COSC 2306
Data Programming

Stacks

Practice : reverse string

```
def reverse_string(str):
    my_stack = Stack()

    for ch in str:
        my_stack.push(ch)

    rev_str = ''
    while not my_stack.isEmpty():
        ch = my_stack.pop()
        rev_str = rev_str + ch

    return rev_str
```

Postfix Expressions Calculator

- Arithmetic notations
  - Infix notation: operator between operands
    - 2 + 3
  - Prefix (Polish) notation: operator precedes operands
    - + 2 3
  - Postfix/Reverse Polish notation: operator follows operands
    - 2 3 +
- Stack use in compliers
  - Translate infix expressions into some form of postfix notation
  - Translate postfix expression into machine code

Postfix Expressions Calculator

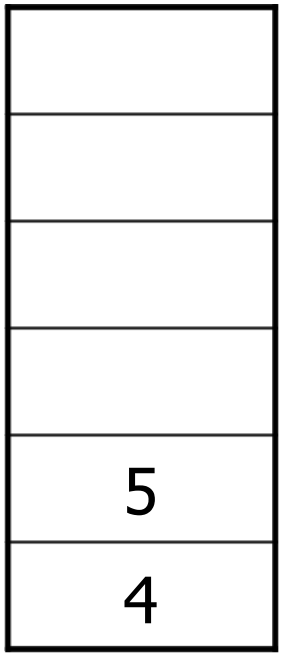
- Computationally convenient, no need of parenthesis
- Rules:
  - If the input is a number, push in stack
  - If the input is an operator, pop 2 items, compute result, push result in stack
- Example: 4 5 * 3 3 * +

