A feladat rövid összefoglalása
Típusellenőrzőt kell írni a számok és szövegek programozási nyelvéhez.

Szintaxis: jegyzetben 3.1-3.2 definíciók.

Típusrendszer: 3.3-3.16.

A program bemenete egy Γ környezet, egy e kifejezés és egy τ típus. Kimenete pedig YES vagy NO annak megfelelően, hogy $\Gamma \vdash e : \tau$ levezethető -e.

A 3.4 lemma biztosítja, hogy e formája alapján megállapítható, hogy mi legyen a levezetés következő lépése. Pl. ha $e = e_1+e_2$, akkor $\Gamma \vdash e : \tau$ akkor vezethető le, ha $\tau = int$ és $\Gamma \vdash e_1 : int$ és $\Gamma \vdash e_2 : int$ is levezethető.

Bemeneti formátum
A bemenetet JSON formátumban kell megadni. Csak valid bemeneti formátumú adatokkal (tehát ABT-kkel) tesztelünk, tehát megfelelő számú argumentuma lesz minden operátornak megadva stb.

A következő ítéleteknek:

$x:str \vdash x : str$
$x:int \vdash let\ x\ in\ y.x+1 : int$
$x:int,y:str \vdash 29 - x : int$
$x:str,y:int \vdash x + y : int$
$x:int \vdash let\ x\ in\ y.y : str$
megfelelő JSON kódok a következők:

```
{"con": {"x":"Str"}, "exp": {"op":"Var","args":"x"}, "ty": "Str"}
{"con": {"x":"Int"}, "exp":
{"op":"Let","args":[{"op":"Var","args":"x"},"y",{"op":"Add","args":[{"op":"Var","args
":"x"},{"op":"IntLit", "args":1}]}]}, "ty": "Int"}
{"con":{"x":"Int", "y":"Str"}, "exp": {"op": "Sub", "args": [{"op": "IntLit",
"args": 29}, {"op": "Var", "args": "x"}]}, "ty": "Int"}
{"con":{"x":"Str", "y":"Int"}, "exp": {"op": "Add", "args": [{"op": "Var", "args":
"x"}, {"op": "Var", "args": "y"}]}, "ty": "Int"}
{"con":{"x":"Int"},"exp":{"op":"Let","args":[{"op": "Var", "args":"x"},"y",{"op":
"Var", "args":"y"}]},"ty":"Str"}
```

A programnak erre a bemenetre ezt a kimenetet kell adnia:

```
YES
YES
YES
NO
NO
```

Általánosságban a JSON kódolást a következőképp adjuk meg. A program bemenete az <input>.

```
<var> ::= <JSON string>
<str> ::= <JSON string>
<int> ::= <JSON egész szám>

<exp> ::=
    { "op": "Var",    "args": <var> }
  | { "op": "IntLit", "args": <int> }
  | { "op": "StrLit", "args": <str> }
  | { "op": "Add",    "args": [<exp>, <exp>] }
  | { "op": "Sub",    "args": [<exp>, <exp>] }
  | { "op": "Cat",    "args": [<exp>, <exp>] }
  | { "op": "Len",    "args": <exp> }
  | { "op": "Let",    "args": [<exp>, <str>, <exp>]}

<ty> ::= "Int" | "Str"

<con> ::= {} | {<var>: <ty>} | {<var>: <ty>, <var>: <ty>} | ...
```

<input> ::= {"con": <con>, "exp": <exp>, "ty": <ty>}

Implementációs részletek

A feladatot a következő négy programozási nyelv valamelyikén lehet megoldani: Haskell, Java, C++ és Python. A megoldásban lehetőség szerint törekedjünk a minél egyszerűbb, rövidebb (minél kevesebb külső függőséget alkalmazó), ámde olvasható és érthető forráskód írására! Az egyes nyelveken készített implementációkkal kapcsolatos egyéb elvárásokat az alábbiakban foglaljuk röviden össze.

Java

Java nyelv esetében két lehetőség adott. Az első esetben a beadott megoldás a gyökérben tartalmazhat egy Main.java állományt, ebben kell lennie egy Main osztálynak, ahol a main() metódus elérhető. A Main osztály ne legyen benne semmilyen package-ben. A programnak ilyenkor tehát a következőképpen fordíthatónak kell lennie:

```
$ javac Main.java
```

valamint a következőképpen futtathatónak:

```
$ echo '{"con": {"x":"Str"}, "exp": {"op":"Var","args":"x"}, "ty": "Str"}' | java
Main
YES
$
```

Külső programkönyvtárakat jar kiterjesztésű fájlokként ugyanabban a könyvtárban kell megadni, mint ahol a Main.java van.

A másik esetben a beadott megoldás a gyökérben tartalmazhat egy pom.xml állományt, amely azt írja le, hogy Maven segítségével miként lehet fordítható és futtatható. A programnak ilyenkor tehát a következőképpen fordíthatónak kell lennie:

```
$ mvn package
```

valamint a következőképpen futtathatónak:

```
$ echo '{"con": {"x":"Str"}, "exp": {"op":"Var","args":"x"}, "ty": "Str"}' | mvn
exec:java
YES
$
```

Feltételezhetjük, hogy a tesztelésnél a JDK 1.7, 1.8 és a Maven 3.1 változata áll rendelkezésre.

C