -2147483648 을 의미하는
10000000 00000000 00000000 00000000 부터

최대 +2147483647 을 의미하는
01111111 11111111 11111111 11111111 로 저장될 수 있는 것이다.

그렇다면 -2147483648
10000000 00000000 00000000 00000000 에서 1을 더 뺀다면?

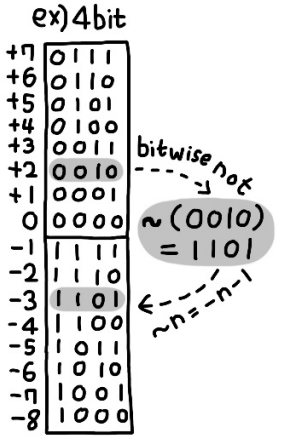
01111111 11111111 11111111 11111111 이 된다.
즉 -2147483649 가 아닌 +2147483647 이 되는 것이다.

이러한 것을 오버플로우(overflow, 넘침)라고 한다.

이러한 내용을 간단히 표시하면, 정수 n이라고 할 때,

~n = -n - 1
-n = ~n + 1 과 같은 관계로 표현된다.

이 관계를 그림으로 그려보면 마치 원형으로 수들이
상대적으로 배치된 것과 같다.



◎ 입력 형식
정수 1개가 입력된다.
-2147483648 ~ +2147483647

◎ 출력 형식
비트 단위로 1 -> 0, 0 -> 1로 바꾼 후 그 값을 10진수로 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
2	-3	0	-1

Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

비트단위로 and 하여 출력하기

입력된 정수 두 개를 비트단위로 and 연산한 후 그 결과를 정수로 출력해보자.
비트단위(bitwise)연산자 &를 사용하면 된다.(and, ampersand, 앰퍼센드라고 읽는다.)

** 비트단위(bitwise)연산자는,
~(bitwise not), &(bitwise and), ¦(bitwise or), ^(bitwise xor),
<<(bitwise left shift), >>(bitwise right shift)
가 있다.

예를 들어 3과 5가 입력되었을 때를 살펴보면
3 : 00000000 00000000 00000000 00000011
5 : 00000000 00000000 00000000 00000101
3 & 5 : 00000000 00000000 00000000 00000001
이 된다.

비트단위 and 연산은 두 비트열이 주어졌을 때,
둘 다 1인 부분의 자리만 1로 만들어주는 것과 같다.

이 연산을 이용하면 어떤 비트열의 특정 부분만 모두 0으로도 만들 수 있는데
192.168.0.31 : 11000000.10101000.00000000.00011111
255.255.255.0 : 11111111.11111111.11111111.00000000

두 개의 ip 주소를 & 연산하면
192.168.0.0 : 110000000.10101000.0000000.00000000 을 계산할 수 있다.

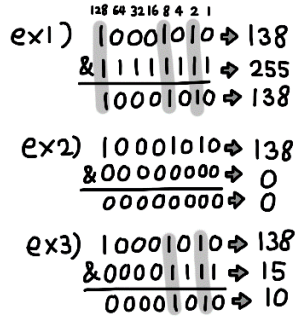
실제로 이 계산은 네트워크에 연결되어 있는 두 개의 컴퓨터가 데이터를 주고받기 위해
같은 네트워크에 있는지 아닌지를 판단하는데 사용된다.

이러한 비트단위 연산은 빠른 계산이 필요한 그래픽처리에서
마스크연산(특정 부분을 가리고 출력하는)을 수행하는 데에도 효과적으로 사용된다.

◎ 입력 형식
두 개의 정수가 공백을 두고 입력된다.
-2147483648 ~ +2147483647

◎ 출력 형식
두 정수를 비트단위(bitwise)로 and 계산을 수행한 결과를 10진수로 출력한다.

입력 예시	출력 예시
3 5	1



비트단위로 or 하여 출력하기

입력된 정수 두 개를 비트단위로 or 연산한 후 그 결과를 정수로 출력해보자.
비트단위(bitwise) 연산자 ¦(or, vertical bar, 버티컬바)를 사용하면 된다.

