University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

MAT Exam Expository Papers

Math in the Middle Institute Partnership

7-2007

Order of Operations and RPN

Greg Vanderbeek

University of Nebraska - Lincoln

Follow this and additional works at: https://digitalcommons.unl.edu/mathmidexppap

Part of the Science and Mathematics Education Commons

Vanderbeek, Greg, "Order of Operations and RPN" (2007). *MAT Exam Expository Papers*. 46.
https://digitalcommons.unl.edu/mathmidexppap/46

This Article is brought to you for free and open access by the Math in the Middle Institute Partnership at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in MAT Exam Expository Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

Master of Arts in Teaching (MAT) Masters Exam

Greg Vanderbeek

In partial fulfillment of the requirements for the Master of Arts in Teaching with a Specialization in the Teaching of Middle Level Mathematics in the Department of Mathematics.
Dr. Gordon Woodward, Advisor

July 2007

Order of Operations and RPN
Expository Paper

Greg Vanderbeek

July 2007

History of the order of operations

There is not a wealth of information regarding the history of the notations and procedures associated with what is now called the “order of operations”. There is evidence that some agreed upon order existed from the beginning of mathematical study. The grammar used in the earliest mathematical writings, before mathematical notation existed, supports the notion of computational order (Peterson, 2000). It is clear that one person did not invent the rules but rather current practices have grown gradually over several centuries and are still evolving.

Prompted by the need for such conventions, the basic rule that multiplication has precedence over addition appears to have arisen without much disagreement as algebraic notation was being developed in the 1600s. Authors of mathematical research books written during this era used various systems of symbols, forcing them to explain their use of conventions. For example, in 1631, Englishman Thomas Harriot wrote about the multiplication of fractions using the notation $\left.\begin{array}{c}\underline{ac}\\ b\\ \underline{b}\end{array}\right| = \frac{acb}{b} = ac$

and for complex fractions he wrote $\frac{\frac{aaa}{b}}{d} = \frac{aaa}{bd}$ (Cajori, 1929).

About the same time, fellow Englishman William Oughtred was placing an unusual emphasis upon the use of mathematical symbols. Oughtred’s beliefs that mathematical notation should replace verbal explanations of calculations are summarized by the following passage:

> “…Which Treatise being not written in the usual synththetical manner, nor with verbous expressions, but in the inventive way of Analitice, and with symbols or notes of things instead of words, seemed unto many very hard; though indeed it was but their owne diffidence, being scared by the newness of the delivery; and not any difficulty in the thing it selfe.” (Cajori, 1929)

Despite the abundance of unique symbol sets for mathematical operations, during this time not much was being written about the development of a *universal* set of rules to create order for the operations.

The continued development of specific rules for establishing computational order did not proceed without some growing pains. In particular, math historian Florian Cajori (1859-1930) pointed out that there was disagreement as to whether multiplication should have precedence over division, or whether they should be treated equally. In addition, Cajori (1929) quoted many writers for whom, in the special case of a factorial-like expression such as **n(n-1)(n-2)**, the multiplication sign seems to serve as a grouping symbol (i.e. parentheses or brackets) allowing them to write **n × n - 1 × n – 2** instead. Cajori pointed out that this was an exception to an established rule by which **nn-1n-2** would be interpreted as the quadratic expression $\mathbf{n^2 - n - 2}$ **.**

The following is a brief description of some early historical events that contributed to the development of our current uses of mathematical notation and computational practices.

In 1892 in *Mental Arithmetic,* M. A. Bailey advises avoiding expressions containing both ÷ and ×.

In 1898 in *Text-Book of Algebra* by G. E. Fisher and I. J. Schwatt, $a \div b \times b$ is interpreted as $(a \div b) \times b$.

In 1907 in *High School Algebra, Elementary Course* by Slaught and Lennes, it is recommended that multiplications in any order be performed first, then divisions as they occur from left to right. So $a \div b \times b = a \div (b \times b)$.

In 1910 in *First Course of Algebra* by Hawkes, Luby, and Touton, the authors agree with Fisher and Schwatt when they write that ÷ and × should be taken in the order in which they occur.

In 1912, *First Year Algebra* by Webster Wells and Walter W. Hart states: "Indicated operations are to be performed in the following order: first, all multiplications and divisions in their order from left to right; then all additions and subtractions from left to right." So $c + a \div b \times b = c + ((a \div b) \times b)$.

