Name: NIM:

Problem Set 2
TK2ICM: *Logic Programming* (CSH4Y3)
Second Term 2018-2019

Day, date	:	Tuesday, February 19, 2019
Duration	:	**60 minutes**
Type	:	***open all***, *individual (no cooperation between/among class participants)*

Instruction:

1. You are not allowed to discuss these problems with other class participants.
2. You may use any reference (books, slides, internet) as well as other students who are not enrolled to this class.
3. Use the predicate name as described in each of the problem. **The name of the predicate must be precisely identical**. Typographical error may lead to the cancellation of your points.
4. Submit your work to the provided slot at CeLoE under the file name `PS2-<your_name>.pl`. For example: `PS2-Albert.pl`. Please see an information regarding your nickname at google classroom.

Problem 1 (20 points) Write the predicate `max3numbers/3` which returns the maximum value of three numbers. For example:

- `max3numbers(0,0,0).` returns
  `0`
  true.
- `max3numbers(2,2,1).` returns
  `2`
  true.
- `max3numbers(2,1,2).` returns
  `2`
  true.
- `max3numbers(1,2,2).` returns
  `2`
  true.
- `max3numbers(1,2,3).` returns
  `3`
  true.
- `max3numbers(1,3,2).` returns
  `3`
  true.
- `max3numbers(3,1,2).` returns
  `3`
  true.
- `max3numbers(-2,-3,-1).` returns
  `-1`
  true.

Note: some side effects, such as the constant **true** or **false**, are admissible.

Problem 2 (20 points) Write the definition for the predicate `mypromise(N)` which returns N lines of the the sentence "`I will study hard for the midterm`" (without quotation marks). The value N must be instantiated and it is assumed to be a positive integer. For example:

(a). `mypromise(3).` returns:

```
I will study hard for the midterm.
I will study hard for the midterm.
I will study hard for the midterm.
```

(b). `mypromise(5).` returns:

```
I will study hard for the midterm.
I will study hard for the midterm.
I will study hard for the midterm.
I will study hard for the midterm.
I will study hard for the midterm.
```

Note: some side effects, such as the constant **true** or **false**, are admissible. However, you must avoid infinite recursive call. If the value N is not a positive integer, then `mypromise(N).` returns **false**.

Problem 3 (20 points) Write the definition for the predicate `factorial(N,Factorial)` which returns **true** whenever $Factorial = N!$. The variable `N` must be instantiated. For example:

- `factorial(0,1)` and `factorial(1,1)`returns **true** (these are the base cases);
- `factorial(2,X)` returns `X = 2`;
- `factorial(3,X)` returns `X = 6`;
- `factorial(4,X)` returns `X = 24`;
- `factorial(10,X)` returns `X = 3628800`.

In general `factorial(+N,X)` returns `X = +N!` for any instantiated value of `+N`. (Hint: for the base case, write `factorial(0,1).` and `factorial(1,1).`.)

Problem 4 (20 points) Suppose $S(n)$ is the following summation:

$$S(n) = 1^3 + 2^3 + 3^3 + \cdots + n^3.$$

Write the predicate `sumcube(N,Sum)` that returns **true** whenever $Sum = 1^3 + 2^3 + \cdots + N^3$. The variable `N` must be instantiated. For example:

- `sumcube(0,0)` and `sumcube(1,1)`returns **true** (these are the base cases);
- `sumcube(2,X)` returns `X = 9`;
- `sumcube(3,X)` returns `X = 36`;
- `sumcube(4,X)` returns `X = 100`;
- `sumcube(10,X)` returns `X = 3025`;
- `sumcube(100,X)` returns `X = 25502500`;

In general `sumcube(+N,X)` returns `X =` $\sum_{i=0}^{+N} i^3$ for any instantiated value of `+N`. (Hint: for the base case, write `sumcube(0,0).` and `sumcube(1,1).`.)

Problem 5 (20 points) Suppose we have the following knowledge base:

```
directTrain(forbach,saarbruecken).
directTrain(freyming,forbach).
directTrain(fahlquemont,stAvold).
directTrain(stAvold,forbach).
directTrain(saarbruecken,dudweiler).
directTrain(metz,fahlquemont).
directTrain(nancy,metz).
```

That is, this knowledge base holds facts about towns it is possible to travel between by taking a direct train. But of course, we can travel further by “chaining together”’ direct train journeys. Write a recursive predicate `travelBetween/2` that tells us when we can travel by train between two towns. For example, when given the query

```
travelBetween(nancy,saarbruecken).
```

it should reply **true**. It is, furthermore, plausible to assume that whenever it is possible to take a direct train from `A` to `B`, it is also possible to take a direct train from `B` to `A`. Hence, the your Prolog program should have the following outputs:

- `travelBetween(saarbruecken,nancy).` returns **true** (possibly yields infinite recursive call, but it is OK for now).
- `travelBetween(nancy,saarbruecken).` returns **true** (possibly yields infinite recursive call, but it is OK for now).
- `travelBetween(nancy,freyming).` returns **true** (possibly yields infinite recursive call, but it is OK for now).
- `travelBetween(freyming,nancy).` returns **true** (possibly yields infinite recursive call, but it is OK for now).
- `travelBetween(nancy,nancy).` returns **true** (possibly yields infinite recursive call, but it is OK for now).