** ¦ 은 파이프(pipe)연산자라고도 불리는 경우가 있다.

** 비트단위(bitwise) 연산자는,
~(bitwise not), &(bitwise and), ¦(bitwise or), ^(bitwise xor),
<<(bitwise left shift), >>(bitwise right shift)
가 있다.

예를 들어 3과 5가 입력되었을 때를 살펴보면
3 : 00000000 00000000 00000000 00000011
5 : 00000000 00000000 00000000 00000101
3 ¦ 5 : 00000000 00000000 00000000 00000111
이 된다.

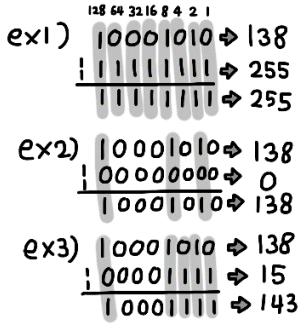
비트단위 or 연산은 둘 중 하나라도 1인 자리를 1로 만들어주는 것과 같다.

이러한 비트단위 연산은 빠른 계산이 필요한 그래픽처리에서도 효과적으로 사용된다.

◎ 입력 형식
두 개의 정수가 공백을 두고 입력된다.
-2147483648 ~ +2147483647

◎ 출력 형식
두 정수를 비트단위(bitwise)로 or 계산을 수행한 결과를 10진수로 출력한다.

입력 예시	출력 예시
3 5	7



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

비트단위로 xor 하여 출력하기

입력된 정수 두 개를 비트단위로 xor 연산한 후 그 결과를 정수로 출력해보자.
비트단위(bitwise) 연산자 ^(xor, circumflex/caret, 서컴플렉스/카릿)를 사용하면 된다.

** 주의 ^은 수학식에서 거듭제곱(power)을 나타내는 기호와 모양은 같지만,
C언어에서는 전혀 다른 배타적 논리합(xor, 서로 다를 때 1)의 의미를 가진다.

** 비트단위(bitwise) 연산자는,
~(bitwise not), &(bitwise and), ¦(bitwise or), ^(bitwise xor),
<<(bitwise left shift), >>(bitwise right shift)
가 있다.

예를 들어 3과 5가 입력되었을 때를 살펴보면
3 : 00000000 00000000 00000000 00000011
5 : 00000000 00000000 00000000 00000101
3 ^ 5 : 00000000 00000000 00000000 00000110
이 된다.
이러한 비트단위 연산은 빠른 계산이 필요한 그래픽처리에서도 효과적으로 사용된다.

구체적으로 설명하자면,
두 장의 이미지가 겹쳐졌을 때 색이 서로 다른 부분만 처리할 수 있다.
배경이 되는 그림과 배경 위에서 움직이는 그림이 있을 때,
두 그림에서 차이만 골라내 배경 위에서 움직이는 그림의 색으로 바꿔주면
전체 그림을 구성하는 모든 점들의 색을 다시 계산해 입히지 않고
보다 효과적으로 그림을 처리할 수 있게 되는 것이다.
비행기 슈팅게임 등을 상상해보면 된다.

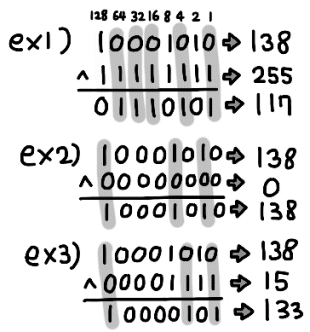
◎ 입력 형식

두 개의 정수가 공백을 두고 입력된다.
-2147483648 ~ +2147483647

◎ 출력 형식

두 정수를 비트단위(bitwise)로 xor 계산을 수행한 결과가 10진수로 출력된다.

입력 예시	출력 예시
3 5	6



두 정수 입력받아 큰 수 출력하기

입력된 두 정수 a, b 중 큰 값을 출력하는 프로그램을 작성해보자.
단, 조건문을 사용하지 않고 3항 연산자 ? 를 사용한다.

