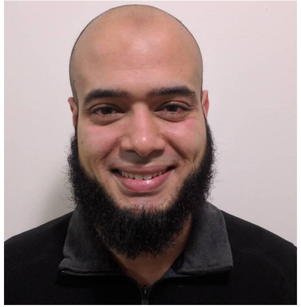
Data Structures

Infix to Postfix 2

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)



Expressions with parentheses

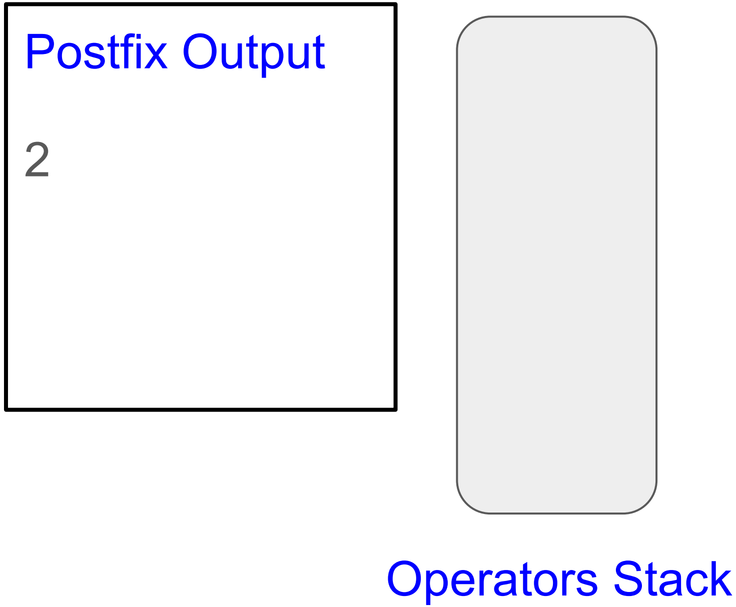
- For expression 2 + 3 * 4 ⇒ 234*+ [which same as 2 + (3*4)]
- What about: (2+3) * 4
  - Observation: the formulation inside the () is **independent** from outside
  - 2+3 ⇒? 23+
  - So we are like A*4 where A is 23+
  - So overall is A4* ⇒ 23+4*
- 2+3-((5+2)*3)
  - A = 5+2 ⇒ 52+
  - B = A*3 ⇒ A3* ⇒ 52+3*
  - 2+3-B ⇒ 23+B- ⇒ 23+52+3*-
- So: we can independently call postfix conversions on these deeper ones first?
  - ~O(n^2)

Greatness of stack

- We know stack has a good sense with reversing tasks
- But it also have a good sense with sub-recursive tasks
- Can we change the stack code to simply consider the () in O(n)
- The idea is simple
  - When you find (, just add it to the stack to indicate a sub-problem
  - Once found), then pop everything tell you find (
  - This way the same code solved the sub-problem (something) easily
  - The first) we meet represents one the deepest expressions

Parsing: **2**+3-((5+2)*3)

