Thompson, P. & Thompson, A. (1987). Computer presentations of structure in algebra. In J. Bergeron, N. Herscovics, & C. Kieran (Eds.), *Proceedings of the Eleventh Annual Meeting of the International Group for the Psychology of Mathematics Education*, *1*, 248-254.

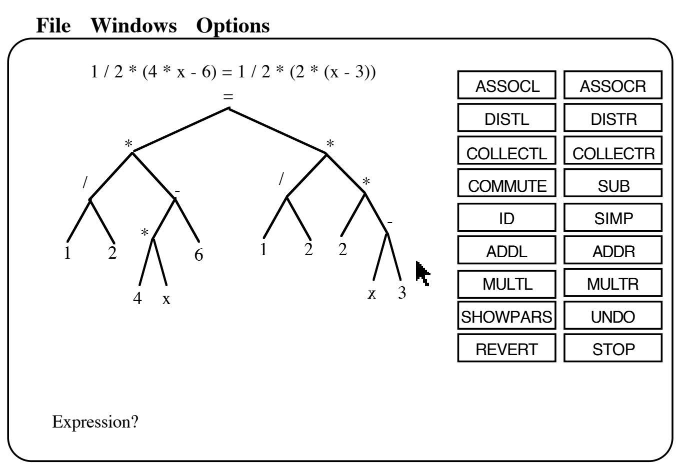
COMPUTER PRESENTATIONS OF STRUCTURE IN ALGEBRA

Patrick W. Thompson
Alba G. Thompson
Illinois State University

Many errors committed by students of algebra appear to be a result of their long-term inattention to structure of expressions and equations. A special computer program was developed that enabled students to manipulate expressions, but which constrained them to acting on expressions only through their structure. Eight leaving-seventh graders used the program for eight days. An analysis of their actions indicated that errors due to inattention to structure occurred largely while they were first learning a field property or identity, and that afterwards such errors were infrequent.

Typical errors found in previous studies of students' errors in algebra suggest that students studying algebra frequently fail to realize that formulas in mathematical symbol systems have an intrinsic structure (Lewis, 1981; Matz, 1982; Sleeman, 1982, 1984, 1985). In algebra, expressions are structured explicitly by the use of parentheses, and implicitly by assuming conventions for the order in which we perform arithmetic operations. It is hypothesized that many of students' errors in manipulating an algebraic expression are due to their inattention to the expression's structure.

To test this hypothesis, we built a program, called EXPRESSIONS, that presents expressions and equations in two formats: in usual (sentential) form and in the form of an expression tree. The figure to the right shows the screen after having entered the equation 4x-6=2(x-3) and then having multiplied both sides by $\frac{1}{2}$. The equation's expressions are shown in sentential notation at the top of the screen. The tree representation of the equation is shown directly below the sentential notation.



To change an expression by the use of a field property or other transformation, students put the mouse pointer on top of one of the buttons along the right side of the screen and then clicked the mouse to select that action. Then they put the pointer on top of the operation in the tree representation of the expression which defined the expression or subexpression to be transformed, and clicked the mouse again. The action was performed on the selected expression or subexpression, and the sentential notation and expression tree were changed accordingly.

To transform an expression by the use of an identity, students put the mouse pointer on top of the ID button, clicked the button, and then clicked the operation sign within the tree which defined the expression to which the identity was to be applied. The computer applied whichever of these identities was appropriate to the chosen expression or subexpression: $a–b = a+ -b$, $\frac{x}{y} = x*\frac{1}{y}$, $-x = -1*x$, or $x = 1*x$, and then updated the expression tree and sentential display accordingly.

Sample

The sample consisted of eight leaving-seventh graders—six males and two females—from the ISU elementary laboratory school and who volunteered to participate in the study. Their mean age was 13 years 1 month; their mean cumulative mathematics score on the Iowa Test of Basic Skills was 74.6. In the last quarter of seventh grade mathematics, five students received an A, one received a B, and two received a C.

Method

The study took place over nine consecutive weekdays in June of 1986. The first session was devoted to administering a pretest; eight sessions (50 minutes each) were given to direct instruction and practice. The pretest involved assessing students' knowledge of the conventions for order of operations (evaluating numeric expressions), their knowledge of field properties, and their knowledge of variables.

Instruction took place in a classroom at ISU, where the instructor used a Macintosh running EXPRESSIONS. The Macintosh was connected to a projector which created a 6' x 6' image of the screen. All instruction was videotaped.

For practice sessions, students were grouped in pairs by matching their cumulative mathematics score on the Iowa Test of Basic Skills. Practice sessions took place with students in two locations: in a computer room and in the classroom, with two students per computer. Students using the classroom computer were videotaped. Each pair of students was videotaped once. A set of booklets containing examples and practice problems were provided to each student. All students used a version of the program that stored their keystrokes and mouse–clicks in a data file which could be "played back" for later analysis.

Instruction proceeded in this order: order of operations in arithmetical expressions; field properties as transformations of arithmetical expressions; identities and derivations. An outline of the eight days of instruction is given in Table 1.

Day	***In Class***	***In Groups***
2	Order of operations; Evaluating expressions	Worksheet 1 (Parts 1 & 2)
3	Parentheses; Expression trees	Worksheet 1 (Part 3); Worksheet 2
4	Discuss Worksheet 2; Commutativity; Associativity; Example from Worksheet 3	————————————
5	————————————————	Worksheet 3 (Part 1)
6	Review commutativity; associativity; introduce distributing and collecting	Worksheet 3 (Part 2)
7	————————————————	Worksheet 3 (Part 2); Worksheet 4
8	Review field properties; introduce identities	————————————
9	————————————————	Worksheet 5

Table 1. Summary of instruction.

The worksheets comprised an integral part of instruction. Table 2 shows the numeric-transformation problems students worked in sessions 6 and 7. Table 3 shows the identity derivation problems students worked in session 9.

	Start With	*Change It To*
N1.	5*(4+3)	3*5 + 4*5
N2.	5*((4+3)+2)	(5*4)+((2+3)*5)
N3.	(7+3)*(6+5)	(7*6+7*5)+(3*6+3*5)
N4.	(6+$^{-}$5)*(6+5)	(6*6) + ($^{-}$5*5)
N5.	3*(8+4) + 9*(4+8)	(9+3)*(8+4)
N6.	3*(6/9) + (6/9)*7	10*(6/9)
N7.	$^{-}$5*3 + (2+3)*5	0 + 10
N8.	(5+9)*(5+9)	5*5 + 90 + 9*9

Table 2. Numeric transformation problems.

