Shunting-yard-Algorithmus

J. Beem, V. Paluri, D. Serban

23. Jänner 2026

Überblick

1 Problemstellung

Überblick

1. Problemstellung
2. Reverse Polish Notation

Überblick

1. Problemstellung
2. Reverse Polish Notation
3. Shunting-yard-Algorithmus
   - Grundidee
   - Visualisierung
   - Laufzeit
   - Implementierung in Java

Problemstellung

Wie verarbeitet ein Computer mathematische Ausdrücke?

Problemstellung

Wie verarbeitet ein Computer mathematische Ausdrücke?

Beispiel: $3 + 4 \cdot 5$

Problemstellung

Wie verarbeitet ein Computer mathematische Ausdrücke?

Beispiel: $3 + 4 \cdot 5$

- Zuerst $4 \cdot 5 = 20$, dann $3 + 20 = 23$

Problemstellung

Wie verarbeitet ein Computer mathematische Ausdrücke?

Beispiel: $3 + 4 \cdot 5$

- Zuerst $4 \cdot 5 = 20$, dann $3 + 20 = 23$
- Computer lesen von links nach rechts

Problemstellung

Wie verarbeitet ein Computer mathematische Ausdrücke?

Beispiel: $3 + 4 \cdot 5$

- Zuerst $4 \cdot 5 = 20$, dann $3 + 20 = 23$
- Computer lesen von links nach rechts

Woher weiß der Computer, welche Operation zuerst ausgeführt werden soll?

Herausforderungen

- Operatorpräzedenz
  - Potenzieren vor Multiplizieren vor Addieren

Herausforderungen

- Operatorpräzedenz
  - Potenzieren vor Multiplizieren vor Addieren
- Klammern ändern die Reihenfolge
  - $(3 + 4) \cdot 5 = 35$
  - $3 + 4 \cdot 5 = 23$

Herausforderungen

- Operatorpräzedenz
  - Potenzieren vor Multiplizieren vor Addieren
- Klammern ändern die Reihenfolge
  - $(3 + 4) \cdot 5 = 35$
  - $3 + 4 \cdot 5 = 23$
- Verschachtelte Ausdrücke
  - $((15/(7 - (1 + 1))) \cdot 3) - (2 + (1 + 1))$

Herausforderungen

- Operatorpräzedenz
  - Potenzieren vor Multiplizieren vor Addieren
- Klammern ändern die Reihenfolge
  - $(3 + 4) \cdot 5 = 35$
  - $3 + 4 \cdot 5 = 23$
- Verschachtelte Ausdrücke
  - $((15/(7 - (1 + 1))) \cdot 3) - (2 + (1 + 1))$

Gibt es eine einfachere Notation für Computer?

Reverse Polish Notation (Postfix)

Reverse Polish Notation (Postfix)

Beispiele:

- 1 + 3 $\rightarrow$ 1 3 +

Reverse Polish Notation (Postfix)

Beispiele:

- `1 + 3` $\rightarrow$ `1 3 +`
- `(4 - 5) * 6` $\rightarrow$ `4 5 - 6 *`

Reverse Polish Notation (Postfix)

Beispiele:

- `1 + 3` $\rightarrow$ `1 3 +`
- `(4 - 5) * 6` $\rightarrow$ `4 5 - 6 *`

- RPN ist deutlich einfacher für Computer

Reverse Polish Notation (Postfix)

Beispiele:

- `1 + 3` $\rightarrow$ `1 3 +`
- `(4 - 5) * 6` $\rightarrow$ `4 5 - 6 *`

- RPN ist deutlich einfacher für Computer
  - Keine Klammern

Reverse Polish Notation (Postfix)

Beispiele:

- `1 + 3` $\rightarrow$ `1 3 +`
- `(4 - 5) * 6` $\rightarrow$ `4 5 - 6 *`

- RPN ist deutlich einfacher für Computer
  - Keine Klammern
  - Keine Operatorpräzedenz-Regeln

Reverse Polish Notation (Postfix)

Beispiele:

- `1 + 3` $\rightarrow$ `1 3 +`
- `(4 - 5) * 6` $\rightarrow$ `4 5 - 6 *`

- RPN ist deutlich einfacher für Computer
  - Keine Klammern
  - Keine Operatorpräzedenz-Regeln
  - Lineare Verarbeitung

RPN Evaluierung

	Stack
15 7 1 1 + - / 3 * 2 1 1 + + -	[15, 7, 2]
15 7 2 - / 3 * 2 1 1 + + -	[15, 5]
15 5 / 3 * 2 1 1 + + -	[3]
3 3 * 2 1 1 + + -	[9]
9 2 1 1 + + -	[9, 2, 2]
9 2 2 + -	[9, 4]
9 4 -	[5]