참고
3개의 요소로 이루어지는 3항(ternary) 연산자는
"조건식 ? (참일 때의 값) : (거짓일 때의 값)" 의 형태로 사용하는 연산자이다.
- 조건식의 계산 결과가 참인 경우에는 ':' 왼쪽의 값 또는 식으로 바뀌고,
- 거짓인 경우에는 ':' 오른쪽의 값 또는 식으로 바뀐다.

예를 들어
`123 > 456 ? 0 : 1`
과 같은 표현식은 123 > 456 의 비교연산 결과가 거짓이므로 1이 된다.

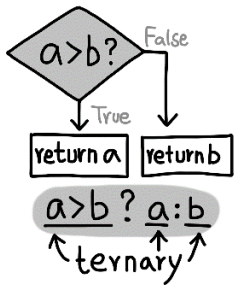
예시
```
printf("%d", a>b ? a:b); //두 값 중 큰 값이 출력된다.
```

예시 코드는 a>b 의 결과가 참(1)이면 (a>b ? a:b)의 결과는 a가 되고,
거짓(0)이면 (a>b ? a:b)의 결과는 b가 된다.

◎ 입력 형식
두 정수가 공백을 두고 입력된다.
-2147483648 ~ +2147483647

◎ 출력 형식
두 정수 중 큰 값을 10진수로 출력한다.

입력 예시	출력 예시
123 456	456



정수 세 개 입력받아 가장 작은 수 출력하기

입력된 세 정수 a, b, c 중 가장 작은 값을 출력하는 프로그램을 작성해보자.
단, 조건문을 사용하지 않고 3항 연산자 ? 를 사용한다.

참고
C언어 소스코드 작성시 모든 요소들은
"순서에 따라 한 단계씩 실행"
"미리 정해진 순서에 따라 하나씩 연산 수행"
"그 때까지 연산된 결과를 이용해 다시 순서에 따라 하나씩 연산 "
...
등의 원리가 적용된다.

따라서 3항 연산자 ? 를 중첩해(괄호로 묶는 등..) 이용하면
여러 값들을 순서대로 비교해 가장 큰/작은 값을 계산할 수 있다.

예를 들어
(a>b ? a:b)>c ? (a>b ? a:b):c 의 계산식은
a, b, c 의 값 중 가장 큰 값으로 계산된다.
잘 이해가 되지 않는다면 어떤 순서에 따라 계산될지 생각해보고
여러 가지 연산자가 동시에 사용된 식이 있을 때,
어떤 우선순위에 따라 순서대로 계산이 되는지 찾아보도록 한다.

"연산자 우선순위"를 검색하면 우선순위와 결합방향이 나온다.
예를 들어 변수에 어떤 값을 대입하는
대입(assign) 연산자 = 의 우선순위는 가장 낮고, 오른쪽에서 왼쪽의 결합방향을 가진다.

따라서,
a = b = c = d = e = f = g = h = i = j = 0;
의 식을 실행하면 오른쪽에서 부터 왼쪽으로 가면서
처음에 j 변수에 0이 대입되고, 다음에 i 변수에 j변수에 저장되어 있는 값이 저장되고,
그 다음에 h 변수에 i변수에 저장되어 있는 값이 저장되고 ...
결국 모든 변수의 값을 0으로 만드는 결과가 된다.

** 3항 연산자는 자주 사용되지는 않지만,
복잡한 계산식이나 조건 처리, 비교 구조를 매우 간단히 표현할 수 있게 해준다.

잘 사용해보면 나름대로의 재미와 묘미가 있는 연산자이다.

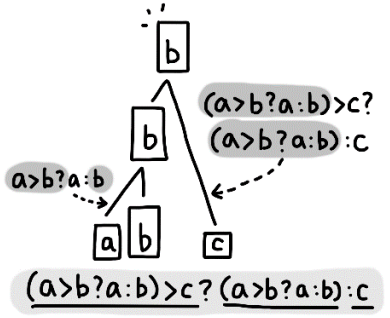
특히, 보다 짧은 코드로 문제를 해결하려고 하는
숏 코딩(coding)에서는 빠질 수 없는 요소이다.
"똑같이 해결할 수 있는 프로그램이지만, 때로는 아주 적은 소스코드 양으로 풀어내는 것을
매우 즐기는 숏 코더들이 있다."

