COMP 62: 460 Comparative Programming Languages

Winter 2024

Instructor: Rashed I. Nekvi

Assignment 2 – Points 20

Due Date**: Sunday, March 03, 2024, 11:59pm**

1. Introduction

This assignment is meant to test your learnings and practical skill development on the implementation of the Interpreter program. You are presented with a right-recursive grammar closely akin to the one given in the Assignment 1. You first have to slightly modify the recursive descent parser in either *C* or *Golang programming (preferred)* language that you developed in Assignment 1, and then, extend your program to interpret the program (evaluate the expression described by the grammar).

2. Grammar

Consider the following grammar:

```
<Exp>: = <Term> | <Term> @ <Exp>
<Term>: = <Number> | <Number> ^ <Term>
<Number>: = <Integer>|! <Integer>|
<Integer>|: = 0 | 1| 2| 3| …| 2^31 -1
```

Here, the symbols: <Exp>, <Term>, <Number> and <Integer> are *non-terminals*. <Exp> is the *start* symbol. The symbols: @, ^, !, and any integer in the range of (0, 1, 2, …, (2^{31} -1)) are the *terminals*. The "|' used in the grammar is neither terminal or non-terminal, simply interpret it as "or" in English.

Notice that this grammar is slightly modified from the grammar provided in the Assignment 1. The difference is in its usage of the *operator* symbols: @, ^ and ! instead of using, respectively, +, *, and -. As you can verify from the sample output outlined in the Section 3 that the modified symbols (@, ^ and !) hold the same meanings as the standard operator symbols (+, *, and -) do which are, respectively, *addition*, *multiplication*, and *negation*.

3. Questions **[5+15= 20]**

Implement an Interpreter program in either *C* or *Golang language* that accepts strings (i.e., expression) from standard input (one per line) until EOF and evaluates the expression. Specifically, answer the following questions:

Question 1: Modify the Scanner and Recursive-descent Parser of your Assignment 1 to reflect the modified grammar given in Section 2. Assume that the whitespaces are ignored.

Question 2: Extend your program from Question 1 to include an Interpreter that evaluates (i.e., interprets) expression in the way shown below in the sample outputs.

Sample Outputs:

```
> 2 @ 5^3
17
> 2  @ 5 ^3
17
> 2 @ 5^!3
-13
> 2 @ 5
7
> 8^2
16
> 2 @ 5^b
17
"2 @ 5^b" contains invalid lexemes and thus is not an expression.
> 2 @ ^5^3
"2 @^5^3" is not an expression
```

Important Instructions:

1. The ">" symbol in the sample output is simply the prompt for input and will be the empty string in your system
2. You can assume that the whitespaces are ignored, and no input string will contain more than 25 lexical units.
3. The "invalid lexemes" message takes priority over the "not an expression" message; that is, the "not an expression" message can be issued only if the input string consists entirely of valid lexemes.
4. Do not build a parse tree to solve this problem. Factor your program into a recursive-descent parser (i.e., like Assignment 1) and an Interpreter as implemented by the *Simple* interpreter (see in Moodle and lecture slides # Ch 4).
5. Normal precedence rules hold such as: the operator "!" (*negation*) has the highest, the "^" (*multiplication*) has the second highest, and the "@" (*addition*) has the lowest.
6. Assume left-to-right associativity.

4. Resources

a) Recursive descend parsing:
   - Chapter 3 and Chapter 4 of the textbook titled "*Programming Languages – Concepts and Implementation*" by S. Perugini.
   - Also review the "Simple Interpreter" discussed in Section 4.2 and the simple program file uploaded in the Moodle course page and in lecture slides for Ch 4.

b) Golang
   - Installation guide: https://go.dev/doc/tutorial/getting-started
   - Web tutorial: www.gobyexample.com

5. Deadline and Submission

a) Deadline: Sunday, March 03, 2024, 11:59pm

b) Submission instructions:
   - Submitted program must run without any error.

- **Name** your program as *yourFirstName_assignment2.go.txt or yourFirstName_assignment2.c.txt* (e.g., rashed_assignment1.go.txt).
- Then **zip** your program file for submission.
- **Login** the Moodle course website for this course (62:460). Find the Assignments menu (at the bottom of page), then find and click Assignment 2.
- Under Assignment 2, **upload** your zipped file. Finally, click the **Submit** button by the due date to finalize your submission.

6. Criteria for Evaluation

- The program runs without error.
- Timely submission
- Correctness and completeness of the program.
- Clarity of the code maintained by proper indentation, comments, and spacing.
- Instructor discretion
- A software for **plagiarism check** will be applied on each submitted work. Please get yourself familiar with BU's plagiarism policy.