

Data Structures and Objects

CSIS 3700

Fall Semester 2017 — CRN 42034

Project 2 — Calculator

Due date: Friday, October 13, 2017

Goal

Develop a program that implements a four-function calculator that performs all arithmetic with fractions.

Details

Your program will read a list of arithmetic expressions, evaluate them and display their results. All numbers in the expression will be integers; however, the results of calculations will be fractions.

Your program must be able to process any valid arithmetic expression that includes the following:

- Nonnegative integer numbers
- The four basic arithmetic operations
- Parentheses
- Variable names, up to 40 variables; names follow C++ naming rules
- Assignment in the form *var = expression*

For each expression, evaluate it, display the result and store the result in the appropriate variable, if necessary.

►Required Objects

A calculator needs two **Stack** objects — one to store numbers and one to store operators. In this program, the number stack — the *numStack* — will store **Fraction** objects and the operator stack — the *opStack* — will store characters.

In order to store and retrieve variable values, a **Dictionary** object will be necessary. The keys are strings and the values are **Fractions**. The exact implementation of the variable dictionary does not matter.

►Calculator Algorithm

The program must read multiple lines from the standard input. Each line contains an arithmetic expression and possibly an assignment to a variable. An algorithm for processing such a line follows in Algorithms 1 and 2.

Algorithm 1 Main calculator algorithm

```
1: procedure EVALUATE(string s)
2:   Clear numStack
3:   Clear opStack
4:   Push $ onto opStack
5:   first  $\leftarrow$  0
6:   dest  $\leftarrow$   $\Lambda$ 

7:   Scan forward for = symbol
8:   if = is found then
9:     first  $\leftarrow$  position of character after =
10:    dest  $\leftarrow$  first name found on line
11:  end if

12:  while first < s.length do
13:    PROCESSSYMBOL(s, first)
14:  end while

15:  while top of opStack is not $ do
16:    Perform top operation
17:  end while

18:  if dest  $\neq$   $\Lambda$  then
19:    Insert or update dictionary, key is dest, value is top of numStack
20:  end if
21:  output top of numStack
22: end procedure
```

Algorithm 2 Processing a symbol in the input string

```
1: procedure PROCESSSYMBOL(string s, int first)
2:   if s[first] is a digit then
3:     Convert digit sequence to Fraction
4:     Push Fraction object onto numStack
5:     Advance first to first character past digit sequence
6:   else if s[first] is a letter then
7:     Extract name into string
8:     Search for name in dictionary, push value onto numStack
9:     Advance first to first character past name
10:  else if s[first] is ( then
11:    Push ( onto opStack
12:    Increment first
13:  else if s[first] is ) then
14:    while top of opStack is not ( do
15:      Perform top operation
16:    end while
17:    Pop ( from top of numStack
18:    Increment first
19:  else if s[first] is an operator then
20:    while top of opStack has precedence over s[first] do
21:      Perform top operation
22:    end while
23:    Push s[first] onto opStack
24:    Increment first
25:  else
26:    Increment first
27:  end if
28: end procedure
```

To process an operator, pop the **opStack** into a variable. Then, pop two values from the **numStack** into two **Fraction** objects. The first value popped is the right operand, the second value is the left operand. Perform the given operation and push the answer onto the **numStack**.

If the expression is well-formed, then at line 18 of Algorithm 1, the **opStack** will only have \$ and the **numStack** will have only one value which is the result of evaluating the expression. If the expression is not well-formed, an exception might be thrown or one of the stacks will have more than one value. In these cases, output an error message.

What to turn in

Turn in your source code and **Makefile**. If you use Code::Blocks, turn in a tarball of your project directory.