In 1913, *Second Course in Algebra*, authors Webster Wells and Walter W. Hart are clarifying what they wrote in their 1912 text: "*Order of Operations.* In a sequence of the fundamental operations on numbers, it is agreed that operations under radical signs or within symbols of grouping shall be performed before all others; that, otherwise, all multiplications and divisions shall be performed first, proceeding from left to right, and afterwards all additions and subtractions, proceeding again from left to right."

In 1917, "The Report of the Committee on the Teaching of Arithmetic in Public Schools," *Mathematical Gazette* 8, p. 238, recommended the use of brackets to avoid ambiguity in such cases.

In *A History of Mathematical Notations* (1928-1929) Florian Cajori writes (vol. 1, page 274), "If an arithmetical or algebraical term contains ÷ and ×, there is at present no agreement as to which sign shall be used first." (Miller, 2006).

Contemporary contributions to the effort of formalizing rigid rules for computational order came primarily from two sources. One was through the development of computer technology. Programming languages used by computers required a set of strict rules for interpreting mathematical expressions. Prior to these technological developments, it was acceptable to simply recognize some forms, like x/yz as ambiguous and ignore them. A second source was the result of widespread mathematics textbook use. Peterson (2000) described the role textbooks played when he wrote "I suspect that the concept, and especially the term "order of operations" and the PEMDAS/BEMDAS mnemonics, were formalized only in this century, or at least in the late 1800s, with the growth of the textbook industry. I think it has been more important to text authors than to mathematicians, who have just informally agreed without needing to state anything officially."

In summary, the rules can be divided into two categories: the natural rules (such as precedence of exponential over multiplicative over additive operations, and the meaning of parentheses), and the artificial rules (left-to-right evaluation, equal precedence for multiplication and division, and so on). The former were present from the beginning of the notation, and probably existed already, though in a somewhat different form, in the geometric and verbal modes of expression that preceded algebraic symbolism. The latter, not having any absolute reason for their acceptance, have had to be gradually agreed upon through usage, and continue to evolve (Peterson, 2000). Today's situation is not all that different from the 1600s.

Limitations of common mnemonics used in beginning algebra

The most common mnemonics used in beginning algebra are "PEMDAS" (Parentheses, Exponents, Multiplication, Division, Addition, Subtraction) and "BEDMAS" (Brackets, Exponents, Multiplication, Division, Addition, Subtraction). These serve as memorization aids and are typically introduced around the 6th grade. While much time and effort are expended by teachers and students learning the order of operations, there are many reasons to question their effectiveness. I will discuss a few of them.

First, when students enter into the middle grades they begin building a fundamental understanding of computation that would survive without the use of the PEMDAS mnemonic. These students learn that multiplication (of integers) is defined as repeated addition. Also, students are gaining an understanding that division is defined as multiplying by the reciprocal and subtraction is defined as adding the negative in combination with the commutative and associative laws of addition and multiplication. Therefore, they are able to compute 4+3*2 by changing it to 4+2+2+2 or 4+3+3. Students are able to make connections between the expressions, finding that the correct answer is produced from the original expression only when the multiplication is done first.

Students are able to simplify calculations by reducing an expression containing all four common binary operators to only addition and multiplication. For example, Wu (2002) argues that middle grades students can convert an expression such as $(9-5)+2$ by changing it to

$(9 + -5) \times \frac{1}{2}$, making rules for division and subtraction redundant and unnecessary. Although I believe it is extremely important for students to have an understanding of the relationship between division and multiplication by the reciprocal for the division of fractions and some mental math strategies, I have found my students are resistant to this practice. Division will never be eliminated therefore students must be prepared to deal with it.

Furthermore, the grouping functions (which are listed as parentheses, extended radical signs, fractions with expressions in the numerator or denominator, and absolute values) allow you to bypass these conventions, anyway. For these reasons, a student should not be restricted by only performing calculations from left to right. The 4-step format (parentheses, exponents, multiplication and division from left to right, addition and subtraction from left to right) is confusing and inefficient. Instead, as H. Wu (2004) suggests, we should teach a much simpler form: exponents first, followed by multiplications, then additions including a thorough explanation of the fundamental properties underlying the process.

