...evaluate an expression using queues.

[<< Back to: comp.lang.c Answers (Abridged) to Frequently Asked Questions (FAQ)]

Question by seema	
Submitted on 3/22/2004	
Related FAQ: comp.lang.c Answers (Abridged) to Frequently Asked Questions (FAQ)	
Rating: ★★★½☆	Rate this question: N/A ▼ VOTE
how to evaluate an expression using queues.	

Answer by zia	
Submitted on 4/13/2004	
Rating: ★★★½☆	Rate this answer: N/A ▼ VOTE

Evaluating expressions.
Below are evaluations of the same expression in three different notations. Time "flows" from top to bottom.

	infix notation	prefix notation	postfix notation
0:	(1 + (2 * 3)	+ 1 * 2 3	1 2 3 * +
1:	(1 + 6)	+ 1 6	1 6 +
2:	7	7	7

The basic properties of rewriting are well-known. At each step the algorithm works

by (1) inspecting parts of an expression, and then (2) replacing that part by a value. So the method actually consumes the original expression. Note also that all rewriting is in-place.
For larger expressions there may be several places where rewriting could take place, and the rewriting could then even proceed concurrently. For infix and prefix expressions it may be necessary to search to the left or to the right to find a suitable expressions to evaluate next. For postfix it is always possible to restrict the search to the right, this is used in the stack machine below.
Evaluating expressions by a queue machine
Here is a method which uses the prefix expression as a queue. Time again flows "flows" from top to bottom.

```
0:        + 1 * 2 3
1:            1 * 2 3 +          (2b)
2:              * 2 3 + 1        (2c)
3:                    + 1 6      (2a)
4:                        7      (2a)
```

The algorithm is quite different from in-place rewriting in the first section and also different from the stack-rewriting in the second section.

WHILE the queue has more than one element DO

(1) Inspect the front element.
(2a) If it is an operator followed by all its arguments,
append the value to the rear of the queue AND THEN
remove operator and arguments from the front of the queue
(2b) If it is an operator not followed by all its arguments,
remove it from the front, copy it to the rear.
(2c) (it is an operand)
remove it from the front, copy it to the rear.

The algorithm resembles rewriting by starting with a prefix expression and ending with a value, but the rewriting is not in-place. The algorithm resembles the stack machine by successively examining the next symbol of the expression, but it does not use an auxiliary structure. The significance of the "AND THEN" emphasis in step (2a) will become apparent shortly.
In the same way, it is just as simple to evaluate a postfix expression as a queue - merely by reading the queue from right to left.

```
0:              1 2 3 * +
1:            + 1 2 3 *        (2b)
2:          6 + 1              (2a)
3:        1 6 +                (2c)
4:      7                      (2a)
```

Of course postfix and prefix are not just reverses of each other, as is seen by examples with non-commutative operators such as subtraction or division. But as the two examples demonstrate, there is no inherent connection between stack evaluation and postfix, or between queue evaluation and prefix. Any such connection is just by convention. This being said, I now follow this convention and speak of stack machines being driven by postfix expressions. In the same way I shall now take queue machines to be certain rewritings of prefix expressions.
For any particular prefix evaluation the efficiency will depend on the ratio of the (2a) steps that really do the work, and the (2b) and (2c) bookkeeping steps that are at best a nuisance. The ratio depends on the patters of (binary) operators b and values v. In the example, of the form

```
b v b v v
```

the ratio was 2:1:1. For expressions of the form
b b v v b v v
the ratio is 3:1:0 which is better. For expressions of the form
b b v b v v v
the ratio is 3:3:3 which is worse. In general, the algorithm is best for expressions representing a balanced binary tree. Since most expressions depart from this ideal, the entire method would seem to be barely more than a cute but silly curiosity.
But let us recapitulate: the other methods use in-place rewriting of either an original infix or prefix or postfix expression, or of a separate stack. For word-size values such as integers this hardly matters because (1) integer operations are indivisible single machine operations, and (2) the result will always fit into the space that has been vacated by the operands. The queue machine is also a rewriting method, but it is not in-place. This has a number of repercussions, to be discussed below.

Answer by lapnguyenxuan	
Submitted on 6/12/2004	
Rating: ☆☆☆☆	Rate this answer: N/A ▼ VOTE
infix convert to posfix	

Answer by sharad	
Submitted on 9/28/2004	
Rating: Not yet rated	Rate this answer: N/A ▼ VOTE
http://forums.devshed.com/showthread.php?t=132345	

Answer by test13	
Submitted on 11/24/2005	
Rating: Not yet rated	Rate this answer: N/A ▼ VOTE
<IMG SRC="/images/library.jpg" HEIGHT=62 WIDTH=150 BORDER="0" ALIGN="MIDDLE" ALT="Internet FAQ Archives">	

Your answer will be published for anyone to see and rate. Your answer will not be displayed immediately. If you'd like to get expert points and benefit from positive ratings, please create a new account or login into an existing account below.

Your name or nickname:	
If you'd like to create a new	

account or access your existing account, put in your password here:	
Your answer:	

Check spelling

FAQS.ORG reserves the right to edit your answer as to improve its clarity. By submitting your answer you authorize FAQS.ORG to publish your answer on the WWW without any restrictions. You agree to hold harmless and indemnify FAQS.ORG against any claims, costs, or damages resulting from publishing your answer.

Submit

[<< Back to: comp.lang.c Answers (Abridged) to Frequently Asked Questions (FAQ)]

[Home | FAQ-Related Q&As | General Q&As | Answered Questions]