# TC2006 Programming Languages

## Assignment 7: Logic programming in Prolog

This assignment will give the opportunity to practice and get familiar with the Prolog programming language. Internally, through Prolog comments, you must include your student ids and names in the source code file, as well as the description of the functions. **Mandatory requirement:** properly use the **cut clause (!).**

***1.*** Program the **nth** predicate that determines if a given integer is a prime number. A prime number is a positive number greater than 1 that is only divisible by itself and 1.
Test cases:

```
?- prime(-5).   => false
?- prime(15).   => false
?- prime(7).    => true
?- prime(19).   => true
```

**2.** Program the **range** predicate that gets an incremental list of integers between two values passed as arguments. Assume that the second argument is a number greater than or equal to the first.
Test cases:

```
?- range(3, 3, R).   => R = [3].
?- range(2, 7, R).   => R = [2,3,4,5,6,7].
```

**3.** Program the **cartesian** predicate which obtains a list of element pairs constructed as the Cartesian product of two sets represented as lists.
Test cases:

```
?- cartesian([], [1, 3, 8], R).     => R = [].
?- cartesian([1, 3, 8], [a, b], R).
          => R = [[1,a],[1,b],[3,a],[3,b],[8,a],[8,b]].
```

**4.** Program the **count_deep** predicate that counts the times that a particular element appears within a nested list.
Test cases:

```
?- count_deep(b, [a,b,c], R).                          => R = 1.
?- count_deep(g, [a,[g,[c,d],[e,b,[g]],h],g], R).      => R = 3.
?- count_deep(z, [a,[b,[c,d],[e,f,[g]],h],i], R).      => R = 0.
```

**5.** Program the **unique_list** predicate that obtains a list with the elements that do not appear repeated within a nested list.
Test cases:

```
?- unique_list([a,b,a,b,a,c], R).          => R = [c].
?- unique_list([a,[b,[c,[a],c],b]], R).    => R = [].
?- unique_list([a,[b,[c,[d]]]], R).        => R = [a,b,c,d].
```

**6.** Program the **greater** predicate that returns a list of elements greater than a given value in a binary tree described with the function:

**tree(Root, LeftSubtree, RightSubtree) .**

Test cases:

```
?- greater(10,nil,H).    => H = [] .
?- greater(10, tree(8, tree(5, tree(2, nil, nil), tree(7, nil, nil)),
              tree(9, nil, tree(15, tree(11, nil, nil), nil))),H).
       => H = [15,11].
?- greater(0, tree(8, tree(5, tree(2, nil, nil), tree (7, nil, nil)),
             tree(9, nil, tree(15, tree(11, nil, nil), nil))),H).
       => H = [8,5,2,7,9,15,11].
```

**7.** Program the **seed** predicate that from a list of numbers creates a binary search tree described with the function:

**tree(Root, LeftSubtree, RightSubtree) .**

In a binary search tree, the root is always greater than or equal to the values in its left subtree while it is less than the values in its right subtree.

Test cases:

```
?- seed([], A).          => A = nil.
?- seed([2,3,1], A).
       => A = tree(2, tree(1, nil, nil), tree(3, nil, nil)).
?- seed([8,5,2,7,9,15,11], A).
       => A = tree(8, tree(5, tree(2, nil, nil), tree(7, nil, nil)),
                      tree(9, nil, tree(15, tree(11, nil, nil), nil))).
```