- Current Token 2
    - Digit
- Rule #1: If digit, add to output



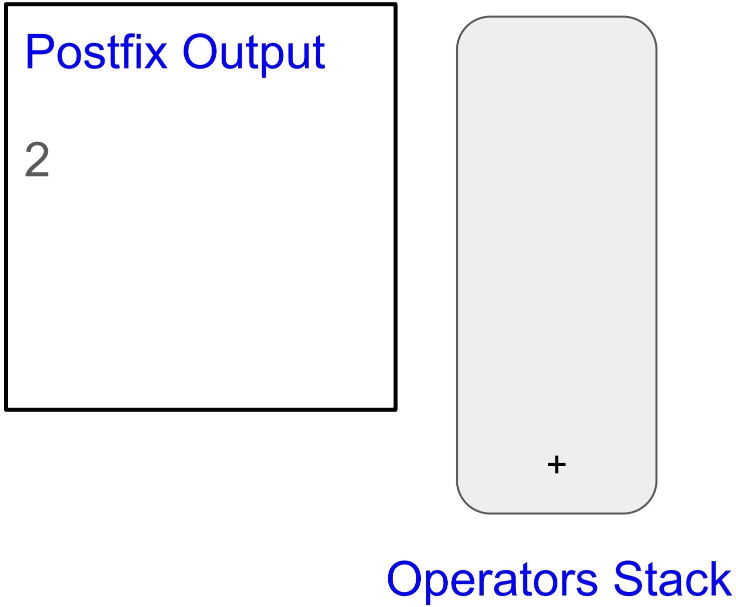
Postfix Output

2

Operators Stack

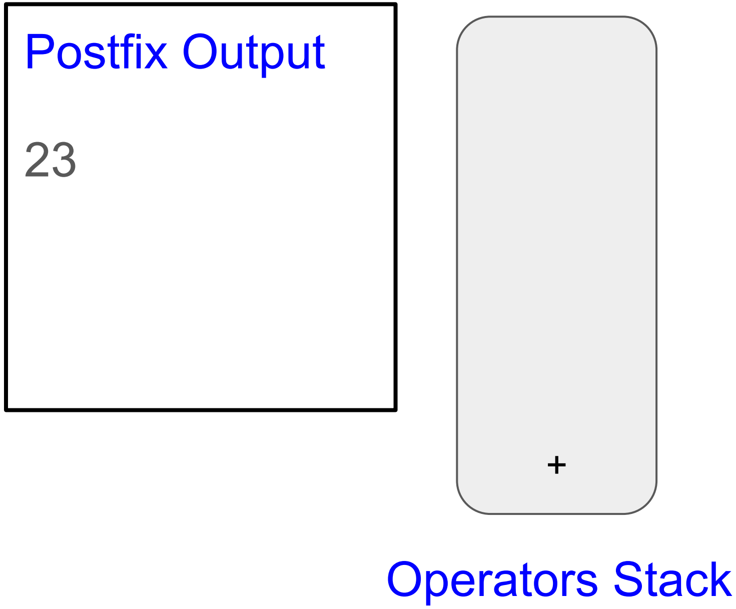
Parsing: 2+3-((5+2)*3)

- Current Token +
  - Operator
- Rule #2: If operator and empty stack, push in the stack



Parsing: 2+**3**-((5+2)*3)

- Current Token 3
  - Digit
- Rule #1: If digit, add to output



Postfix Output

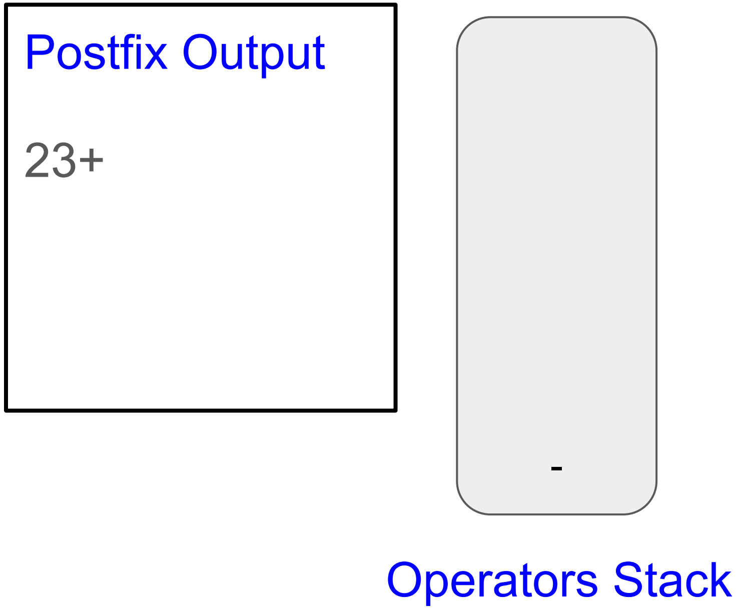
23

+

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token -
  - Operator
- Rule #4: as long as precedence (cur) <= top, pop top and add to postfix
- Finally, add current token to the stack



