# Data Structures and Objects
# CSIS 3700

*Fall Semester 2018 — CRN 41930*

---

Project 2 — Fraction Calculator
*Due date: Friday, October 12, 2018*

## *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 **Fraction**s. 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

```
procedure EVALUATE(string $s$)
    Clear numStack
    Clear opStack
    Push $ onto opStack
    $first \leftarrow 0$
    $dest \leftarrow \Lambda$

    Scan forward for = symbol
    if = is found then
        $first \leftarrow$ position of character after =
        $dest \leftarrow$ first name found on line
    end if

    while $first < s.length$ do
        PROCESSSYMBOL($s, first$)
    end while

    while top of opStack is not $ do
        Perform top operation
    end while

    if $dest \neq \Lambda$ then
        Insert or update dictionary, key is $dest$, value is top of numStack
    end if
    output top of numStack
end procedure
```

**Algorithm 2** Processing a symbol in the input string

```
procedure ProcessSymbol(string $s$, int $first$)
    if $s[first]$ is a digit then
        Convert digit sequence to Fraction
        Push Fraction object onto numStack
        Advance $first$ to first character past digit sequence
    else if $s[first]$ is a letter then
        Extract name into string
        Search for name in dictionary, push value onto numStack
        Advance $first$ to first character past name
    else if $s[first]$ is ( then
        Push ( onto opStack
        Increment $first$
    else if $s[first]$ is ) then
        while top of opStack is not ( do
            Perform top operation
        end while
        Pop ( from top of numStack
        Increment $first$
    else if $s[first]$ is an operator then
        while top of opStack has precedence over $s[first]$ do
            Perform top operation
        end while
        Push $s[first]$ onto opStack
        Increment $first$
    else
        Increment $first$
    end if
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.