Jake Goldstein

HW 08 Written

1. Infix expressions
   1. 1 - 2 + 3 ^ 2

Input Stack Output

1 1

- - 1

2 - 1 2

+ + 1 2 -

3 + 1 2 - 3

^ + ^ 1 2 - 3

2 1 2 -3 2 ^ +

* 1. (2 ^ 3) ^ 2

Input Stack Output

( (

2 ( 2

^ ( ^ 2

3 ( ^ 2 3

) 2 3 ^

^ ^ 2 3 ^

2 2 3 ^ 2 ^

* 1. 2 ^ 3 ^ 2

Input Stack Output

2 2

^ ^ 2

3 ^ 2 3

^ ^ ^ 2 3

2 2 3 2 ^ ^

* 1. (2 + 6) / 3 - (32 + 4 \* 7) \* 2

Input Stack Output

( (

2 ( 2

+ ( + 2

6 ( + 2 6

) 2 6 +

/ /

3 / 2 6 + 3

- - 2 6 + 3 /

( - ( 2 6 + 3 /

32 - ( 2 6 + 3 / 32

+ - ( + 2 6 + 3 / 32

4 - ( + 2 6 + 3 / 32 4

\* - ( + \* 2 6 + 3 / 32 4

7 - ( + \* 2 6 + 3 / 32 4 7

) - 2 6 + 3 / 32 4 7 \* +

\* - \* 2 6 + 3 / 32 4 7 \* +

2 2 6 + 3 / 32 4 7 \* + 2 \* -

* 1. 3 + 2 - 4 + 5

Input Stack Output

3 3

+ + 3

2 + 3 2

- - 3 2 +

4 - 3 2 + 4

+ + 3 2 + 4 -

5 3 2 + 4 – 5 +

* 1. (3 + 2) ^ 4 ^ (3 \* 2 + 4)

Input Stack Output

( (

3 ( 3

+ ( + 3

2 ( + 3 2

) 3 2 +

^ ^ 3 2 +

4 ^ 3 2 + 4

^ ^ ^ 3 2 + 4

( ^ ^ ( 3 2 + 4

3 ^ ^ ( 3 2 + 4 3

\* ^ ^ ( \* 3 2 + 4 3

2 ^ ^ ( \* 3 2 + 4 3 2

+ ^ ^ ( + 3 2 + 4 3 2 \*

4 ^ ^ ( + 3 2 + 4 3 2 \* 4

) 3 2 + 4 3 2 \* 4 + ^ ^

1. Postfix expressions
   1. 4 2 + 3 3 ^ -

Input Stack

4 4

2 4 2

+ 6

3 6 3

3 6 3 3

^ 6 27

* -21
  1. 3 2 ^ 3 2 \* -

Input Stack

3 3

2 3 2

^ 9

3 9 3

2 9 3 2

\* 9 6

- 3

* 1. 4 2 3 \* - 3 2 ^ - 6 +

Input Stack

4 4

2 4 2

3 4 2 3

\* 4 6

- -2

3 -2 3

2 -2 3 2

^ -2 9

* -11

6 -11 6

+ -5

* 1. 4 3 + 2 \* 1 -

Input Stack

4 4

3 4 3

+ 7

2 7 2

\* 14

1 14 1

- 13

* 1. 3 5 \* 1 + 4 / 6 +

Input Stack

3 3

5 3 5

\* 15

1 15 1

+ 16

4 16 4

/ 4

6 4 6

+ 10

1. enum TokenType { EOL, VALUE, OPAREN, CPAREN, EXP, MULT, DIV, MOD, PLUS, MINUS };

// PREC\_TABLE matches order of Token enumeration

struct Precedence {

int inputSymbol;

int topOfStack; };

vector PREC\_TABLE = {

{ 0, -1 }, { 0, 0 }, // EOL, VALUE

{ 100, 0 }, { 0, 99 }, // OPAREN, CPAREN

{ 6, 5 }, // EXP

{ 3, 4 }, { 3, 4 }, { 3 , 4 } // MULT, DIV, MOD

{ 1, 2 }, { 1, 2 } // PLUS, MINUS

};

1. Printed: exp, exp, mult

The stack (The item on the far right is the top): eol, plus, div

1. For the tree
   1. \*
   2. –
   3. 4 5 8 3
   4. \* + -
   5. 1
   6. 2
   7. 7
   8. 4 –
   9. +
   10. 4 + 5 – 8 \* 3
   11. \* + 4 – 5 8 3
   12. 4 5 8 - + 3 \*
2. The searching efficiency of the tree is Big-Oh of the height. So if the tree is all on one side it is essentially just linear and therefore it will be O(n), where n is the number of elements in the tree. However, if the tree is balanced out more or less evenly, then the run time will be O(log(n)). This is what H describes.
3. 3
4. Making a tree
   1. Minimal height order: 3 1 2 0 6 5 7

3

1 6

0 2 5 7

* 1. Maximal height order: 0 1 2 3 4 5 6 7

0

1

2

3

4

5

6

7

1. Binary tree

389

39 666

30 47 400 999

9 33 45



315

45 666

30 47 389 999

1. 33 400

315

39 666

30 47 389 999

9 33 45 400

398