Postfix Output

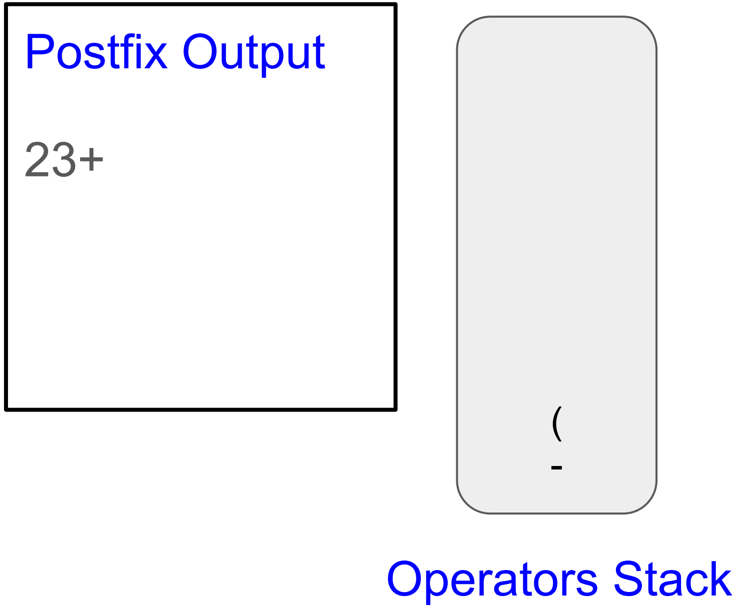
23+

-

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token (
    - Operator
- Rule: if (, just add it
    - Signals a new subproblem



Postfix Output

23+

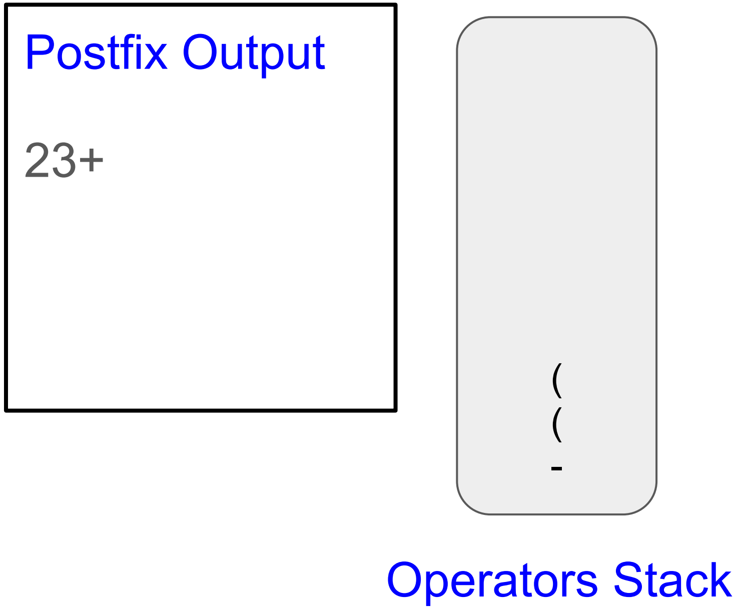
(

-

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token (
    - Operator
- Rule: if (, just add it
    - Signals a new subproblem



Postfix Output

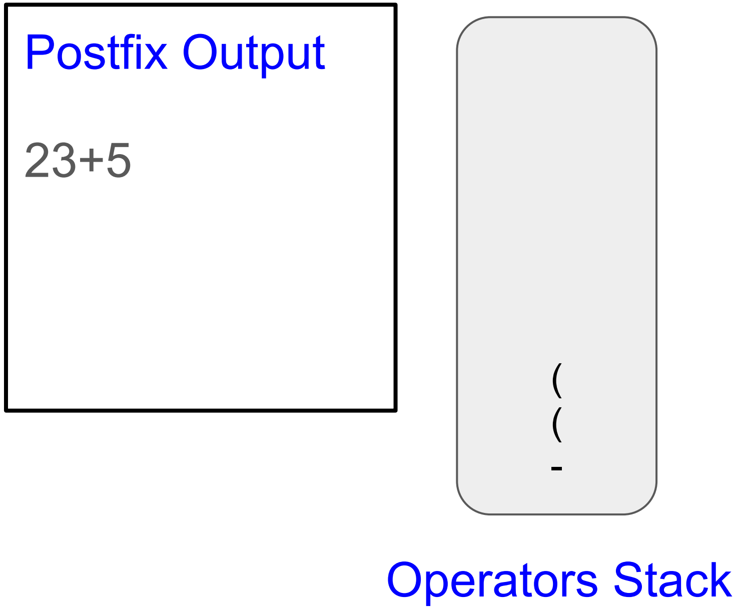
23+

(
(
-

Operators Stack

Parsing: 2+3-((**5**+2)*3)

