CARLETON UNIVERSITY Department of Systems and Computer Engineering
SYSC 3101 Programming Languages

1 Scheme Friday, February 5th 2016.

Write in SCHEME, a compiler for arithmetic expressions of the form:

```
<op operand1 operand2>
```

where `op` is either `+`, `-`, `*`, or `/` and the operands are either numbers or nested expressions. An example of a legal expression is `(* (+ 3 6) (- 7 9))`. Assume that the target machine has instructions:

```
(move value register)
(add register-1 register-2)
(subtract register-1 register-2)
(times register-1 register-2)
(divide register-1 register-2)
```

All arithmetic operations will leave the result in the first register argument. To simplify, assume an unlimited number of registers. Your compiler should take an arithmetic expression and return a list of the machine operations. For instance, your compiler should output the following machine instructions for the example above [`(* (+ 3 6) (- 7 9))`]:-

```
move 3 register-1
move 6 register-2
add register-1 register-2
move 7 register-3
move 9 register-4
subtract register-3 register-4
times register-1 register-3
```

Given an arithmetic expression *expr*, your compiler should:

1. Indicate whether *expr* is grammatically correct, `<arithmetic-expression>`.
2. If *expr* is not grammatically correct, return a syntax error message and if possible explain the problem.
3. If *expr* is a grammatically correct arithmetic expression, gives the list of machine codes.

Your main function will have no arguments, and will be named **`compile`**. This is necessary to make testing your code feasible. The test will be done as follows:

```
> ( compile )
(* 2 3)
move 2 register-1
move 3 register-2
times register-1 register-2
>
```

Hints:

1. The arithmetic expression can be specified as:

   <arithmetic-expression> → (<op> <arithmetic-expression> <arithmetic-expression>)
   | <constant>
   <op> → + | - | * | /

2. You can use stack to implement your compiler. In that case, you will have to design two functions – push and pop. push will stack an element into the stack while pop will un-stack an element.