숏 코딩은 일종의 재미이기는 하지만,
프로그래밍언어의 밑바닥 기초, 세세한 처리 과정에 대한 이해,
컴파일러의 소스코드 해석과 변환 등에 대한 경험과 지식이 필요하다.

◎ 입력 형식
세 개의 정수가 공백으로 구분되어 입력된다.
-2147483648 ~ +2147483648

◎ 출력 형식
가장 작은 값을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
1 2 3	1	3 -1 5	-1



정수 세 개 입력받아 짝수만 출력하기

세 정수 a, b, c가 입력되었을 때, 짝수만 출력해보자.

참고
if(조건)
{ //조건이 참일 때에만 실행되는 구역(코드블록의 시작)
  실행1;
  실행2;
...
} //코드블록의 끝

의 조건/선택 실행 구조는
주어진 "조건"을 검사해 참인 경우에만 코드블록 안에 작성한 내용들을 순서대로 실행한다.

예시
```
if(a%2==0)
{
  printf("%d", a);
}

if(b%2==0)
{
  printf("%d", b);
}

if(c%2==0)
{
  printf("%d", c);
}
```

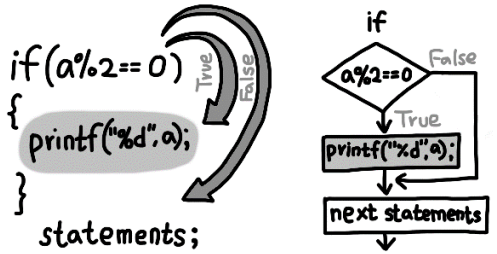
** a%2==0 은 a%2가 먼저 수행되고 그 결과를 0과 비교해 참/거짓으로 계산해 준다.

a%2==0의 의미는 a를 2로 나눈 나머지가 0이라면,
즉 "a가 짝수라면?" 이라는 의미로 해석할 수 있다.

◎ 입력 형식
세 정수 a, b, c 가 공백을 두고 입력된다.
0 ~ +2147483647 범위의 정수들이 입력되며 적어도 1개는 짝수이다.

◎ 출력 형식
짝수만 순서대로 줄을 바꿔 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
1 2 3	2	1 2 4	2 4



정수 세 개 입력받아 짝/홀 출력하기

세 정수 a, b, c가 입력되었을 때, 짝(even)/홀(odd)을 출력해보자.

참고
if(조건)
{ //코드블록1 시작
  ... 실행...;
  ...
}
else
{ //코드블록2 시작
  ... 다른실행...;
  ...
}
의 조건/선택 실행구조는 주어진 "조건"을 검사해 참인 경우에는 코드블록1을 실행하고, 거짓인 경우에는 코드블록2를 실행한다.

예시
```
if(a%2==0)
{
  printf("%s", "even");
}
else
{
  printf("%s", "odd");
}
```

** else는 if 없이 혼자 사용되지 않는다. 또한, else 다음에 조건이 없는 이유는?
   참이 아니면 거짓이고, 거짓이 아니면 참이기 때문에...
   즉 if문의 조건식의 판별 결과는 2가지 경우(참 또는 거짓)로 계산되는데,
   else 부분은 if문의 조건식의 결과가 거짓인 경우에 해당하기 때문이다.

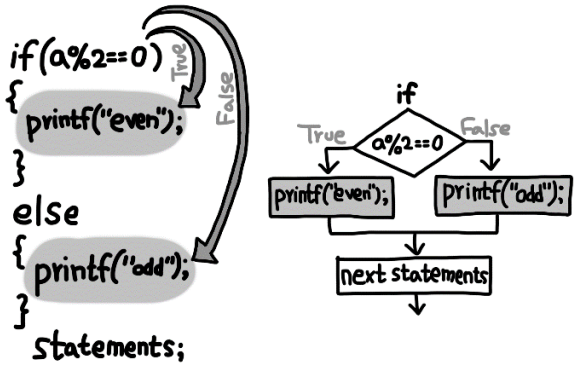
** 실행해야 할 명령들이 여러 개일 때 코드블록 기호 { }로 묶어 주는데,
   코드블록의 내용이 논리적으로 1개 단위라면 코드블록 기호를 생략할 수 있다.

```
if(a%2==0) printf("%s", "even");
else printf("%s", "odd");
```

◎ 입력 형식
세 정수 a, b, c 가 공백을 두고 입력된다.
0 <= a, b, c <= +2147483647

◎ 출력 형식
입력된 순서대로 짝(even)/홀(odd)을 줄을 바꿔 출력한다.