- Current Token 5
    - Digit
- Rule #1: If digit, add to output



Postfix Output

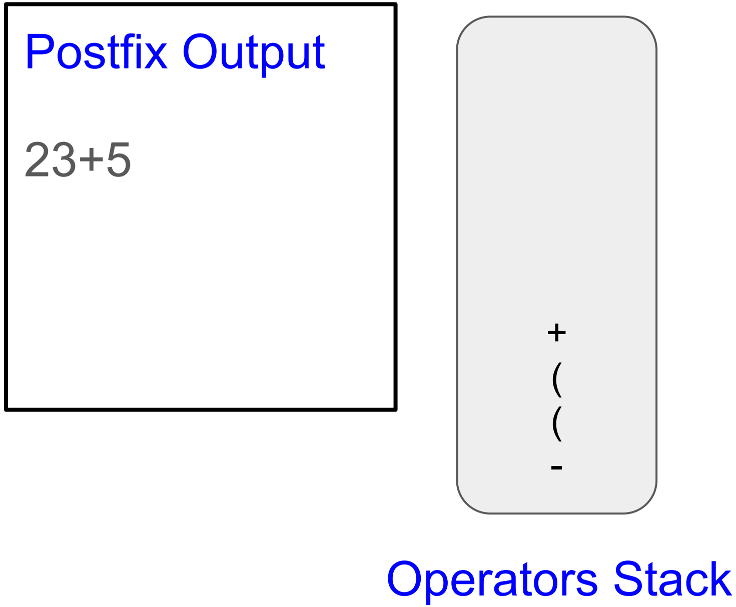
23+5

(
(
-

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token +
  - Operator
- Rule: If the top is (, just add the new operator to the stack



Postfix Output

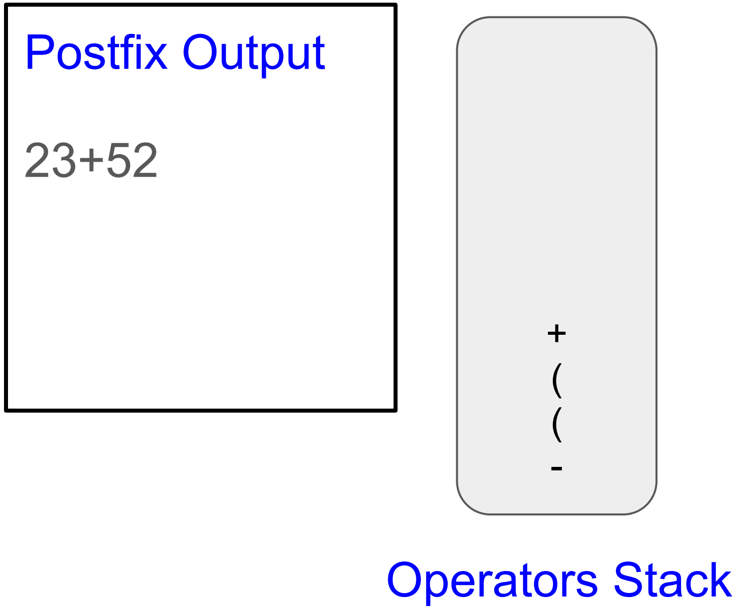
23+5

+
(
(
-

Operators Stack

Parsing: 2+3-((5+**2**)*3)

- Current Token 2
  - Digit
- Rule #1: If digit, add to output



Postfix Output

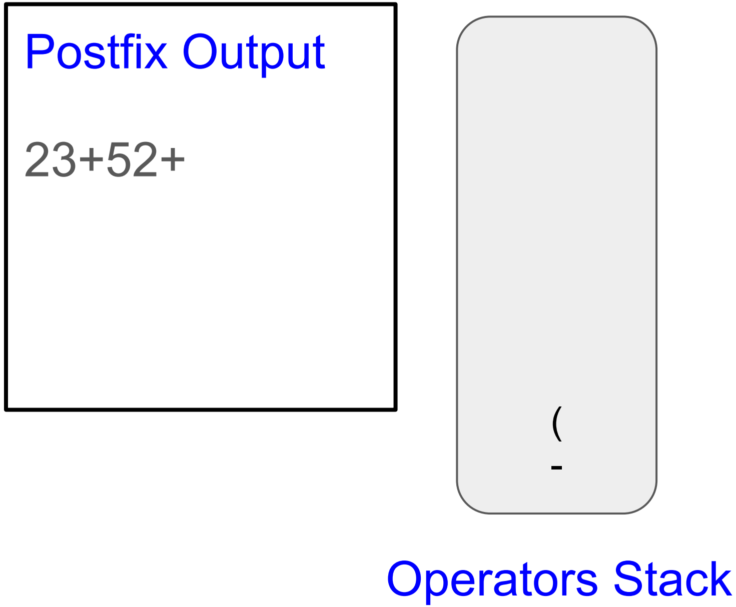
23+52

+
(
(
-

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token)
    - Digit
- Rule: If), then a sub-problem is done
    - Pop all operators tell find (which was sub-problem begin



