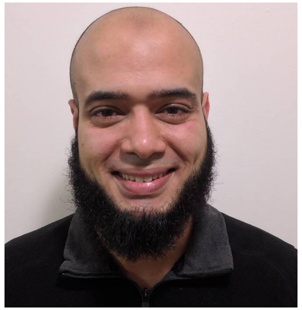
Data Structures

Infix to Postfix

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

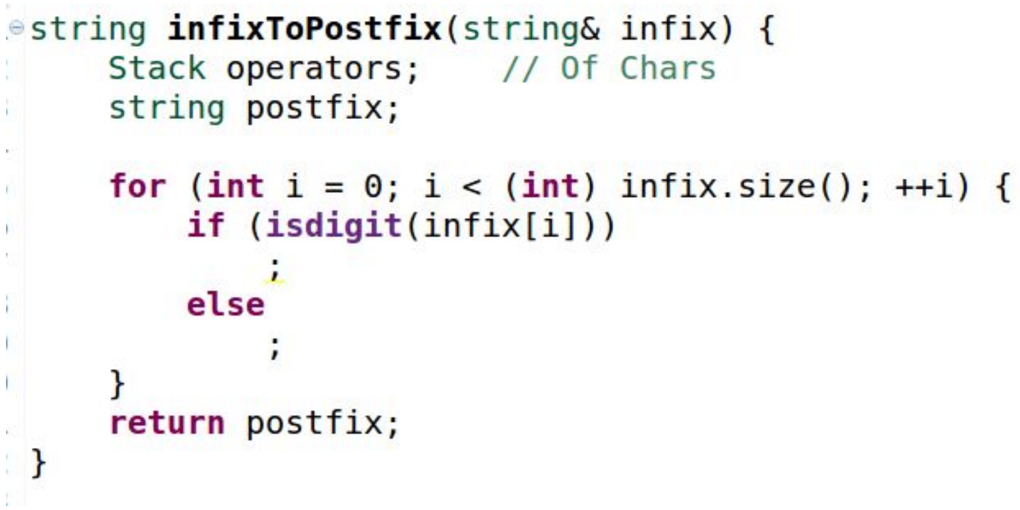


Infix to Postfix Conversion

- Task: Given an infix expression, convert it to a postfix expression
    - 1+2*3 ⇒ 123*+
- For simplicity, let's first consider these constraints
    - Input is a string **without** spaces. Output is a string
    - All numbers are represented as single digits, with no positive/negative sign
        - E.g. {0, 1, 2, ...9} but not -5 or +7
    - Our only operators are + - * /: observe that all of them have left to right associativity
        - Remember: / * has higher precedence than + -
- Shunting-yard algorithm
    - The algorithm was invented by **Edsger Dijkstra** to do the conversion
    - We can both convert and evaluate using stacks
    - Parsed elements (numbers or operators) are called **tokens**

Infix to Postfix Algorithm

- We will maintain a string for the output and a stack of operators
  - So the stack will have only operators: + - * /
- We iterate on the output, moving token by token
  - Each token is either a number (single digit) or an operator (+ - * /)

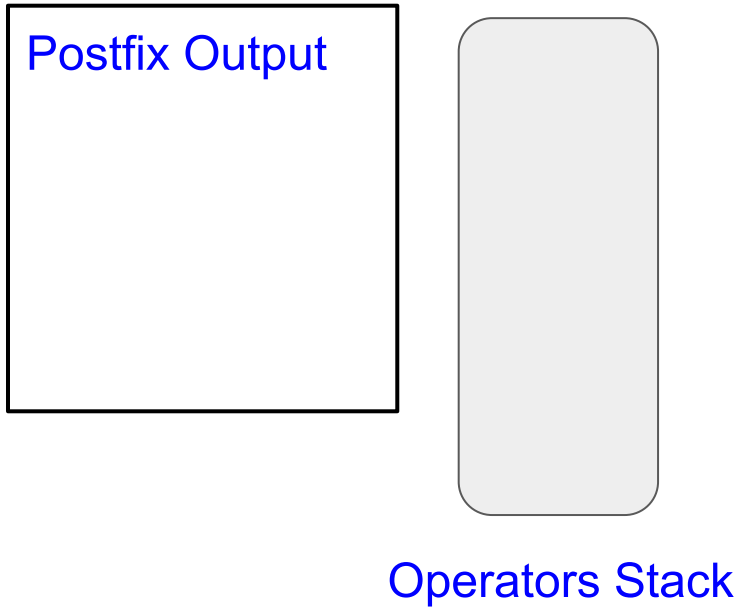


```
string infixToPostfix(string& infix) {
    Stack operators;    // Of Chars
    string postfix;

    for (int i = 0; i < (int) infix.size(); ++i) {
        if (isdigit(infix[i]))
            ;
        else
            ;
    }
    return postfix;
}
```

