Interpretation and Compilation

TEST 1A

Luis Caires

Universidade Nova de Lisboa

15 October 2014

Test Statement

The goal of the exercise is to extend an interpreter for a simple expression language with additional constructs for manipulating strings.

The basic project files will be available in the CLIP when the test starts.

The basic language includes arithmetic operations, identifiers and declaration blocks, as defined by the following grammar:

```
program ::= exp ;;
exp ::= exp + exp |
        exp - exp |
        exp * exp |
        exp / exp |
        ( exp )   |
        num |
        id  |
        decl (id = exp)+ in exp end
```

Test Statement

The grammar is to be extended with

```
< STR: ("\"" ["0"-"9","a"-"z","A"-"Z"]* "\"") >

exp ::= ... |
        STR |
        exp @ exp | tostr exp
```

The **`STR`** token denotes a string literal, e.g., `"Luis Caires"`.

The expression `e1@e2` computes the concatenation of the results (expected to be string values) of evaluating `e1` and `e2`.

The expression **`tostr`** `e` expects `e` to evaluate to an integer value and then returns its string representation of `e1`.

In the next page we illustrate the evaluation of some sample expressions.

Examples

```
“hello”@” “@”world”;; // input (black)
“hello world”           // evaluation result (red)
“two is not “@tostr (1+2);;
“two is not 3“
decl x=”apples” y=”oranges” in x@” and “@y end;;
”apples and oranges”
decl x=”o” in
 decl y=x@x in
  “H” @ decl z=y@y in z@z end @ “!”;;
“Hoooooooo!”
decl x=”X” in
 decl y=x@x in y@ tostr (y+1) @ tostr (1+y) @y;;
“XX100001000XX”
```

Test Statement

NOTE: Before doing anything else, compile and run the provided code.

What you are expected to do (extending the provided source files):

Extend the JavaCC parser. Extend the LL grammar to deal with the additional three constructs string literal, concatenation, and tostr.

AST construction. Define the additional AST node classes.

Representation of string values. Define a class to represent string values with its required operations, extending the **IValue** interface.

Extend the interpreter. Define the **IValue eval(Environ e)** method in the new ASTNode classes.

Test your interpreter. It should give the right answers for the examples shown and any other we may think of.