$\rightarrow 5$

Shunting-yard-Algorithmus

dt.: Rangierbahnhof-Algorithmus

Shunting-yard-Algorithmus

dt.: Rangierbahnhof-Algorithmus

- Erfunden von Edsger Dijkstra (1961)
- Ziel: Infix $\rightarrow$ Postfix (RPN)
  - oder: Infix $\rightarrow$ Abstract Syntax Tree
  - oder: Infix $\rightarrow$ Stackmaschinencode

Wikimedia Commons (Public Domain)

Shunting-yard-Algorithmus

dt.: Rangierbahnhof-Algorithmus

- Erfunden von Edsger Dijkstra (1961)
- Ziel: Infix $\rightarrow$ Postfix (RPN)
  - oder: Infix $\rightarrow$ Abstract Syntax Tree
  - oder: Infix $\rightarrow$ Stackmaschinencode

Wikimedia Commons (Public Domain)

Eine mögliche Anwendung (Taschenrechner):

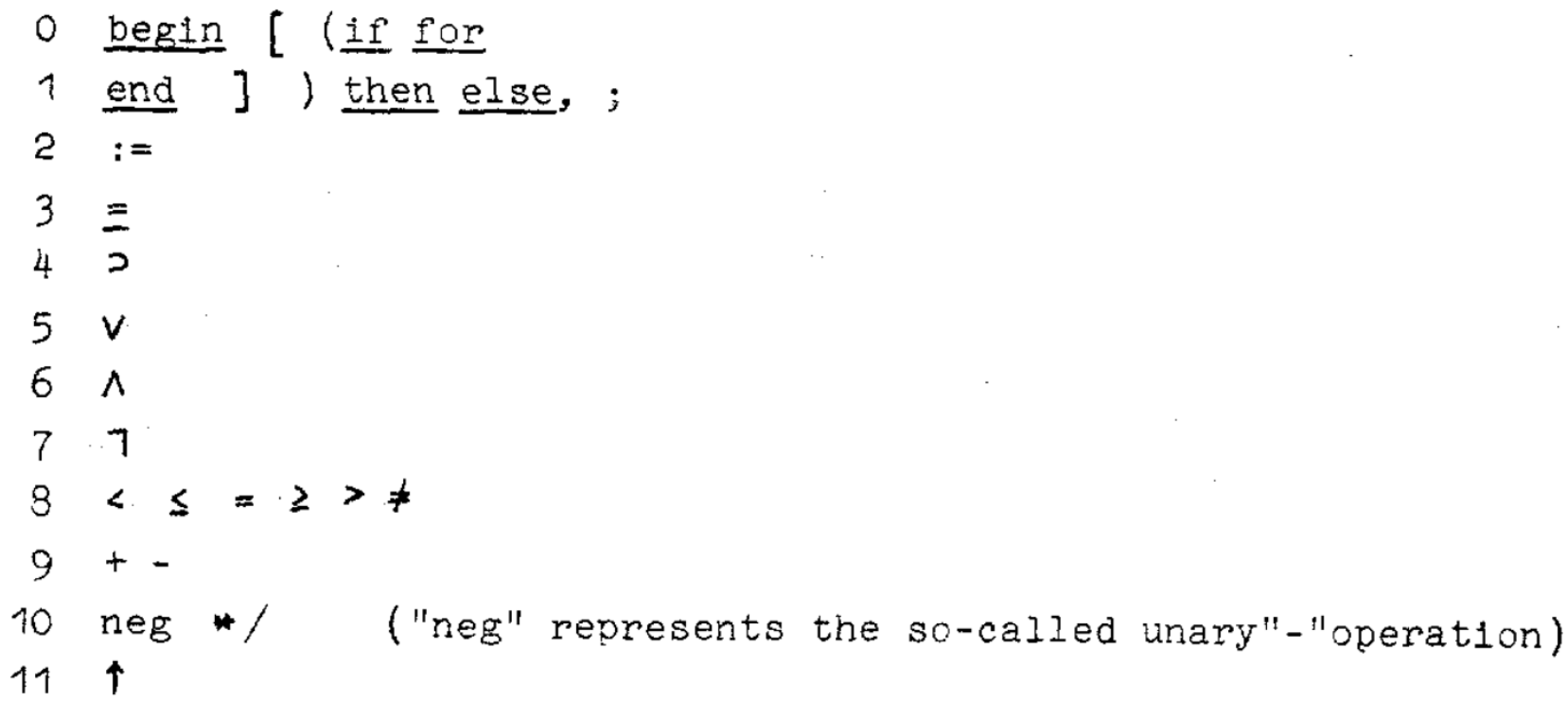
Eingabe $\longrightarrow$ `"(6 - 4) * 3"`