Postfix Output

23+52+

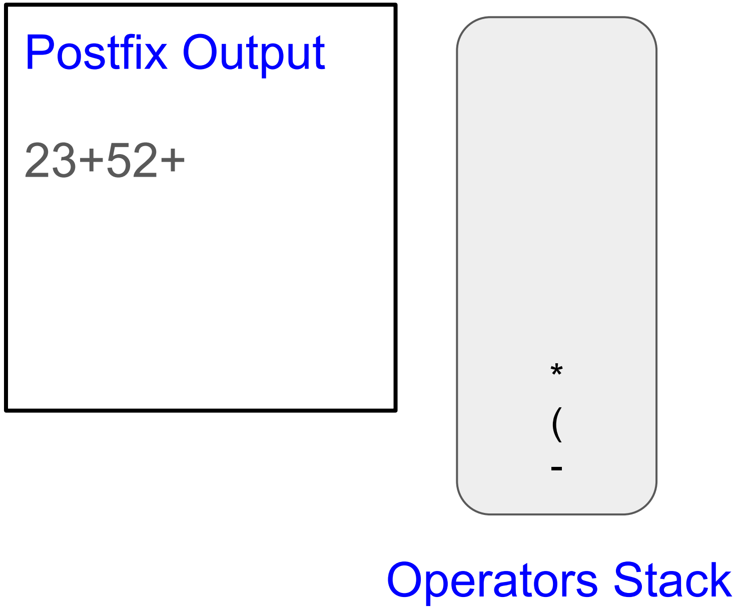
(

-

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token *
  - Operator
- Rule: If the top is (, just add the new operator to the stack



Postfix Output

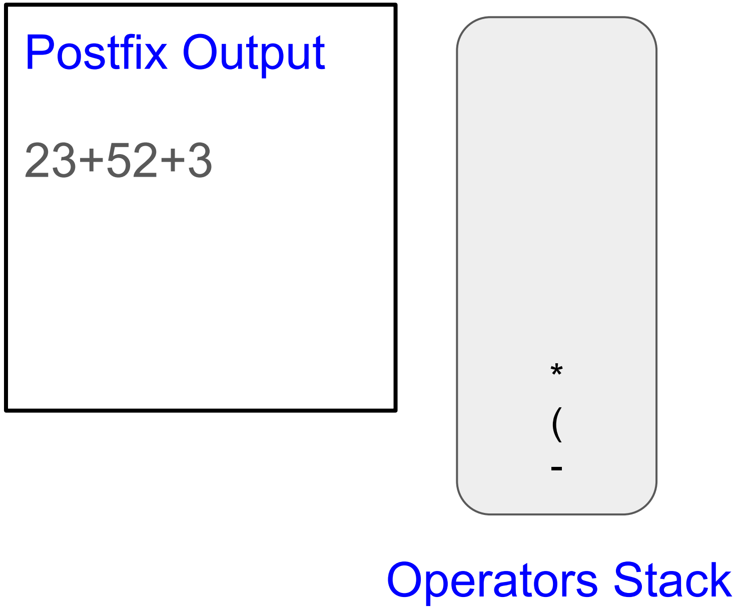
23+52+

*
(
-

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token 3
  - Digit
- Rule #1: If digit, add to output



