RPN

CptS 355 - Programming Language Design

Washington State University

Home
Kliks
Calendar
Syllabus
Resources
People
Project turn-in

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

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 as in the last example; in that case, we suspend evaluation of the expression of the first operator (*) in order to evaluate that of the second (+). Once we know the result of the + we can continue computing the *.

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

- 1 ~
- 1 2 +
- 3 1 2 + *

When we begin talking about the PostScript language, we will be discussing a programming language that uses a notation closely related to RPN. First, 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 scientific 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. I bought an HP-35 at the WSU Bookie in 1973 for $295. I still have it and it still works though the batteries are long-since dead. 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;
    }
}
```

For non-commutative operators (/, -) you need to arrange to compute the left operand first.
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.

Online resources: Boehm Constructive Reals calculator: http://www.hpl.hp.com/personal/Hans_Boehm/new_crcalc/CRCalc.html. You want the "Default RPN" calculator listed on this page. You can ignore the discussion on the page about constructive reals -- that is not a topic for this class.
Example: recall the quadratic formula `(-b±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. (% is used to begin a comment to the end of the line)

```
4 4 * 4 32 chs * - sqrt % discriminant
dup                     % make a copy on the stack
4 chs + 2 /             % first answer (4)
pop                     % get rid of first answer
4 + chs 2 /             % second answer (-8)
% now let's check the answers
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 experiment 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).