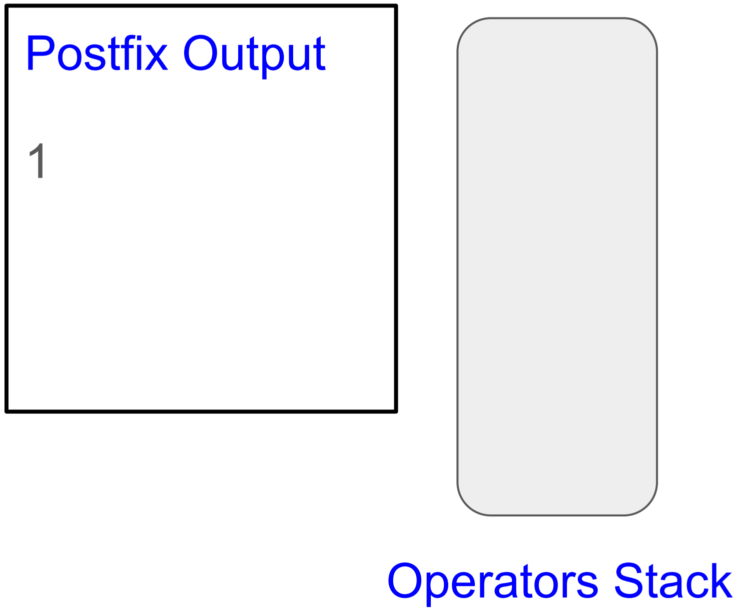
Parsing: 1+3*5-8/2

- We initially have an empty operators stack, and an empty string representing our postfix
- The tokens we'll be handling from the string are:
    - 1
    - +
    - 3
    - *
    - 5
    - -
    - 8
    - /
    - 2



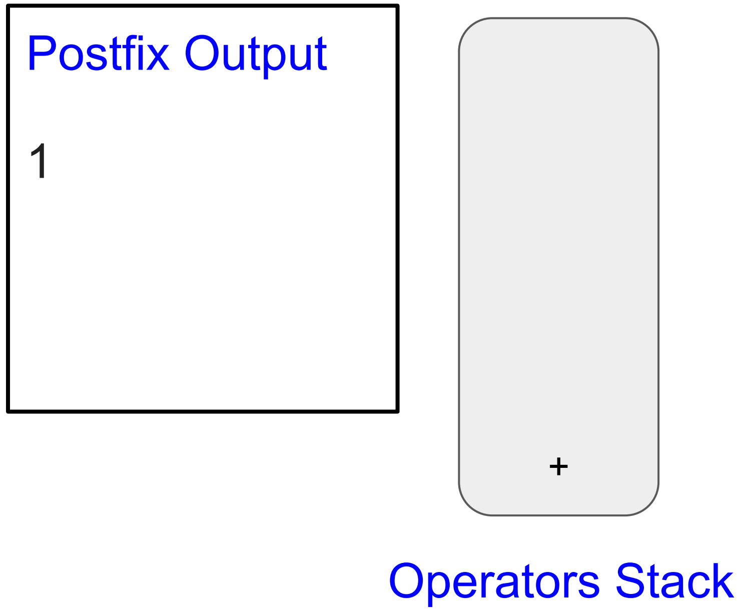
Parsing: **1**+3*5-8/2

- Current Token 1
  - Digit
- Rule #1: if the token is a digit, simply add it immediately to the 'output' string



