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 3

SIMULATE SYMBOL TABLE BY HASH TABLE

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 the simulations of symbol table via list and tree. To optimize searching progress, hash table is among one of the most suitable data structures. Moreover, in this assignment, students are also introduced how to build a symbol table for languages using type inference. Type inference programming languages, in general, is a programming language, in which, when declaring identifiers, it does not require explicitly declare the corresponding types. However, the way that type of an identifier is set is related to statements or expressions, in which the identifier is used.

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

3 Description

3.1 Input

Mỗi testcase là một tập tin đầu vào bao gồm các dòng lệnh thiết lập tham số cho bảng băm (được mô tả ở mục **??**) và các lệnh tương tác với bảng ghi đối tượng (được mô tả ở mục 3.6). Sinh viên có thể thấy được ví dụ về các testcase thông qua mục này.

Every testcase is an input file, including lines of code used to set parameter for hash table

(specified in section **??**); and those of code used to interact with symbol table (specified in section 3.6). 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 hash table 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. Level of the block that the identifier belongs to

Students must re-design storing information to fit to the specification.

When interacting with hash table, we need to encode an object's inforamtion into a key. The encrypted key is a integer of the form $\overline{ca_1a_2a_3a_4...a_n}$, where:

- c is the level of the block that the identifier belongs to.

- a_1, a_2, ..., a_n respectively are the decimal value in the ASCII table of each character in the identifier, after subtracting 48.

Example 1: We have an identifier-"xB" at level 1. Hence, the key of this symbol is:

- Level of block is 1. Therefore, $c = 1$.
- $x = 120$ is encrypted into $120 - 48 = 72$
  $B = 66$ is encrypted into $66 - 48 = 18$

Therefore, xB//1 will be encrypted into 17218.

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**, goes with the undeclared identifier.
2. Redeclared error **Redeclared**, goes with the redeclared identifier.
3. Invalid declaration error **InvalidDeclaration**, goes with the invalid declaration identifier.
4. Type mis-matching error **TypeMismatch**, goes with the command causing error.
5. Table overflow error **Overflow**, goes with the command causing error.
6. Block not closing error **UnclosedBlock**, goes with level of not closing block (specified in section 3.6.4).
7. Corresponding block not found error **UnknownBlock**.

The program will stop interacting if any error happens.

3.5 Hash table's parameter settings

Hash table used in this assignment is the one using open addressing to solve collisions. Hence, the parameter settings for it, including setting the size, the hash function and the probing function are crucial to the assignment. This setting is always located on the very first line in the testcases, and only appears once. A setting is written on one line, and always starts with a code corresponding to a probing method. Moreover, there are multiple parameters in one command. The first one is separated from the code by one space, and the others are also separated from

each other by one space. All of the parameters must be in the format of a maximum-6-digit natural number. In addition, there are no other delimiters and trailing characters.

Contrary to the above rules and the format outlined below are all wrong settings. The simulation immediately throws an InvalidInstruction error, together with the wrong setting line and terminates the program.

3.5.1 Linear probing method - LINEAR

- Format: LINEAR <m> <c>
  where
  - <m> is the size of the hash table.
  - <c> is a constant used in probing progress.
- Meaning: Then, the hash function and the corresponding probing function are

$$h(k) = k \mod m$$
$$hp(k, i) = (h(k) + ci) \mod m$$

  where k is the key and i is the number of probings.

3.5.2 Quadratic probing method - QUADRATIC

- Format: QUADRATIC <m> <c_1> <c_2>
  where:
  - <m> is the size of the hash table.
  - <c_1>, <c_2> are constants used in probing progress.
- Meaning: Then, the hash function and the corresponding probing function are

$$h(k) = k \mod m$$
$$hp(k, i) = (h(k) + c_1 i + c_2 i^2) \mod m$$

  where k is the key and i is the number of probings.

3.5.3 Double hashing - DOUBLE

- Format: DOUBLE <m> <c>
  trong đó:

  - $<m>$ is the size of the hash table.
  - $<c>$ is a constant used in probing progress.

- Meaning: Then, the hash function and the corresponding probing function are

$$h_1(k) = k \mod m$$
$$h_2(k) = 1 + (k \mod (m-2))$$
$$hp(k,i) = (h_1(k) + cih_2(k)) \mod m$$

  where k is the key and i is the number of probings.

3.6 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.

3.6.1 Insert a symbol into the symbol table - INSERT

- Format: **INSERT <identifier_name> <num_of_parameters>?**
  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 numberic characters.
  - <num_of_parameters>? is the number of parameters which has to pass to a function if identifier is declared as a function. This section is only presented when the identifier is a function, otherwise this section is disappeared. The function declaration only occurs in global scope (level 0).
- Meaning: Add a new identifier to the symbol table. In comparision to C/C++, it is similar to declare a new variable. However, after having declared a variable, its type has been not specified yet, which is likely to the concept of **auto** and **decltype**.
- Value to print to the screen: the number of slots need to go through **before** reaching the empty slot which can place the value.
- Possible errors:

- **Redeclared** if re-declare a variable.
- **InvalidDeclaration** if declare a function in block whose level is not equal to 0.
- **Overflow** if the declaration cannot find a suitable empty slot in hash table.

