

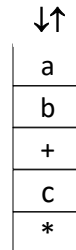
An Algebraic Specification of a simple LIFO stack.

(Following the axioms, I have supplied three separate proofs for your scrutiny)

Consider the following instance of a stack:

Say, the evaluation of expression “(a + b) * c” is being considered.

In RPN implemented via a LIFO stack structure:



Operations:

init (create a new stack)

put (place an element on the stack)

pop (process and remove an element from the stack)

top (check which element is currently at the top of the stack)

type (check if the element at the top of the stack is an operand or an operator)

done (check whether or not a stack contains any terms, i.e. whether an expression has been fully evaluated)

Signatures:

init () -> stack	[basic]
put (s, t) -> queue	[basic]
pop (s) -> queue	[extra]
top (s) -> t	[inspector]
type (s) -> op	[inspector]
done (s) -> Boolean	[inspector]

Axioms:

[basics x (extras + inspectors) => 2x(1+3) = 8 axioms]

- 1) pop (init ()) = error
- 2) pop (put (s,t)) = if done (s) = 1 then init () else s
- 3) top (init ()) = error
- 4) top (put (s,t)) = t
- 5) type (init ()) = error
- 6) type (put (s,t)) = type (t)
- 7) done (init ()) = TRUE
- 8) done (put (s,t)) = FALSE

Proofs:

(Proof 1) To prove that, from the original stack, processing the terms “a+b” will not result in a fully evaluated expression.

The initial stack:

put (put (put (put (put (init ()),*),c),+),b),a)

Operation to be proven on the initial stack:

done (pop (pop (pop (put (put (put (put (put (init ()),*),c),+),b),a))))

Iteratively (3 times) using axiom 2 (else clause) on the initial stack and required operations results in:

done (pop (pop (put (put (put (put (put (init ()),*),c),+),b))))

```
done (pop (put (put (put (init ()),*),c),+)))  
done (put (put (init ()),*),c))
```

Using axiom 8 on the above:

FALSE (proven as required)

(Proof 2) To prove that, from the original stack, after processing the terms “a+b” the next term to process will be an operand.

The initial stack:

```
put (put (put (put (put (init ()),*),c),+),b),a)
```

Operation to be proven on the initial stack:

```
type (pop (pop (pop (put (put (put (put (init ()),*),c),+),b),a))))
```

Iteratively (3 times) using axiom 2 (else clause) on the initial stack and required operations results in:

```
type (pop (pop (put (put (put (init ()),*),c),+),b)))
```

```
type (pop (put (put (put (init ()),*),c),+)))
```

```
type (put (put (init ()),*),c))
```

Using axiom 6 on the above yields:

```
type (c)
```

The resulting value of this axiomatic operation is “**operand**” (proven as required)

*(Proof 3) To prove that, from the original stack, processing the terms “(a+b)*c” will result in a fully evaluated expression.*

The initial stack:

```
put (put (put (put (put (init ()),*),c),+),b),a)
```

Operation to be proven on the initial stack:

```
done (pop (pop (pop (pop (pop (put (put (put (put (init ()),*),c),+),b),a))))))
```

Iteratively (5 times) using axiom 2 (else clause) on the initial stack and required operations results in:

```
done (pop (pop (pop (pop (put (put (put (put (init ()),*),c),+),b))))))
```

```
done (pop (pop (pop (put (put (put (init ()),*),c),+))))
```

```
done (pop (pop (put (put (init ()),*),c)))
```

```
done (pop (put (init ()),*))
```

```
done (init ())
```

Using axiom 7 on the above:

TRUE (proven as required)