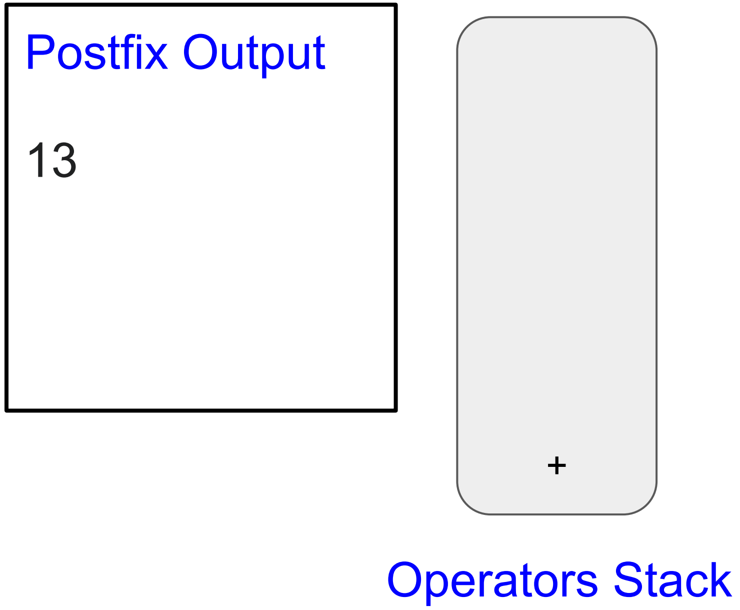
Parsing: 1**+**3*5-8/2

- Current Token +
    - Operator
- Rule #2: if our token is an operator,
  AND our stack is empty,
  push the operator token into the stack



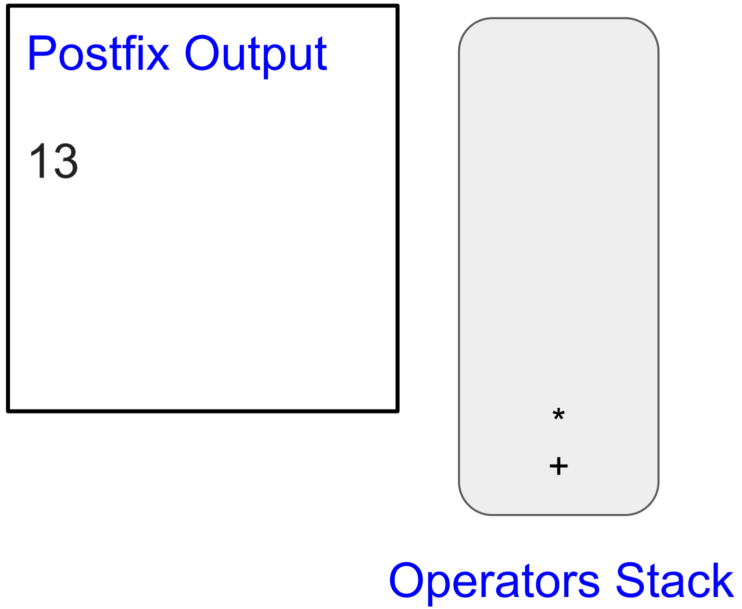
Parsing: 1+**3***5-8/2

- Current Token 3
  - Digit
- Rule #1: again, if our token is a digit, immediately add it to our output string