	Start With	*Change It To*
I1.	(z - q)*u	z*u - q*u
I2.	r*(s/t)	(r*s)/t
I3.	-(p + q)	-p + -q
I4.	(a + b)/c	a/c + b/c
I5.	6x + x	7x
I6.	5x - x	4x
I7.	x + x	2x

Table 3. Identity derivation problems

Results

Pretest

Six of the eight students processed numeric expressions from left to right, ignoring conventions for order of operations (e.g., 8 – 6 + 5 * 3 evaluates to 21), when grouping was not given explicitly. All eight were familiar with commutativity. Seven were familiar with associativity in its simplest form. None was familiar with distributivity. Six differentiated among expressions and equations on the basis of superficial characteristics (e.g. "y+2=5 is different from x-2=5 and x+2=5 because it uses y and the others use x.")

Analysis of computer use

EXPRESSIONS was modified to store all interactions. The stored files were then later rerun for analysis. Students' actions were categorized according to the following scheme:

- A — Appropriate transformation applied at an appropriate place in the expression, given the current and goal expressions.
- IA — Inappropriate transformation, e.g. trying to use the distributive property on *(a*b)+c*.
- AWP — Appropriate action, but applied in a wrong place. This was inferred if a student tried the same transformation twice in a row, first trying it at an inappropriate place in the expression and then applying it appropriately.
- CD — Confused direction. An action was placed in this category if a directional transformation was appropriate (such as using the associative property of multiplication to change the grouping from being on the left to being on the right) but the student chose the wrong direction.

Transforming Numeric Expressions

Table 4 shows the percents of students' actions falling within each category while working the numeric transformation problems (Table 2). Table 5 shows the percents of students' actions falling within each category while working the identity derivation problems (Table 3).

In many cases, the majority of inappropriate actions occurred early in a problem, suggesting that students were exploring the effects of the available transformations upon expressions. To eliminate the effects of exploratory errors upon the percents in Tables 4 and 5, the data were reanalyzed by the same categorization scheme as previously, but with this exception: All actions prior to two consecutive appropriate actions were discarded. Tables 6 and 7 show the percents of "non-exploratory" actions falling within each of the categories.

Problem	*A*	*IA*	*AWP*	*CD*
N1	87	0	0	13
N2	75	25	0	0
N3	69	9	17	6
N4	60	31	8	1
N5	82	0	18	0
N6	88	0	0	12
N7	56	39	5	0
N8	89	11	0	0

Table 4. Numeric transformations: Percent per category of all actions. All students completed all problems.

Problem	*A*	*IA*	*AWP*	*CD*
I1	41	48	6	6
I2	*	*	*	*
I3	*	*	*	*
I4	69	23	0	8
I5	56	38	6	0
I6	81	12	6	0
I7	80	20	0	0

Table 5. Identity derivations: Percent per category of all actions; "*" indicates incomplete data.

The differences between Tables 4 and 6 and between Tables 5 and 7 suggest that students' errors were due to initial play involved in understanding the problems, understanding the available transformations, and making connections between the two. Once students internalized the transformations' structural constraints, they were less likely to commit errors and were more efficient in their solution strategies.

Problem	*A*	*IA*	*AWP*	*CD*
N1	100	0	0	0
N2	95	5	0	0
N3	85	7	4	4
N4	92	4	0	4
N5	82	0	18	0
N6	88	0	0	12
N7	100	0	0	0
N8	94	6	0	0

Table 6. Numeric transformations: Percent per category of non-exploratory actions.

Problem	*A*	*IA*	*AWP*	*CD*
I1	58	39	3	0
I2	*	*	*	*
I3	*	*	*	*
I4	86	0	0	14
I5	100	0	0	0
I6	100	0	0	0
I7	100	0	0	0

Table 7. Identity derivations: Percent per category of non-exploratory actions; "*" indicates incomplete data .

Exploratory errors were commonly either irrelevant to the problem being solved (e.g., "what does this button do?") or were attempts at doing something that might take an expression closer to its goal state. For example, one error was to try to use associativity to change $(a+b)*c$ into $a+(b*c)$, to which the computer "responded" by doing nothing. The students wanted b to be multiplied by c, and apparently concluded that the associative property would do that regrouping for them. Also, it was common for students to repeat an errorful action. It appeared that repeating an action supported students in their attempts to reflect on the reasoning they used in first choosing the action, and supported them in understanding the reason that the chosen transformation did not accomplish whatever they had in mind.

Discussion

Previous studies of students' errorful manipulation of expressions and equations proposed that their errors are due to mal-formed rules—perturbations of correct rules. This study asked whether or not such errors were due to students' inattention to structural features of expressions and transformations thereupon. The results suggest that mal-rules need not be a natural occurrence when students operate in an environment that supports explicit attention to expressions' structures, and where structure also imposes constraints on students' actions. We cannot say from the results presented here that errors reported in previous studies were due to students' inattention to structure, but these results indicate that attention to structure is an important consideration.

Students could attempt errorful transformations of expressions while using the computer, but the computer would not carry them out. It appeared that they interpreted this context as one where experimentation became natural and beneficial. We would like to think that students disposition to experimentation was a result of the software and the use made of it. However, it also could have been a result of the instructor's style of instruction, or it could have been that this particular group of students was predisposed to experimentation and reflection.

A limitation of the study is that students were not assessed outside of the computer environment. It is quite conceivable that had these students been left to their own devices, they would have committed errors on paper and pencil that they learned not to make while using the computer. The issue of transfer from computer to noncomputer environments requires extensive research.

Another limitation of the study is that we do not know the depth of commitment that these students had when they "proved" that two expressions were equivalent, or when they derived an identity. Did students think of an identity as a theorem that could be applied in other contexts? We do not know.

A feature of structure which we could not address here with data, but which was addressed explicitly in the study, was that of variable. Many problems (all of those in Tables 2 and 3) were designed so that students would have to treat a subexpression as a unit. When applying field properties and identities to expressions, students regularly needed to substitute a subexpression in an expression for a letter in the canonical statement of a property or identity. They became quite adept at this. Also, students felt no discomfort when letters were first introduced in to-be-transformed expressions. Apparently, by having them transform numerical expressions, they became used to the idea that expressions could be manipulated regardless of their constituent elements. Thus, when letters were introduced, students saw no obstacle in continuing what they had already learned to do with numerical expressions. The approach wherein manipulating algebraic expressions is presented as a natural extension of manipulating numerical expressions deserves further research.

