VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATA STRUCTURES AND ALGORITHMS - CO2003

ASSIGNMENT 1

SIMULATE SYMBOL TABLE BY LIST

Author: MEng. Tran Ngoc Bao Duy

Ho Chi Minh city, 08/2021

ASSIGNMENT'S SPECIFICATION

Version 1.1

1 Assignment's outcome

After completing this assignment, students review and make good use of:

- Designing and using recursion
- Object Oriented Programming (OOP)
- List data structures

2 Introduction

Symbol table is a crucial data structure, made and maintained by compilers to trace semantics of identifiers (e.g information about name, type, scope, e.t.c).

Among the stages that the compiler takes to convert source code into executable code, semantic analysis is one of the most critical stage to validate correctness of the source code, such as checking whether a variable has been declared prior to being used, or whether the assignment of a value to a variable is type matched, e.t.c. The semantic analysis stage demands the symbol table in order to trace the information required in those checks.

In this assignment, students are required to implement a simulation of a record table, using list data structures.

3 Description

3.1 Input

Every testcase is an input file, 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 accomplish this assignment, students should:

1. Read carefully this specification
2. Download initial.zip file and extract it. After that, students will receive files: main.h, main.cpp, SymbolTable.h, SymbolTable.cpp, error.h. Students are not allowed to modify the files that are not in the submission list.
3. Modify the files SymbolTable.h, SymbolTable.cpp to accomplish this assignment, but make sure to achieve these two requirements:
   - There is at least one SymbolTable class having instance method public **void run(string testcase)** because this method is the input to the solution. For each testcase, an instance of this class is created and the run method is called with the file name of the text file (containing an interaction with the symbol table) as a parameter.
   - There is only one include command in the SymbolTable.h file, which is **include "main.h"**, and one include command in the SymbolTable.cpp file, which is **include "SymbolTable.h"**. Also, no other **include**s are allowed in these files.
4. Students are required to design and use their data stuctures based on the acknowledged list data structures.
5. Students must release all dynamically allocated memory when the program ends.

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 file except for the UnclosedBlock and UnknownBlock errors. The program will stop and not continue to interact if any error occurs.

3.5 Interation commands

A command is written on one line and always begins with a code. In addition, a command can have no, one or two parameters. The first parameter in the command, if any, will be separated from the code by exactly one space. The second parameter of the code, if any, is separated from the first by a space. Additionally, there are no other trailing and delimiting characters.

Contrary to the above regulations, the others are all wrong commands, the simulation will immediately throw the InvalidInstruction error with the wrong 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 file 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 file 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:

```
success
success
Redeclared:  INSERT x string
```

3.5.2 Assign value to symbol - ASSIGN

- Format: **ASSIGN <identifier_name> <value>**
  trong ó:
  - <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, n123, n456, n789 are number constants, and n123a, n123.5, n123.8.7 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 both the <identifier_name> and <value> section.
  - **TypeMismatch** if the type of the assigned value and the identifier are different.

Example 3: For input file 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.

```
success
success
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 UnknownBlock if we close it but can't find its starting block.

Example 4: For input file 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:

```
success
success
success
success
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 file 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 file 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 file 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 Submission

Students are required to submit only 2 files: SymbolTable.h and SymbolTable.cpp prior to the given deadline in the link "Assignment 1 - Submission". There are some simple testcases used to check students' work to ensure that their results are compilable and runnable. Students can submit as many times as they want, but only the final submission will be graded. Because the system cannot bear the load when too many students submit their work at once, students should submit their work as soon as possible. Students will take all responsibility for their risk if they submit their work near the deadline. When the submission deadline is over, the system will close so students will not be able to submit their work any more. Other methods for submission will not be accepted.

5 Other regulations

- Students must complete this assignment on their own and must prevent others from stealing their results. Otherwise, the student treat as cheating according to the regulations of the school for cheating.
- Any decision made by the teachers in charge of this assignment is the final decision.
- Students will not be given any testcases after grading, but will only be provided with information of the testcase design strategie and the distribution of the correct number of students according to each test case.

6 Changelog

- Update the rule for number literal.

——————————END——————————