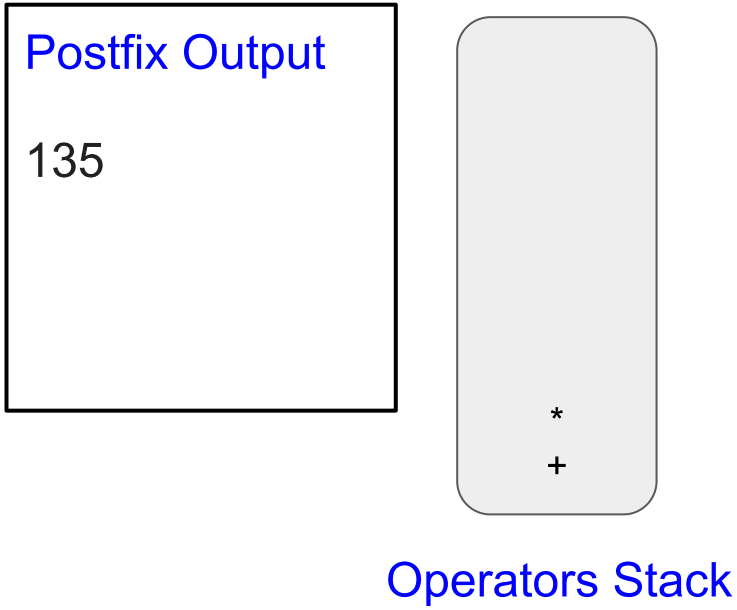
Parsing: 1+3*5-8/2

- Current Token *
    - Operator
- Rule #3: if we have a non-empty stack, and if our current operator token (i.e. the operator * in this case) is of **higher precedence** than the operator at the top of the stack, simply add it to the stack



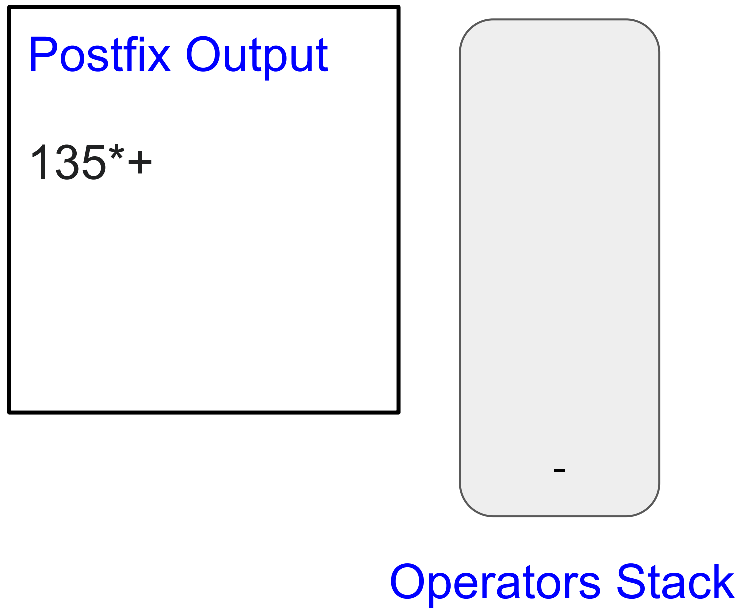
Parsing: 1+3***5**-8/2

- Current Token 5
  - Digit
- Rule #1: once again, if the token is a digit, simply add it to the output, if evaluating postfix notation