The use of expression trees as one of the representational systems within the computer program proved to be a positive feature of instruction. Students found expression trees to be quite intuitive. When doing Worksheet 1, which focused upon evaluating expressions given in sentential form, students used EXPRESSIONS only to check their answers. They were told only that they needed to click SIMP and then click the top of the tree to evaluate an expression. We found four students who constructed expression trees for complex expressions as an aid to evaluating them, *even though there had been no discussion about how expression trees are constructed, and these students had never before seen an expression tree*.

Finally, it should be noted that in eight days of instruction these leaving-seventh grade students went from essentially no working knowledge of order of operations to deriving algebraic identities, and did so with some depth of understanding. Even with the limitations stated earlier in this discussion, the fact that such coverage is possible makes us question assumptions that are built into traditional junior high school pre-algebra and algebra curricula about what one can expect of junior high school students in the United States.

References

Sleeman, D. H. (1982). Assessing competence in basic algebra. In D. H. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 186-199). New York: Academic Press.

Sleeman, D. H. (1984) An attempt to understand students' understanding of basic algebra. *Cognitive Science*, *8*, 387-412.

Lewis, C. (1981). Skill in algebra. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 85-110). Hillsdale, NJ: Erlbaum.

Matz, M. (1982). Towards a process model for high school algebra errors. In D. Sleeman, & J. S. Brown (Eds.), *Intelligent tutoring systems*. New York: Academic Press.

Sleeman, D. H. (1985). Basic algebra revisited: A study with 14-year olds. *International Journal of Man-Machine Studies*, *22*, 127-149.

Quick Sheet

General - Button Group 1

SIMP	Do arithmetic (if all numbers) or simplify expression.
UNDO	Undo last action. UNDO undoes itself.
SHOWPARS	All expressions/equations are printed with parentheses. Order of operations is explicit. SHOWPARS changes itself to HIDEPARS.
HIDEPARS	All expressions/equations are printed without parentheses, except where necessary to avoid ambiguity. Order of operations is implicit. HIDEPARS changes itself to SHOWPARS.
HISTORY	Print a history of current expression/equation at the printer. Make sure printer is on before clicking HISTORY.
REVERT	Toggle between initial and current expression.
STOP	Stop the program. Select START from the Options Menu to start again.

Modifying Expressions - Button Group 2

All of Button Group 1, plus the following

ASSOCL	Associate-from-the-left. Move parentheses from grouping on the left to grouping on the right. Change (a+b)+c to a+(b+c).
ASSOCR	Associate-from-the-right. Move parentheses from grouping on the right to grouping on the left. Change a+(b+c) to (a+b)+c.
DISTL	Distribute-from-the-left. Apply the left distributive property of multiplication over addition. Change a(b+c) to ab+ac. Inverse of COLLECTL.
DISTR	Distribute-from-the-right. Apply the right distributive property of multiplication over addition. Change (a+b)c to ac+bc. Inverse of COLLECTR.
COLLECTL	Collect from the left. Change ab+ac to a(b+c). Inverse of DISTL.
COLLECTR	Collect from the right. Change ac+bc to (a+b)c. Inverse of DISTR.
ID	Change by identity. Changes (neg x) to -1 * x x - y to x + -y x/y to x * 1/y 1/u*1/v to 1/(u*v) √(x*y) to √x * √y *and vice-versa*.
SUB	Substitute one thing for another everywhere. Can substitute a number, letter, or expression for a number, letter, or expression.

Modifying Equations - Button Group 3

All of Button Groups 1 and 2, plus the following.

ADDL	Add the same thing to both sides of an equation on their left.
ADDR	Add the same thing to both sides of an equation on their right.
MULTL	Multiply both sides of an equation by the same thing on their left.
MULTR	Multiply both sides of an equation by the same thing on their right.

Modifying Equations - Button Group 4

The buttons in Button Group 3 are replaced by the single button:

OPERATE	Apply the same operation to both sides of an equation. Put a question mark (?) in place of the expressions of the equation. For example, click OPERATE, then enter COS(?)+3? to compose both sides of the equation with the function COS(X)+3X.

Allowable Operations and Functions

Binary operations:	= + - * / ^
Unary operations:	- √ ln( ) log( ) sin( ) cos( ) tan( ) sec( )
	csc( ) arctan( ) arccos( ) arcsin( ) exp( )

Optional Key Substitutes

So that you need not have to use the SHIFT key, you may obtain the following symbols by pressing the indicated keys.

To Get	*Press*
+	;
*	'
(	[
)	]
√	v

Drawing Boxes

Whenever the program is awaiting text input, you may draw boxes on the screen by clicking and dragging. This is useful when you wish to draw attention to a specific subtree in an expression tree or draw attention to a subexpression in an expression that is in sentential notation.

Quit The Program

Click STOP if the program is runing. Select RESET from the Options Menu, then select QUIT from the File Menu.

Implied Multiplication

You do not need to put a multiplication sign (*) between a number and a letter, anything and a left parenthesis, or between two numbers or two letters when they are separated by a space. The expression “2X-5” is the same as “2*X-5,” the expression “(5-7)(X+2)” is the same as “(5-7)*(X+2)”, and the expression “X Y” is the same as “X*Y.”

Worksheet 1

Order of Operations

Expressions that involve only numbers are evaluated according the the rules given below. These rules are called *conventions,* which means that there could be different rules, but these are the ones that everyone has agreed to use.

1. First, if there are parentheses, do what is inside them. To do what is inside parentheses, apply the rules listed here, including this one. After you have done what is inside parentheses, cross out that part of the expression and replace it with its result.

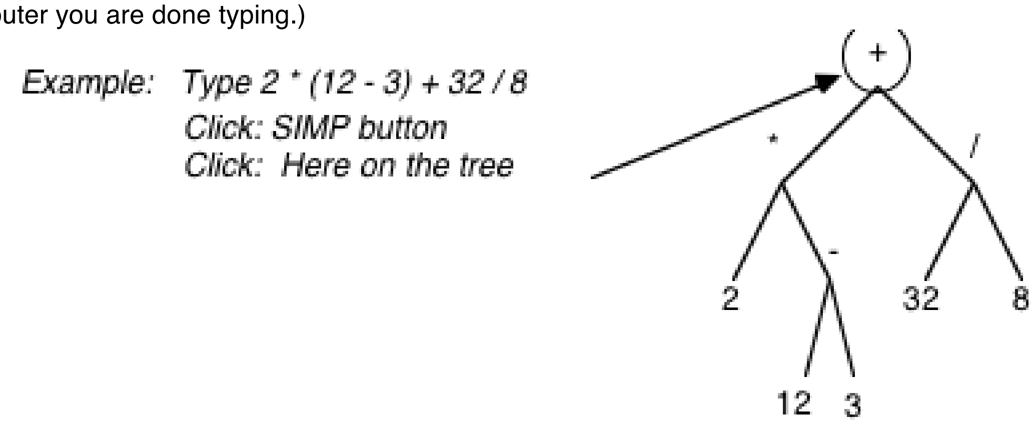
2. Second, perform every negation. Cross out that part of the expression and replace it with its result.

