VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ADVANCED PROGRAMMING - CO2039

ASSIGNMENT 3

SIMULATE SYMBOL TABLE BY LIST

Author: MEng. Tran Ngoc Bao Duy, BEng. Thi Khac Quan

Ho Chi Minh City, 03/2025

ASSIGNMENT'S SPECIFICATION

Version 1.1

1 Assignment's outcome

Upon completion of this major assignment, students will be able to review and proficiently apply:

- Fundamental concepts of functional programming.
- Utilization of higher-order functions.
- Techniques for employing list data structures within the functional programming paradigm.

2 Introduction

The **Symbol table** is a fundamental data structure that is created, maintained, and utilized by compilers to track the semantics of identifiers, such as storing information related to names, types, scopes, and so on.

Among the various stages a compiler performs to translate source code into executable code, the semantic analysis phase is crucial for verifying the correctness of the program. For instance, it checks whether a variable has been declared before use, or whether the assignment of a value to a variable is type-compatible. The **semantic analysis** phase requires the symbol table to trace the information necessary for such validations.

In this major assignment, students are required to implement a simulation of a symbol table using list-based data structures.

3 Description

3.1 Input

Every testcase is an input, including lines of code, which are used to interact with symbol table. These are specified in section 3.5. Students can find example of testcases in this section.

3.2 Requirements

To complete this assignment, students must:

1. Read through the entire description file.
2. Download the file `initial.zip` and extract it. After extraction, students will receive the files: `main.py`, `Symbol.py`, `SymbolTable.py`, `TestSuite.py`, `TestUtils.py`. Students are not allowed to rename any of these files, as renamed files will not be included in the submission scope.
3. Modify the file **SymbolTable.py** to complete the assignment, while ensuring the following two requirements:
    - At least one function def simulate(list_of_commands) must exist in this file, as it serves as the entry point for the solution. For each testcase, this function will be called with an input parameter which is a list of commands (see `TestUtils.py` for reference).
    - Students may only import exactly three external modules into this file: **from StaticError import** *, **from Symbol import** *, and **from functools import** *. Any other **import** statements in these files are strictly prohibited.
4. Write at least 50 testcases in the file **TestSuite.py** to validate their implementation.

3.3 Information of a symbol in the symbol table

Information of a symbol consists of:

1. Name of the identifier
2. Corresponding type of the identifier

3.4 Semantic errors

During the interation, some semantic errors can be checked and thrown (via **throw** command in C/C++ programming language) if found:

1. Undeclared error **Undeclared**.
2. Redeclared error **Redeclared**.
3. Block not closing error **UnclosedBlock**, combined with level of not closing block (specified in section 3.5.3).

4. Corresponding block not found error **UnknownBlock**.

These errors are all accompanied by the corresponding command in the text string in the input except for the UnclosedBlock and UnknownBlock errors. The program will stop and not continue to interact if any error occurs.

3.5 Interation commands

Each command must be written on a single line and start with a code. A command may have zero, one, or two parameters. If present, the first parameter must follow the code with exactly one space, and the second parameter must follow the first with exactly one space. No extra spaces or delimiters are allowed.

Any command that violates these rules or the specific format required for each command type is considered invalid. In such cases, the simulation will immediately raise an InvalidInstruction error with the faulty command line and terminate.

3.5.1 Insert a symbol into the symbol table - INSERT

- Format: **INSERT <identifier_name> <type>**
  where:
  - <identifier_name> is the name of an identifier, which is a string of characters that begins with a lowercase character followed by characters consisting of lowercase, uppercase, underscore characters _ and numeric characters.
  - <type> is the corresponding type of the identifier. There are two types of type, **number** or **string** to declare numeric and string types respectively.
- Meaning: Add a new identifier to the symbol table. Compared with C/C++, this is similar to declaring a new variable.
- Value to print to the screen: **success** if successfully added to the table, otherwise throw the corresponding error.
- Possible errors: **Redeclared**.

Example 1: For input includes:

```
INSERT a1 number
INSERT b2 string
```

Since there are no duplicate names (re-declared) errors, the program prints out:

```
success
```

```
success
```

Example 2: For input includes:

```
INSERT x number
INSERT y string
INSERT x string
```

Because the identifier x has been added in line 1, but also continues to be added in line 3, it causes a Redeclared error, so the program prints out:

```
Redeclared: INSERT x string
```

3.5.2 Assign value to symbol - ASSIGN

- Format: **ASSIGN <identifier_name> <value>**
  with:
  - <identifier_name> is the name of an identifier and must follow the rules outlined in 3.5.1.
  - <value> is a value assigned to a variable, which can take three forms:
    - Number constant: begins with the letter n and is followed by a series of numbers. For example, 123, 456, 789 are number constants, and 123a, 123.5, 123.8.7 are not. Number constant is considered to be of type number.
    - String constant: begins with an apostrophe ('), followed by a string consisting of numeric characters, alphabetical characters, spaces, and ends with an apostrophe. For example, 'abc', 'a 12 C' are string constants, and 'abc_1', 'abC@u' are not. String constant is considered to be of type string.
    - A different identifier that has been declared.
- Meaning: Check whether it is suitable to assign a simple value to an identifier
- Value to print to the screen: **success** if successfully assigned, otherwise throw the corresponding error.
- Possible errors:
  - **Undeclared** if an undeclared identifier appears in either the <identifier_name> or <value> section.
  - **TypeMismatch** if the type of the assigned value and the identifier are different.

