Project 5: Interpreting SpartyTalk

DUE: November 11, 2022 11:59 PM EDT**.** ***Please make sure to always make backups of your files!***

Description

In this project, we continue implementing the language called *SpartyTalk* by finishing the interpreter of its basic syntax. The implementation of this project should be based on the lexer implemented in Project 1 and the parser implemented in Project 3.

First, let us do a quick recap of what *SpartyTalk* is. The following is a sample program written in *SpartyTalk*:

```
gogreen;
nvar a = -10.5;
svar b = “hello\n”;
svar c = “world”;
svar d = b + c;
nvar e = a * 2;
nvar f = 3.5 / a;
f = f / 7.5 * 3;
spartysays “hi “ + e;
gowhite;
```

The language separates instructions with semicolons, just like in C/C++ or Rust. Instead of having a **`main()`** function, the starting point of execution in *SpartyTalk* is the **`gogreen`** instruction, and the end of the execution is determined by the **`gowhite`** instruction.

SpartyTalk is a *strongly-typed* language, and it does not do *type inference* like Python or Rust (i.e., guessing types based on the assigned value). Instead, it requires explicit specification of type during the declaration/initialization of a variable, just like in C/C++. We currently have two basic data types: *numbers* and *strings*. We use the **`svar`** keyword to declare a string variable and **`nvar`** keyword for declaring a numeric variable. *SpartyTalk* does not like ambiguity, so the variables must be initialized (e.g., assigned a value during the declaration).

SpartyTalk produces output using the **`spartysays`** command. Also, our language currently supports four operations: **`+`**, **`-`**, **`*`**, and **`/`**. If **`+`** is used with numbers, it performs arithmetic addition. If **`+`** is used with strings, it performs concatenation. If **`+`** is used between a string and a number, it converts the number to a string and performs concatenation (like in Python).

In this project, we will continue implementing the parser for the basic *SpartyTalk* BNF grammar, as shown below. Please note that throughout the semester we will be adding more rules to this grammar, so this is not the final *SpartyTalk* grammar.

```
<program> ::= “gogreen” “;” <statements> “gowhite” “;”
<statements> ::= <statement> | <statements> <statement>

<statement> ::= “spartysays” <expression> “;”
              | “nvar” <identifier> “=” <expression> “;”
              | “svar” <identifier> “=” <expression> “;”
              | <identifier> “=” <expression> “;”

<expression> ::= <identifier>
              | <number>
              | <string>
              | “(“ <expression> “)”
              | <expression> “+” <expression>
              | <expression> “-” <expression>
              | <expression> “*” <expression>
              | <expression> “/” <expression>

<identifier> ::= ([a-zA-Z][a-zA-Z0-9]*)
<number> ::= ([+\-]?[0-9]+(\.[0-9]+)?)
<string> ::= ("[^"]*")
```

In this project, you are expected to continue implementing the function **`interpret_spartytalk().`** Different from Project 4, the argument of this function is now the program itself (not the IR). Also, in this project, **`interpret_spartytalk()`** is not expected to return anything. Instead, it will execute the program by tracing all variables, evaluating all expressions, and printing the output of **`spartysays`** commands to the standard output.

For example, if the argument of **`interpret_spartytalk()`** is this program:

```
gogreen;
nvar a = 10;
nvar b = a * 2.2;
spartysays b;
gowhite;
```

Then the result of the execution should be the following string printed to the standard output (followed by a newline, like in Python’s **`print()`**):

```
22.0
```

Implementation

Continue the implementation of **`interpret_spartytalk()`** function in **`solution.py`**. The function takes a single argument — a *SpartyTalk* program in the form of a string. In Project 4, this function returned a list. However, in this project, **`interpret_spartytalk()`** should not return anything. Instead, **`interpret_spartytalk()`** is expected to interpret the program and print the evaluated expressions of **`spartysays`** commands to the standard output (if any). In this project, we temporarily assume that all inputs to **`interpret_spartytalk()`** are valid.

Testing and Grading

The solution will be graded using 40 autograding tests: 20 tests in **`test_open.py`** and 20 additional hidden tests that will be used by the instructors while grading. The hidden tests *will not* introduce any new challenges on top of the ones already tested by the open tests. To run the tests, invoke the **`pytest`** command while in the project directory. Each test is worth 2 points, resulting in 80 total possible points. Please read the open tests to better understand the requirements of the implementation, *but do not modify the tests*. **Submit the solution to D2L.**

Have fun!