

RPN Perspective

by

**John Kennedy
Mathematics Department
Santa Monica College
1900 Pico Blvd.
Santa Monica, CA 90405**

rkennedy@ix.netcom.com

Except for this comment explaining that it is blank for some deliberate reason, this page is intentionally blank!

RPN Perspective

The following is a reprint of an article published in the PPC Calculator Journal, Volume 9 Number 5, August 1982, pp26-29.

“Here and elsewhere we shall not obtain the best insight into things until we actually see them growing from the beginning.”

- ARISTOTLE -

“Among chosen combinations the most fertile will often be those formed of elements drawn from domains which are far apart.”

- HENRI POINCARÉ -

“A good notation has a subtlety and suggestiveness which at times make it seem like a live teacher.”

- BERTRAND RUSSELL -

“In other words, use of symbolic manipulations does not necessarily give one any technique for solving the problem that was not already present in the intuitive case; it merely makes the existing techniques more flexible, more effective, and more apparent. This is characteristic of the use of symbols.”

- J. BARKLEY ROSSER -

This article is provided for those readers interested in learning the exact origins of the Polish notation. With the introduction of handheld computers employing high level languages it may be the case that **RPN** calculators as we have known them are a dying breed. I certainly hope this is not the case and do not intend this statement as a prediction. Nonetheless, the role that **RPN** has played in the family of calculator/computer languages is changing, and with the introduction of a new machine that could become a significant influence in the **PPC** world, I think it is appropriate to have a background article concerning the Reverse Polish Notation.

This article was actually written three years ago and was to have been an appendix in a book to be written by Richard Nelson and myself called *“Programming Techniques For RPN Calculators.”* However, the **PPC ROM Project** began at that time and we were never able to start work on that book. (We hope anyone who has held the **PPC ROM** manual in their hands will understand why our proposed book has never seen the light of day.)

At the **PPC** Western Calculator conference held in Santa Clara in 1979 I presented a paper titled **RPN HISTORY**. In that paper I cited some references to early applications of the Polish notation on computers. However, it was not possible in that short paper to give a detailed account of exactly what the Polish notation was when it was introduced. The value of that paper was pointing out that while Jan Lukasiewicz (1878-1956) has been properly credited as the originator of the Polish notation, most of the

historical dates given in the early 1950's are either inaccurate or entirely misleading. The Polish notation originated in 1928 (or earlier) long before the first electronic digital computer.

One of the reasons I became interested in the Polish notation had to do with a game I was given about the time I started high school. The game is called **WFF'N PROOF** and is designed to help you think “logically”. I never did learn how to play the game but it has been on my bookshelf for almost 20 years and it uses the exact Polish notation introduced by Lukasiewicz in 1929. I have had the game since 1962 but it was not until 1979 that I realized the intimate connection between my **HP** calculators and **WFF'N PROOF**.

Another reason for my interest in the Polish notation had to do with my first correspondence with John McGechie. John is Professor of Philosophy at Monash University in Australia and is better known by **PPC** members as the editor of **PPCTN**. John and I began corresponding about logic and calculators before he became so involved in **PPC** activities. John had written some articles in the Australian Logic Teachers Journal as well as some excellent logic programs for the **HP-19C**. He understood better than I what the Polish notation had to do with symbolic logic.

I had also written an HP-67 program to play **WFF'N PROOF** and to calculate truth tables. I discovered for myself that in symbolic logic, *well-formed formulas* (hence the abbreviation **WFF** in the above game) are easily recognized when written in Polish notation. John McGechie also provided me with valuable references concerning Jan Lukasiewicz as well as a bibliography on symbolic logic and calculators/computers.

A complete history describing the interconnection between pure symbolic logic and Boolean algebra, electronic switching circuits and modern digital computer design and operation would be a very large task. This article is only an attempt to explain the relationship between mathematical and logical notations. This will not make you a better programmer but you may gain a new perspective of the abbreviation **RPN**. A background in symbolic logic would be helpful but is not required.

Many people know that **RPN** stands for *Reverse Polish Notation*, but few really understand the foundations of this system of notation or know of its significance in the world of calculators and computers. Jan Lukasiewicz was the Polish logician who invented a parenthesis-free notation used to describe logical expressions in the subject generally known as symbolic logic. Because Lukasiewicz's nationality is much easier to remember and pronounce compared to his last name, his system of notation has come to be known as the Polish notation.