Another reason to de-emphasize teaching the order of operations via the PEMDAS mnemonic is because it lures teachers into teaching ambiguous computations such as $4 + 5 \times 6 + 10$. These computations should not be done just for the purpose of teaching the order of operations because they are seldom encountered beyond a math classroom. In a realistic context, this problem would be done by computing the subexpressions in the proper order or parentheses would be used to clarify order. A problem like this is designed to trick a person and encourages students to rely on a calculator to compute the correct answer.

Finally, too much time is spent teaching terminology and conventions and not enough time is spent teaching mathematics. As described by Wu (2004), “too much importance is attached to the name of a concept or the literal execution of a procedure without attempting to learn the mathematical significance of the concept itself or the reason or spirit behind the procedure." Wu (2004) explains further when he writes "In the context of mathematics learning, knowing that a number is called "the additive inverse" of another number is of zero value compared with knowing how to make use of the uniqueness of additive inverse to explain, for example, why –(r+s) = -r – s and (-r)(-s) = +rs for all rational numbers r and s."

<u>RPN</u>

What is RPN and how does it work?

In the 1920's, Polish mathematician Jan Lukasiewicz developed a formal logic system which allowed mathematical expressions to be specified without parentheses by placing the operators before (prefix notation) or after (postfix notation) the operands. This logic system was later at the root of the idea of the recursive stack, a last-in, first-out computer memory store invented by Australian philosopher and computer scientist Charles Hamblin in the mid-1950s. A similar concept underlies a form of postfix notation which was first implemented in Hewlett Packard calculators. HP adjusted the postfix notation for a calculator keyboard, added a stack to hold the operands and functions to reorder the stack. HP dubbed the result Reverse Polish Notation (RPN) in honor of Lukasiewicz.

In Reverse Polish notation the operators follow their operands; for instance, to add three and four one would write "3 4 +" rather than "3 + 4". If there are multiple operations, the operator is given immediately after its second operand; so the expression written "3 − 4 + 5" in conventional infix notation would be written "3 4 − 5 +" in RPN: first subtract 4 from 3, then add 5 to that. RPN does not require parentheses to force order unlike infix notation. For example, the infix expression 6 × (9 + 3) becomes 6 9 3 + × in RPN.

An Example of a calculation done in RPN

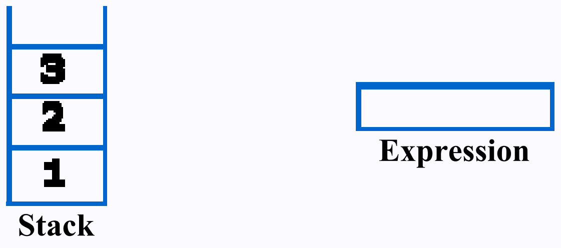
The following example illustrates the process an RPN interpreter executes to calculate expressions. The memory stores used by RPN interpreters are called stacks. Operands are pushed onto the last in, first out (LIFO) stack, and when an operation is performed, its operands are popped from a stack and its result pushed back on.

Postfix Evaluation:
In normal algebra we use the infix notation like a + b * c. The corresponding postfix notation is a b c * +. The algorithm for the conversion is as follows:

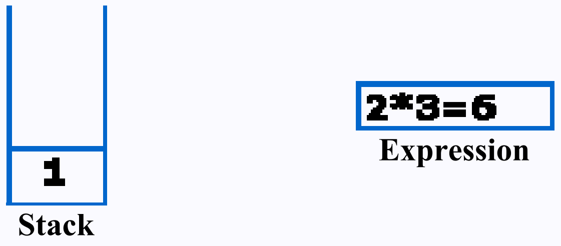
- Scan the postfix string from left to right.
- If the scanned character is an operand, add it to the stack.
- If the scanned character is an operator then the top most element of the stack is popped and stored in a variable *temp*. Next, the stacked is popped a second time and the result stored in the variable *topStack*. The popped operands are used to evaluate the expression *topStack*(Operator)*temp*. The value of this expression is then pushed back onto the stack. Repeat this step until all the characters are scanned.
- After all characters are scanned, we will have only one element in the stack. The remaining element is the final result of the calculation.

