Assignment #2 on Prolog

Date Due: February 26, 2026

Total: 100 marks

Include for all programs both rules/code and execution. Don't use code from the internet. Write your own code.

Please use only the type of constructions that you see the slides/videos/examples. Don't use built-in predicates that will solve the problem for you[1]; that will defeat the purpose of the assignment. You are allowed to use the predicates that are present in the slides (with or without changes).

Don't use predicates that emulate procedural programming constructions like for loops(any of them: logical or counter control loops) or conditionals(all kind). Use just the BNF syntax present in the slides – no other extensions. You need to think in Prolog, not in a procedural programming language. By using procedural programming constructions you'll get 10% of the mark or 0.

The content of the slides is enough to complete the assignment. You must send the code(text) together with instructions of how to run the programs (the `Readme` file), and a text capturing the execution (the `Run.txt` file) (no binary files!)[2]. Do not use AI. The use of AI for solving the assignment gives you a 0 for the assignment or the whole course. Just use the general format required for all assignment submissions, as it is described in the slides.

1. (24 marks) Write a solution for a function/predicate that computes the following list defined as:

   (a) The result of combining two empty lists is empty list.

   (b) Adding an element of the list to the right for the first list, generates lists that are the result of appending that element to the result of combining the first list with the second list.

   (c) Adding an element of the list to the right for the second list, generates lists that are the result of appending that element to the result of combining the first list with the second list.

   (d) elements obtained using the above rules should be obtained only once.

---

[1] Some versions of Prolog may have that predicates built in

[2] pictures/screenshots are binary files

```
|? - combine([a,b,c],[1,2,3],X).

X = [a,b,c,1,2,3] ? ;

X = [a,b,1,c,2,3] ? ;

X = [a,1,b,c,2,3] ? ;
....
X = [1,2,3,a,b,c]

yes
```

2. (46 marks) Write a predicate to interactively guess if a number has sum property in relation to a list stored internaly in at most k tries. The number k is read from standard input. After k is entered, the user must enter a number x and k values i_1, i_2, i_k, $1 \leq i_1, i_2, \ldots, i_k \leq n$.

   A number x has sum property with respect to the list $[A_1, \ldots A_n]$ if the indexes satisfy this condition $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ and $x = \sum_{i=0}^{k} A_{i_k}$.

   If the number has sum property, the programs ends(user wins), if not, the program displays the number of tries left and the user tries to guess the number again.

   In case the user types a number that is lower than the sum of all numbers in the list this should be displayed. After each try, the number of remaining tries should be displayed.

   If the numbers i_1, i_2, i_k, $1 \leq i_1, i_2, \ldots, i_k \leq n$ entered by the user do not have the property that $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, the program decreses the number of tries by two and the user must guess again.

   Outputs can be obtained with predicates `format(' string')`, `write(X)`, and `nl`, and for an input we can use the predicate `read(X)` (integer input will be followed by a "."). The list of numbers should be hardcoded (inside the program).

   For example:

```
game.
6.
You have 6 tries to guess a number with sum property.
60.
1.
2.
3.
4.
5.
6.
The max sum is lower.
You have 5 tries to guess a number with sum property.
20.
1.
3.
5.
7.
4.
```

```
You have 3 tries to guess a number with sum property.
10.
1.
4.
6.
You guessed right.
yes.
```

3. (40 marks) Write a predicate to interactively mixes two lists (stored internally) into one list depending on an the value of an integer k is read from standard input. The first k elements are obtained as follows:

   (a) The first element of the list is the maximum of the first element of the first list and the k-th element of the second list.

   (b) The second element of the list is the minimum of the second element of the first list and the $k-1$-th element of the second list.

   (c) the rest of the elements are obtained by alternatively applying the previous two rules.

   If k is less then the minimum of the lengths of both lists the user must input the number again.

   The next elements are obtained by mixing the rest of the elements, one from the second list one from the first list, starting with $(k+1)$th elements. If one list is longer then the other one, the missing elements from the shorter list are ignored.

   For example:

```
game.
Number k:
4.
Initial lists [1,4,5,7,9,-10,-3,2,8,9]  [7,14,1,-7,-9,10,3,-2]
Result [7,4,5,-7,9,-9, -10,10 -3,3, 2,-2,8 ,9]
```

```
game.
Number k:
12
The numer is to big enter a lower number
11
The numer is to big enter a lower number
3
Initial lists [11,4,5,7,9]  [7,14,1,-7,-9,10,3,-2]
Result [11,4,5,7,-7,9,-9,10,3,-2]
```