Example 2: For input file includes:

```
LINEAR: 19 1
INSERT a1
INSERT b2
INSERT rj 2
```

Due to no duplicate names (re-declaration), the program prints out:

```
0
0
1
```

The symbol table is:

Slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Value									b2//0	rj//0							a1//0		

Example 3: For input file includes:

```
LINEAR: 19 1
INSERT x
INSERT y
INSERT x
```

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:

```
0
0
Redeclared: x
```

3.6.2 Assign value to symbol - ASSIGN

- Format: **ASSIGN <identifier_name> <value>**
  where:
  - <identifier_name> is the name of an identifier and must be follow the rules outlined in 3.6.1.
  - <value> is the 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 constant is considered to be of type number.
    - String constant: begin with an apostrophe ('), followed by a string consisting of numeric characters, alphabetical characters, spaces, and end with an apostrophe. 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.
    - A different 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:
  - If both sides of the assign statement have types, we must validate the suitable when assigning a simple value to an identifier. The validation process starts at <value> first and <identifier_name> later.
  - If at least one of two sides of the assign statement does not have a type, we infer its type by the following rules:
    - The type of the unknown type side is inferred to the type of the known type one.
    - The type of the arguments in the function type identifier must be inferred to the type of the parameters respectively if these parameters have type and vice versa.
    - The return type in the function type identifier must be inferred to the type of the assigned identifier and vice versa.

- Value to print to the screen: The number of slots the program need to reach before have found the slot containing the information about the symbol of each identifier which appear in the command if validate successful or infer.
- Possible error:
  - **Undeclared** if an undeclared identifier appears in either <identifier_name> or <value> section.
  - **TypeMismatch** if:
    - the type of the assigned value and the identifier are different.

    - The function call has the identifier whose type is not function type.
    - The type of parameters is not the same as the type of arguments respectively.
    - The return type of function is not the same as the type of the assigned identifier.
  - **TypeCannotBeInferred** if:
    - Both sides of the assign statement (if the type of identifier is function type, the side type is the return type of the function) are not inferred a specific type.
    - The arguments and the parameters are not inferred particular type respectively.

Example 4: For input file includes:

```
LINEAR: 19 1
INSERT x
INSERT sum 2
ASSIGN x 1
ASSIGN x sum(5,x)
INSERT z
INSERT foo 1
ASSIGN z foo('abc')
```

The identifier sum is declared as a two arguments function, foo is declared as a one arguments function. The inference process goes like this:

- In the line 3, type of x is **number**.
- In the line 4:
  - The first argument type of the function has the same type as the type of number constant 5, which means **number** type.
  - The second argument type of function has the same type as the type of x, which means **number**.
  - The return type of function sum has the same type as the type of x, which means **number**.
- In the line 7:
  - The first parameter of the function has the same type as the type of string constant 'abc', which means **string** type.
  - The return type of foo and the type of z cannot be inferred, which causes **TypeCannotBeInferred** error.

3.6.3 Function call - CALL

- Formal: **CALL** <call_exp>
  where, <call_exp> is the call expression as describing in 3.6.2.
- Meaning: function call is not assigned to any identifier. The inference process for this function call is similar to any expression. However, return type must be (must be inferred to) **void**.
- Value to print to the screen: The total of slots must be reached before having found the slot containing information about the symbol of each identifier, which appears in the command if success check or infer.
- Possible error: **Undeclared**, **TypeMismatch**, **TypeCannotBeInferred**.

3.6.4 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-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 reached.
  - All blocks have a defined level. When it comes to global block, its level is equal to 0 and will increment with its child blocks (sub-blocks).
  - When getting out of a block, all declared identifiers in that block must be removed from the symbol table.
- Value to print to the screen: The program will print nothing when it comes to opening and closing blocks.
- Possible erros: UnclosedBlock can be thrown if we don't close an opening block or UnknownBlock if we close it but can't find its starting block.

Example 5: For input file includes:

```
LINEAR 19 1
INSERT x1
INSERT y1
BEGIN
INSERT x1
```

```
BEGIN
INSERT u7
END
END
```

The sysmbol table after adding u7 is:

Slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Value	u7//0									y1//0		x1//1							x1//0

3.6.5 Search for a symbol corresponding to an identifier - LOOKUP

- Format: **LOOKUP <identifier_name>**
  where, <identifier_name>is a name of an identifier and must follow the rules outlined in 3.6.1.
- Meaning: find whether an identifier is in the symbol table or not. Compared to C/C++, it is similar to find and use a variable.
- Value to print to the screen: The index of the current slot containing the identifier.
- Possible errors: **Undeclared** if the identifier cannot be found in all scopes of the symbol table.

Example 6: For input file includes:

```
LINEAR 19 1
INSERT x1
INSERT y1
BEGIN
INSERT x1
BEGIN
INSERT u7
LOOKUP x1
END
END
```

x1//1 will be found when running the command LOOKUP x1. As a result, this command will print out 11.

3.6.6 Print the symbol table - PRINT

- Format: **PRINT**
- Meaning: Print the index and the corresponding value of each position containing data in the symbol table.
- Value to print to the screen: the slot and the identifier with the corresponding block level are printed and separated by a space on the same line. All slots are separated by a semicolon ; with no trailing space.

Example 7: For input file includes:

```
LINEAR 19 1
INSERT x1
INSERT y1
BEGIN
INSERT x1
BEGIN
INSERT u7
PRINT
END
END
```

The PRINT line will print out:

```
0 u7//0;9 y1//0;11 x1//1;18 x1//0
```

4 Submission

Students are required to submit only 2 files: SymbolTable.h and SymbolTable.cpp prior to the given deadline in the link "Assignment 3 - 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——————————