Example:
Postfix (RPN) String: 1 2 3 * + 4 -

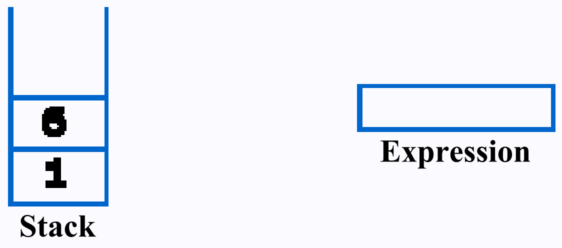
Initially the Stack is empty. Now, the first three characters scanned are 1, 2, and 3, which are operands. Thus they will be pushed into the stack in that order.



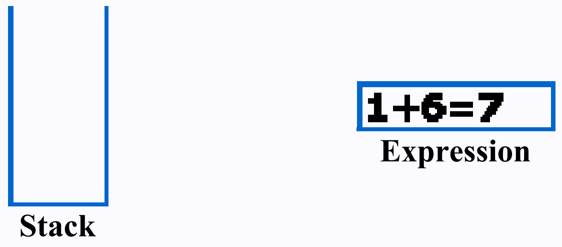
Next character scanned is "*", which is an operator. Thus, we pop the top two elements from the stack and perform the "*" operation with the two operands.



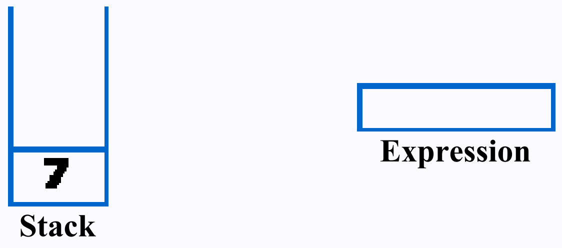
The value of the expression (2*3) that has been evaluated (6) is pushed into the stack.



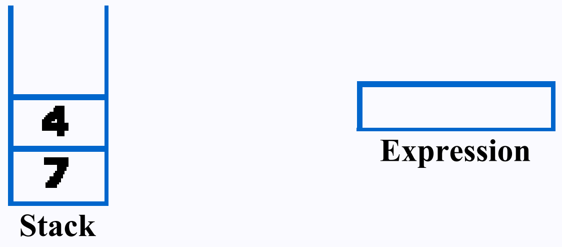
Next character scanned is "+", which is an operator. Thus, we pop the top two elements from the stack and perform the "+" operation with the two operands.



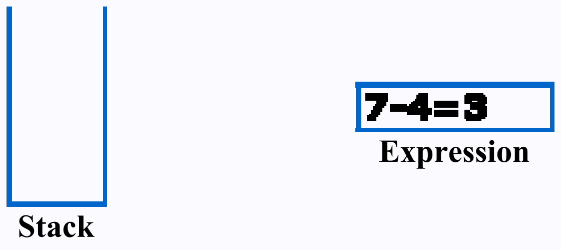
The value of the expression (1+6) that has been evaluated (7) is pushed into the stack.



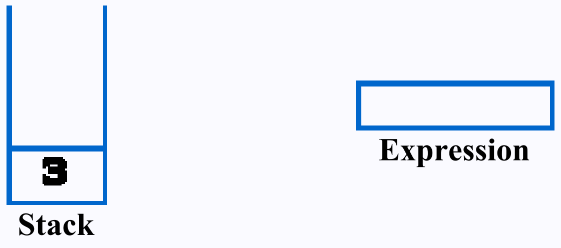
Next character scanned is "4", which is added to the stack.



Next character scanned is "-", which is an operator. Thus, we pop the top two elements from the stack and perform the "-" operation with the two operands.



The value of the expression (7-4) that has been evaluated (3) is pushed into the stack.



Now, since all the characters are scanned, the remaining element in the stack (there will be only one element in the stack) will be returned.

End result :

- Postfix String : 123*+4-
- Result : 3

(Pillai, 2002)

Why did HP use RPN?

In the years that followed the development of postfix notation, computer scientists realized that RPN was very efficient for computer use since calculations with postfix notation are done faster, requiring less processor time, than those with infix notation. As a postfix expression is scanned from left to right, operands are placed into the LIFO stack and operators may be immediately applied to the operands at the top of the stack. By contrast, expressions with parentheses and precedence (infix notation) require that operators be delayed until some later point. For example, the expression (75 - (6 + 4) * 3) / 5 requires the interpreter, human or machine, to make decisions about the operands and operators. This occurs only after the entire expression is scanned from left to right which demands extra processing time. Thus, the compilers of most modern computers of this time converted statements to RPN for execution. (In fact, some computer manufacturers designed their computers around postfix notation.)