Postfix Expressions Calculator

- Example: 4 5 * 3 3 * +

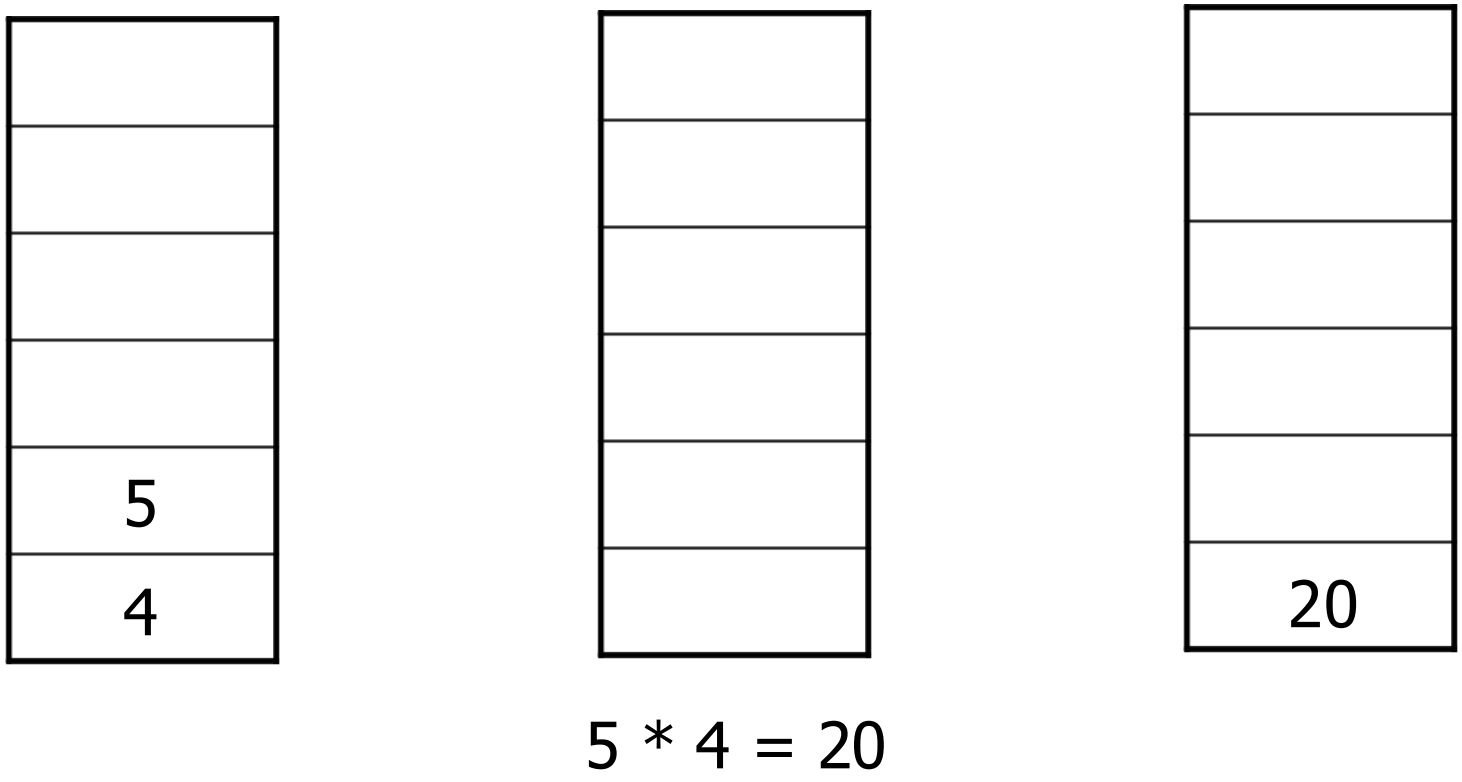
Postfix Expressions Calculator

- Example: 4 5 * 3 3 * +



Postfix Expressions Calculator

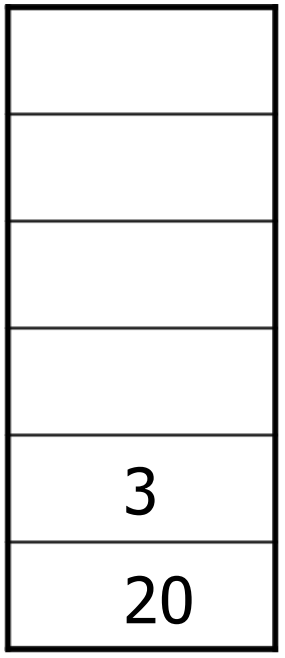
- Example: 4 5 * 3 3 * +



5 * 4 = 20

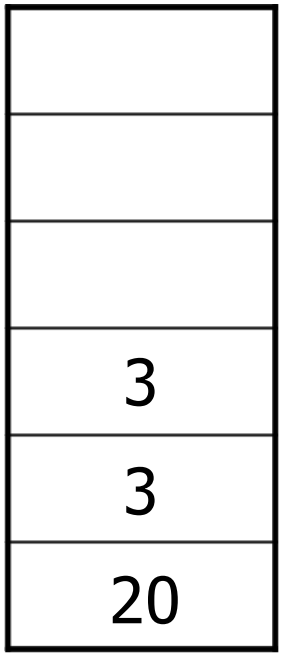
Postfix Expressions Calculator

- Example: 4 5 * 3 3 * +



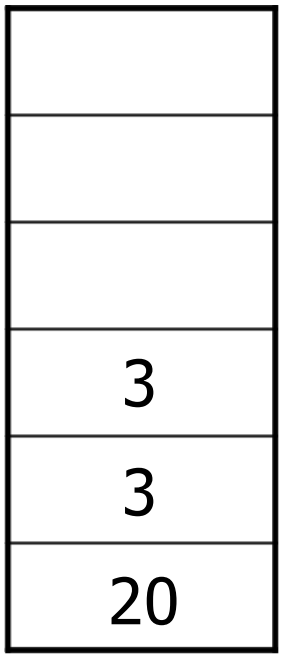
Postfix Expressions Calculator

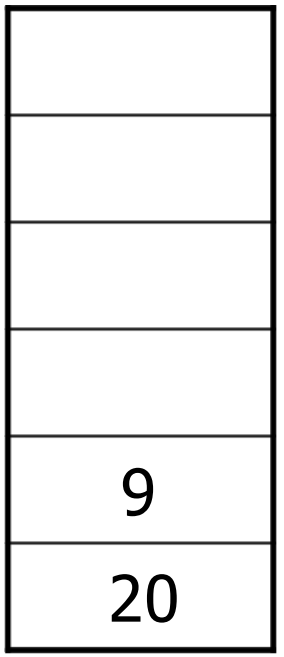
- Example: 4 5 * 3 3 * +



Postfix Expressions Calculator

- Example: 4 5 * 3 3 * +

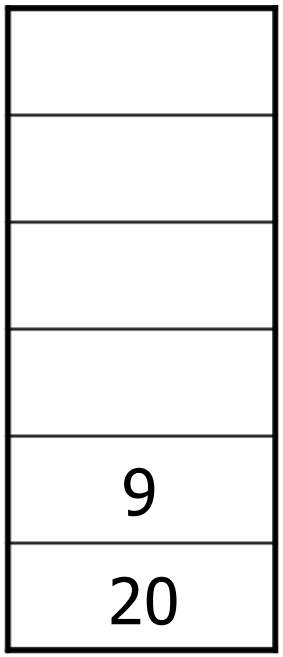


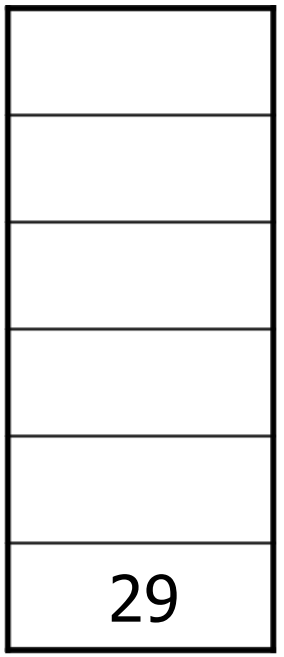


3 * 3 = 9

Postfix Expressions Calculator

- Example: 4 5 * 3 3 * +





20 + 9 = 29

Postfix Calculator Implementation

```
def evaluate_postfix(str):

    my_stack = Stack()
    str = str.replace(' ','')
    numbers = '0123456789'
    operators = '+-*/'

...

    final = my_stack.pop()
    return final

print(evaluate_postfix('5 4 + 3 3 + *'))

>54
```