Postfix Output

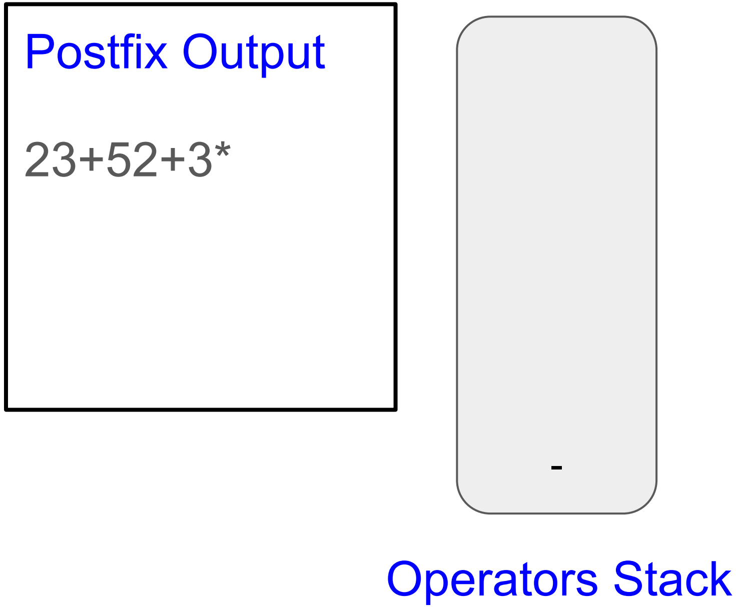
23+52+3

*
(
-

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token)
  - Digit
- Rule: If), then a sub-problem is done
  - Pop all operators tell find (which was sub-problem begin



Postfix Output

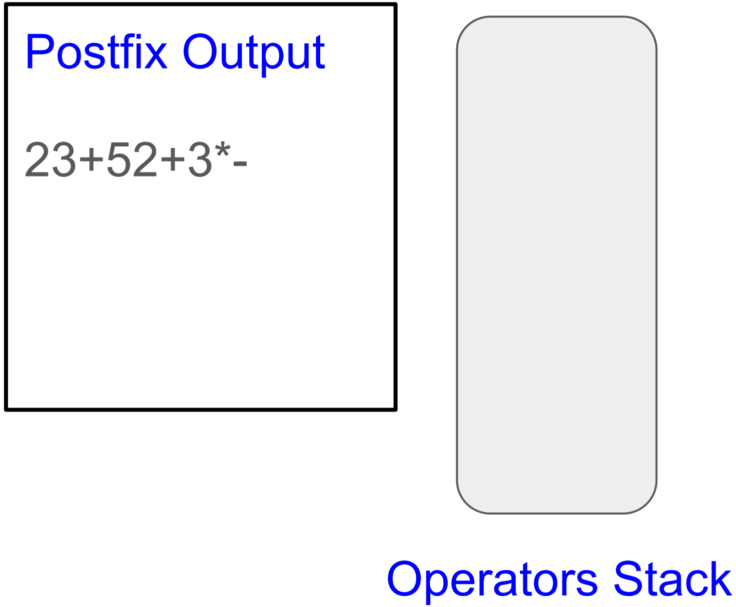
23+52+3*

-

Operators Stack

Parsing: 2+3-((5+2)*3)

- Current Token NONE
- Rule #5: If finished, in order pop each item and add to postfix
- Final expression 23+52+3*-
- Your turn: take 10 minutes to modify our previous code

