

Notes on post-fix notation (also known as “Reverse Polish Notation”)

12th January 2004

A way of writing mathematical expressions developed in the 1920’s by Polish mathematician Jan Lukasiewicz.

Comes in two forms “prefix” and “post-fix”

Prefix notation/Polish notation: operator operand, or operator operand operand

Examples: ~ 1 , $+ 1 2$, $* 3 + 1 2$;

Notice how you may find an operator where you expect an operand; in that case, evaluate the expression corresponding to that operator in order to evaluate the first operator (*, in this case).

In the post-fix variant (Reverse Polish notation) operators *follow* the operands

Examples: $1 \sim$, $1 2 +$, $3 1 2 + *$

On Friday, when we begin talking about the PostScript language, we will be discussing a programming language that uses a notation closely related to RPN.

Today, we will just talk about RPN though to begin to get a sense of how it works.

RPN became widely known in the engineering and scientific community in the mid 1970s when Hewlett-Packard introduced the HP-35 scientific calculator – the first pocket calculator. To use the HP-35 you enter mathematical expressions in RPN. I’m sure that the appeal for HP was that the calculator itself was considerably simplified by this choice, but for the user there are also considerable efficiencies once you get used to it.

Here is my HP-35 bought in 1973 for \$295 from the Bookie. You can come up and take a look at it later. It’s too small to use to demonstrate, so we’ll look at a java applet calculator instead. The HP-35’s release started the rapid demise of the slide rule as a tool of scientists and engineers.

Even today, HP’s high-end scientific calculators offer RPN entry as an option. If you have such a calculator I recommend learning to use this mode – it is less error prone than using parentheses.

Algorithm for entering formulas in RPN:

```

while not done {
  enter a number
  while true {
    if a one-operand operation is possible {
      enter it;
    } else if a two-operand operation is possible {
      enter it;
    } else break;
  }
}

```

Example:

```

(4 + 2 * 5) / (1 + 3 * 2)
=>
4 2 5 * + 1 3 2 * + /

```

How does it work: numbers that are entered directly as well as results of operations are kept on a *stack*. Each operation consumes its operands from the stack and replaces them with the result of the operation.

Illustrate with Boehm Constructive Reals calculator: http://www.hpl.hp.com/personal/Hans_Boehm/new_crcalc/Run

Example: recall the quadratic formula $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Let's solve $x^2 + 4x - 32 = 0$. $a = 1$, $b = 4$, $c = -32$. Observe that we'll need the square root twice and this is maybe the hardest to enter so let's start with it.

```

4 4 * 4 32 chs * - sqrt dup
4 chs + 2 / ----- first answer (4)
pop ----- get rid of first answer
4 + chs 2 / ----- second answer (-8)
----- check
4 4 * 4 4 * + 32 - ----- 0 as expected
8 chs dup * 4 8 chs * 32 - ----- 0 as expected

```

Suggestion: go online and find a RPN calculator to play with between now and Friday. The one I'm using here comes up in the first few results if you search for CRCalc and HP in Google. Another good search is "RPN calculator applet".

(The RPN algorithm is derived from a web page by W. Marshall Leach, Jr., Prof. of EECE at Georgia Tech).