Assignment 4: Expression Trees

This assignment uses the following six files

`evalExp.c`, `evalExp.h`, `recognizeExp.c`, `recognizeExp.h`, `scanner.c`, `scanner.h`

from assignment 2, extended with the two files

`prefixExp.c` and `prefixExp.h`.

The contents of these last two files are discussed in Section 2.5.2 of the notes. All files and a `Makefile` are available on Themis as `assignment4code.zip`.

Part 1 (3 points for tests)

Copy the files `prefixExp.h` and `prefixExp.c` to new files `infixExp.h` and `infixExp.c`. Rename and modify all relevant functions such that infix (instead of prefix) expressions can be recognized and evaluated using expression trees. Please ensure that all your new functions in `infixExp.c` have different names than those in `prefixExp.c`. If an existing function does not have to be modified (for example, `valueIdentifier`), then delete it from `infixExp.*` and import it from `prefixExp.h`.

Also make a new file `mainInfix.c` based on `mainPref.c`. Now you can run '`make infix`' and other commands provided in the `Makefile`.

Note: only upload the three new files to Themis, not the eight files listed above! They will be inserted by Themis automatically, so you can still call any functions defined in them.

An example of input and resulting output:

input	resulting **output**

`2 + 3 * 4 - 5 / 6`
`(2+3) * (4-5) /6`
`2 * x * x + 3 * x + 7`
`var1 - 3 * var2 /var3`
`+ 2 3`
`!`
	`give an expression: 2 + 3 * 4 - 5 / 6`
	`in infix notation: ((2 + (3 * 4)) - (5 / 6))`
	`the value is 13.1667`
	
	`give an expression: ( 2 + 3 ) * ( 4 - 5 ) / 6`
	`in infix notation: (((2 + 3) * (4 - 5)) / 6)`
	`the value is -0.833333`
	
	`give an expression: 2 * x * x + 3 * x + 7`
	`in infix notation: ((((2 * x) * x) + (3 * x)) + 7)`
	`this is not a numerical expression`
	
	`give an expression: var1 - 3 * var2 / var3`
	`in infix notation: (var1 - ((3 * var2) / var3))`
	`this is not a numerical expression`
	
	`give an expression: + 2 3`
	`this is not an expression`
	
	`give an expression: good bye`

Part 2 (5 points for tests, 2 points for code review)

1. Define a function `simplify` that simplifies expression trees, according to the following requirements where E is an arbitrary expression:

   $0 * E$ and $E * 0$ are simplified to 0;
   $0 + E$, $E + 0$, $E - 0$, $1 * E$, $E * 1$ and $E/1$ are simplified to E.

2. Define a function `differentiate` that, given the simplified expression tree of an expression E, determines the expression tree of the derivative dE/dx. Use the function `simplify` to simplify the result of `differentiate`. For the computation of the derivative, the following rules apply:

$$\begin{array}{lcl} dn/dx & = & 0 \quad \text{if } n \text{ is a number} \\ dy/dx & = & 0 \quad \text{if } y \text{ is an identifier different from } x \\ dx/dx & = & 1 \\ d(E_1 + E_2)/dx & = & dE_1/dx + dE_2/dx \\ d(E_1 - E_2)/dx & = & dE_1/dx - dE_2/dx \\ d(E_1 * E_2)/dx & = & (dE_1/dx) * E_2 + E_1 * (dE_2/dx) \\ d(E_1/E_2)/dx & = & ((dE_1/dx) * E_2 - E_1 * (dE_2/dx))/(E_2 * E_2) \end{array}$$

   Observe that, in the case of multiplication and division, both the original function and its derivative are involved. As a consequence, you will need a C function to make a copy of an expression tree.

3. Extend the functionality of the program of Part 1 for non-numerical expressions with the functions `simplify` and `differentiate`.

Upload all new files to Themis (including Part 1), but not the eight files listed above. An example of input and resulting output:

input	resulting **output**

`2 / (3 * 4) - (5 + 6)`
`2*x*x + 5*x - 6`
`0*x + 1*x*1*x*1 - 0`
`(x/1 - 2)/(x + 1)`
`2*y*y + 3*y - 7`
`!`
	`give an expression: 2 / ( 3 * 4 ) - ( 5 + 6 )`
	`in infix notation: ((2 / (3 * 4)) - (5 + 6))`
	`the value is -10.8333`
	
	`give an expression: 2 * x * x + 5 * x - 6`
	`in infix notation: ((((2 * x) * x) + (5 * x)) - 6)`
	`this is not a numerical expression`
	`simplified: ((((2 * x) * x) + (5 * x)) - 6)`
	`derivative to x: (((2 * x) + (2 * x)) + 5)`
	
	`give an expression: 0 * x + 1 * x * 1 * x * 1 - 0`
	`in infix notation: (((0 * x) + ((((1 * x) * 1) * x) * 1)) - 0)`
	`this is not a numerical expression`
	`simplified: (x * x)`
	`derivative to x: (x + x)`
	
	`give an expression: ( x / 1 - 2 ) / ( x + 1 )`
	`in infix notation: (((x / 1) - 2) / (x + 1))`
	`this is not a numerical expression`
	`simplified: ((x - 2) / (x + 1))`
	`derivative to x: (((x + 1) - (x - 2)) / ((x + 1) * (x + 1)))`
	
	`give an expression: 2 * y * y + 3 * y - 7`
	`in infix notation: ((((2 * y) * y) + (3 * y)) - 7)`
	`this is not a numerical expression`
	`simplified: ((((2 * y) * y) + (3 * y)) - 7)`
	`derivative to x: 0`
	
	`give an expression: good bye`