입력 예시	출력 예시
1 2 8	odd even even



정수 한 개 입력받아 분석하기

정수 한 개가 입력되었을 때, 음(minus)/양(plus)과 짝(even)/홀(odd)을 출력해보자.

참고
조건/선택 실행 구조의 안에 조건/선택 실행 구조를 다시 "중첩"할 수 있다.
이는 "중첩(nested)"의 원리가 적용되는 내용으로 아래와 같은 구조가 가능하다.

```
if(조건1)
{
  if(조건2) //조건문의 중첩
  {
    ...
  }
  else
  {
    ...
  }
}
else
{
  if(조건2)
  {
    ...
  }
  else
  {
    ...
  }
}
```

위와 같은 조건/선택 실행구조는 조건/선택 실행구조를 중첩해
서로 다른 4가지(2가지*2가지)의 경우에 대해 다른 실행을 할 수 있도록 해준다.

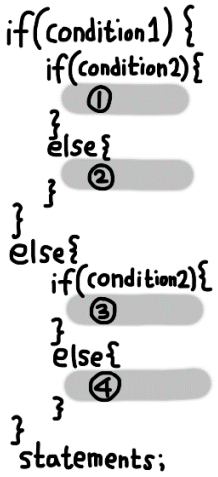
** 어떤 조건들이 맞았을 때에는 해당부분의 코드블록 내용만 실행되고,
   전체 조건/선택 실행구조를 빠져나간다. 즉, 다른 부분들은 실행되지 않는다.

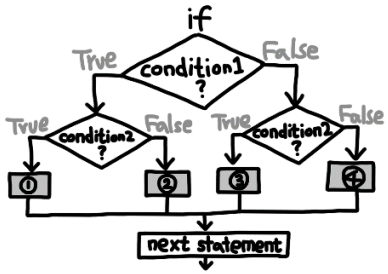
** 소스코드의 들여쓰기는 사람이 보고 이해하기 쉽도록 하는 것으로써
   들여쓰기나 줄바꿈을 하지 않더라도 동일하게 컴파일된다.

◎ 입력 형식
정수 한 개가 입력된다.
-2147483648 ~ +2147483647, 단 0은 입력되지 않는다.

◎ 출력 형식
입력된 정수에 대해
첫 줄에 minus 나 plus 를 출력하고,
두 번째 줄에 odd 나 even 을 출력한다.

입력 예시	출력 예시
-2147483648	minus even





Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

정수 한 개 입력받아 평가 출력하기

점수(정수, 0 ~ 100)를 입력받아 평가를 출력해보자.

평가 기준
점수 범위 : 평가
90 ~ 100 : A
70 ~ 89 : B
40 ~ 69 : C
0 ~ 39 : D
로 평가되어야 한다.

참고
여러 조건들을 순서대로 비교하면서 처리하기 위해서 조건문을 중첩할 수 있다.
만약, 순서대로 검사하고 실행하기 위해 아래와 같이 중첩 시킨다면,

```
if(조건1)
{
 ...
}
else
{
 if(조건2)
 {
  ...
 }
 else
 {
  ...
 }
}
```

중첩된 구조는 논리적으로 1단위이기 때문에 코드블록 기호를 생략하면
아래와 같은 구조로 다시 표현될 수 있다.

```
if(조건1) { ... ; }
else if(조건2) { ... ; }
else { ...; }
```

이와 같이 조건을 계속 붙여나가면..

```
if(조건1) { ... ; }
else if(조건2) { ... ; }
else if(조건3) { ... ; }
else if(조건4) { ... ; }
else if(조건5) { ... ; }
else if(조건6) { ... ; }
else if(조건...) { ... ; }
else { ... ; }
```

위와 같은 구조를 만들어 순서대로 조건을 검사할 수 있다.
어떤 조건이 참이면, 그 부분의 내용을 실행하고 조건/선택 구조를 빠져나간다.

