**190405**

**\*관계 연산자**

부동소수점에서 관계연산자가 같게 나와야 할 결과가 같지 않다고 나올 경우가 있다

예제 1)

double d1(100 - 99.99); //0.01

double d2(10 - 9.99); //0.01

cout << setprecision(17);

cout << "d1 is " << d1 << endl; //0.01000000000005116

cout << "d2 is " << d2 << endl; //0.00999999999997868

**\*Short Circuit Evaluation** (Algorithm) : 논리 연산에 있어 결과가 확실히 예측이 되었을 때 남은 연산을 수행하지 않는 것을 말함.

ㄴ AND의 경우 false가 나오면 뒤에 나오는 연산은 생략함. -> 무조건 False 이기 때문.

ㄴ OR의 경우 True가 나오면 뒤에 나오는 연산은 생략한다. -> 무조건 True 이기 때문.

예제 2)

int x = 2, y = 2;

if (x == 1 && y++ == 2)

{

//do Something

/\*

--------------------->

우선 순위 계산 순서 [{x == 1} && {(y++) == 2}]

①x는 1이 아니기 때문에 False

②논리AND(&&)는 알고리즘 상 False 가 나와버리면 뒤에 있는 표현식을 처리하지 않는다.

③그렇기에 핵심은 y++가 연산되지 않는다는 것.

④그래서 결과 출력 시 2가 나온다.

\*/

}

cout << y << endl; //2

**\*De Morgan’s Law**

!(x && y) == !x || !y

!(x || y) == !x && !y

**\*XOR Exclusive-OR (다르면 T, 같으면 F)**

INPUT RESULT

0 0 F

0 1 T

1 0 T

1 1 F

//Xor Algorithm

if (x != y)

{

// T

}

else // F

**\*Binary Numbers**

Decimal To Binary 새로운 방법

Ex) 148

2 4 8 16 32 64 **128** 256 512 1024 2048 4096 …

① 148보다 작거나 같은 수를 찾는다. -> 148 >= 128

② 128는 2^7이므로 이진수의 8번째 자리가 1이다.

③ 148에서 128을 뺀다. -> 20

④ 20보다 작거나 같은 수를 찾는다 -> 20 >= 16

⑤ 반복하여 차가 0이 될 때까지 한다.

\*Add <-> XOR 연산.

\*Two’s Complement

-5를 이진수로 표현하는 법

0 0 0 0 0 0 0 0

부호

비트

**5를 이진수로 표현 후 보수 취한 뒤 +1을 해준다.**

+1을 더해주는 이유 : 두 가지 0이 되는 것을 막기 위함.

가령, -0을 표현하기 위해 0에 보수를 취하고 +1을 더한다고 했을 때…

0000 0000

1111 1111 //Complement

1 0000 0000 +0이 되어 버린다.