WORKSHEET

A6.6 Reverse Polish notation

Theory of computation · A level · OCR H446 2.3.1, AQA 7517 4.3.3.1 · about 25 min

BugBotLab

Name _____ Class _____ Date _____

What this lesson is about

Infix and postfix, converting both ways, evaluating RPN with a stack, and the shunting-yard algorithm.

- Infix and postfix
- Why and where RPN is used
- Converting infix to RPN by hand
- Evaluating RPN with a stack
- Converting infix to RPN with a program

Questions

6 marks in all

1. Convert (4 + 6) × 3 − 2 to Reverse Polish notation. Use * for ×, and separate tokens with spaces. [1 mark]

2. Evaluate the RPN expression 7 2 3 * - 4 + [1 mark]

3. When evaluating RPN with a stack, an operator pops two values. For 9 3 -, which is the right-hand operand? [1 mark]

- ☐ **A** 9, the first popped
- ☐ **B** 3, the first popped
- ☐ **C** 9, the second popped
- ☐ **D** 3, the second popped

4. Why is Reverse Polish notation used? [1 mark]

Tick every answer that is true.

- ☐ **A** It needs no brackets
- ☐ **B** It needs no operator precedence rules
- ☐ **C** It can be evaluated left to right with a stack
- ☐ **D** It uses fewer operands than infix

5. Convert the RPN expression 5 1 2 + 4 * + to infix, with only the brackets that are needed. Use * for ×. [1 mark]

6. What does this program print? [1 mark]

```
stack = []
for token in "3 4 2 * +".split():
    if token in "+*":
        right = stack.pop()
        left = stack.pop()
        stack.append(left + right if token == "+" else left * right)
    else:
        stack.append(int(token))
    print(stack)
```

The task: an RPN calculator

Write both halves of a calculator for expressions with whole numbers, `+`, `-`, `*`, `/` and brackets. In every expression, tokens are separated by single spaces, including the brackets. - `to_rpn(infix)` takes an infix string such as `"( 3 + 4 ) * 2"` and returns the RPN string with single spaces, such as `"3 4 + 2 *"`. Use the shunting-yard algorithm with a stack. All four operators work left to right. - `evaluate(rpn)` takes an RPN string and returns its value, using a list as a stack. `/` is ordinary division; every test divides exactly. - Do not use Python's `eval`. - For each expression in `tests`, in order, print the infix, `->`, the RPN, `=`, then the value as a whole number. For example `3 + 4 * 2 -> 3 4 2 * + = 11`. Four lines in all.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

tests = ["3 + 4 * 2", "( 3 + 4 ) * 2", "( 5 - 1 ) * ( 2 + 6 ) / 4", "8 - 2 - 3"]

def to_rpn(infix):
    output = []
    stack = []
    return ""
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a6-6-reverse-polish-notation/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Convert `(a + b) × (c − d) / e` to RPN by hand, then check it with your `to_rpn`.
2. Make `evaluate` print `invalid` instead of crashing for `4 +` and for `1 2 3 +`.
3. Add `^` for powers. Powers work right to left: `2 ^ 3 ^ 2` is `2 ^ (3 ^ 2)`. What must change in step 2 of the algorithm?