Module : Compilation L3 ASYRIA
2019-2020
Thierry Goubier

Sujet de TP : Interpréteur

Sujet à rendre, noté, sous la forme d'un package pour Pharo/SmaCC.

Énoncé :

1. Appliquer le cours 5 / préparer l'interpréteur
2. A partir du projet existant, écrire un interpréteur pour le langage de programmation suivant :
   - Un fichier est une séquence d'expressions séparées par des ;
   - Un commentaire est n'importe quoi entre { et }.
   - Constantes numériques entières.
   - La structure de contrôle if condition then expression else expression (else optionnel).
   - Des déclarations de variables de la forme type nom de variable, avec type : integer.
   - Des expressions permettant d'affecter une valeur à une variable (opérateur :=)
   - Une condition est une expression entière avec 0 faux, et n'importe quelle valeur non 0 est vraie.
   - Les opérateurs + * - /
   - Des scopes begin ... end permettant de redéfinir des variables
   - Tout est expression et retourne une valeur
   - Des fonctions peuvent être définies avec le mot clé function et être appelées
   - La valeur de retour de la fonction est la valeur de la variable portant son nom.
   - L'interpréteur affiche comme résultat final la valeur de la dernière expression du programme

L'interpréteur à rendre sera le package CompilationTP-Nom1Nom2.mcz.

Les tests suivants sont à implémenter. La note sera fonction du nombre de tests à vert.

```
test1
     "test that 3 + 4; equals 7"

     self assert: (InterpreteurParser parse: '3 + 4;') = 7

test2
     "test that 4 - 3; equals 1"

     self assert: (InterpreteurParser parse: '4 - 3;') = 1

test3
     "test two statements:"
```

```
    self assert: (InterpreteurParser parse:
‘3 + 4;
4 – 3;’) = 1

test4
    "test a variable."

    self assert: (InterpreteurParser parse: 'var a: integer;
3 + 4;') = 7

test5
    "test a variable."

    self assert: (InterpreteurParser parse: 'var a: integer;
a := 3 + 4;') = 7

test6
    "test a variable."

    self assert: (InterpreteurParser parse: 'var a: integer;
a := 3 + 4;
a - 5;') = 2

test7
    "more variables."

    self assert: (InterpreteurParser parse: '
var x: integer;
var y: integer;

x = 3;
y = 5;
x + y;') = 8

test8
    "undeclared variable."
    <expectedFailure>

    self assert: (InterpreteurParser parse: '
var x: integer;

x := 3;
y := 5;
x + y;') = 8

test9
    "variable without a value is set at 0."

    self assert: (InterpreteurParser parse: '
var x: integer;
var y: integer;
```

```
y := 5;
x + y;') = 5
```

test10

```
     "if else."

     self assert: (InterpreteurParser parse: '
var x: integer;
var y: integer;

x := 1;
if x then
     y := 5;
else
     y := 4;
y;') = 5
```

test11

```
     "if else."

     self assert: (InterpreteurParser parse: '
var x: integer;
var y: integer;

x := 0;
if x then
     y := 5;
else
     y := 4;
y') = 4
```

test12

```
     "scopes."

     self assert: (InterpreteurParser parse: '
var x: integer;
var y: integer;

x := 3;
y := 1;

begin
     var y: integer;
     y := 4;
     x := x + y;
end;
x := x + y;
x;') = 8
```

test13

```
     "scopes."
```

```
     self assert: (InterpreteurParser parse: '
var x: integer;
var y: integer;

x := 3;
y := 1;

begin
     y := 4;
     x := x + y;
end;
x := x + y;
x;') = 11
```

test14

```
     "if with scopes."

     self assert: (InterpreteurParser parse: '
var x: integer;
var y: integer;

x := 0;
y := 1;

if x
then
     begin
          var y: integer;
          y := 4;
          x := x + y;
     end
else
     begin
          y := 3;
          x := x + y;
     end;
x := x + y;
x;') = 6
```

test15

```
     "if with scopes."

     self assert: (InterpreteurParser parse: '
var x: integer;
var y: integer;

x := 3;
y := 1;

if x
then
     begin
```

```
          var y:integer;
          y := 4;
          x := x + y;
     end
else
     begin
          y := 3;
          x := x + y;
     end;
x := x + y;
x;') = 8
```

test16

```
     "functions."

     self assert: (InterpreteurParser parse: '
var y:integer;
function f(x:integer):integer;
begin
     f := x * x;
end;

y := f(2);
') = 4
```

test17

```
     "functions, scoped."

     self assert: (InterpreteurParser parse: '
int x;

function f(x:integer):integer;
begin
     f:= x * x;
end;
x := 0;
x := f(3);
') = 9
```

test18

```
     "Factorial, recursive."

     self assert: (InterpreteurParser parse: '
function factorial(n:integer):integer {
  if n then
    factorial := n * factorial(n - 1);
  else
    factorial := 1;
}

factorial(4);
') = 24
```