3. Third, perform every multiplication or division in the order they occur as you read from left to right. Cross out that part of the expression and replace it with its result.

4. Fourth, perform every addition or subtraction in the order they occur as you read from left to right. Cross out that part of the expression and replace it with its result.

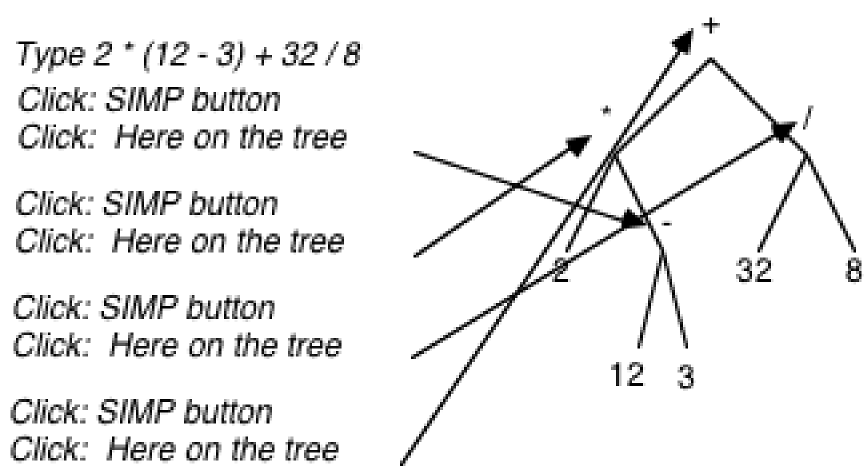
Practice

Evaluate each expression according to the rules listed above. Then check your answer using EXPRESSIONS. (Note: Any time you are told to type something, always end your typing by pressing RETURN or ENTER. Pressing RETURN or ENTER tells the computer you are done typing.)



This will replace the tree by the result of evaluating the whole expression.

You can check your steps by clicking each operation from bottom to top, as shown below.



When you click on the minus sign above 12 and 3, that part of the expression is replaced by 9, which is the *result* of 12 - 3. Then you are left with 2*9 and 32/8, connected by a plus sign.

Part I

Evaluate each of the expressions given below. Notice that the exercises are given in pairs. Each expression on the right involves the same numbers and operations as the expression on its left, but the grouping is different from the conventional grouping.

1.	16-9+2	2.	16-(9+2)
3.	20-12/2	4.	(20-12)/2
5.	3*8-7	6.	3*(8-7)
7.	2*6+2*4	8.	2*(6+2)*4
9.	8+3*6/9	10.	(8+3)*6/9
11.	6+2*8-2*3/4	12.	(6+2)*(8-2)*(3/4)
13.	2*12+2*7+2*8+10	14.	2*(12+2*(7+2*(8+10)))

Name __________

Part II

Put parentheses in the expressions given below so that the explicit groupings match the implicit groupings that are determined by the conventions for order of operations.

*Example: 11+4*2 ————> 11+(4*2)* — Do 4*2 first. No more parentheses needed

*Example: 9-2*7/3+4 ————> 9 - (2*7)/3+4* — Do 2*7 first
*————> 9 - ((2*7)/3)+4* — Do (2*7)/3 second
*————> (9-((2*7)/3))+4* — Do 9 - ((2*7)/3) third
No more parentheses needed

	<u>Without Parentheses</u>	<u>With Parentheses</u> (Draw them in yourself)
1.	1 + 2 + 3 + 4	1 + 2 + 3 + 4
2.	4 - 2 + 5 / 3	4 - 2 + 5 / 3
3.	3 * 18 - 6 / 3 + 10	3 * 18 - 6 / 3 + 10
4.	5 + 4 * 2 / 8 - 6	5 + 4 * 2 / 8 - 6
5.	2 / 3 + 7 - 5 * 8 / 4	2 / 3 + 7 - 5 * 8 / 4
6.	8 - 4 + 7 - 6 + 2	8 - 4 + 7 - 6 + 2
7.	7 * 6 - 5 / 4 + 3 / 2	7 * 6 - 5 / 4 + 3 / 2

To Check Your Answers, Do This:

- Start the EXPRESSIONS program if it is not already started.
- Click **SHOWPARS** if that button is showing. (If **HIDEPARS** is showing, then skip this step.)
- Type each expression into EXPRESSIONS as it is given <u>*without*</u> parentheses.
- Each time you enter an expression, compare where EXPRESSIONS puts parentheses with where you put them.

Name __________

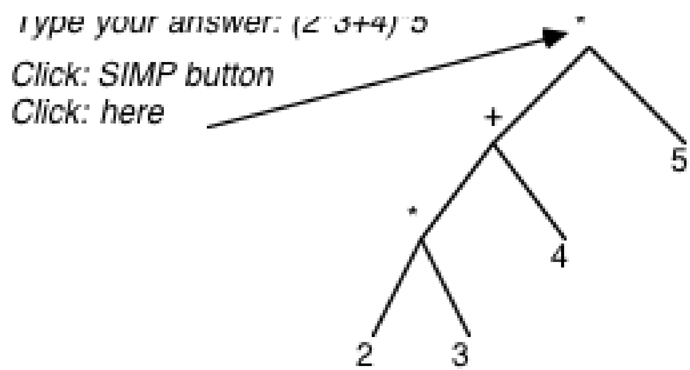
Part III

Place parentheses within each expression so that, when evaluated, it gives the indicated result. Check your answer (1) by evaluating the expression, or (2) by entering your expression into EXPRESSIONS *with* parentheses, clicking the **SIMP** button, and then clicking the top of the tree.

