COMP 2140 Assignment 4: Postfix Expressions with Variables

Helen Cameron

Due: Wednesday 23 November 2016 at 10:00 p.m.

Important Announcement

Until you agree to the honesty declaration (see the instructions at the end of Assignment 1 under "Honesty Declaration"), you cannot hand in any assignments — that is, until you agree to the honesty declaration, you will not be able to see the assignment dropboxes in UM Learn.

How to Get Help

Your course instructor is helpful: Email me at `Helen.Cameron@cs.umanitoba.ca` or come to my office hours at 2:30–3:30 p.m. Mon/Wed/Fri or 4:00–4:30 p.m. Tues/Thurs in E2 477 EITC (or by appointment).

For email, please remember to put "[comp2140]" in the subject and use a meaningful subject, and to send from your UofM email account.

Course discussion groups on UM Learn: A discussion group for this assignment is available in the COMP 2140 course site on UM Learn (click on "Discussions" under "Communications"). Post questions and comments related to Assignment 4 there, and I will respond. Please do not post solutions, not even snippets of solutions there, or anywhere else. I strongly suggest that you read the assignment discussion group for helpful information.

Computer science help centre: The staff in the help centre can help you (but not give you assignment solutions!). See their website at `http://www.cs.umanitoba.ca/undergraduate/computer-science-help-centre.php` for location and hours. You can also email them at `helpctr@cs.umanitoba.ca`.

Programming Standards

When writing code for this course, follow the programming standards, available on this course's website on `UMLearn`. Failure to do so will result in the loss of marks.

Assignment Question

Remember: You will need to read this assignment many times to understand all the details of the program you need to write.

For this assignment, you will be adding functionality to the solution to Assignment 3. **You must start with my solution to Assignment 3.** My solution to Assignment 3 will become available on Friday 11 November 2016 after 11:00 p.m. — that is, after the dropbox for Assignment 3 closes. I will put my solution to Assignment 3 in the content browser on UM Learn where you got the Assignment 3 question.

Assignment 3 had you evaluating postfix expressions that used constant integer operands. Assignment 4 requires you to add variables:

- Each postfix expression in the input becomes an assignment statement that assigns the value of a postfix expression to a variable.
- A postfix expression can now include variables as operands, as well as constant integer operands.

For example, an input line could contain the following:

```
beet = 10 30 + apple apple + /
```

You will store in a `Table` the variables that have been assigned values (further details on the `Table` implementation below). Each variable will be stored as a pair: the variable's identifier (a `String`) and its value (an `int`); — thus, you must also have a `Variable` class (further details on that class and allowable variable identifiers are below).

A variable is "declared" the first time it has a value assigned to it in a valid assignment statement (an assignment statement with no errors in it). So if the variable `Table` doesn't already contain the variable being assigned to, you should create a new `Variable` to store the variable's identifier and value, and insert the `Variable` in the variable `Table`. Otherwise, if the variable being assigned to is found in the variable `Table`, then simply change its value in its `Variable` instance to the new value.

When a valid variable identifier is used in a postfix expression on the right side of an assignment statement, you must retrieve the variable's value from the variable `Table` to use as the operand. (If it is not in the `Table`, then the postfix expression contains an undeclared variable and the whole expression is invalid, and so is the assignment statement that contains it.)

Changing the Assignment 3 solution: You will need to add the `Variable` class and the `Table` class, and you will have to modify the `Expression` class and the application class to handle the changes described above.

You will have to change the name of the application class to comply with the hand-in instructions (below).

You must change as little as possible in my solution—just make the changes needed to add the new functionality required and make no other changes. You are NOT allowed to make changes because you prefer to do something differently than I did. For example, you are not permitted to change the `Node` class or the `Stack` class at all. (This requirement that you change as little as possible is exactly how programmers are expected to work on large ongoing projects. Every change to a (large) working program could introduce a bug — so changes are kept to the minimum to reduce the chances of introducing bugs. Furthermore, code changes require other programmers working on the program to have to adjust their understanding of how the program works—we want to interfere with their understanding as little as possible.)

The Variable Class

The `Variable` class should be very simple. Each `Variable` instance should contain the variable's identifier (a `String`) and its value (an `int`).

The class should contain at least a constructor, an `equals` method (that compares the two `Variables`' identifiers — or maybe compares the calling `Variable`'s identifier to a `String`), a `toString` method, and a `static` method to tell you if a `String` is a valid variable identifier or not.

Note: to call a `static` method of a class from outside the class, you must use the class name. For example, the `parseInt()` method in the `Integer` class is a `static` method, so if you want to call it from one of your classes, you must say `Integer.parseInt(someString)`.

Valid variable identifier: A variable identifier must begin with a lowercase letter (`a-z`) and be followed by a mix of zero or more lowercase letters (`a-z`), uppercase letters (`A-Z`), or digits (`0-9`). Some valid identifiers: `z` and `bigFoot` and `fahrenheit451`. Some invalid identifiers: `23skidoo` and `raining-cats-and-dogs` and `_couldn't_you_think_of_something_shorter?`.