At the time Hewlett Packard introduced its first calculator using RPN, other pocket calculators typically used a partial algebraic model. That meant they could evaluate simple expressions like 3 + 4 but couldn't handle anything that involved parentheses or order of operations. The technology available didn't allow for full algebraic compilers in pocket calculators.

The development of RPN allowed HP to produce a pocket calculator that could evaluate arbitrary, complex expressions using the available technology. For many, learning a new style of entry was a small price to pay to be able to evaluate these expressions on a calculator. Once the technology to produce algebraic compilers could fit into a pocket calculator, most RPN users had decided that RPN was more efficient and consistent for the user as well as for the calculator. Also, because subexpressions are evaluated as they are entered, entry errors are more obvious with RPN. On an algebraic calculator, omitting an opening parenthesis may not lead to a calculation error until much later when an entire expression has been evaluated.

Another advantage to RPN is consistency between machines. Early algebraic calculator models had differing limits of the complexity of the expressions they could evaluate. For example, TI catalogs from the late 70's listed how many levels of parentheses and pending operations each model could handle. Even today if you begin to use an algebraic calculator, you need to determine just "how algebraic" it really is. Some calculators cannot handle a problem written as $\frac{4+6}{6-3}$ even though they claim to allow expressions to be entered as they are written.

Converting infix notation to RPN

Conversion algorithm

The following is an algorithm (illustrated by a flowchart in Appendix A) for converting an expression in infix notation to one in postfix, or RPN. This algorithm will work on any expression that uses simple binary operators such as $+$, $-$, $\times$, $\div$ and parentheses. The process begins when the infix expression is read as a string of characters. Each character is either added to the postfix output string or placed in temporary storage called a *stack* where it awaits future action. There are four results possible for each character scanned:

Operator: If the character is an operator, and there's nothing else on the operator stack, it is pushed onto the stack. If there is something on the stack, the precedence of the current operator is compared with the one on the stack. If the stack's operator is greater or equal to the current one, the top operator from the stack is popped and added to the postfix string. If its precedence is less, the current operator is pushed on the operator stack and the next character is read from the infix string. The current operator continues to be compared to the stack until one of the following occurs:

1. the end of the stack is reached
2. the top of the stack is a left parenthesis
3. the current operator is compared to one with lower precedence