A description of his notation was published in two of his works in 1929. One of those works consisted of notes from lectures delivered at Warsaw University in the autumn trimester of the academic year 1928/29. It is significant that in listing the more important new results of his research in mathematical logic, Lukasiewicz chose his parenthesis-free notation as the first item on the list. This idea must have originated in the year 1928 (or earlier) and this date is almost 20 years prior to the completion of the first electronic computer. Thus the origin of the Polish notation was not directly related to mathematical computations or computers.

Symbolic logic is concerned in part with developing techniques to determine whether or not certain sentences follow from certain others. Simple sentences have a truth value within a given context, and

simple sentences can be regarded as the building blocks from which compound sentences can be constructed. In sentential or propositional calculus, simple sentences are combined by the use of logical connectives. The five standard logical connectives are:

“and” “or” “not” “if...then...” and “if and only if”

Symbolism is introduced into logic by using single letters of the alphabet to represent sentences and using special symbols to represent the logical connectives. As an example of translation into symbolic form, consider the sentence:

If George drinks and George does not smoke cigarettes then George is in good health.

This is a compound sentence made up from three simple sentences. We may let each of the single letters p , q , and r denote one of the simple sentences:

p : George drinks.
 q : George smokes cigarettes.
 r : George is in good health.

If in addition we let the symbol “ \sim ” stand for “not”, let “ \Rightarrow ” stand for “if...then...” and let “ \wedge ” stand for “and”, then the above sentence may be written in symbolic form as

$$(p \wedge \sim q) \Rightarrow r$$

As further illustration, if we let the letter s denote the sentence:

s : George should quit his bad habits.

then the symbolic expression $(p \wedge q) \Rightarrow (\sim r \wedge s)$ would translate into ordinary English as:

If George drinks and smokes cigarettes then George is not in good health and he should quit his bad habits.

We have used parentheses in the above two examples of symbolic expressions to indicate proper grouping of sentence variables and logical connectives. The need for doing so is to eliminate any ambiguity which might be associated with such symbolic forms. In this sense the reason for employing parentheses as grouping symbols is identical to that for using parentheses in mathematical expressions.

The mathematical expression $2 + 3 \times 7$ may be subject to misinterpretation if one does not agree to, or is ignorant of, standard mathematical conventions. If we perform the operations as we read from left to right then we would first add 2 and 3 to get 5, and then multiply 5 by 7 to come up with a final answer of 35. Of course we could just as well begin by multiplying 3 and 7 to get 21, and then add 2 to that result and have 23 for a final answer. Hence the expression $2 + 3 \times 7$ is ambiguous and one way to remove the ambiguity is to introduce parentheses. If the intended meaning is to add first, we would write $(2 + 3) \times 7$ and if we mean to multiply first we would write $2 + (3 \times 7)$.

Of course this example is quite simple but it clearly demonstrates the need and use of parentheses as grouping symbols. In more complicated examples we may need to introduce several sets of parentheses to give an expression an intended meaning. But as more parentheses are introduced, a given expression becomes more difficult to read. Strictly speaking, it is not incorrect to include more parentheses than necessary, but the idea is to use a minimum number of parentheses and yet still maintain the intended meaning or interpretation. For this reason certain conventions have been adopted in mathematics.

One of these conventions is that we should perform multiplications and divisions before additions and subtractions. With this idea in mind the expression $2 + 3 \times 7$ is no longer ambiguous and its intended meaning can be expressed without any parentheses at all. Of course if the intended meaning is to add first then we must use parentheses and write $(2 + 3) \times 7$. The example $(2 + 3) \times 7$ does not violate the convention that we should multiply before adding. We would multiply times 7 before adding if we only knew what the first number is to be multiplied by 7. The only way to figure that number is to perform the operation within parentheses first.