*Example: 2 * 3 + 4 * 5 = 50* (put parentheses where necessary to get 50)

*(2 * 3 + 4) * 5 = 50* (your answer)

To check yourself:



1. 10 - 2 + 4 / 2 = 6

2. 10 - 2 + 4 / 2 = 7

3. 9 + 6 / 3 - 1 = 12

4. 9 + 6 / 3 - 1 = 4

5. 10 / 2 - 1 * 3 + 4 = 16

6. 10 / 2 - 1 * 3 + 4 = 28

7. 8 / 2 + 6 - 4 / 2 = -1

8. 8 / 2 + 6 - 4 / 2 = 1

9. 8 * 6 - 3 + 1 * 4 = 41

10. 8 * 6 - 3 + 1 * 4 = 64

Worksheet 2

Expression Trees

An expression tree shows the *structure* of an expression—how the expression's parts fit together.

An expression tree is made by following these steps:

1. Put an operation sign at the top.

2. Draw two line segments that drop to the left and to the right of the operation sign. These will connect the operation sign with the things that are being operated upon (called *operands*).

3. Put the operands below the operation sign at the ends of the line segments.

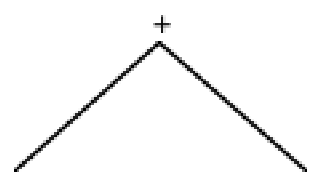
The "trick" to constructing complicated expression trees is to understand that, in step 3, if an operand is an expression, you will have to construct another expression tree.

*Example: Draw an expression tree for 3*5+2*

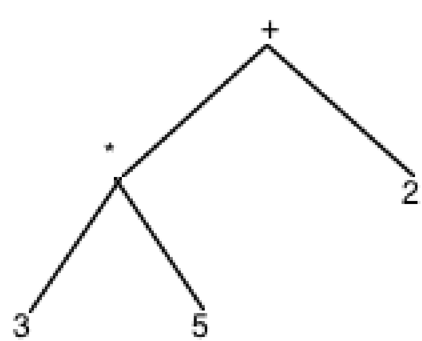
First, rewrite the expression with parentheses.

*3*5+2 ————> (3*5)+2*

Second, write the top operation sign with two branches.



Third, put the things being operated upon at the ends of the branches. If an expression is one of the things being operated upon, then repeat these steps to write it as an expression tree.

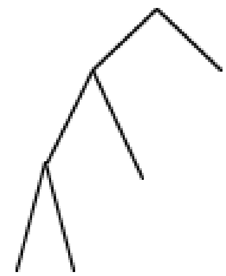


Part I (Partial trees)

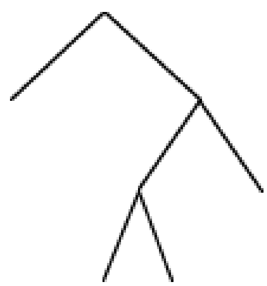
Complete the expression tree for each expression. Check yourself by entering the expression into EXPRESSIONS and comparing its tree with yours.

1. 5 - 10 * 2

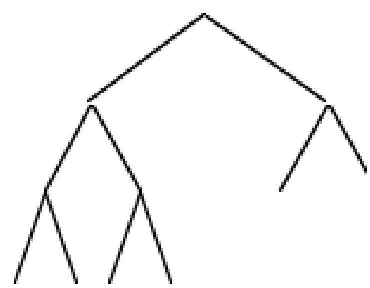
2. 4 / 2 + 3 - 4



3. 4 / (2 + 3 - 4)



4. 5 * 4 + 6 / 3 - 7 * 2



Part II

Draw an expression tree for each expression. Check yourself by entering the expression into EXPRESSIONS and comparing its tree with yours.

1. 11 - 2 * 9 / 3

2. 6 * (11 - 8) + 2 * 5

3. 2 * 12 - 4 * 8

4. 2 * (12 - 4) * 8

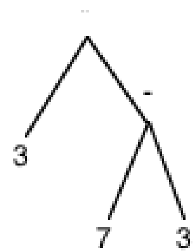
5. 9 - 3 + 3 / 4 - 2 * 5

6. 9 - (3 + 3 / 4 - 2) * 5

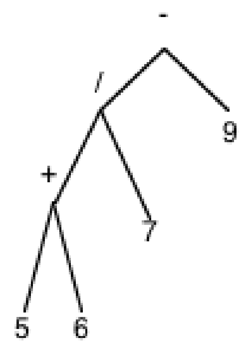
Part III

For each tree, enter an expression into EXPRESSIONS that will make the tree.

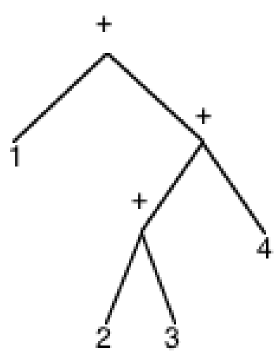
1.



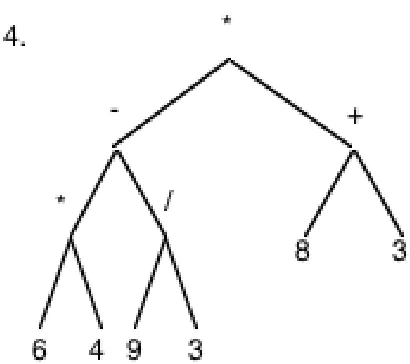
2.



3.



4.



Worksheet 3

Properties of Operations

*Select **Button Group 2** from the Options menu before doing these examples and exercises. If EXPRESSIONS is running, you will need to stop it to select Button Group 2.*

Be sure to have <u>parentheses showing</u>.

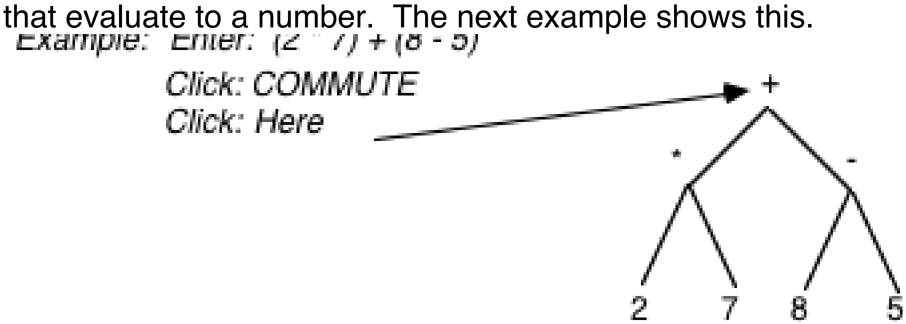
When something is always true about an operation it is called a *property* of that operation. Addition and multiplication have several properties. One is called *commutativity*; the other is called *associativity.*

Part I: Commutative Property of Addition
Commutative Property of Multiplication

When we write something like (5+4)=(4+5), we are stating that the two expressions evaluate to the same number. When we write (x+y)=(y+x) we are stating that regardless of the numbers we add, if we change their order they will evaluate to the same sum.

