Order of operations

From Wikipedia, the free encyclopedia

In mathematics and computer programming, the **order of operations** (or **operator precedence**) is a collection of rules that reflect conventions about which procedures to perform first in order to evaluate a given mathematical expression.

For example, in mathematics and most computer languages, multiplication is granted a higher precedence than addition, and it has been this way since the introduction of modern algebraic notation.[1][2] Thus, the expression $2 + 3 \times 4$ is interpreted to have the value $2 + (3 \times 4) = 14$, not $(2 + 3) \times 4 = 20$. With the introduction of exponents in the 16th and 17th centuries, they were given precedence over both addition and multiplication and could be placed only as a superscript to the right of their base.[1] Thus $3 + 5^2 = 28$ and $3 \times 5^2 = 75$.

These conventions exist to eliminate ambiguity while allowing notation to be as brief as possible. Where it is desired to override the precedence conventions, or even simply to emphasize them, parentheses ( ) (sometimes replaced by brackets [ ] or braces { } for readability) can indicate an alternate order or reinforce the default order to avoid confusion. For example, $(2 + 3) \times 4 = 20$ forces addition to precede multiplication, and $(3 + 5)^2 = 64$ forces addition to precede exponentiation.

Contents

Definition[edit]

The order of operations used throughout mathematics, science, technology and many computer programming languages is expressed here:[1]

1. exponents and roots
2. multiplication and division
3. addition and subtraction

This means that if a mathematical expression is preceded by one binary operator and followed by another, the operator higher on the list should be applied first.

The commutative and associative laws of addition and multiplication allow adding terms in any order, and multiplying factors in any order—but mixed operations must obey the standard order of operations.

In some contexts, it is helpful to replace a division by multiplication by the reciprocal (multiplicative inverse) and a subtraction by addition of the opposite (additive inverse). For example, in computer algebra, this allows manipulating fewer binary operations and makes it easier to use commutativity and associativity when simplifying large expressions – for more details, see Computer algebra § Simplification. Thus $3 \div 4 = 3 \times \frac{1}{4}$; in other words, the quotient of 3 and 4 equals the product of 3 and $\frac{1}{4}$. Also $3 - 4 = 3 + (-4)$; in other words the difference of 3 and 4 equals the sum of 3 and −4. Thus, 1 − 3 + 7 can be thought of as the sum of 1 + (−3) + 7, and the three summands may be added in any order, in all cases giving 5 as the result.

The root symbol √ requires a symbol of grouping around the radicand. The usual symbol of grouping is a bar (called vinculum) over the radicand. Other functions use parentheses around the input to avoid ambiguity. The parentheses are sometimes omitted if the input is a monomial. Thus, sin 3x = sin(3x), but sin x + y = sin(x) + y, because x + y is not a monomial.[1] Some calculators and programming languages require parentheses around function inputs, some do not.

Symbols of grouping can be used to override the usual order of operations.[1] Grouped symbols can be treated as a single expression.[1] Symbols of grouping can be removed using the associative and distributive laws, also they can be removed if the expression inside the symbol of grouping is sufficiently simplified so no ambiguity results from their removal.

Examples[edit]

A horizontal fractional line also acts as a symbol of grouping:

For ease in reading, other grouping symbols, such as curly braces { } or square brackets [ ], are often used along with parentheses ( ). For example:

Exceptions[edit]

Unary minus sign[edit]

There are differing conventions concerning the unary operator − (usually read "minus"). In written or printed mathematics, the expression -3^2 is interpreted to mean $0 - (3^2) = -9$,[1][3]

Some applications and programming languages, notably Microsoft Excel (and other spreadsheet applications) and the programming language bc, unary operators have a higher priority than binary operators, that is, the unary minus has higher precedence than exponentiation, so in those languages -3^2 will be interpreted as $(-3)^2 = 9$.[4] This does not apply to the binary minus operator −; for example while the formulas =-2^2 and =0+-2^2 return 4 in Microsoft Excel, the formula =0-2^2 returns −4. In cases where there is the possibility that the notation might be misinterpreted, a binary minus operation can be enforced by explicitly specifying a leading 0 (as in 0-2^2 instead of just -2^2), or parentheses can be used to clarify the intended meaning.