이렇게 조건들을 순서대로 검사할 때에는
중간에 범위가 빠지지 않았는지 꼼꼼하게 생각하고 조건들을 만드는 것이 중요하다.
이는 마치 수학에서 빠진 범위 없이 부등식을 만드는 것과 유사하다.

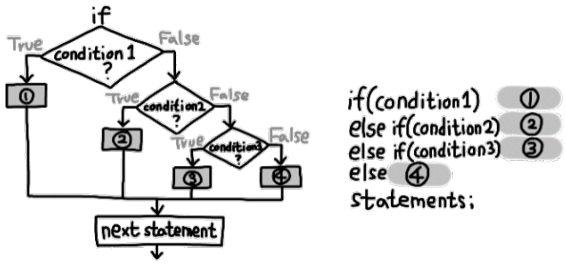
◎ 입력 형식

정수(0 ~ 100) 한 개가 입력된다.

◎ 출력 형식

평가 결과를 출력한다.

입력 예시	출력 예시
73	B



평가 입력받아 다르게 출력하기

평가를 문자(A, B, C, D, ...)로 입력받아 내용을 다르게 출력해보자.

평가 내용
평가 : 내용
A : best!!!
B : good!!
C : run!
D : slowly~
나머지 문자들 : what?

참고
조건/선택문을 복합적으로 구성해 출력할 수도 있지만,

```
switch(정수값)
{
    case 'A': //문자 'A'가 정수값 65('A'의 아스키 값)로 저장되기 때문에 가능하다.
    ...;
    break;

    case 'B':
    ...;
    break;

    case 'C':
    ...;
    break;

    default:
    ...;
}
```

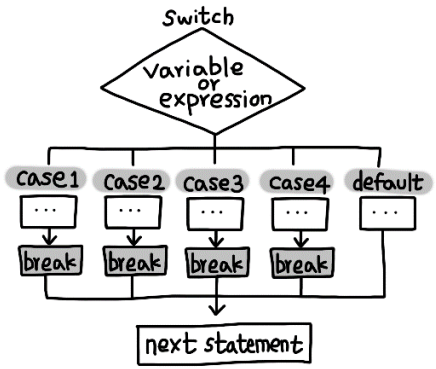
위와 같은 switch( ) ... case... break; 제어문을 사용할 수 있다.

** break; 를 사용하지 않으면 이후의 명령들도 계속 실행된다.
   default: 는 제시된 case 를 제외한 나머지 모든 경우에 실행된다.
   switch( ) 에 주어지는 값은 "정수"값만 가능하며,
   문자도 아스키코드 정수값이기 때문에 가능하다.

◎ 입력 형식
영문자 한 개가 입력된다.
(A, B, C, D 등의 한 문자가 입력된다.)

◎ 출력 형식
평가내용에 따라 다른 내용이 출력된다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
A	best!!!	X	what?



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

월 입력받아 계절 출력하기

월이 입력될 때 계절 이름이 출력되도록 해보자.

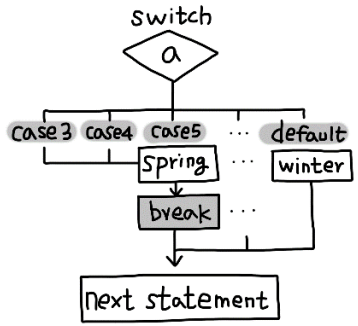
예
월 : 계절 이름
12, 1, 2 : winter
3, 4, 5 : spring
6, 7, 8 : summer
9, 10, 11 : fall

참고
swtich( ).. case ... break; 제어문에서
break;를 제거하면 멈추지 않고 다음 명령이 실행되는 특성을 이용할 수 있다.

```
switch(a)
{
   ...
   case 3:
   case 4:
   case 5:
    printf("spring");
   break;
   ...
}
```

로 작성하면, 3, 4, 5가 입력되었을 때 모두 "spring"이 출력된다.

