181101

//실력이 뛰어나다는 건, 어려운 문제를 잘 해결하는 것이 아니라,

그것은 누구나 신경 쓰고 조심하고 집중해서 문제에 임하게 되는 것이고,

기본에 얼마나 충실하느냐야. 기본은 소홀하기 쉽지. 간과하기 쉽고.

그렇기 때문에야말로, 기본에서 그 사람의 실력과 인격이 드러나는 거야.

팩토리얼 같은 경우, 수가 산술급수적으로 늘어나기 때문에,

자료형에 신경 쓰지 않으면 오버플로우가 발생해.

팩토리얼 구하는데 int로 선언하면 10단위만 넘어가도 바로 넘어불지.

//for 구문 조건식에, 꼭 i만 들어가야 하는 건 아니야.

for ( i = 1; s <= 4000; i = i + 1) 이렇게 해도 돼.

초기값, 조건식, 증감식, 각각에 뭐가 들어갈지는 너무나 자유로워.

꼭 초기값, 조건식, 증감식 각각에 모두 하나의 변수만

들어갈 수 있는 건 아니야.

'초기값 또는 조건식 또는 증감식에 들어 있는 특정 변수' or '비어 있음'

이게 아니라, 모든 변수 뭐가 들어가든 상관 없어.

비어있든, a가 들어가든, x가 들어가든, m 이 들어가든.

그저, 내가 알고 싶은 녀석.

//

for ( i = 1; i <= 10; i = i + 1)

{

if (i == 0)

{

1. continue; 2. break; 3.return;

}

}

1. continue는 for 구문은 유지한 채, 자신이 수행 중이던 한 반복 횟수만 중단 후 다음 횟수 재개.

2. break는 자신이 위치한 곳으로부터, (자신이 속한 for구문 내에서의)그 이후의 연산은 뭐가 있든 상관 없이, 한 겹의 for 구문 밖으로만 탈출. 그 for 구문을 나와서, 그 이후의 연산을 그대로 진행.

3. return은 for구문이든, 그 뒤에 뭐가 있든, 그냥 연산 자체를 종료. 프로그램 자체를 종료.

//while은 이어서 나오는 소괄호 안에 조건식만 들어가. 그러니까 초기값은 while 들어가기 전에 미리 정의해놔야 해.

do while 은, 일단 한 번은 수행해. do. 그리고 중괄호 안의 연산을 수행하고, while 에 달아놓은 조건을 만족한다면, 소괄호 안이 true라면 다시 do로 올라가서 더 반복.

만약 while 의 조건이 애초부터 안 맞는다면, 그냥 while 구문을 사용한다면 실행 자체를 안 하겠지만, do while 구문을 사용한다면 일단 한 번은 수행하고 멈추겠지.

//줄 바꾸는 명령어의 위치 중요성

while ( x < 101)

{

if ( 100 % x == 0)

{

System.out.print(x + "\t");

m = m + 1;

}

if ( m % 3 == 0)

{

System.out.println();

}

x = x + 1;

}

두 번째 if 문의 위치가, 첫 번째 if문과 독립적으로 존재한다면 어떻게 될까.

m = 6, x = 17 이었어봐. 자 이제 x = 18 되면서 첫 번째 if 문의 조건은 만족하지

못하기 때문에 건너뛰겠지만, m은 그대로 6이기 때문에, 두 번째 if 문은 수행해.

그리고 마지막 독립 구문인 x = x + 1도 수행해. 그러면 다음 x = 19가 되겠지.

다시 수행해보자. x = 19이기 때문에 첫 번째 if문은 또 건너 뛰어. 따라서

m 은 변함이 없어. (왜냐하면 m의 값에 대한 서술은 첫 번째 if 문 안에만 있는데,

그걸 건너뛰니까.) 따라서 m 은 그대로 6이고, 두 번째 if문의 조건을 만족하기 때문에

두 번째 if 문도 수행해. 이렇게, 계속해서 줄바꿈이 일어나면서, 가운데가 뛰겠지.

따라서, 줄 바꿈을 고려하는 것 자체를, 첫 번째 if을 만족했을 경우에만 고려하도록

만든다면? 두 번째 if문을, 첫 번째 if 문 안으로 종속시킨다면?

181102

//되게 중요한 거 알려줄게. 우리 프로그래밍 짜잖아. 무슨 특정 프로그램? 도구? 메모장? 같은 데에서 짜잖아. 근데 엔터나 스페이스바는 인식을 못하잖아.(의미 단위 구분하는 것 말고는.) 그럼 우리 컴퓨터 입장에서 생각해볼까? 그들은 사실은!!! 스페이스바나 엔터 등은 인식하지 못하므로 없는 것과 마찬가지이기 때문에,　우리가 입력하는 프로그래밍 언어들을, 모두 한 줄로 인식하는 것과 같아!! 그렇기 때문에, 앞에 있는 연산부터 인식하는 거고, 괄호로 의미 단위를 구분하기 때문에 괄호 짝을 맞춰주지 않으면 멍청해지는 거고, 등등의 현상이 일어나는 것이었어!!

/for구문이나 while구문 등, 반복 중에 특정 조건을 만족하면 해당 구문의 밖으로 나갈 수 있지. 그런데 for 구문을 중첩해서 여러 겹을 써 놓고, 가장 안쪽에 if문을 썼다고 해보자. 그 상황에서 특정 조건을 만족하면 반복문 전체를 끝내고 싶어. if구문 안에 break를 넣을 수 있겠지만, 반복문 전체를 끝내려면 각 층계의 for구문마다 if를 넣어 줘야해. 아오, 귀찮아. 따라서, break를 만나면, 해당 break를 특정 층계의 for구문에게 적용하도록 만들 수 있어. 그것은 for 구문에 이름을 붙여 주는 거야. 그리고 break에도 이름

//

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

for(i = 1; i < 6; i = i + 1)

{

for (m = 6; m > i; m = m - 1)

{

System.out.print(" ");

}

for (m = 1; m < 2 \* i; m = m + 1)

{

System.out.print("\*");

}

System.out.println();

}

내가 직접 생각해 낸 코드! 이것으로 보건대, 프로그래머에게 컨디션 관리는 얼마나 중요한가!

머리가 맑은 상태에서는 번뜩이는 아이디어가 샘솟는다! 생각이 예리해진다!

팀워크란 또 얼마나 중요한가! 상대방에게 내가 어떤 기능을 하는 코드를 어디까지 짰고

어디에서 뭐 때문에 막혔는지, 차근차근 하나하나 꼬치꼬치 설명하다 보면,

나를 막고 있던 벽은 저절로 허물어진다!!