The above convention does not resolve the ambiguity of the simple expression $16 \div 8 \div 2$. If we perform the operations as we read from left to right this expression reduces to the number 1. But if we perform $8 \div 2$ first, then the final answer will come out to be 4. So the above convention is usually refined to state that we should perform multiplications and divisions first, in left to right order, before performing any additions or subtractions, in left to right order, all of these operations being carried out within sets of grouping symbols. In actual practice we usually find it easiest to begin by performing first, all those operations within the innermost set of parentheses, and then work our way outwards until the expression is complete.

Now let's get back to George. In symbolic logic we can also run into the same ambiguity of expressions that occurs in mathematics. This time our variables will denote the following simple sentences.

- p : George will play the piano.
- q : George will sing.
- r : George's sister will sing.

We also let the symbol “ \vee ” denote the logical connective “or”. Then the symbolic expression

$$p \vee q \wedge r$$

is ambiguous, for when we insert grouping symbols in two different ways we come up with two different interpretations. When we translate $p \vee (q \wedge r)$ into English we have:

George will play the piano or George and his sister will sing.

When we translate $(p \vee q) \wedge r$ into English we have

George will play the piano or sing and his sister will sing.

The meanings of these two translations are obviously different, since in the second case George's sister is going to sing regardless of what George does. But in the first case it may happen that George's sister will be sitting on the sidelines watching George play the piano with his mouth closed.

Just as in mathematics, in symbolic logic there are certain conventions that dictate which logical connectives take precedence over others. These conventions also help keep the number of parentheses to a minimum. We need not discuss these conventions in detail since our only purpose is to introduce enough basic ideas to be able to formulate the genuine Polish notation.

Many standard texts on symbolic logic don't use parentheses for grouping symbols. They employ a dot notation in which one or more dots may take the place of one or more sets of parentheses. The meaning of the symbolic expression

$$(p \vee q) \wedge (q \vee r)$$

would be the same if we simply dropped the outer parentheses and wrote

$$p \vee q \wedge q \vee r$$

instead. And now the inner parentheses may be replaced by two dots so that

$$(p \vee q) \wedge (q \vee r)$$

could be written in dot notation as

$$p \vee q \cdot \wedge \cdot q \vee r$$

We have mentioned the existence of the dot notation only to point out that there are systems of symbolic logic which don't employ parentheses as grouping symbols. However, this last statement is not to be taken to mean the same thing when we say Lukasiewicz's notation is a parenthesis-free notation. For the systems which use the dot notation have almost literally traded dots for parentheses. The dots are still necessary to act as grouping symbols.

As an aside, the sentential calculus can be expanded to include what is called the predicate calculus. In the predicate calculus simple sentences may be further broken down into terms. What are called existential and universal quantifiers are used to handle terms. Associated with each quantifier is the concept known as the scope or range of the quantifier. Most of those systems of symbolic logic which use the dot notation do so in order to reserve parentheses to indicate the scope of quantifiers.

What makes Lukasiewicz's notation unique is that it is not only a parenthesis-free and dot-free notation, in fact, the notation is free of any kind of grouping notation whatever! Lukasiewicz used capital letters to denote the logical connectives.

C replaces	\Rightarrow	C is for conditional
A replaces	\vee	A is for alternation
K replaces	\wedge	K is for conjunction
N replaces	\sim	N is for negation
E replaces	\Leftrightarrow	E is for equivalence or “if and only if”

The simple conditional “if p then q ”, which in standard notation would be denoted by $p \Rightarrow q$ is denoted by Cpq in Polish notation. $p \wedge q$ in Polish notation is Kpq , while $p \vee q$ is Apq . $\sim p$ is simply Np .

In this symbolism, the capital letters which denote the logical connectives act as operators and the lower case letters act as the operands. The operator is placed to the left of the operands in the Polish notation.

The order in which the operands are placed to the right of the operator are important in the case of Cpq . Cpq denotes “if p then q ” whereas Cqp denotes “if q then p ”. Complex symbolic sentences can be formed unambiguously and without parentheses using the Polish notation. It is very significant that Lukasiewicz pointed out that his notation is also easily written using a standard typewriter.