** 12, 1, 2 는 어떻게 처리해야 할지 여러 가지로 생각해 보아야 한다.



◎ 입력 형식
월을 의미하는 한 개의 정수가 입력된다.(1 ~ 12)

◎ 출력 형식
계절 이름을 출력한다.

입력 예시1	출력 예시1	입력 예시2	출력 예시2
12	winter	3	spring

Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

0 입력될 때까지 무한 출력하기1

정수가 순서대로 입력된다.
-2147483648 ~ +2147483647, 단 개수는 알 수 없다.

0이 아니면 입력된 정수를 출력하고, 0이 입력되면 출력을 중단해보자.
while( ), for( ), do~while( ) 등의 반복문을 사용할 수 없다.

참고
goto 명령문을 사용하면 간단한 반복 실행을 만들 수 있다.
반복 실행 부분을 빠져나오기 위해(즉 무한 반복을 방지하기 위해)
반복 실행 되는 도중에 조건을 검사해야 한다.

goto 이름:
이 명령은 이름: 이 작성된 곳으로 프로그램의 실행 흐름을 바꾸어 준다.

"이름:" 과 같이 콜론(:)이 붙어있는 부분을 이름표(label, 레이블)라고 한다.
레이블은 특별한 선언 없이 사용할 수 있으며 언더바(_)나 영문자로 시작하면 된다.
레이블은 한 단어처럼 공백없이 모두 붙여 써야 한다.

switch( ) ... case ... : ... break; 에서
case ... : 도 일종의 레이블이라고 생각할 수 있다.

goto 레이블:
명령을 사용하면 반복되는 부분을 여러 개 자유롭게 만들 수 있다.

goto 명령은 반복 실행을 만들어낼 수 있는 가장 간단한 명령이지만,
복잡하게(스파게티 코드) 사용하는 경우, 이해가 어렵고 오류가 생기기 쉽다.

예시

```
  int n;
reload:     //레이블은 콜론(:)으로 끝내고, 일반적으로 들여쓰기를 하지 않는다.
  scanf("%d", &n);
  printf("%d", n);
  if(n!=0) goto reload; //reload라고 적혀있는 레이블로 실행 이동
```

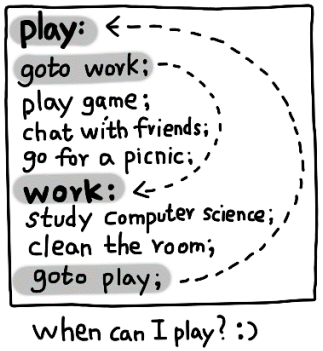
◎ 입력 형식

정수가 순서대로 입력된다.
-2147483648 ~ +2147483647, 단 개수는 알 수 없다.

◎ 출력 형식

입력된 정수를 줄을 바꿔 하나씩 출력하는데, 0이 입력되면 종료한다.
(0은 출력하지 않는다.)

입력 예시	출력 예시
7 4 2 3 0 1 5 6 9 10 8	7 4 2 3



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

정수 입력받아 계속 출력하기

n개의 정수가 순서대로 입력된다.
-2147483648 ~ +2147483647, 단 n의 최대 개수는 알 수 없다.

n개의 입력된 정수를 순서대로 출력해보자.

while( ), for( ), do~while( ) 등의 반복문을 사용할 수 없다.

예시

```
  int n, m;
  scanf("%d", &n);
reget: //레이블은 콜론( : ) 으로 끝난다.
  scanf("%d", &m);
  printf("%d\n", m);
  if(n-- != 0) goto reget; //reget:으로 이동, n의 값 1만큼 감소
```

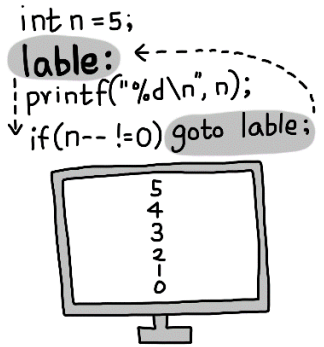
◎ 입력 형식

첫 줄에 정수의 개수 n이 입력되고,
두 번째 줄에 n개의 정수가 공백을 두고 입력된다.
-2147483648 ~ +2147483647, 단 n의 최대 개수는 알 수 없다.

