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 2

SIMULATE SYMBOL TABLE BY SPLAY TREE

Author: MEng. Tran Ngoc Bao Duy

Ho Chi Minh city, 08/2021

ASSIGNMENT'S SPECIFICATION

Version 1.0

1 Assignment's outcome

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

- Designing and using recursion
- Object Oriented Programming (OOP)
- Searching algorithms and hash 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).

In the previous assignment, students were required to implement simulations of symbol table via list data structure. However, indexing speed for checking purpose in this kind of data structure is not efficient. Whenever the source program has plenty of variables, saved in various scopes, it will become ineffective. On the other hand, in practice, programmers often tend to use newly declared or recently used identifiers for the next command lines, making the process of indexing those identifiers becomes more popular.

In this assignment, students are required to implement a simulation of a symbol table, using splay tree data structures to facilitate those disadvantages mentioned above.

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 structures based on the acknowledged splay tree data structures.
5. Students must release all dynamically allocated memories 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. Level of the block that the identifier belongs to
3. Corresponding type of the identifier

On the binary search tree, searching for key of nodes takes place frequently. To compare keys of symbol, we sequentially do as follow:

- Compare levels of blocks that identifiers belong to, return the key of the node which has the greater value and vice versa.
- In case the levels of blocks are equal, compare the names of identifiers, return the result by the following steps:

- Equal if names of identifiers are the same.
- Greater than if the first non-matching character of the first string is greater than the corresponding character of the second string.
- Less than in the other opposite cases.

Students should use **compare** method of **string** class library to implement when comparing strings.

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. Invalid declaration error **InvalidDeclaration**.
4. Type mis-matching error **TypeMismatch**.
5. Block not closing error **UnclosedBlock**, combined with level of not closing block (specified in section 3.5.3).
6. 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 to the symbol table - INSERT

- Format: **INSERT** <**identifier_name**> <**type**> <**static**>

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 three types of type, which are **number**, **string** or **function type** to declare numberic and string types. The function type consists of 2 parts: list of arguments' type which can be empty, and return type. These parts are separated by an arrow ->. List of arguments' type starts with an open parenthesis, followed by list arguments' type, which can number or string type, separated by a comma, ends with a close parenthesis. Return type can be number or string type. Function type can only be declared in global block (level is equal to 0). For example: (number,number)->string means this is a function which accepts 2 number type parameters and return a string type.
- <static> can have either **true** or **false**. If <static> is equal to **true**, the identifier belongs to global scope, which means the block level of this identifier is always equal to 0.

- Meaning: add a new identifier to the symbol table. Compared with C/C++, this is similar to declare a new variable.
- Value to print to the screen: <**num_comp**> <**num_splay**>, where <num_comp> is the number of comparison with every nodes in the tree, <num_splay> is the number of splay operations has been made if success, otherwise throw the corresponding error.
- Possible errors:
  - **Redeclared** if declare a identifier which has been declared before.
  - **InvalidDeclaration** if declare a function in a block whose level is not equal to 0

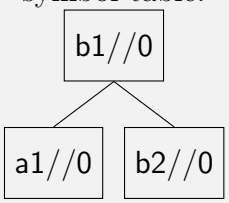
Example 1: For input file includes:

```
INSERT a1 number false
INSERT b2 string false
INSERT b1 (number,number)->string false
```

Because there are no duplicate names (redeclaration), the program prints out:

```
0 0
1 1
2 1
```

The splay tree representing the symbol table:



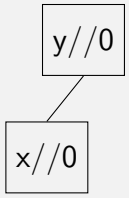
Example 2: For input file includes

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

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

```
0 0
1 1
Redeclared: INSERT x string false
```

The splay tree representing the symbol table (before the error happened):



3.5.2 Assign value to symbol - ASSIGN

- Format: **ASSIGN <identifier_name> <value>**
  where:
  - <identifier_name> is the name of an idenfier 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: a series of numbers. For example: 123, 456, 789 are number constants, while 123a, 123.5, 123.8.7 are not number constants. Number constant is considered to be of type number.
    - String constant: begin with an apostrophe (’), followed by a string consinting of numberic characters, alphabetical characters, spaces, and end with an apostro-

phe. For example: 'abc', 'a 12 C' are string constants, while 'abc_1', 'abC@u' are not. String constant is considered to be of type string.