- If the input is a number, push in stack
- If the input is an operator, pop 2 items, compute result, push result in stack

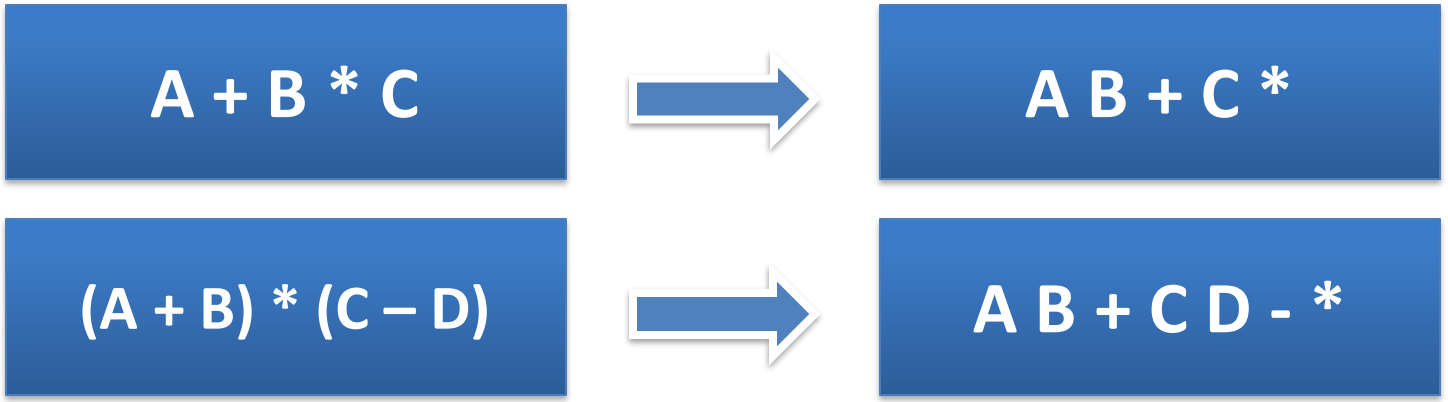
Postfix Calculator Implementation

```
def evaluate_postfix(str):

    my_stack = Stack()
    str = str.replace(' ','')
    numbers = '0123456789'
    operators = '+-*/'

    for ch in str:
        if ch in numbers:
            my_stack.push(int(ch))
        elif ch in operators:
            vals = []
            i = 0
            while not my_stack.isEmpty() and i < 2:
                vals.append(my_stack.pop())
                i += 1
            if ch == '+':
                result = vals[0] + vals[1]
                my_stack.push(result)
            if ch == '-':
                result = vals[0] - vals[1]
                my_stack.push(result)
            if ch == '*':
                result = vals[0] * vals[1]
                my_stack.push(result)
            if ch == '/':
                result = vals[0] / vals[1]
                my_stack.push(result)

    final = my_stack.pop()
    return final

print(evaluate_postfix('5 4 + 3 3 + *'))
```

- If the input is a number, push in stack
- If the input is an operator, pop 2 items, compute result, push result in stack

```
>54
```

From infix to postfix

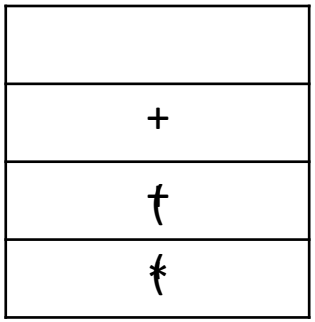


- Start from an operation complete with brackets
- If left bracket, push in stack
- If operand, add to expression
- If operator, push in stack
  - pop other operators from stack if they have higher precedence
  - push the current operator to stack
- If right bracket, pop from stack until left bracket is found
  - add operators to expression
  - discard the brackets
- Pop and write remaining operators in the stack after scanning

From infix to postfix

- Example: (5 + 4) * (3 + 3)

5 4 + 3 3 + *



Start from an operation complete with brackets

If left bracket, push in stack

If operand, add to expression

If operator, push in stack

pop other operators from stack if they have higher precedence

push the current operator to stack

If right bracket, pop from stack until left bracket is found

add operators to expression

discard the brackets

- Pop and write remaining operators in the stack after scanning

- You try: (A + B) * C - (D - E) * (F + G)

- Solution: A B + C * D E – F G + * -

Another Stack Implementation

```
class Stack:
    def __init__(self):
        self._items = []

    def is_empty(self):
#       return self._items == []
        if len(self._items) > 0:
            return False
        else:
            return True

    def push(self, item):
#       self._items.append(item)        O(1)
        self._items.insert(0, item)     #O(n)

    def pop(self):
#       return self._items.pop()        O(1)
        return self._items.pop(0)       #O(n)

    def peek(self):
#       return self.items[len(self.items) - 1]
        return self._items[0]

    def size(self):
        return len(self._items)
```