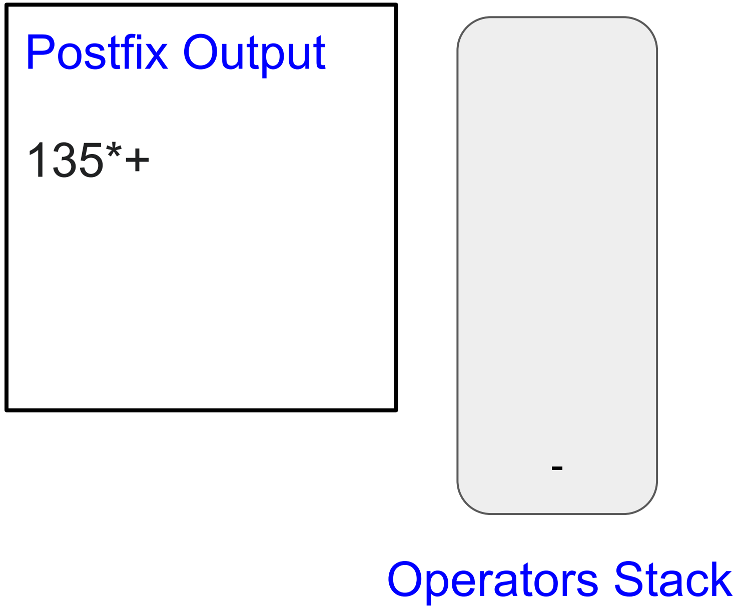
Parsing: 1+3*5-8/2

- Current Token -
  - Operator
- Rule #4: for as long as the precedence of our current operator (in this case, the operator -) is of lower or equal precedence to the operator at the top of the stack, we pop the top of the stack, and add it to the postfix notation string
  - - vs * ?   Smaller. Pop
  - - vs +?   Equal. Pop
- Finally, add current token to the stack



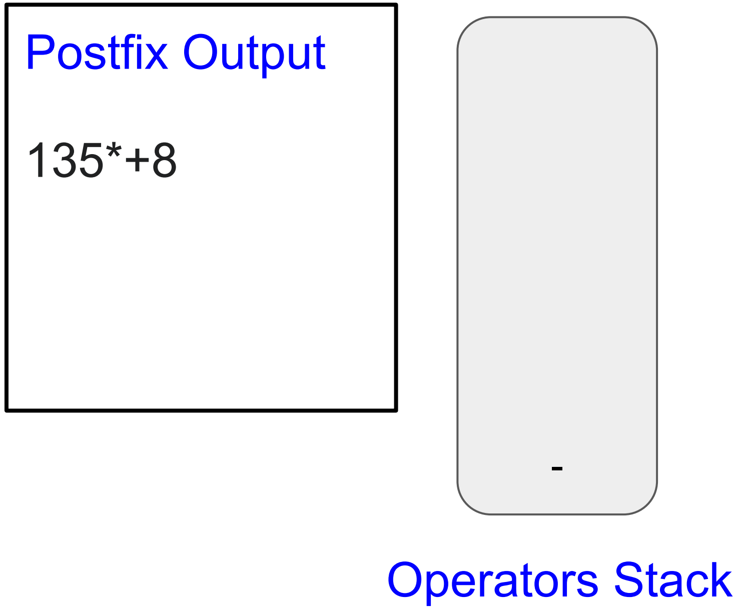
Parsing: 1+3*5-8/2

- Why was * popped?
  - Since * was the top element in the stack, and is of **higher** precedence than the current operator -, it must be added to the postfix string before -
  - Now 3 and 5 will be multiplied: 3*5 = 15
- Why was + popped?
  - It has **equal** precedence to the current operator -, and this operator has **left to right** associativity, so it must also be added to the postfix string before -
  - Now 1 and 15 will be added: 1 + 15 = 16



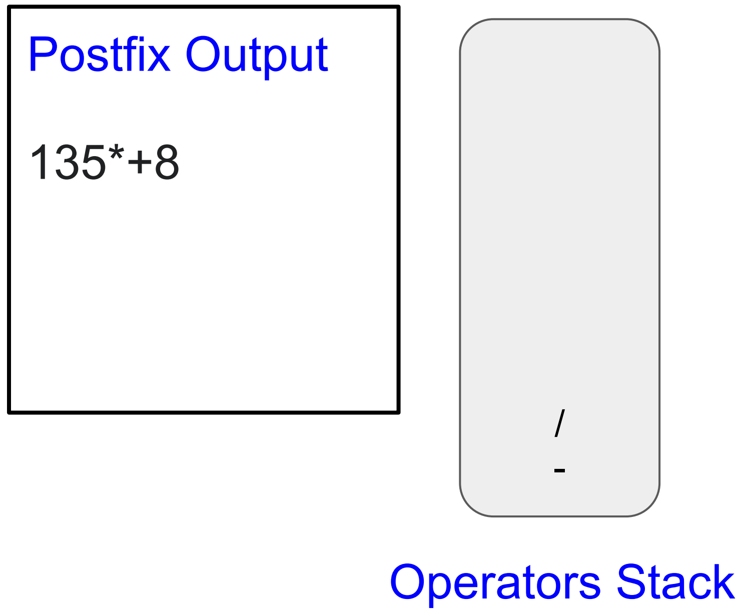
Parsing: 1+3*5-**8**/2

- Current Token 8
    - Digit
- Rule #1: if the token is a digit, add it to the postfix output string