◎ 출력 형식

n개의 정수를 한 개씩 줄을 바꿔 출력한다.

입력 예시	출력 예시
5	1
1 2 3 4 5	2
	3
	4
	5



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

0 입력될 때까지 무한 출력하기2

정수가 순서대로 입력된다.
-2147483648 ~ +2147483647, 단 개수는 알 수 없다.

0이 아니면 입력된 정수를 출력하고, 0이 입력되면 출력을 중단해보자.

```
if(조건)
{ //코드블록
  ...;
}
```

구조를 사용하면 주어진 조건이 참인 경우만 코드블록 부분이 실행된다.

비슷하게 어떤 조건에 따라 반복적으로 실행시킬 때에는 if를 while로 바꾸기만 하면

```
while(조건)
{ //코드블록
  ...;
}
```

와 같은 방법으로 코드블록의 내용을 반복적으로 수행시킬 수 있다.

수행되는 과정은
1. 먼저 조건을 검사한다.
2. 코드블록을 실행한다.
3. 다시 조건을 검사한다.
4. 코드블록을 실행한다.
...

와 같이 주어진 조건이 참인 동안 계속적으로 반복 실행한다.
이렇게 while(조건){...}을 이용하면 goto 명령을 사용하지 않고
반복을 만들어낼 수 있다.

예시

```
int n=1;   //처음 조건 검사를 넘어가기 위해 0이 아닌 값 입력
while(n!=0)
{
  scanf("%d", &n);
  printf("%d", n);
}
```

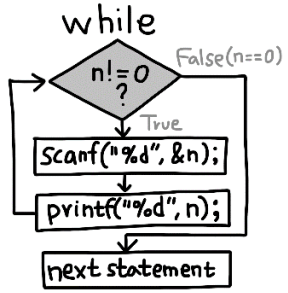
◎ 입력 형식

정수가 순서대로 입력된다.
-2147483648 ~ +2147483647, 단 개수는 알 수 없다.

◎ 출력 형식

입력된 정수를 줄을 바꿔 하나씩 출력하는데, 0이 입력되면 종료한다.
(0은 출력하지 않는다.)

입력 예시	출력 예시
7 4 2 3 0 1 5 6 9 10 8	7
	4
	2
	3



정수 한 개 입력받아 카운트다운 출력하기1

정수(1 ~ 100) 한 개가 입력되었을 때 카운트다운을 출력해보자.

```
while(조건)
{
  ...
}
```
구조를 사용하자.

예시
```
int n;
scanf("%d", &n);
while(n!=0)
{
  printf("%d", n);
  n=n-1;    //n--;와 같다.
}
```

◎ 입력 형식

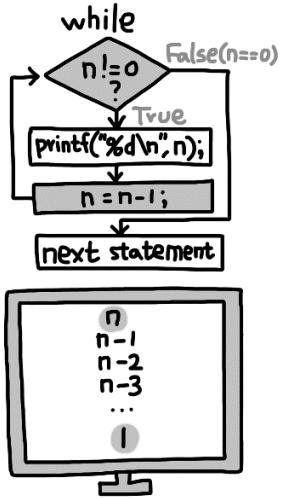
정수 한 개가 입력된다.

1 ~ 100

◎ 출력 형식

1씩 줄이면서 한 줄에 하나씩 1이 될 때까지 출력한다.

입력 예시	출력 예시
5	5
	4
	3
	2
	1



Thought && (Idea ¦¦ Curiosity)

(Offline Coding) ¦¦ (Kernel Codes)

정수 한 개 입력받아 카운트다운 출력하기2

정수(1 ~ 100) 한 개가 입력되었을 때 카운트다운을 출력해보자.

예시

```
int n;
scanf("%d", &n);
while(n!=0)
{
  n=n-1; //n--; 와 같다.
  printf("%d", n);
}
```

◎ 입력 형식

정수 한 개가 입력된다.

1 ~ 100

◎ 출력 형식

1씩 줄이면서 한 줄에 하나씩 0이 될 때까지 출력한다.

입력 예시	출력 예시
5	4
	3
	2
	1
	0