The symbolic formula $(p \wedge \sim q) \Rightarrow r$ when written in the Polish notation becomes $CKpNqr$. This is a well-formed formula (**WFF**) whose construction can be understood by beginning with the expression in parentheses in the standard notation. There we have $p \wedge \sim q$. This means “ p and not q ”. Nq denotes “not q ” and when we form the conjunction of Nq with p we write the letter K to the left of the two terms whose conjunction we wish to form. We thus have $KpNq$ as the antecedent of the above conditional whose consequent is r . To form the complete expression we write both $KpNq$ and r to the right of the letter C . The final form is then $CKpNqr$.

$(p \wedge q) \Rightarrow (\sim r \wedge s)$ becomes $CKpqKNrs$. The antecedent and consequent of this basic conditional statement are Kpq and $KNrs$. The conditional is formed by writing these two terms to the right of the letter C . It can now be easily seen why the Polish notation does not require parentheses.

If we next consider the performances of George and his sister we will see how the symbolic formula $p \vee q \wedge r$ may be rendered unambiguous using the Polish notation. For the formula $p \vee (q \wedge r)$ becomes $ApKqr$ in Polish notation, while the other variation, $(p \vee q) \wedge r$, becomes $KApqr$. Note that in $ApKqr$ the connective A applies to p and Kqr , while the connective K applies to q and r . But in $KApqr$, the connective A applies only to p and q whereas the connective K applies to Apq and r .

As a further aid to understanding the Polish notation we note that not every symbolic string is a **WFF**. The formula $ApqCr$ is not a **WFF** for at least two reasons. First, Cr is incomplete because C is a two-place connective and yet only one character appears to its right in Cr . If we were to insert the letter s for the missing operand in Cr , the original formula would become $ApqCr s$ which is still not a **WFF**. Apq and $Cr s$ are two complete syntactically correct atomic units, but they can only be combined by inserting another two-place connective to their left. Adding K on the left makes $ApqCr s$ turn into $KApqCr s$ which is a **WFF**. In the traditional notation, $KApqCr s$ would become $(p \vee q) \wedge (r \Rightarrow s)$.

We previously indicated that the reasons for using parentheses in symbolic logic were identical to the reasons for using parentheses in ordinary mathematical notation. In fact, there is a very close relationship between the **WFFs** in sentential calculus and formulas in mathematics. Every **WFF** can be translated into a mathematical formula whose only operations are addition, multiplication, and negation. The reverse is also true. By translating sentential calculus **WFFs** into mathematical formulas we can see more clearly their algebraic character.

The algebra of logic obeys many (but not all) of the laws of arithmetic. When we translate a formula from sentential calculus into a mathematical formula the variables no longer represent sentences, but rather represent numerical quantities which are the truth values of the translated sentences. We will restrict our arithmetic to the two numbers 0 and 1. $0 = FALSE$ and $1 = TRUE$. The logical connectives will be translated into familiar mathematical operations.

For example, logical conjunction is translated into multiplication. The **WFF** $p \wedge q$ translates as $p \times q$. Note that if p is *TRUE* and q is *FALSE* then the conjunction $p \wedge q$ is *FALSE*. Substituting 1 for p and 0 for q in the mathematical formula $p \times q$ yields $1 \times 0 = 0$ which also represents a *FALSE* result. As a matter of fact, the only time $p \times q$ is *TRUE* is when p and q are individually *TRUE*. Similarly, the only time $p \times q$ yields a numerical value 1 is when both p and q take on the value 1.

Alternation is translated into addition. The connective “or” as used in logic always refers to the non-exclusive case. Thus “ $p \vee q$ ” reads as “ p or q and possibly both”. In order for $p \vee q$ to be *TRUE* at least one of p and q must be *TRUE*. The only time $p \vee q$ is *FALSE* is when both p and q are *FALSE*. Note that the only time $p + q = 0$ is when both p and q are zero. One slight complication with $p + q$ occurs when both p and q are equal to 1. In this case, true addition yields a value of 2. Thus the $+$ in the algebra of logic differs slightly from ordinary addition. For our purposes, we have to define $1 + 1 = 1$, but otherwise, you may assume ordinary addition is the operation involved.

Since negating a *TRUE* sentence yields a *FALSE* one, and vice versa, we also have the peculiar arithmetic statements that $-1 = 0$ and $-0 = 1$. $\sim p$ translates as $-p$.