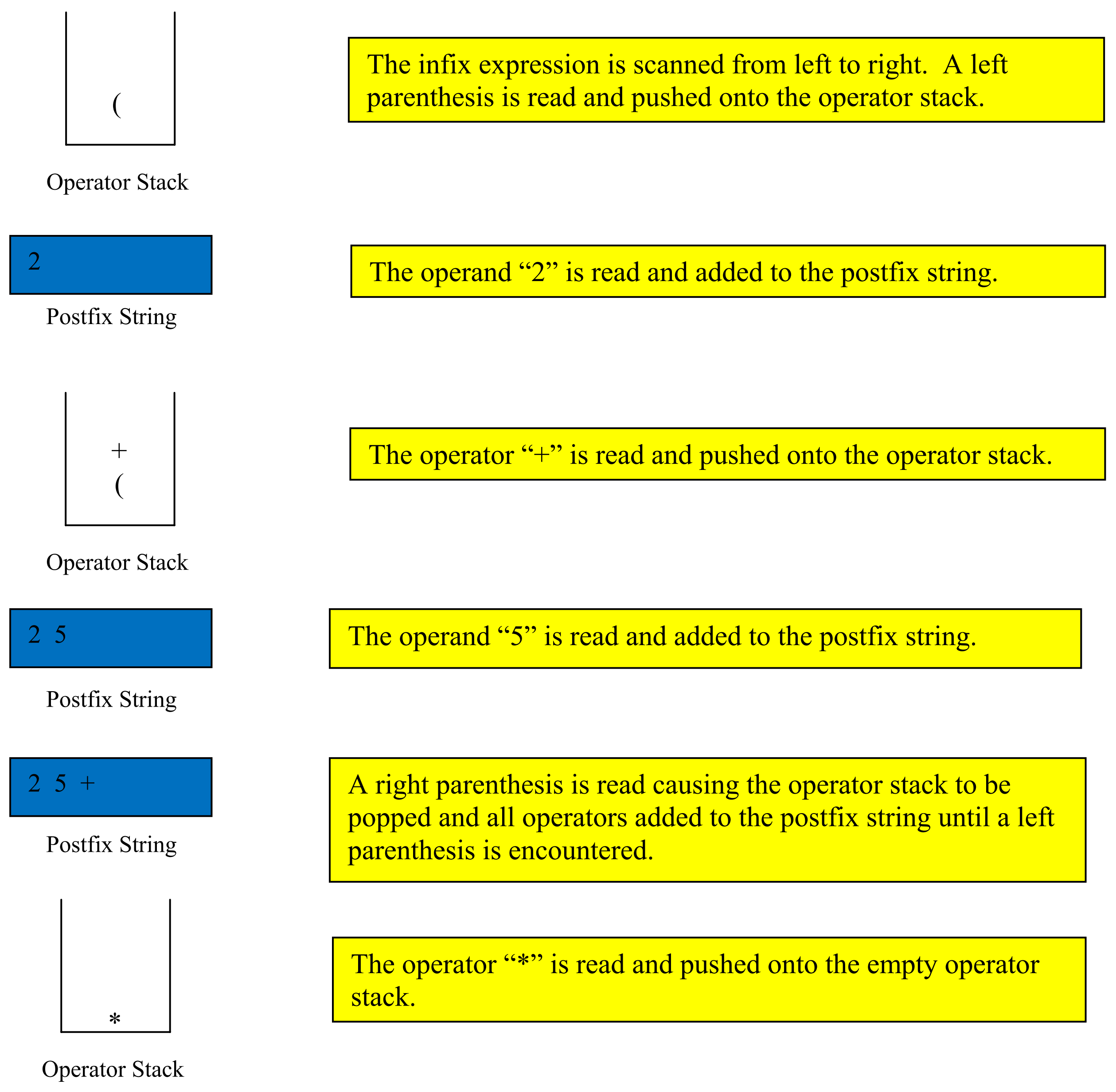
Left Parenthesis: If the character is a left parenthesis it is pushed it onto the operator stack.

Right Parenthesis: If the character is a right parenthesis then all the characters are popped off the stack until a left parenthesis or the end of the stack is encountered.

Operand: If the character is an operand it is added to the postfix string.

When the entire infix string has been scanned, all remaining operators are popped from the operator stack and added to the postfix string. The following diagram illustrates the state of the stacks and postfix string as an expression written in infix notation is converted to RPN.

$infix\ expression: (2 + 5) \cdot 3 + 1$



The infix expression is scanned from left to right. A left parenthesis is read and pushed onto the operator stack.

The operand “2” is read and added to the postfix string.

The operator “+” is read and pushed onto the operator stack.

The operand “5” is read and added to the postfix string.

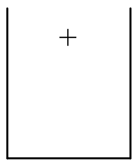
A right parenthesis is read causing the operator stack to be popped and all operators added to the postfix string until a left parenthesis is encountered.

The operator “*” is read and pushed onto the empty operator stack.

2 5 + 3

Postfix String

The operand “3” is read and added to the postfix string.



Operator Stack

The operator “+” is read and compared to the top operator in the stack. Since “*” has a higher precedence than “+” the “*” is popped and added to the postfix string before the “+” is pushed onto the operator stack..

2 5 + 3 *

Postfix String

2 5 + 3 * 1

Postfix String

The operand “1” is read and added to the postfix string.

2 5 + 3 * 1 +

Postfix String

The end of the infix string has been reached. The operators in the stack are popped and added to the postfix string.

Here are more examples of converting infix notation to RPN.