The Table Class

You will be storing `Variables` in the table — the identifier of a variable is the variable's key — identifiers are unique. (Two variables with the same identifier are, in fact, the *same* variable. Do not store duplicates!)

The `Table` must be implemented as a hash table — use size 37 for the array. The hash table must use separate chaining to resolve collisions (as discussed in class) — the array must be an array of `Nodes`. To hash a `String` key to an array index, you must use the polynomial hash code and implement it with Horner's method (as discussed in class). You must also use small prime `a` = 13 in the polynomial hash code.

The small prime number `a` should be an instance variable that is set by the constructor — the table size and the value of `a` should be parameters to the constructor.

The `Node` class is already written for you — you must use the existing `Node` class. Each `Node` stores an `Object` item, so that `Node`s can be used both by the `Stack` (to store `Integer`s) and by the `Table` (to store `Variable`s). Therefore, you need to cast to type `Variable` an item retrieved from a `Node`. (Why didn't I use generics for the `Node`s? Because arrays and generic types do not mix well.)

The `Table` class must contain at least the following methods:

- A constructor (to construct an empty table). Its parameters should be the size of the array and the value of the small prime `a` used in the polynomial hash code.
- The `private` hash method to hash an identifier to an array index.
- A search method to tell you if the table contains a variable with a given identifier (return the `Variable` if it does, and `null` if it doesn't).
- An insert method to insert a new `Variable` into the table.
- A method to print out all the `Variable`s stored in the table. The output does not need to be sorted in any way.

Input

The input file contains on its first line an integer (and nothing else). That integer tells you how many assignment statements the file contains. The remaining lines of the file contain assignment statements, one per line. You should use a simple `for`-loop to read in the assignment statements.

An assignment statement consists of a variable identifier followed by an equals sign followed by a postfix expression. All these pieces, including the tokens in the postfix expression, are separated by blanks. Of course, the postfix expression should be stored in an `Expression` (NOT including the variable being assigned to and the equals sign).

I will provide a small input file (`tinyInput4.txt`) and a larger input file (`largerInput4.txt`). Your program should work correctly on both of them.

Note that the provided application class prompts for the input file name already.

Output

You should, of course, print out the usual title and trailer messages.

For each assignment statement, you should print out the variable to be assigned to, the postfix expression, and, finally, the value of the expression (or error messages if the something erroneous is detected).

After all the assignment statements have been processed, you should print out all variables and their values.

Sample Output

```
Comp 2140  Assignment 4   Fall 2016
Evaluating postfix expressions with variables.
Model Solution

Enter the postfix expressions file name (.txt files only):
tinyInput4.txt

*********************************************

The assignment statements and the values of their right sides
```

```
============================================================

The variable to be assigned to: apple
The postfix expression on the right side: -3 4 +
The value of the expression: 1

The variable to be assigned to: beet
The postfix expression on the right side: 10 30 + apple apple + /
The value of the expression: 20

The variable to be assigned to: apple
The postfix expression on the right side: 16 6 - 8 3 - /
The value of the expression: 2

The variable to be assigned to: cherry
The postfix expression on the right side: 16 beet 4 - 8 / - apple +
The value of the expression: 16

The variable to be assigned to: dandelion4
The postfix expression on the right side: 3 e +
ERROR: undeclared variable e
Expression erroneous, value cannot be computed; invalid assignment.

Invalid variable ID (23skidoo) on the left side of the assignment statement; invalid assignment.
Input line: 23skidoo = 13

No "=" in assignment statement; invalid assignment.
Input line: constantinople 1 1+

Variables and Their Values
--------------------------
Variable beet = 20
Variable cherry = 16
Variable apple = 2

Program ends.
```

Hand-in Instructions

Go to COMP2140 in UM Learn, then click "Dropbox" under "Assessments" at the top. You will find a dropbox folder called "Assignment 4". (If you cannot see the assignment dropbox, follow the directions under "Honesty Declaration" on Assignment 1.) Click the link and follow the instructions. Please note the following:

- Submit ONE .java file only. The .java file must contain all the source code. The .java file must be named
  `A4<your last name><your student id>.java` (e.g., `A4Cameron1234567.java`).
- Please do not submit anything else.
- You can submit as many times as you like, but only the most recent submission is kept.
- We only accept homework submissions via UM Learn. Please DO NOT try to email your homework to the instructor or TA or markers — it will not be accepted.

- We reserve the right to refuse to grade the homework or to deduct marks if you do not follow instructions.
- **Assignments become late immediately after the posted due date and time.** Late assignments will be accepted up to 49 hours after that time, at a penalty of 2% per hour or portion thereof. After 49 hours, an assignment is worth 0 marks and will no longer be accepted.