You can change the order in which you add or multiply two numbers and the expression's value will not change. That is, you can change ♦ + Δ to Δ + ♦, or you can change ♦ * Δ to Δ * ♦, and the result will not change.

Please keep in mind that ♦ and Δ do not have to be numbers. They can be expressions that evaluate to a number. The next example shows this.

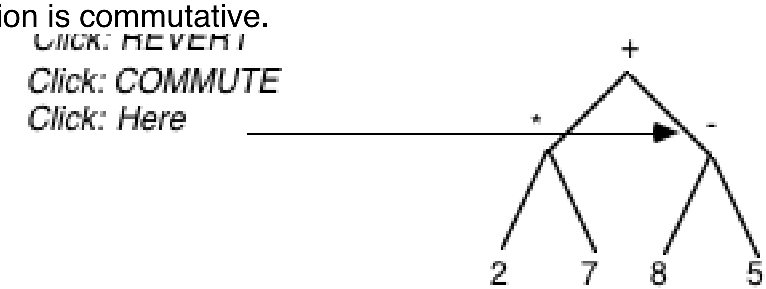


After applying the commutative property of addition to (2 * 7) + (8 - 5), the resulting expression is (8 - 5) + (2 * 7).

You can apply the commutative property of multiplication to the sub-expression (2 * 7).

The result is the expression (8 - 5) + (7 * 2). The expression (8 - 5) + (7 * 2) is *equivalent* to (8 - 5) + (2 * 7), because they still evaluate to the same number. To see this, click **SIMP**, then click the top of the tree. The result is 17. Then click **REVERT**. You now see the original expression. Click **SIMP** and click the top of the tree. The result is still 17.

Not every operation is commutative.



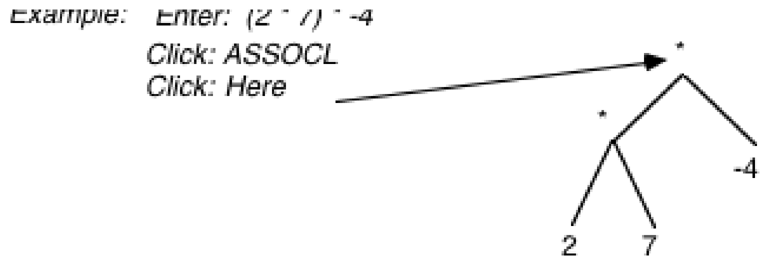
Nothing happened when you clicked the subtraction sign. This is because changing (8 - 5) to (5 - 8) would have changed the value of the expression. That is, *subtraction is not commutative, because changing the order of the operands may change the value of the expression.* Think of another operation that is not commutative.

<u>Associativity</u>

You can change the way that three numbers are grouped (or *associated*) when they are all added or when they are all multiplied and you will not change the expression's value.

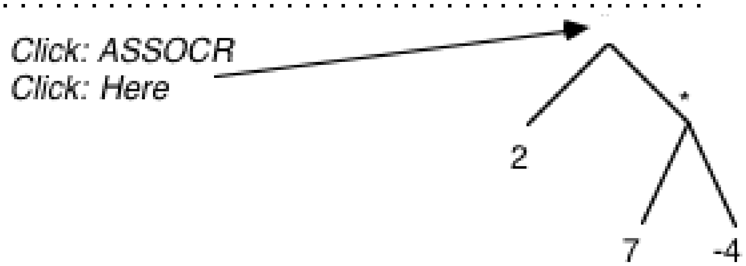
Re-associate <u>from the left</u>: *(a + b) + c ——> a + (b + c)*
*(a * b) * c ——> a * (b * c)*

Re-associate <u>from the right</u>: *a + (b + c) ——> (a + b) + c*
*a * (b * c) ——> (a * b) * c*



ASSOCL re-associates from the left, changing (2 * 7) * -4 to the expression 2 * (7 * -4).

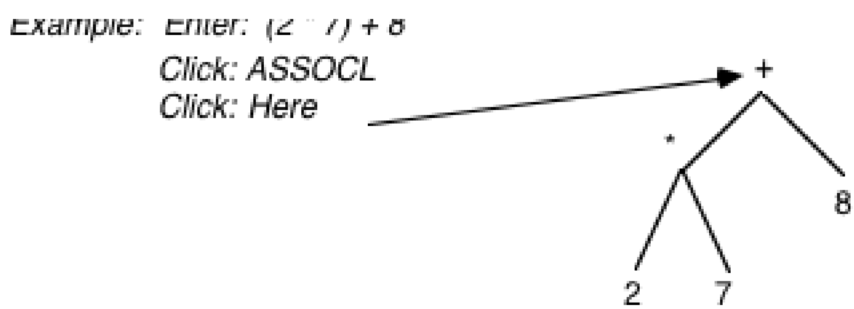
. . . .



ASSOCR re-associates from the right, changing 2 * (7 * -4) to the expression (2 * 7) * -4.

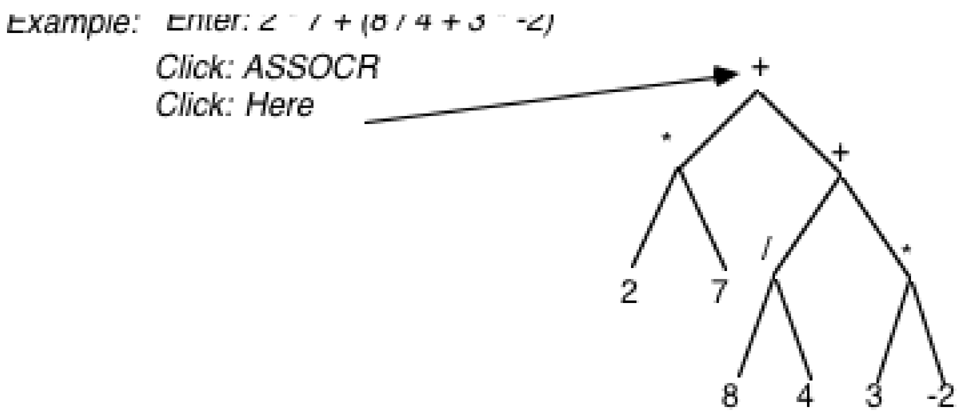
. . . .