Standard Algebraic Expression (Infix)	**Reverse Polish Notation (Postfix)**
8 + 5	**8 5 +**
13 – 7 + 9	**13 7 – 9 +**
3+ 6 × 4	**3 6 4 × +**
(3+ 6) × 4	**3 6 + 4 ×**
8((3 + 4) − 5)	**8 3 4 + 5 − ×**
5 + ((1 + 2) × 4) − 3	**5 1 2 + 4 × + 3 −**
(6 (7 – 2)) ÷ 3 + 9 × 4	**6 7 2 − × 3 ÷ 9 4 × +**

Summary

When teaching the order of operations, the focus should be on fundamental mathematical principles instead of memorized mnemonic devices. The familiar PEMDAS acronym could be reduced to “multiply before you add” if students develop a complete understanding of the relationships among common operators. RPN is a viable teaching tool to help build a solid computational foundation. RPN forces a shift in responsibility for operations order from the person performing the calculation to the person notating the calculation. Since each operation given in RPN creates implied parentheses around the two previous values, skill is required by the user to translate infix to RPN by correctly placing the operators within the operands for the expression to be evaluated as desired. Once an expression is correctly written in RPN, it can be quickly calculated with minimal chance for error. For example, the RPN expression 3 5 + 7 * 8 7 - / is equivalent to ((3 5 +) 7 *) (8 7 –) /. The absence of parentheses with this type of notation reduces the number of characters processed by the interpreter making it more efficient than infix. Since RPN forces subexpression to be calculated at an intermediate level, the user is able to check for errors periodically rather than wait until the entire expression has been evaluated as is done with infix notation. It is my belief that RPN could be used effectively to deepen students’ understanding of computation and could be a valuable tool for teaching computational order.

