CPSC 3400 Languages and Computation
Winter 2024

Homework 5
Due: Wednesday, Feb 21 at 10:00pm

Write a function `simplify` (in `hw5.fsx`) that simplifies an algebraic expression. The expression using a similar expression discriminated union used for symbolic differentiation in class (shown below). It removes the divide and power operators but adds a second variable Y.

```
type Expression =
    | X
    | Y
    | Const of float
    | Neg of Expression
    | Add of Expression * Expression
    | Sub of Expression * Expression
    | Mul of Expression * Expression
```

To start, the file `hw5.fsx` is available on `cs1`:

`/home/fac/bdiazacosta/cpsc3400/hw5/hw5.fsx`

It contains this type definition, a function `exprToString` that converts an expression to a more readable string format, and contains tests for the examples shown in this document.

It must perform the following algebraic simplifications:

- Addition involving two numbers. `Add (Const 5.0, Const 3.0)` → `Const 8.0`
- Subtraction involving two numbers. `Sub (Const 5.0, Const 3.0)` → `Const 2.0`
- Multiplication involving two numbers.
  `Mul (Const 5.0, Const 3.0)` → `Const 15.0`
- Negation involving a number.
  `Neg (Const 4.0)` → `Const -4.0`; `Neg (Const -9.0)` → `Const 9.0`
- Addition with zero. `Add (X, Const 0.0)` → `X`; `Add (Const 0.0, Y)` → `Y`
- Subtraction with zero. `Sub (X, Const 0.0)` → `X`; `Sub (Const 0.0, Y)` → `Neg Y`
- Subtraction of identical terms. `Sub (Y, Y)` → `Const 0.0`
- Multiplication with zero.
  `Mul (X, Const 0.0)` → `Const 0.0`; `Mul (Const 0.0, Y)` → `Const 0.0`
- Multiplication with one. `Mul (X, Const 1.0)` → `X`; `Mul (Const 1.0, Y)` → `Y`
- Double negation. `Neg (Neg X)` → `X`

The basic examples here are tests `t1-t15` in the provided file.

Unlike symbolic differentiation, `simplify` must be applied recursively. For example, simplifying the following expression requires three simplifications to arrive at 0.

```
Sub (Mul (Const 1.0, X), Add (X, Const 0.0))

Sub (X, Add (X, Const 0.0))   simplify the multiply
Sub (X, X)                    simplify the add
Const 0.0                     simplify the subtract
```

This expression is test `t16` in the provided file.

Additional examples from the interactive interpreter:

```
> simplify (Add (Mul (Const 4.0, Const 3.0), Sub (Const 11.0, Const 5.0)));;
val it : Expression = Const 18.0
> simplify (Sub (Sub (Add (X, Const 1.0), Add (X, Const 1.0)), Add (Y, X)));;
val it : Expression = Neg (Add (Y,X))
> simplify (Sub (Const 0.0, Neg (Mul (Const 1.0, X))));;
val it : Expression = X
> simplify (Mul (Add (X, Const 1.0), Neg (Sub (Mul (Const 2.0, Y), X))));;
val it : Expression = Mul (Add (X,Const 1.0),Neg (Sub (Mul (Const 2.0,Y),X)))
```

The same four examples in algebraic form:

Original Expression	***Simplified Expression***
$(4 \times 3) + (11 - 5)$	18
$\big((x + 1) - (x + 1)\big) - (y + x)$	$-(y + x)$
$0 - \big(-(1 \times x)\big)$	x
$(x + 1) \times \Big(-\big((2 \times y) - x\big)\Big)$	$(x + 1) \times \Big(-\big((2 \times y) - x\big)\Big)$ *no simplification possible*

These four examples are tests `t17-t20` in the provided file.

Additional notes:

- For comparisons of the same expression, only look for exact matches. You do not have to look for algebraically equivalent expressions. For instance `Add (Const 3.0, X)` is an exact match of `Add (Const 3.0, X)` but not `Add (X, Const 3.0)`.
- Some input expressions may not be able to be simplified (such as the last expression in the table above) – simply return the non-simplified expression.
- Do not modify the `Expression` type and `exprToString` function. Any modifications will cause tests to fail, possibly resulting in a very low score.
- Do not consider testing to be complete with the 20 provided tests. Be sure to add your own tests.

Grading

Grading is based on the following rubric and is primarily based on functionality based on how many tests pass or fail. If your program passes the test, you get full credit for that test. If your program fails the test, you get no credit for that test.

simplify	*50 points*
• 20 provided tests:	-1 for each failing test
• 10 additional tests:	-3 for each failing test

The following additional deductions are possible:

- Programs that modify the `Expression` type and/or `exprToString` function may result in a deduction even if all tests pass.
- Programs may lose up to 5 points if they exhibit poor readability or style.
- Using F# constructs or system functions outside the subset in class will result in a deduction. Severe deductions are possible if the use significantly simplifies one of the functions and/or results in a style that is not pure functional programming. I also reserve the right to require a resubmission.
- Programs that contain syntax errors will receive a zero.

Submitting your Assignment

To submit your assignment, you will need to transfer your file to `cs1`. On `cs1`, run the following script in the directory with your program:

```
/home/fac/bdiazacosta/submit/cpsc3400/hw5_submit
```

This will copy the files `hw5.fsx` to a directory that can be accessed by the instructor. Please be sure to keep the same file name or the submission program will not work. Only the last assignment submitted before the due date and time will be graded. ***Late submissions are not accepted and result in a zero.***