Example 3: For input includes:

```
INSERT x number
INSERT y string
ASSIGN x 15
ASSIGN y 17
ASSIGN x 'abc'
```

The assignment on the third line does not raise an error, but on the fourth line, a type error occurrs because of assigning the number constant to a string-typed identifier. The program runs to this line and terminates.

```
TypeMismatch: ASSIGN y 17
```

3.5.3 Open and close block - BEGIN/ END

- Format: **BEGIN/ END**.
- Meaning: open and close a new block is the same as open and close { } in C/C++. When opening a new block, there are a few rules as follow:
  - It is allowed to re-declare previously declared identifier's name.
  - When searching for an identifier, we must search it in the innermost block. If it is not there, continue searching in the parent block iteratively until the global block is met
  - All blocks have a defined level. When it comes to global block, its level is 0 and will increment with its child blocks (sub-blocks).
- Value to print to the screen: The program will print nothing when it comes to opening and closing blocks.
- Possible Errors: UnclosedBlock can be thrown if we don't close an opened block, or Unknown-Block if we close it but can't find its starting block.

Example 4: For input includes:

```
INSERT x number
INSERT y string
BEGIN
INSERT x number
BEGIN
INSERT y string
END
END
```

The program prints out:

```
success
success
success
success
```

Since when running, there is a level-1 block on line 3, we can re-declare variable x on line 4. A level-2 block starts on line 5, which allows us to re-declare variable y without throwing any errors.
Having said that, if the END command on the last line is removed, the program will print out:

```
UnclosedBlock: 1
```

since we do not have any commands to close the block that has level 1.

3.5.4 Search for a symbol corresponding to an identifier - LOOKUP

- Format: **LOOKUP <identifier_name>**
  where, <identifier_name> is the name of an identifier and must follow the rules outlined in 3.5.1.
- Meaning: Finding whether an identifier is in the symbol table. It is alike finding and using a variable in C/C++ programming language.
- Value to print to the screen: level of the block containing the identifier if found, otherwise throw the corresponding error.
- Possible errors: **Undeclared** if the identifier cannot be found in all scopes of the symbol table.

Example 5: For input includes:

```
INSERT x number
INSERT y string
BEGIN
INSERT x number
LOOKUP x
LOOKUP y
END
```

This program will print out:

```
success
```

```
success
success
1
0
```

Since x is found in the child block. y is not found in the child block but can be found in the parent block.

3.5.5 Print active identifiers in the current scope in forward direction - PRINT

- Format: **PRINT**
- Meaning: Prints all the identifiers that can be found in the current scope in the declared order from the first line to the current line.
- Value to print to the screen: identifier and level of its respective block are printed and separated by a space on the same line with no trailing space.

Example 6: For input includes:

```
INSERT x number
INSERT y string
BEGIN
INSERT x number
INSERT z number
PRINT
END
```

The program will print out:

```
success
success
success
y//0 x//1 z//1
```

PRINT lines in level-1 block and variable x is re-declared there. Hence, only x in this level-1 scope can been seen. The other x in parent scope (level-0 scope) cannot be seen.

3.5.6 Print active identifiers in the current scope in reverse order - RPRINT

- Format: **RPRINT**
- Meaning: Print all the identifiers that can be found in the current scope in order, from child scope to parent scope.
- Value to print to the screen: identifier and level of its respective block are printed and separated by a space on the same line with no trailing space.

Example 7: For input includes:

```
INSERT x number
INSERT y string
BEGIN
INSERT x number
INSERT z number
RPRINT
END
```

The program will print out:

```
success
success
success
z//1 x//1 y//0
```

RPRINT lines in level-1 block and variable x is re-declared there. Hence, only x in this level-1 scope can been seen. The other x in parent scope (level-0 scope) cannot be seen. All are printed in reverse order.

4 Coding Rules

In this assignment, students are required to fully utilize functional programming concepts. Therefore, student submissions must adhere to the following requirements:

1. It is not allowed to use any additional modules other than the ones already provided.
2. The student's program must consist solely of functions defined using the def keyword, without the use of global variables or custom data types (i.e., no class declarations).

3. The use of loops such as `for`, `while`, etc., is prohibited. Instead, students must use higher-order functions and list comprehensions.
4. Within each function body, students may declare variables, but each variable must be assigned exactly once and must not be reassigned later, in order to ensure data immutability.

Student submissions will be checked and graded by the system. If any violations are detected, the submission will be automatically rejected.

5 Submission

Students are required to submit only two files: `SymbolTable.py` and `TestSuite.py`, before the deadline specified in the ”Assignment 3 - Submission” link. A set of simple test cases will be used to verify that the submitted solution can be compiled and executed correctly. Students may submit their work as many times as they wish; however, only the final submission will be used for grading and evaluation.

Due to potential system overload when a large number of students submit simultaneously, it is strongly recommended that students submit their work as early as possible. Late submissions are at the student’s own risk. Once the submission deadline has passed, the system will be closed, and no further submissions will be accepted. Submissions through alternative means will not be considered under any circumstances.

6 Other regulations

- Students must complete this assignment independently and must not allow others to copy their work. Any violation will be treated as academic dishonesty, in accordance with the university’s regulations.
- All decisions made by the instructors responsible for this assignment are final.
- Test cases will not be disclosed after grading. However, students will be informed about the design strategies of the test cases and the distribution of correct responses across them.

7 Changelog

——————END——————