Principles of Programming Languages [PLP-2014]
Exercises on Programming Languages - May 26, 2015

1) Strings are represented differently in various programming languages. For example, they are represented
   a) as character arrays in C,
   b) as character lists in Haskell, and
   c) as objects in JAVA.

For each of those languages, describe how can you perform the operations listed below. For each operation, indicate the primitive provided by the language (if it exists, and if you know it), and, more importantly, the algorithm to compute it, in pseudocode.
   d) length
   e) concatenation
   f) equality comparison

2) Let be given the following type definitions in a programming language which uses structural type equivalence:

```
type T1 = struct{ int a;  bool b; };
type T2 = struct{ int a;  bool b; }
type T3 = struct{ T2 u; T1 v;  }
type T4 = struct{ T1 u; T2 v; }
```

In the scope of the declarations **`T3 a`** and **`T4 b`**, is the assignment **`a = b`** permitted? Justify your answer.

3) How are "type compatibility", "coercion" and "casting" related? Provide a couple of examples to illustrate such relationships.

4) Which type is assigned to each of the following functions using polymorphic type inference?

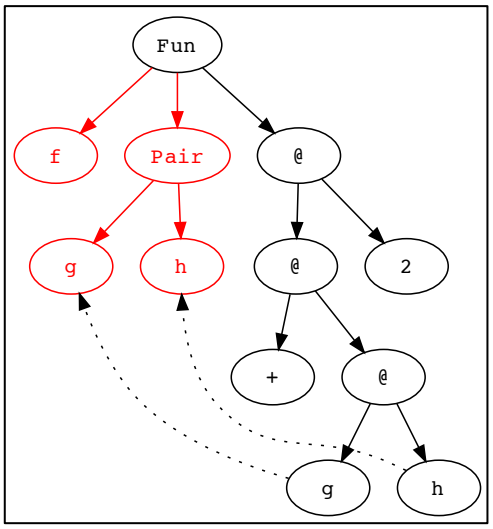
```
fun G(f,x){return f(f(x));}
fun H(t,x,y){if (t(x)) return x;  else return y;}
fun K(x,y){return x;}
```

5) Show in your favorite programming language that the integer type can be defined as a recursive type starting with just the value **`null`**. Then write a function that checks if two integers defined in this way are equal.

6) Describe three different ways of allocating in memory a 2-dimensional array A of dimensions N x M. Assuming that indexes range in {0, …, N-1} and {0, …, M-1} respectively, give the formula for accessing an arbitrary element `A[i][j]` for each of the three proposed allocation schemes.

7) Using the notation of Cartesian products, mappings, and disjoint unions, write down
   a) the set of values and the cardinality of each of the following C types:
      - `enum Suit {club, diamond, heart, spade};`
      - `struct Card {Suit s; byte r;};`
      - `struct Turn { bool pass; Card play; };`
   b) the set of values and the cardinality of each of the following ADA types:
      - `type Suit is (club, diamond, heart, spade);`
      - `type Rank is range 2 .. 14;`
      - `type Card is record  s: Suit;  r: Rank; end record;`
      -
```
type Turn (pass: Boolean) is record
        case pass is
             when false => play: Card;
             when true => null;
        end case;
end record;
```

8) Use the parse graph to the right to calculate the type for the function
   `f(g,h) = g(h) + 2`
   Assume that **2** has type **Integer** and **+** has type **Integer → Integer → Integer**.



9) Infer the types of the following Haskell functions, given that the type of **"+"** is **Integer → Integer → Integer**:

```
apptolist z f []     = z
apptolist z f (x:xs) = f x (apptolist z f xs)
add []     = 0
add (x:xs) = x + add xs
sum xs = apptolist 0 (+) xs
```

Compare the functions **add** and **sum**.

10) The third line in the next box shows the type inferred by the Haskell compiler for the **isInDouble** function. Explain the meaning of all the components of the type, and show how the compiler inferred them.

```
isInDouble n [] = False
isInDouble n (x:xs) = (x == 2*n) || isInDouble n xs

isInDouble :: (Num a, Eq a) => a -> [a] -> Bool
```

11) Type classes in Haskell are used to support *ad-hoc* polymorphism or *subtype* polymorphism? Justify your answer.

12) *Innermost* and *outermost* evaluation strategies may require a different number of steps to evaluate an expression. Show how many steps are necessary to evaluate the expression **square (1 + 2)** using the rule **square (x) -> x * x** and the obvious rules for addition and multiplication, by using:
   a) innermost (applicative) evaluation
   b) outermost (normal order) evaluation
   c) outermost evaluation with memoization

13) By exploiting the syntax for *list comprehension* of Haskell, write expressions that denote:
   a) The list of squares of natural numbers from 0 to 100;
   b) The list of all *Pythagorean triples* up to *n*, i.e., of all triples *(x, y, z)* such that *x, y, z* <= *n* and x*x + y*y = z*z.

14) Certain monads in Haskell are better interpreted as containers. Consider from this perspective the type constructor **Set** with the obvious meaning, and discuss what are the steps to turn it into a Haskell monad, stressing how definitions would differ from those of the list monad.

15) Compare lambda expressions in Haskell and in Java. Under which aspects are they similar? And what are the main differences? Is Java 8 a higher-order language?

16) Describe three different kinds of sources for Java 8 streams, one of which should be able to generate an infinite stream of objects.

17) Describe three different kinds of intermediate operations for Java 8 streams, discussing whether they are stateless or stateful.

18) Describe three different kinds of terminal operations for Java 8 streams.