Name: NIM:

Problem Set 2

TK2ICM: *Logic Programming* (CSH4Y3)

Second Term 2019-2020

Day, date	:	Monday, February 17, 2020
Duration	:	**75 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 LMS under the file name PS2-<iGracias_ID>.pl. For example: PS2-albert.pl if your iGracias ID is albert.

1 Maximum of Three Numbers

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.

2 My Promise

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**.

3 Factorial

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).`.)

Remark 1 For any non-negative integer N, the value of $N!$ (N factorial) is defined as follows:

$$N! = \begin{cases} 1, & \text{if } N = 0 \\ N \cdot (N-1)!, & \text{if } N \geq 1. \end{cases}$$

For example, we have $0! = 1$, $1! = 1$, $2! = 2 \cdot 1 = 2$, and $3! = 3 \cdot 2 \cdot 1 = 6$.

4 Cubic Sum

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`, because $S(2) = 1^3 + 2^3 = 9$;
- `sumcube(3,X)` returns `X = 36`, because $S(3) = 1^3 + 2^3 + 3^3 = 36$;
- `sumcube(4,X)` returns `X = 100`, because $S(4) = 1^3 + 2^3 + 3^3 + 4^3 = 100$;
- `sumcube(10,X)` returns `X = 3025`, because $S(10) = 1^3 + 2^3 + \cdots + 10^3 = 3025$;
- `sumcube(100,X)` returns `X = 25502500`, because $S(100) = 1^3+2^3+\cdots+100^3 = 25502500$.

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).`.)

5 Maze

Problem 5 Imagine that the following knowledge base describes a maze. The facts determine which points are connected, i.e., *from which point you can get to which other point in one step*. Furthermore, imagine that all paths are one-way streets, so that **you can only walk them in one direction**. Thus, for example you can get from point 1 to point 2, but not the other way round.

```
connected(1,2).
connected(3,4).
connected(5,6).
connected(7,8).
connected(9,10).
connected(12,13).
connected(13,14).
connected(15,16).
connected(17,18).
connected(19,20).
connected(4,1).
connected(6,3).
connected(4,7).
connected(6,11).
connected(14,9).
connected(11,15).
connected(16,12).
connected(14,17).
connected(16,19).
```

Write a predicate `path/2` that tells us from which point in the maze we can get to which other point when chaining together connections given in the above knowledge base. For example:

- `path(5,10).` returns
  true;
  false.

  (we can go from 5 to 10 using the following path: $5 \to 6 \to 11 \to 15 \to 16 \to 12 \to 13 \to 14 \to 10$, ignore the **false** output)

- `path(1,19).` returns
  false.

  (we cannot go from 1 to 19, i.e., there is no path from 1 to 19)

- `path(13,X).` returns
  `X = 14;`
  `X = 9;`
  `X = 17;`
  `X = 10;`
  `X = 18;`
  false.

  (from 13, we can go to 14, 9, 17, 10, and 18)

- `path(X,19).` returns
  `X = 16;`
  `X = 5;`
  `X = 15;`
  `X = 6;`
  `X = 11;`
  false.

  (we can go to 19 from following points: 16, 5, 15, 6, and 11)

- `path(X,5).` returns
  false .

  (there is no path from any point to 5)

Note: carefully define your predicate to avoid stack overflow or infinite recursion.