
I. Language features

- (a) [20 marks] Fill in the following table with either "yes" or "no" in each cell. When deciding whether a language has a particular feature, only consider the built-in (default) feature set, not features provided by extra modules, or added by the user.

	Haskell	Java / C / C++	Prolog
Backtracking			
Pattern recognition			
Loops			
Ad-hoc polymorphism			
Mutable (changeable) variables			
Recursion			
Higher order functions			
Lambda expressions			
Strong typing			
Output parameters			

- (b) [1 mark] Rewrite the following Haskell code in Prolog.

```
1st = [x | x <- [1..9], (p x)]
```

(c) [4 marks] Rewrite the following Java code in Prolog.

```
int f( int[] x ) {
  int l = 0;
  for (i = 0; i <= x.length; i++)
    switch (x[i]) {
      case 1:
      case 2:
        l += 2;
        break;
      case 3:
      case 4:
        l++;
      case 5:
        l--;
        break;
      default:
        break;
    }
  return l;
}
```

II. Consider the following Haskell code.

```
f x y = foldl (\k l -> (div k 2) + 1) 0
           [a + b + c | a <- [2..x], b <- [y..a^2], c <- (reverse [a..b])]
```

(a) [8 marks] Rewrite this function in Prolog.

(b) [7 marks] Rewrite this function in an imperative language of your choice (or pseudocode). YOU MAY NOT USE AN ARRAY OR A LIST IN THE IMPERATIVE VERSION.

III. Prolog has a feature called *negation-by-failure*. That is, according to the predicate logic underpinning Prolog, the negation of a predicate occurs when the predicate fails.

- (a) [4 marks] Write a function `not/1` that negates the truth value of its sole parameter. For example, given the below definition for `p/1`, then `not(p(a))` should always fail, and `not(p(c))` should always succeed. You may wish to make use of the built-in predicate `fail/0` which always fails. However you may **NOT** make use of the built-in `\+` operator.

```
p( a ).  
p( b ).
```

- (b) [9 marks] Java, C and C++ all have some version of a *string tokenizer*. A string tokenizer is a function that takes a string and (sometimes) some character to use as a delimiter (often defaults to whitespace), then returns, one-by-one, all the substrings of the original string that are set apart by that delimiter. Write a list tokenizer in Prolog called `tokenize/3` that can tokenize a list as follows. You must make use of the `append/3` predicate.

```
| ?- tokenize( X, [3,4,0,7,4,0,2,0,4,5,9,0,1], 0 ).  
  
X = [3,4] ? a  
  
X = [7,4]  
  
X = [2]  
  
X = [4,5,9]  
  
X = [1]
```

-
- (c) [2 marks] There is a function in Haskell that does this. What is it called?
(alternatively: write a function in Haskell that will tokenize a string).

IV. Use the following database to answer both parts of this question.

a(1, 2).
a(2, 3).
a(2, 4).
a(6, 1).

b(1, 2).
b(1, 4).
b(2, 4).
b(3, 5).
b(5, 6).

(a) [9 marks] Given the following definition for $c/2$, what is the result when the query $c(X, Y)$ is made?

$c(A, B) :- a(A, B).$
 $c(A, B) :- b(A, C), c(C, B).$

(b) [9 marks] If we now change the definition of $c/2$, to the following, what is the result when the query $c(X, Y)$ is made?

$c(A, B) :- a(A, B).$
 $c(A, B) :- b(A, C), !, c(C, B).$

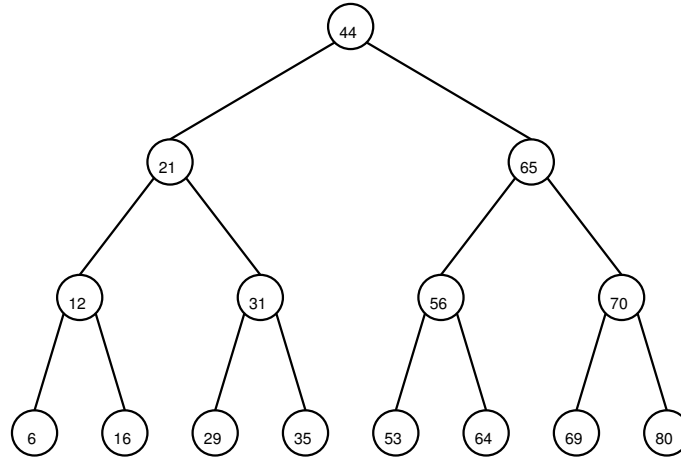
V. Consider the following lambda expression.

$$(\lambda xyz.(yzx) - x) 4 (\lambda xy.(\lambda x.2 * x)y + x) 2$$

(a) [7 marks] Draw the tree resulting from this expression.

(b) [5 marks] Reduce this lambda expression as far as possible.

VI. A *binary search tree* (sometimes also called a *dictionary*) is a binary tree with the following property: for every node in the tree, all elements of the left subtree have a value less than that node, and all elements of the right subtree have a value greater than that node. See the following image for an example of a binary search tree (BST). The aim of this question is that you implement a working BST in Prolog. Note it is not required that a BST always be balanced. For this question, you should assume the recursive definition of a binary tree which is either `nil`, or `t(a,t1,t2)` where a is an integer and the left and right children $t1$ and $t2$ are themselves trees.



- (a) [5 marks] Write a function `addnode/3` that takes a BST, and an integer, then adds that integer to the tree. If the integer is already in the tree, then the tree should just be returned unchanged.

(b) [4 marks] Write a function `searchnode/2` that takes a BST, and an integer and succeeds if that integer is in the tree. `searchnode` should fail if the integer is not in the tree.

(c) [4 marks] Write a function `listtotree/2` that takes a list of integers, and creates a BST out of them. That is, it should add each element of the list, one-by-one, to a tree which is initialized to `nil` (the empty tree).

(d) [1 mark] Write a function `searchlist/2` that takes a list of integers, and an integer, and succeeds if that integer is in the list, failing otherwise.

VII. Unification in Prolog.

(a) [7 marks] Examine the following Prolog code:

```
{1} foo( A, B, C ) :-  
{2}     D = A,  
{3}     E =.. [C, A, 2],  
{4}     G = a( B, E ),  
{5}     a( 1, D ) = G.
```

If one were to pose the query: `foo(X, Y, b)`, would this succeed? If so, then what will be the value of `Y`? If it were to fail, why would it fail?

(b) [1 mark] Change the operator so the following query succeeds:
`2 = 1+1.`

(b) [1 mark] Change the operator so the following query succeeds:
`1 = 3/3.`

(b) [1 mark] Change the operator so the following query succeeds:
`'A' =< b.`

VIII. Examine the following Prolog code:

```
f(X,Y) :- X > 0,
         f(X,Y,2).

f(1, [], _) :- !.
f(X, [Y|Ys], Y) :- Z is X // Y,
                  X =:= Z * Y,
                  !,
                  f(Z, Ys, Y).
f(X, Y, Z) :- g(X, Z, W),
             f(X, Y, W).

g(_, 2, 3) :- !.
g(X, Y, Z) :- Y * Y < X,
             !,
             Z is Y + 2.
g(X, _, X).
```

[4 marks] What is the value of X after posing the query `f(3, X)`.?

[4 marks] What is the value of X after posing the query `f(4, X)`.?

[4 marks] What is the value of X after posing the query $f(5, X)$.?

[4 marks] What is the value of X after posing the query $f(6, X)$.?