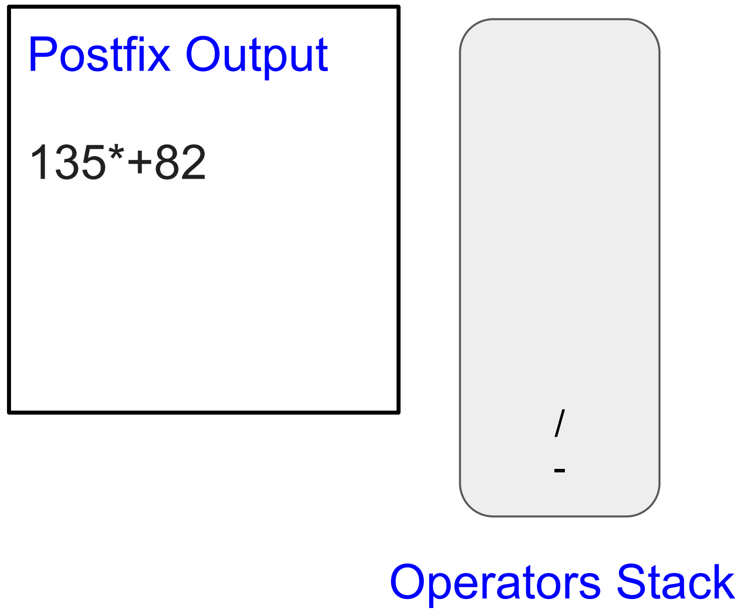
Parsing: 1+3*5-8/2

- Current Token /
  - Operator
- Rule #3: if the current token is an operator, AND is of **higher precedence** than the top of the stack, just add it to our postfix output



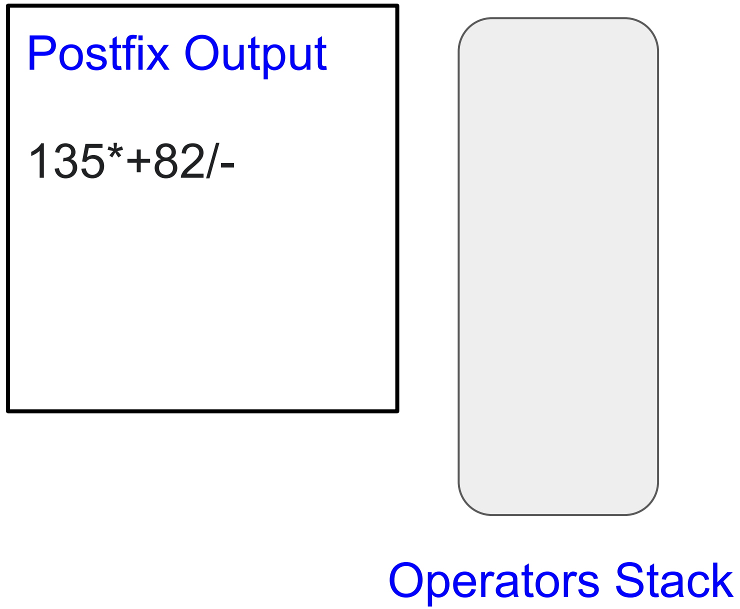
Parsing: 1+3*5-8/**2**

- Current Token 2
  - Digit
- Rule #1: if our current token is just a digit, add it to the postfix string output



Parsing: 1+3*5-8/2

- **No** current token!
- Rule #5: If no more tokens, we need to pop the operators stack in order, adding each token/operator popped to the postfix notation string
- The final expression is: 135*+82/-
- Overall, there are 5 simple rules to follow
- Your turn: take 20 minutes coding it



“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”