Introduction to exercise 5a

CS-460: Programming Languages

Robert Bruce

What is the Shunting Yard algorithm?

The Shunting Yard algorithm converts mathematical* expressions in infix notation into either prefix or postfix notation while preserving operator precedence.

E. W. Dijkstra invented and published the Shunting Yard Algorithm in 1963 (reference is included at the end of this presentation).

**Note: we can adapt the Shunting Yard algorithm to convert Boolean expressions from infix notation to postfix notation too!*

Why is the Shunting Yard algorithm useful?

- Computing an expression in infix notation can be difficult while observing operator precedence.
- Converting the expression into postfix notation while observing operator precedence makes evaluating the expression much easier.
- The conversion process from infix to postfix will require a temporary working stack.
- The evaluation of the postfix expression will also require a temporary working stack.
- The stack is a Last-In-First-Out (LIFO) data structure.

The Shunting Yard algorithm

- In the Files folder under Canvas, I've written the Shunting Yard algorithm in a C-like programing language.
- The file is titled, "numerical_expression_postfix_algorithm.txt".

Converting from infix to postfix using the Shunting Yard algorithm

Example of an expression in infix notation:

```
1 - 2 * 3 + 4 * 5 / 5
```

Converting the following expression from infix to postfix using the Shunting Yard algorithm:

`1 - 2 * 3 + 4 * 5 / 5`

STEP	INPUT TOKEN	OUTPUT	STACK
1	1		
2		1	
3	-	1	
4		1	-
5	2	1	-
6		1 2	-
7	*	1 2	-
8		1 2	- *
9	3	1 2	- *
10		1 2 3	- *
11	+	1 2 3 *	-
12		1 2 3 * -	+
13	4	1 2 3 * - 4	+
14		1 2 3 * - 4	+
15	*	1 2 3 * - 4	+
16		1 2 3 * - 4	+ *
17	5	1 2 3 * - 4	+ *
18		1 2 3 * - 4 5	+ *
19	/	1 2 3 * - 4 5	+ *
20		1 2 3 * - 4 5 *	+ /
21	5	1 2 3 * - 4 5 * 5	+ /
22		1 2 3 * - 4 5 * 5 /	
23		1 2 3 * - 4 5 * 5 / +	

Converting from infix to postfix using the Shunting Yard algorithm

Example of an expression in infix notation:

```
1 - 2 * 3 + 4 * 5 / 5
```

Same expression in postfix notation:

```
1 2 3 * - 4 5 * 5 / +
```

Computing a postfix expression using a stack

Example: computing an expression in infix notation while observing operator precedence:

```
1 - 2 * 3 + 4 * 5 / 5
= 1-(2*3)+((4*5)/5)
= 1-6+(20/5)
= 1-6+4
= -5+4
= -1
```

Computing a postfix expression using a stack

Example: computing in postfix notation :

```
1 2 3 * - 4 5 * 5 / +
```

Computing the following postfix expression using a stack: `1 2 3 * - 4 5 * 5 / +`

STEP	INPUT TOKEN	STACK	NOTES
1	1	1	Push (1)
2	2	1 2	Push (2)
3	3	1 2 3	Push (3)
4	*	1 2 3 *	Push (*)
5		1 2 3	Pop (*)
6		1 2	Pop (3)
7		1	Pop (2)
8		1	Compute: 2*3 = 6
9		1 6	Push (6)
10	-	1 6 -	Push (-)
11		1 6	Pop (-)
12		1	Pop (6)
13		1	Pop (1)
14			Compute: 1-6 = -5
15		-5	Push (-5)
16	4	-5 4	Push (4)
17	5	-5 4 5	Push (5)
18	*	-5 4 5 *	Push (*)
19		-5 4 5	Pop (*)
20		-5 4	Pop (5)
21		-5	Pop (4)
22		-5	Compute: 4*5 = 20
23		-5 20	Push (20)
24	5	-5 20 5	Push (5)
25	/	-5 20 5 /	Push (/)
26		-5 20 5	Pop (/)
27		-5 20	Pop (5)
28		-5	Pop (20)
29		-5	Compute: 20/5 = 4
30		-5 4	Push (4)
31	+	-5 4 +	Push (+)
32		-5 4	Pop (+)
33		-5	Pop (4)
34			Pop (-5)
35			Compute: -5+4 = -1
36		-1	Push (-1)
37			Pop (-1)
38			Computation complete: The answer is -1.

Shunting Yard algorithm for Boolean expressions?

- On Thursday, I will discuss how to convert Boolean expressions from infix notation into postfix.
- You will need to evaluate numerical expressions in your interpreter (i.e. assignment statements)
- You will also need to evaluate Boolean expressions in your interpreter (i.e the Boolean expressions inside an *if* statement, *for* statement, *while* statement, etc.)
- Boolean expressions also exhibit operator precedence.
- Stay tuned...

Reference

E. W. Dijkstra, "Making a Translator for ALGOL-60," *Annual Review in Automatic Programming*, vol. 3, pp. 347-356, 1963, doi:10.1016/S0066-4138(63)80016-6