The associative and commutative laws hold for both \wedge and \vee . We also have two distributive laws, one for \wedge over \vee and the other for \vee over \wedge . There is even a principle of duality which states that any formula that is valid in our new system of algebra has a dual statement which is also valid. The dual statement is obtained by switching $+$ for \times and switching \times for $+$. When we apply this principle of duality to the usual distributive law $p \times (q + r) = p \times q + p \times r$ we obtain the unfamiliar distributive law $p + (q \times r) = (p + q) \times (p + r)$.

The following is a list of examples which show the results of translating ordinary sentential calculus formulas into their mathematical equivalents.

$$\begin{aligned}
 p \vee q & \text{ becomes } p + q \\
 p \wedge q & \text{ becomes } p \times q \\
 p \vee (q \wedge r) & \text{ becomes } p + (q \times r)
 \end{aligned}$$

$$(p \wedge \sim r) \vee s \text{ becomes } p \times (-r) + s$$

$$(p \wedge q) \vee (q \wedge \sim r) \text{ becomes } (p \times q) + (q \times (-r))$$

$$(p \vee q) \wedge (q \vee r) \text{ becomes } (p + q) \times (q + r)$$

We could give many more examples as well as discuss the full translation process, but our purpose here is only to make the historical connection between the Polish notation and ordinary mathematical notation.

In 1847 George Boole published a mathematical analysis of logic which applies algebra to syllogisms. His algebra is not what is known today as Boolean algebra. Boolean algebra was devised by Peirce and Schroder and applied to the sentential calculus. But one important aspect of Boole's work was that it used the symbols of algebra in a new way. Up until that time, algebra was thought of as applying only to real numbers. Boole made the connection between logic and algebra and Schroder and Peirce extended those results. Lukasiewicz was also a historian of logic. Although the Polish notation was originally applied to sentential calculus, Lukasiewicz was well aware of the translation of formulas from symbolic logic into ordinary mathematical notation.

The connection will now become most apparent if we review the above examples of the translation process in terms of the Polish notation. We will simply re-write the above logic formulas in terms of the Polish notation and then replace the Polish symbols A , K , and N by their mathematical equivalents $+$, \times , and $-$.

$$Apq \text{ becomes } + pq$$

$$Kpq \text{ becomes } \times pq$$

$$ApKqr \text{ becomes } + p \times qr$$

$$AKpNrs \text{ becomes } + \times p - rs$$

$$AKpqKqNr \text{ becomes } + \times pq \times q - r$$

$$KApqAqr \text{ becomes } \times + pq + qr$$

The reader with sufficient experience will immediately make the connection between these last pseudo-mathematical expressions and how calculations are performed on **RPN** calculators. Reading the expressions from right to left tells you what quantities are to be entered into the calculation and what operations are to be performed.

Now consider the four analogous and equivalent expressions:

$$p \vee (q \wedge r) \quad p + (q \times r) \quad ApKqr \quad + p \times qr$$

Reading the fourth expression backwards tells you precisely how to compute the second expression. Following the standard mathematical convention, we should enter r and q into the calculation first, and multiply, before entering p and adding. The backward reading can be avoided by writing the terms of the expression in reverse order.

Eureka! Reverse Polish Notation!

Writing the 4th expression in reverse yields $rq \times p +$ which is the **RPN** equivalent of the above four notations. The third notation $ApKqr$ is genuine Polish notation in the symbolic logic as formulated by Lukasiewicz in 1929.

In actual practice, mathematical formulas are not written in **RPN**, but they may be thought of as being written that way, especially when the formula is evaluated on an **RPN** calculator. Once the basic principles behind **RPN** are understood, it is a simple matter to mentally convert from ordinary mathematical notation to **RPN** while a formula is being read and evaluated. The translation process is as simple for complex formulas as it is for elementary formulas.

Three notations are now commonly used to write the relationship between operators and operands. Whether the actual operations are logical or mathematical or something else is of little consequence. The terms prefix, infix and postfix apply to notations involving dyadic (binary) operations. We will simply illustrate all three forms by the example of addition. The sum of a and b can be written as: $+ ab$, or $a + b$, or $ab +$.