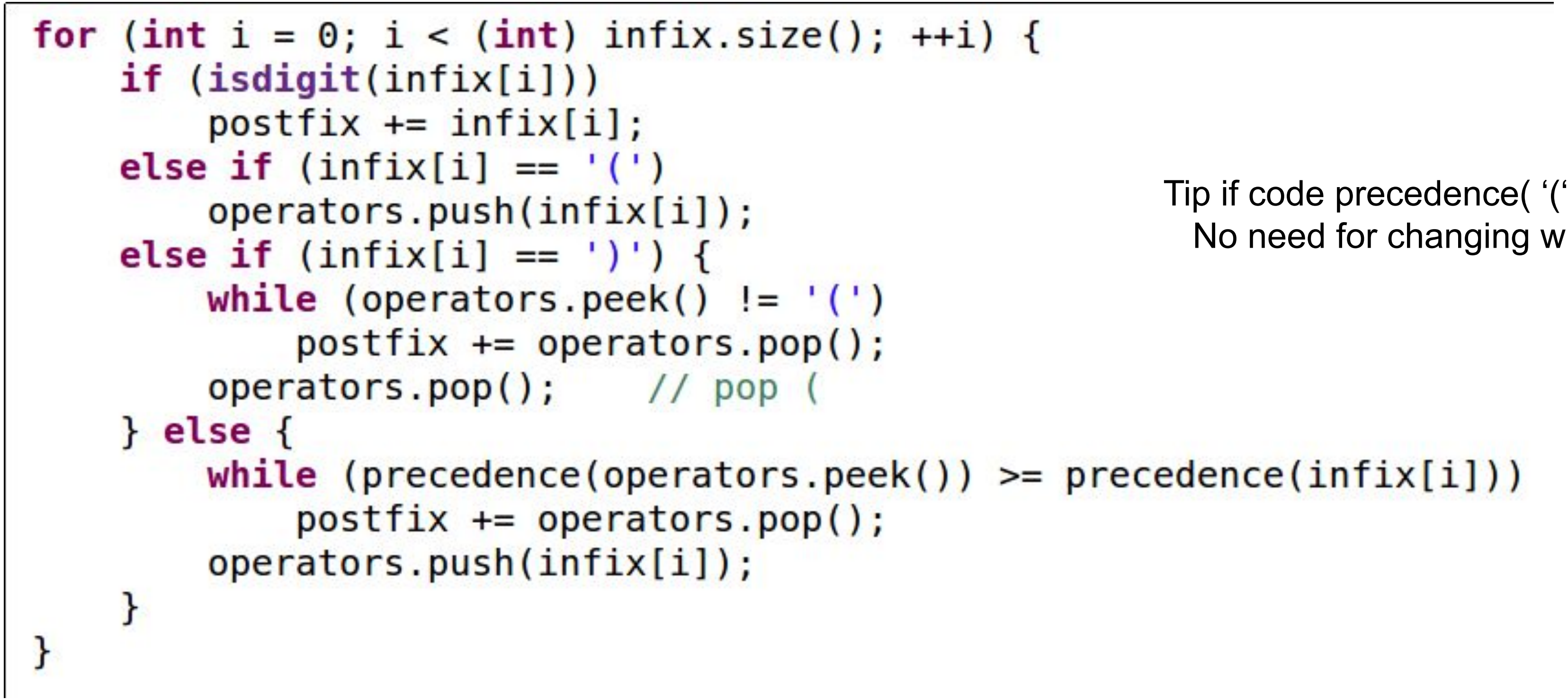


Postfix Output

23+52+3*-

Operators Stack

Infix to Postfix



```
for (int i = 0; i < (int) infix.size(); ++i) {
    if (isdigit(infix[i]))
        postfix += infix[i];
    else if (infix[i] == '(')
        operators.push(infix[i]);
    else if (infix[i] == ')') {
        while (operators.peek() != '(')
            postfix += operators.pop();
        operators.pop();    // pop (
    } else {
        while (precedence(operators.peek()) >= precedence(infix[i]))
            postfix += operators.pop();
        operators.push(infix[i]);
    }
}
```

Tip if code precedence(‘(‘) = 0
No need for changing while

Right to Left associativity

- So far we handled operators: + - * /
  - All left to right, meaning if 2 operators of **equal** precedence then **most left** one applied first
- In terms of the algorithm, we learned that 2 cases add to the stack
  - Rule A: Stack precedence(top) > precedence(cur), e.g. * vs +
  - Rule B: Stack precedence(top) == precedence(cur), e.g. + vs + and + vs -
- But what about operator like ^
  - In math: 2^3^4 is evaluated 2^**(3^4)** NOT (2^3)^4
  - This is **right to left** precedence, that is the **most right** ^ is applied first
  - ^ has higher precedence that + - * /
    - Rule A applies e.g. ^ vs +   and ^ vs *
    - However, *and the only difference*, rule B doesn’t apply (don’t pop from stack): *Homework*

M-M Conversions

- Given that we have 3 types, we can have many to many conversions
  - **Infix to postfix**, Infix to Prefix
  - Postfix to Infix, Postfix to prefix
  - Prefix to Infix, Prefix to Postfix
- Practically, **infix to postfix** is important to ease **evaluating** expressions
- Feel free to think about some of these conversions and how to code

In reality

- For simplicity, we assumed constraints on expressions
  - Educationally enough to administer the concepts
- In practice an expression could be like: (-25+5log(11! *5^3^12))
  - Observe numbers are **several digits**
  - A number could be negative: now - can be both **binary and unary**
    - *Unary operator has higher precedence than ^ + - * /*
  - Observe **functions** such as log and factorial
  - It is more of *implementation skills* rather than other stack concepts
  - One challenge how to parse: we need something to give us separate inputs
    - (-25 + 5 log (11 ! * 5 ^ 3 ^ 2))
    - We call every parsed item **token**, this is the most annoying part to extract

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”