Not every set of operations on three numbers is associative.



Nothing happened when you clicked the plus sign. In order to apply associativity, the operation signs must be both addition or both multiplication.

The operands do not have to be numbers. They can be expressions.



ASSOCR changed 2*7 + (8/4 + 3*-2) to (2*7 + 8/4) + 3*-2. This follows the pattern of changing *a + (b + c)* to *(a + b) + c.* In this case 2*7 is *a,* 8/4 is *b*, and 3*-2 is *c.* All that matters is that you have three things, and that they are all added or multiplied.

Exercises

When you use EXPRESSIONS with these exercises, be sure to select **Always Show Pars** and **Button Group 2** from the **Options** menu.

In each of the exercises below, write the list of buttons and operation signs that you would click to change the first expression into the second. Use the format shown in the example. *Do not use EXPRESSIONS with an exercise until you have planned your solution.* Here are the buttons you can use:

ASSOCL	ASSOCR
COMMUTE	SIMP

Example

Start with:	*Change it to:*
1+(2 + 3)	*(3 + 1) + 2*

Expression: 1 + (2 + 3)	*Button:* COMMUTE
Expression: 1 + (3 + 2)	*Button:* ASSOCR
Expression: (1 + 3) + 2	*Button:* COMMUTE
Expression: (3 + 1) + 2	*Button:* _______

Start with: Change it to:

5 * 4 + 3 + 2 3 + 4 * 5 + 2

Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________

Start with: Change it to:

2 + -7 * 4 + 3 3 + 2 + 4 * -7

Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________

Start with: (1 + 2) + (3 + 4)

Change it to: (1 + 4) + (2 + 3)

Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________

Start with: ((8/3)*(2-8))*((7+6)*(3*2))

Change it to: ((8/3)*(3*2))*((2-8)*(7+6))

Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________

Part II: Distributivity of Multiplication Over Addition

A statement that is always true about multiplication and addition is that

*a * (b + c) and (a*b) + (a*c)*

evaluate to the same number, regardless of the values of *a, b,* and *c.* If we turn the statement around, we can say that

*(a*b) + (a*c) and a * (b + c)*

evaluate to the same number. Either way we state it, this property allows us to change expressions in these ways:

Start with: a*(b + c)	Change it to: a*b + a*c
Start with: a*b + a*c	Change it to: a*(b + c)

Starting with a*(b + c) and changing it to a*b + a*c is called *distributing multiplication over addition from the left.*

Starting with a*b + a*c and changing it to a*(b + c) is called *collecting terms from the left.*

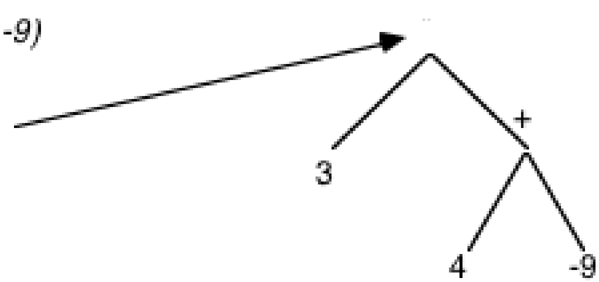
Distributing and collecting can also be done from the right.

Start with: (b + c)*a	Change it to: b*a + c*a
Start with: b*a + c*a	Change it to: (b + c)*a

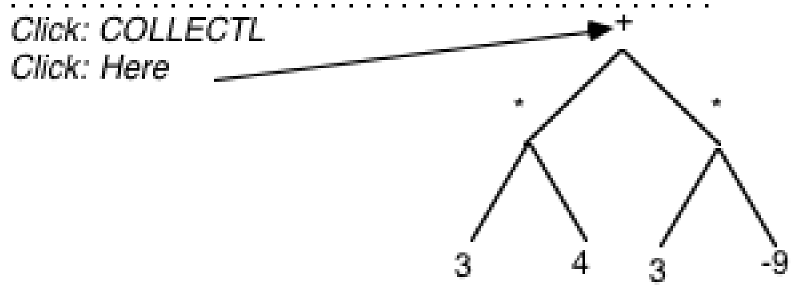
Starting with (b + c)*a and changing it to b*a + c*a is called *distributing multiplication over addition from the right.*

Starting with b*a + c*a and changing it to (b + c)*a is called *collecting terms from the right.*

Example: *Type: 3*(4 + -9)*
Click: DISTL
Click: Here

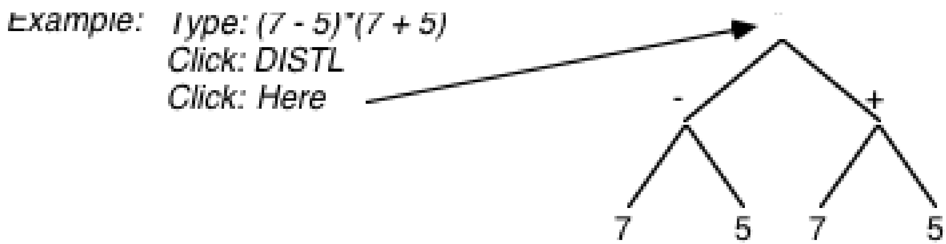


Clicking DISTL and the top of this tree changes 3*(4 + -9) into (3 * 4) + (3 * –9). COLLECTL will undo the effect of DISTL, as shown below.

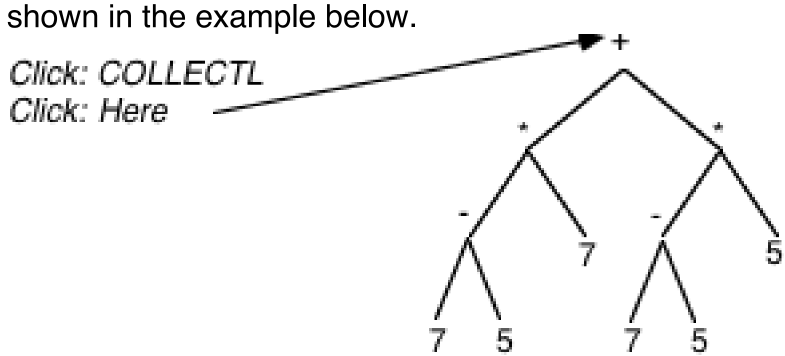


Clicking COLLECTL and the top of this tree changes 3*4 + 3*-9 into 3*(4 + -9).