The first of these is the prefix notation since as we read from left to right the operation precedes the operands. The middle is the common infix notation; the operation is placed between the two operands. The third example with the operation written last is postfix notation. Observe that operations with single operands such as negation or square root can only be written in prefix or postfix notation.

The original Polish notation is a prefix notation and the **RPN** calculator notation is a postfix notation. It is somewhat surprising to learn that compilers and translators for computer languages such as **FORTRAN**, **BASIC**, **ALGOL**, and **Pascal** employ reverse Polish notation that is transparent to the user who writes mathematical expressions using the infix notation. A formula written in the reverse Polish notation has the advantage that it can be tested for syntactical correctness while it is being evaluated. Any operations that are found as the formula is scanned from left to right may be executed immediately without requiring further examination of the formula. The Polish notation is also closely related to tree structures used in computer science.

As a final historical note, in 1951 Lukasiewicz wrote the book **ARISTOTLE'S SYLLOGISTIC FROM THE STANDPOINT OF MODERN FORMAL LOGIC**. This book is the most often misquoted source of the Polish notation. It is a book about logic, not mathematics. On page 78 Lukasiewicz again explains his logical symbolism which he clearly states he has used since 1929. For the reader's benefit he also mentions that such a notation could be used in mathematics as well as logic. Using the associative law of addition as an example, he devotes a total of three lines showing the parallel. The example is not written in reverse Polish notation and these three lines hardly constitute the basis of the Polish notation.

For the original source, see Lukasiewicz's 1929 *ELEMENTS OF MATHEMATICAL LOGIC* listed below. The first chapter of this book is a historical discussion of the relationship between mathematical and philosophical logic. The more prominent works discussed are by Boole, Peirce, Schroder, Frege, Peano, Whitehead, and Russell. Chapter 2 is devoted to defining the primitive terms and axioms of the sentential calculus. Lukasiewicz devotes no less than 20 pages to his system of notation and its basic operations. On page 24 he uses the associative law of addition as an example of how the notation would be applied to arithmetic. This is the first known published source of what is now called the Polish notation. The preface to this book was written in February 1929 and thus it is certain that the Polish notation was invented in 1928 or earlier. This notation is very similar to a notation used by Lukasiewicz in connection with syllogisms.

List of References:

Bauer, F.L. and Samelson, K. "Sequential Formula Translation" *COMMUNICATIONS OF THE ACM*, Volume 2, February 1960, pp. 76-83.

Berkeley, E.C., *GIANT BRAINS OR MACHINES THAT THINK*, New York, John Wiley, 1949 Chapter 9.

Burks, Arthur W., Warren, Don W., Wright, Jesse B., "An Analysis of a Logical Machine Using Parenthesis-Free Notation" *MATHEMATICAL TABLES AND OTHER AIDS TO COMPUTATION*, Vol. 8 No. 46, April 1954 pp. 53-57. (Now called *MATHEMATICS OF COMPUTATION*)

Gardner, Martin, *LOGIC MACHINES AND DIAGRAMS*, New York, McGraw Hill, 1958.

Girling, B. and Moring, H.G., *LOGIC AND LOGIC DESIGN*, Bucks England, International Textbook Company Limited 1973, pp. 118-125.

Hamblin, C.L. "Translation to and From Polish Notation", *THE COMPUTER JOURNAL*, Vol. 5 No. 3, October 1962, pp. 210-213.

Lukasiewicz, Jan, *ELEMENTY LOGIKI MATEMATYCZNEJ*, Warsaw, 1st Edition 1929. English translation: *ELEMENTS OF MATHEMATICAL LOGIC*, translated from Polish by Olgierd Wojtasiewicz, New York, The MacMillan Company, 1963.

McGechie, John E., "The Hewlett-Packard Pupil, Elementary Formal Logic on a Programmable Calculator," *AUSTRALIAN LOGIC TEACHERS JOURNAL*, Feb. 1979 Vol. 3 No. 2, August 1979 Vol. 4 No. 1. See also Vol. 4 No. 2.

Sheridan, P.B., "The Arithmetic Translator-Compiler of the IBM Fortran Automatic Coding System," *COMMUNICATIONS OF THE ACM*, Volume 2, Number 2, 1959, pp. 9-21.

John Kennedy (918)