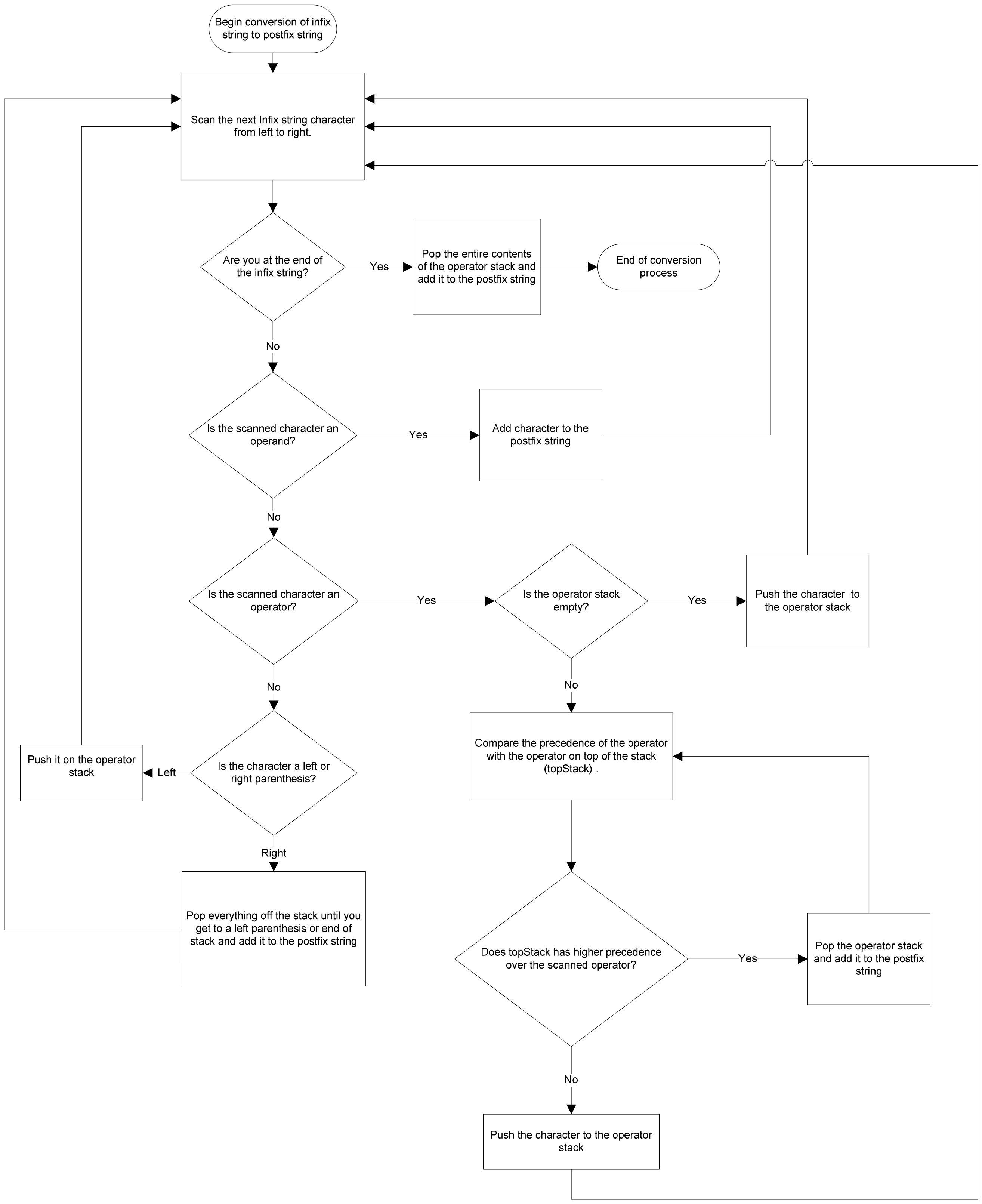
Advantages of RPN

- Reading from left to right, calculations can occur as soon as an operator is read; processing can begin before the entire expression is read.
- Parentheses and brackets are not necessary. The RPN expression 6 2 – 3 + is equivalent to the infix expression (6 – 2) + 3 with no ambiguity.
- RPN allows the user to observe intermediary results as subexpressions are evaluated. This allows the user to check for errors during the calculation process rather than waiting until the end.
- RPN is translated and executed by the same basic algorithm all around the world unlike infix.

Disadvantages of RPN

- Infix notation has been the standard for teaching computational skills for a long time making a switch to postfix difficult and impractical. (However, it is easy for a computer to convert infix notation to postfix).
- Adjacent numbers have to have a space between them, which requires precise handwriting to prevent confusion (for instance, 12 34 + could look a lot like 123 4 +)
- RPN calculators are comparatively expensive and rare compared to an infix calculator. When an RPN calculator is unavailable, frequent users of RPN calculators may find use of infix calculators difficult due to habit.

Appendix A : Conversion Algorithm Flowchart (Infix notation to RPN)


Begin conversion of infix string to postfix string
Scan the next Infix string character from left to right.
Are you at the end of the infix string?
Yes
Pop the entire contents of the operator stack and add it to the postfix string
End of conversion process
No
Is the scanned character an operand?
Yes
Add character to the postfix string
No
Is the scanned character an operator?
Yes
Is the operator stack empty?
Yes
Push the character to the operator stack
No
No
Is the character a left or right parenthesis?
Left
Push it on the operator stack
Right
Pop everything off the stack until you get to a left parenthesis or end of stack and add it to the postfix string
Compare the precedence of the operator with the operator on top of the stack (topStack) .
Does topStack has higher precedence over the scanned operator?
Yes
Pop the operator stack and add it to the postfix string
No
Push the character to the operator stack

References

Cajori, F., PH.D. (1929). *A history of mathematical notations*. Chicago, IL: The Open Court Publishing Company.

Miller, J. (2006). *Earliest Uses of Various Mathematical Symbols*. Retrieved July 2, 2007, from Gulf High School, New Port Richey, FL Web site: http://members.aol.com/jeff570/mathsym.html

Peterson, Dr. (2000) *History of the Order of Operations*. Retrieved July 2, 2007, from The Math Forum: Ask Dr. Math Web site: http://mathforum.org/library/drmath/view/52582.html

Pillai, P. (n.d.). *Infix/Postfix -- convert infix expression to postfix expression*. Retrieved July 5, 2007, from http://scriptasylum.com/tutorials/infix_postfix/infix_postfix.html

Reverse Polish Notation. (n.d.). Retrieved July 2, 2007, from Wikipedia, the free encyclopedia Website: http://en.wikipedia.org/wiki/Reverse_Polish_notation

RPN. (n.d.). Retrieved July 2, 2007, from The Museum of HP Calculators Web site:http://www.hpmuseum.org/rpn.htm

RPN, An Introduction to Reverse Polish Notation. (2007). Retrieved July 2, 2007, from Hewlett Packard Company Web site: http://www.hp.com/calculators/news/rpn

Wu, H. (2004). *"Order of operations" and other oddities in school mathematics*. Retrieved July 3, 2007, from Department of Mathematics, University of California-Berkeley Web site:http://math.berkeley.edu/~wu/