  - Another identifier that has been declared.
  - A function call begins with identifier of function type, followed by an open parenthesis, a list of parameters which can be empty (parameters only accept number constant, string constant, declared identifier. They are separated by a comma), and a close parenthesis. Example: foo(1,2) or baz(a,1) is a valid function call.

- Meaning: Check whether it is valid to assign a simple value to an identifier. The validation process starts with <value> first and <identifier_name> later.
- Value to print to the screen: **<num_comp> <num_splay>**, where <num_comp> is the number of comparison with every nodes on the tree, <num_splay> is the number of splay operations have been made if success, otherwise throw the corresponding error.
- Possible errors:
  - **Undeclared** if an undeclared identifier appears in either the <identifier_name> or the <value> section.
  - **TypeMismatch** if:
    - Type of assigned value and the identifier are different.
    - The identifier of the function call is not a function type one.
    - The type of each argument must be the same as the corresponding parameter.

Example 3: For input file includes:

```
INSERT x number false
INSERT sum (number,number)->number false
ASSIGN x sum(1,1)
ASSIGN z sum(12,'abc')
```

The assignment at line 3 throws a TypeMismatch error, which causes the program print out:

```
0 0
1 1
3 1
TypeMismatch: ASSIGN z sum(12,'abc')
```

Although the identifier z has not been declared yet, the function call sum(12,'abc') has the second argument's type is string constant, which is not the same as the type of the second parameter (number).

3.5.3 Open and close block - BEGIN/ END

- Format: **BEGIN/ END**.
- Meaning: open and close a new block, which 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-declared 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 reached.
  - All blocks have a defined level. When it comes to globak block, its level is 0 and will increment with its child blocks (sub-blocks).
  - When getting out of a block, all identifiers, which are not declared as static, must be removed from the symbol table in the order of their appearance.
- Value to print to the screen: the program will print nothing when it comes to opening and closing blocks.
- Possile errors: **UnclosedBlock** can be thrown if we don't close an opened block or **UnknownBlock** can be thrown if we close a block but can't find its starting block.

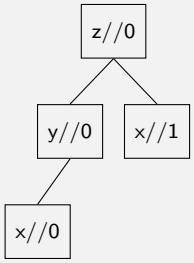
Example 4: For input file includes:

```
INSERT x number false
INSERT y string false
BEGIN
INSERT x number false
BEGIN
INSERT z string true
END
END
```

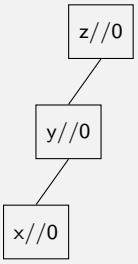
The program will print out:

```
0 0
1 1
1 1
2 1
```

After having added z at line 6, the splay tree representing the symbol table is:



After having got out of the block having level 2 at line 7, since z is an identifier which is declared as static, it is not removed from the tree. When getting out of block having level 1 at line 8, because x//1 is not a static identifier, it will be removed from the tree:



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. In comparison to C/C++, it is similar to find and use a variable.
- 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.

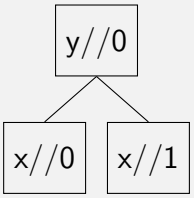
Example 5: For the input file includes:

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

This program will print out:

```
0 0
1 1
1 1
0
```

After having looked up y at line 5, the splay representing the symbol table:



3.5.5 Print the pre-oder of the splay tree representing the symbol table - PRINT

- Format: **PRINT**
- Meaning: Print the pre-order of the splay tree representing the symbol table.
- Value to print to the screen: identifiers and level of theirs respective block are printed and separated by a space on the same line with no trailing space.

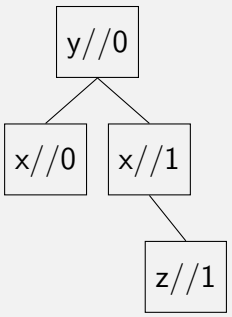
Example 6: For the input file includes:

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

The program will print out:

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

After having looked up y at line 6, the splay tree representing the symbol table is:



4 Submission

Students are required to submit only 2 files: SymbolTable.h and SymbolTable.cpp prior to the given deadline in the link "Assignment 2 - Submission". There are some simple testcases used to check students' work to ensure that their codes 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 works as soon as possible. Students will take all responsibility for their risk if they submit their works 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, student will be considered as cheating according to the

regulations of the school for cheating.

- Any decision from the teachers who are responsible for this assignment is the final decision.
- Test cases won't be public after grading. However, the information of testcase degisn strategy and the distribution of the number of correct submissions for each test case will be given.

——————————END——————————