The letters *a, b,* and *c* do not need to stand for numbers in the general statements of distributing and collecting.



Here, (7-5) is distributed over (7 + 5), giving (7 - 5)*7 + (7 - 5)*5. COLLECTL will undo the effect of DISTL, as shown in the example below.



The expression (7-5) appears on the left in both products within the sum. Thus, COLLECTL will change the expression into (7 - 5)*(7 + 5).

Name __________

Exercises

When you use EXPRESSIONS with these exercises, be sure to select **Always Show Pars** and **Button Group 2** from the **Options** menu.

In each of the exercises below, write the list of buttons and operation signs that you would click to change the first expression into the second. Use the format you saw in Part I. *Do not use EXPRESSIONS with an exercise until you have planned your solution.* Here are the buttons you can use:

ASSOCL	ASSOCR
DISTL	DISTR
COLLECTL	COLLECTR
COMMUTE	SIMP

<u>Start with:</u> | <u>Change it to:</u>

5 * (4 + 3) | 3*5 + 4*5

Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________
Expression: ____ Button: ____________

Start with: Change it to:

5 * ((4 + 3) + 2) (5*4) + ((2 + 3)*5)

Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________

Start with: Change it to:

(7 + 3)*(6 + 5) (7*6 + 7*5) + (3*6 + 3*5)

Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________
Expression: ____Button: ____________

Start with: | Change it to:

(6 + -5)*(6 + 5) | (6*6) + (-5*5)

Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________

Start with: | Change it to:

3*(8 + 4) + 9*(4 + 8) | (9 + 3)*(8 + 4)

Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________
Expression: ______________________________ Button: __________

Worksheet 4

Applying Properties of Operations

Do this worksheet *with* EXPRESSIONS. Before beginning, be sure to select **Button Group 2** and **Always Show Pars** from the **Options** menu.

Write each of your steps (as in Worksheet 3) so that, when done, you have a record of the derivation which turns the "Start with" expression into its target expression.

*(You may need to use **Simp** on some subexpressions.)*

Start With	*Change It To*
3*(6/9) + (6/9)*7	10*(6/9)
-5*3 + (2+3)*5	0 + 10
(5 + 9)*(5 + 9)	25 + 2*45 + 81
(6 + -3)*(6 + 3)	6*6 + -3*3

Worksheet 5

Identities

When an expression can be written in two ways without changing its meaning, we say that the two ways of writing it are identical to each other. The statement that says the two ways of writing an expression are actually the same is called an identity. Here are some examples:

(neg y)	=	-1 * y [that is, -y = -1*y]
a - b	=	a + -b
u/v	=	u * (1/v)
(a/b) * (c/d)	=	(a * c)/(b * d)
x	=	1 * x

The identity x = 1*x says that any number is the same as one times itself. The identity a - b = a + -b says that the difference of two numbers is the same as the the first plus the negative of the second.

The identities listed above are built into expressions. To use them, click the ID button, then click the place in the tree that you wish to change with an identity. expressions will figure out which identity, if any, is appropriate.

When you know a lot of identities, you are able to recognize when two expressions are actually the same even though they might look quite different. Also, you are able to use the identities that you know to discover new identities.

Example 1: Are (a*b)/c and a*(b/c) identical expressions?

Click: SHOWPARS (if this button is showing)

Type: (a*b)/c

Click: ID

Click: The division sign in (a*b)/c
The expression is changed into (a*b)*(1/c). (See the list given above.)

Click: ASSOCL

Click: The top multiplication sign
The expression is changed from (a*b)*(1/c) to a*(b*(1/c))

Click: ID

Click: The multiplication sign in b*(1/c)
The expression b*(1/c) is changed to b/c. (See the list given above. ID works both ways.)

We have shown that (a*b)/c is identical to a*(b/c). The equation (a*b)/c = a*(b/c) is an identity.

Example 2: Does multiplication distribute over subtraction? That is, is it true that a*(b-c) equals a*b - a*c?

Click: SHOWPARS (if this button is showing)

Type: a*(b - c)

Click: ID

Click: The minus sign between b and c.
The expression b - c is changed into b + -c.

Click: DISTL

Click: The times sign of a*(b + -c).
The expression a*(b + -c) is changed into a*b + a*-c

Click: ID

Click: neg
-c (neg c) is changed into -1*c

Click: ASSOCR

Click: The top multiplication sign of a*(-1*c)
The expression is regrouped into (a*-1)*c

Click: COMMUTE

Click: The multiplication sign of a*-1
The expression is changed to -1*a

Click: ASSOCL

Click: The top multiplication sign of (-1*a)*c
The expression is regrouped into -1*(a*c)

Click: ID

Click: The top multiplication sign in -1*(a*c)
The expression is changed into -(a*c)

Click: ID

Click: The addition sign in a*b + -(a*c)
The expression is changed into a*b - (a*c)) (See the list given above. ID works both ways.)

We started with a*(b - c) and ended with a*b - a*c. This means that we have shown that the expressions a*(b - c) and a*b - a*c are identical. So, the equation

a*(b-c) = a*b - a*c

is an identity.

Exercises

I. Start with one of each pair and use any of **ASSOCL**, **ASSOCR**, **COMMUTE**, **DISTL**, **DISTR**, **COLLECTL**, **COLLECTR**, **SIMP**, and **ID** to change it to the other. Refer to the list of identities on page 1 when using **ID**.

Write each of your steps (as in Worksheet 3) so that you have a record of how you transformed the given expression into a target expression.

1.	(z - q)*u	z*u - q*u
2.	r*(s/t)	(r*s)/t
3.	-(p + q)	-p + -q
4.	(a + b)/c	a/c + b/c
5.	6x + x	7x
6.	5x - x	4x
7.	x + x	2x
8.	(u + v)(u - v)	u*u - v*v

You will need to use SIMP on this one to get rid of some terms.

II. The following equations are not identities. For each pair of expressions:

a. Tell what property would have to be violated to change one expression into the other.

b. Replace the letters with numbers, then evaluate the expressions to show that the two expressions do not always evaluate to the same number.

1. 2(x + 5)	2x + 5
2. (x - y) + z	x - (y + z)
3. (u/v)*w	u/(v*w)
4. (n-m)*x	x*(m-n)
4. (r/s)/t	r/(s/t)
5. q+(e/f)	(q+e)/f