CSCI 3415 – Principles of Programming Languages
Dr. Williams
Program 3 – Go
Due Friday, November 18, 2016

From the Go Programming Language web site:

> Go is expressive, concise, clean, and efficient. Its concurrency mechanisms make it easy to write programs that get the most out of multicore and networked machines, while its novel type system enables flexible and modular program construction. Go compiles quickly to machine code yet has the convenience of garbage collection and the power of run-time reflection. It's a fast, statically typed, compiled language that feels like a dynamically typed, interpreted language.

In this program you will implement a simple calculator in Go that uses reflection to support both.

Part 1 – Download Go

From the home page of the Go Programming Language web site, select the Download Go link on the right side of the page. This takes you to a web page with installation instructions for the various platforms – Windows, Mac OS X, or Linux. Then select the Download Go link on that page and select the appropriate download for your platform. Follow the install instructions for your platform.

Make sure you follow the instructions in the "Set up your work environment" section, which directs you to the How to Write Go Code page. At this point you are ready to develop Go applications.

Part II – Calculator

For this program you will be implementing a simple calculator. By simple I mean it only need to support the binary operators +, -, *, and / with their standard definitions (i.e., addition, subtraction, multiplication, and division) and parenthesis. Multiplication and division have higher precedence than addition and subtraction, all of them have left to right associativity, and parentheses can be used to override the precedence.

The program will accept a string containing the expression to be evaluated and print the result of evaluating the expression. Some simple example are:

```
>2*3
6
>2*3.5-2
5.0
>2*(3.5-2)
3.0
```

You should evaluate the expression in a single left to right scan of the input string, using an operator and an operand stack to hold intermediate results during the evaluation. You may handle parenthesis either by a recursive call or by using the stack.

You can decide how to handle whitespace in an expression. For example, you could completely ignore them, ignore them in limited cases (like around operators and operands), or always return an error. [My preference would be to allow spaces around operators and operands, but not in an operand. So, 20 + 30 is valid, but 2 0 + 3 0 is not.] Specify how you process them.

Consider any characters other than space, 0-9, ., +, -, *, and / as an error.

Numbers can be integers (e.g., 2, which is 2.0) or reals (e.g., 2.0). You don't have to worry about scientific notation for inputs. You don't have to worry about negative numbers for inputs – they are most easily handled using unary -, which we don't have.

You might consider developing your code in steps – like we did in class.

- Start with just integers and +, -, *, and / - that is, no parentheses, just integer parsing, and limited error checking. This will get the basic parsing and evaluation algorithm in place.
- Add parenthesis handling. This can either use recursion to evaluate a parenthesized subexpression, or push the open parenthesis on the operator stack and handle it in the code.
- Add error checking.
- [This is required for a grade of C.]
- Add floats. Initially, treat everything as floating point – that is, no mixed arithmetic.
- [This is required for a grade of B.]
- Add reflection to support mixed integer and floating point calculations.
- [This is required for a grade of A.]

Part III – Reflection

Package reflect implements run-time reflection, allowing a program to manipulate objects with arbitrary types. The typical use is to take a value with static type interface{} and extract its dynamic type information by calling TypeOf, which returns a Type.

See "The Laws of Reflection" for an introduction to reflection in Go: https://blog.golang.org/laws-of-reflection.

Part IV – Specifics

Each student will develop a Go program to implement a simple mixed integer / float calculator as described above. Each program submission will include a report describing the problem, your approach to solving it (pseudo code, etc), the source code, and sample outputs. Your report should cite any sources of materials you used in solving the problem.