Mixed division and multiplication[edit]

Similarly, there can be ambiguity in the use of the slash symbol / in expressions such as 1/2*x*.[5] If one rewrites this expression as 1 ÷ 2*x* and then interprets the division symbol as indicating multiplication by the reciprocal, this becomes:

$1 \div 2 \times x = 1 \times ½ \times x = ½ \times x$.

With this interpretation 1 ÷ 2*x* is equal to (1 ÷ 2)*x*.[1][6] However, in some of the academic literature, multiplication denoted by juxtaposition (also known as implied multiplication) is interpreted as having higher precedence than division, so that 1 ÷ 2*x* equals 1 ÷ (2*x*), not (1 ÷ 2)*x*.

For example, the manuscript submission instructions for the *Physical Review* journals state that multiplication is of higher precedence than division with a slash,[7] and this is also the convention observed in prominent physics textbooks such as the *Course of Theoretical Physics* by Landau and Lifshitz and the *Feynman Lectures on Physics*.[a]

Mnemonics[edit]

Mnemonics are often used to help students remember the rules, involving the first letters of words representing various operations. Different mnemonics are in use in different countries.[8][9][10]

- In the United States, the acronym ***PEMDAS*** is common. It stands for *P*arentheses, *E*xponents, *M*ultiplication/*D*ivision, *A*ddition/*S*ubtraction. PEMDAS is often expanded to the mnemonic "**Please Excuse My Dear Aunt Sally**".[5]
- Canada and New Zealand use ***BEDMAS***, standing for *B*rackets, *E*xponents, *D*ivision/*M*ultiplication, *A*ddition/*S*ubtraction.
- Most common in the UK, India, Bangladesh and Australia[11] and some other English-speaking countries are ***BODMAS*** meaning *B*rackets, *O*rder, *D*ivision/*M*ultiplication, *A*ddition/*S*ubtraction. Nigeria and some other West African countries also use BODMAS. Similarly in the UK, ***BIDMAS*** is used, standing for *B*rackets, *I*ndices, *D*ivision/*M*ultiplication, *A*ddition/*S*ubtraction.

These mnemonics may be misleading when written this way.[5] For example, misinterpreting any of the above rules to mean "addition first, subtraction afterward" would incorrectly evaluate the expression[5]

10 − 3 + 2.

The correct value is 9 (and not 5, as if the addition would be carried out first and the result used with the subtraction afterwards).

Special cases[edit]

Serial exponentiation[edit]

If exponentiation is indicated by stacked symbols, the usual rule is to work from the top down, because exponentiation is right-associative in mathematics thus:[1][12]

$a^{b^c} = a^{(b^c)}$

which typically is not equal to $(a^b)^c$.

However, some computer systems may resolve the ambiguous expression differently.[13] For example, Microsoft Excel evaluates *a*^*b*^*c* as $(a^b)^c$, which is opposite of normally accepted convention of top-down order of execution for exponentiation. Thus 4^3^2 is evaluated to 4,096 instead of 262,144.

Another difference in Microsoft Excel is -*a*^*b* which is evaluated as (-*a*)^*b* instead of -(*a*^*b*). For compatibility, the same behavior is observed on LibreOffice. The computational programming language MATLAB is another example of a computer system resolving the stacked exponentiation in the non-standard way.

Serial division[edit]

A similar ambiguity exists in the case of serial division, for example, the expression 10 ÷ 5 ÷ 2 can either be interpreted as

10 ÷ (5 ÷ 2) = 4

or as

(10 ÷ 5) ÷ 2 = 1

The left-to-right operation convention would resolve the ambiguity in favor of the last expression. Further, the mathematical habit of combining factors and representing division as multiplication by a reciprocal both greatly reduce the frequency of ambiguous division. However, when two long expressions are combined by division, the correct order of operations can be lost in the notation.

Calculators[edit]

Main article: Calculator input methods

Different calculators follow different orders of operations. Many simple calculators without a stack implement chain input working left to right without any priority given to different operators, for example typing

1 + 2 × 3 yields 9,

while more sophisticated calculators will use a more standard priority, for example typing

1 + 2 × 3 yields 7.

The *Microsoft Calculator* program uses the former in its standard view and the latter in its scientific and programmer views.

Chain input expects two operands and an operator. When the next operator is pressed, the expression is immediately evaluated and the answer becomes the left hand of the next operator. Advanced calculators allow entry of the whole expression, grouped as necessary, and evaluates only when the user uses the equals sign.

Calculators may associate exponents to the left or to the right depending on the model or the evaluation mode. For example, the expression *a*^*b*^*c* is interpreted as $a^{(b^c)}$ on the TI-92 and the TI-30XS MultiView in "Mathprint mode", whereas it is interpreted as $(a^b)^c$ on the TI-30XII and the TI-30XS MultiView in "Classic mode".

An expression like 1/2*x* is interpreted as 1/(2*x*) by TI-82, but as (1/2)*x* by TI-83 and every other TI calculator released since 1996,[14] as well as by all Hewlett-Packard calculators with algebraic notation. While the first interpretation may be expected by some users, only the latter is in agreement with the standard rule that multiplication and division are of equal precedence,[15][16] so 1/2*x* is read one divided by two and the answer multiplied by *x*.

When the user is unsure how a calculator will interpret an expression, it is a good idea to use parentheses so there is no ambiguity.

Calculators that utilize reverse Polish notation (RPN), also known as postfix notation, use a stack to enter formulas without the need for parentheses.[5]

Programming languages[edit]

Some programming languages use precedence levels that conform to the order commonly used in mathematics,[13] though others, such as APL, Smalltalk or Occam, have no operator precedence rules (in APL, evaluation is strictly right to left; in Smalltalk and Occam, it is strictly left to right).

In addition, because many operators are not associative, the order within any single level is usually defined by grouping left to right so that 16/4/4 is interpreted as (16/4)/4 = 1 rather than 16/(4/4) = 16; such operators are perhaps misleadingly referred to as "left associative". Exceptions exist; for example, languages with operators corresponding to the cons operation on lists usually make them group right to left ("right associative"), e.g. in Haskell, 1:2:3:4:[] == 1:(2:(3:(4:[]))) == [1,2,3,4].

The logical bitwise operators in C (and all programming languages that borrow precedence rules from C, for example, C++, Perl and PHP) have a precedence level that the creator of the C language considered unsatisfactory.[17] However, many programmers have become accustomed to this order. The relative precedence levels of operators found in many C-style languages are as follows:

1	() [] -> . ::	Function call, scope, array/member access
2	! ~ - + * & sizeof *type cast* ++ --	(most) unary operators, sizeof and type casts (right to left)
3	* / % MOD	Multiplication, division, modulo
4	+ -	Addition and subtraction
5	<< >>	Bitwise shift left and right
6	< <= > >=	Comparisons: less-than and greater-than
7	== !=	Comparisons: equal and not equal
8	&	Bitwise AND
9	^	Bitwise exclusive OR (XOR)
10	|	Bitwise inclusive (normal) OR
11	&&	Logical AND
12	||	Logical OR
13	? :	Conditional expression (ternary)
14	= += -= *= /= %= &= |= ^= <<= >>=	Assignment operators (right to left)
15	,	Comma operator

Examples: (Note: in the examples below, '≡' is used to mean "is equivalent to", and not to be interpreted as an actual assignment operator used as part of the example expression.)

- !A + !B ≡ (!A) + (!B)
- ++A + !B ≡ (++A) + (!B)
- A + B * C ≡ A + (B * C)
- A || B && C ≡ A || (B && C)
- A && B == C ≡ A && (B == C)
- A & B == C ≡ A & (B == C)

Source-to-source compilers that compile to multiple languages need to explicitly deal with the issue of different order of operations across languages. Haxe for example standardizes the order and enforces it by inserting brackets where it is appropriate.[18]

The accuracy of software developer knowledge about binary operator precedence has been found to closely follow their frequency of occurrence in source code.[19]

See also[edit]

- Associativity
- Common operator notation (for a more formal description)
- Commutativity
- Distributivity
- Hyperoperation
- Operator (programming)
- Operator associativity
- Operator overloading
- Operator precedence in C and C++
- Polish notation
- Reverse Polish notation

Notes[edit]

1. **^** For example, the third edition of *Mechanics* by Landau and Lifshitz contains expressions such as $hP_z/2\pi$ (p. 22), and the first volume of the *Feynman Lectures* contains expressions such as $1/2\sqrt{N}$ (p. 6–7). In both books these expressions are written with the convention that the solidus is evaluated last.

References[edit]

1. ^ Jump up to:*a b c d e f g h i* *Bronstein, Ilja Nikolaevič; Semendjajew, Konstantin Adolfovič (1987) [1945]. "2.4.1.1.". In Grosche, Günter; Ziegler, Viktor; Ziegler, Dorothea. Taschenbuch der Mathematik(in German).* ***1****. Translated by Ziegler, Viktor. Weiß, Jürgen (23 ed.). Thun and Frankfurt am Main: Verlag Harri Deutsch (and B. G. Teubner Verlagsgesellschaft, Leipzig). pp. 115–120. ISBN 3-87144-492-8.*
2. **^** *"Ask Dr. Math". Math Forum. 22 November 2000. Retrieved 5 March 2012.*
3. **^** *Allen R. Angel. Elementary Algebra for College Students (8 ed.). Chapter 1, Section 9, Objective 3.*
4. **^** *"Formula Returns Unexpected Positive Value". Support.microsoft.com. 15 August 2005. Retrieved 5 March 2012.*
5. ^ Jump up to:*a b c d e* *Ball, John A. (1978). Algorithms for RPN calculators (1 ed.). Cambridge, Massachusetts, USA: Wiley-Interscience, John Wiley & Sons, Inc. p. 31. ISBN 0-471-03070-8.*
6. **^** "division and multiplication have the same priority", http://www.mathcentre.ac.uk/resources/uploaded/mc-ty-rules-2009-1.pdf
7. **^** *"Physical Review Style and Notation Guide" (PDF). American Physical Society. Section IV–E–2–e. Retrieved 5 August 2012.*
8. **^** http://www.mathcentre.ac.uk/resources/uploaded/mc-ty-rules-2009-1.pdf
9. **^** *"Please Excuse My Dear Aunt Sally (PEMDAS)--Forever!". Education Week - Coach G's Teaching Tips. 1 January 2011.*
10. **^** *"What is PEMDAS? - Definition, Rule & Examples". Study.com.*
11. **^**http://syllabus.bos.nsw.edu.au/assets/global/files/maths_s3_sampleu1.doc
12. **^** *Olver, Frank W. J.; Lozier, Daniel W.; Boisvert, Ronald F.; Clark, Charles W., eds. (2010). NIST Handbook of Mathematical Functions. National Institute of Standards and Technology (NIST), U.S. Department of Commerce, Cambridge University Press. ISBN 978-0-521-19225-5. MR 2723248.*[1]
13. ^ Jump up to:*a b* Exponentiation Associativity and Standard Math NotationCodeplea. 23 Aug 2016. Retrieved 20 Sep 2016.
14. **^** *"Implied Multiplication Versus Explicit Multiplication on TI Graphing Calculators". Texas Instruments. 2011-01-16. 11773. Archived from the original on 2016-04-17. Retrieved 2015-08-24.*
15. **^** *Zachary, Joseph L. (1997). "Introduction to scientific programming - Computational problem solving using Maple and C - Operator precedence worksheet". Retrieved 2015-08-25.*

16. **^** *Zachary, Joseph L. (1997). "Introduction to scientific programming - Computational problem solving using Mathematica and C - Operator precedence notebook". Retrieved 2015-08-25.*

17. **^** Dennis M. Ritchie: The Development of the C Language. In History of Programming Languages, 2nd ed., ACM Press 1996.

18. **^** 6÷2(1+2)=? Andy Li's Blog. 2 May 2011. Retrieved 31 December 2012.

19. **^** "Developer beliefs about binary operator precedence" Derek M. Jones, CVu 18(4):14–21