Lexer $\longrightarrow$ `["(", 6, "-", 4, ")", "*", 3]`

SYA $\longrightarrow$ `[6, 4, "-", 3, "*"]`

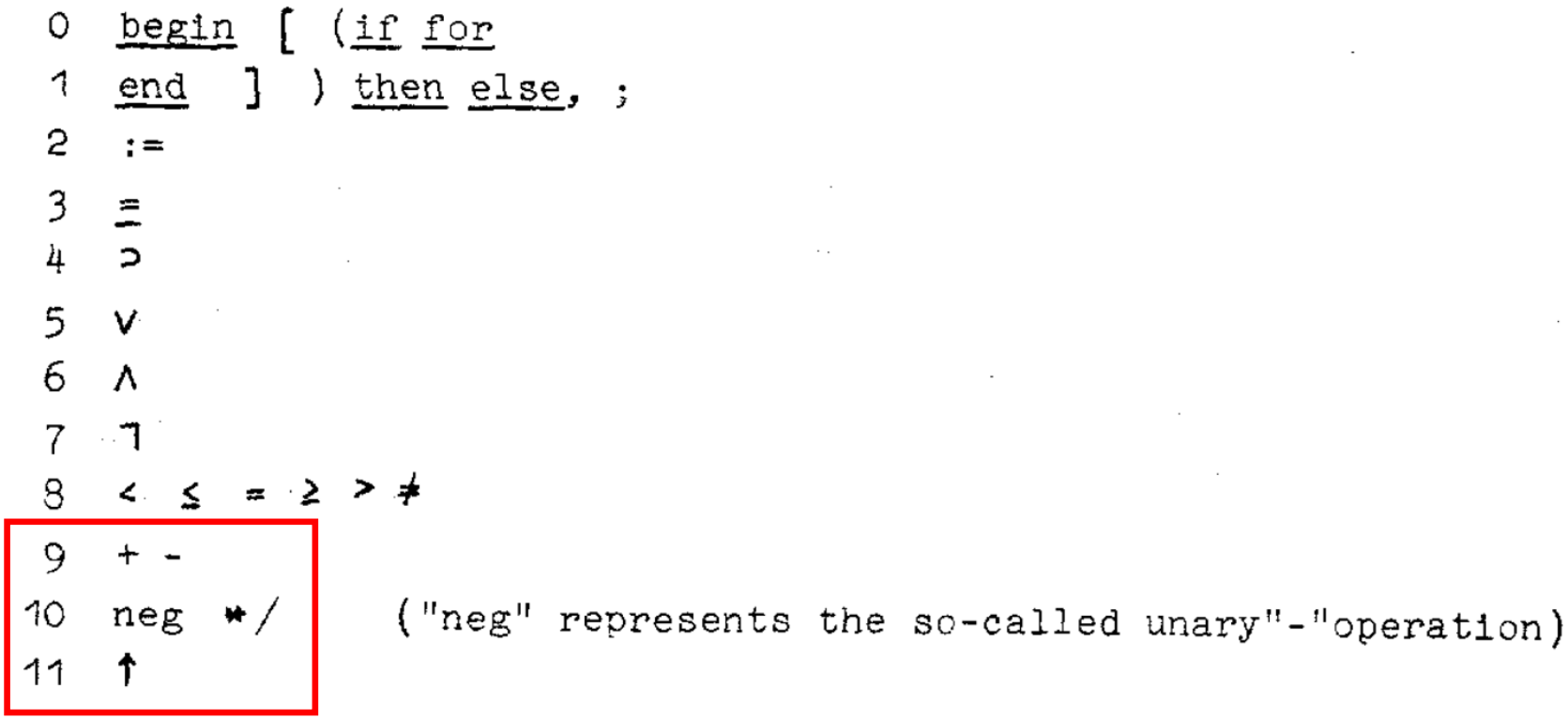
Eval. $\longrightarrow$ `6`

Ursprüngliche Anwendung



Edsger W. Dijkstra (1961). *Algol 60 translation : An Algol 60 translator for the x1 and Making a translator for Algol 60*. Techn. Ber. 35. Mathematisch Centrum, Amsterdam. URL: http://www.cs.utexas.edu/users/EWD/MCReps/MR35.PDF

Ursprüngliche Anwendung



Edsger W. Dijkstra (1961). *Algol 60 translation : An Algol 60 translator for the x1 and Making a translator for Algol 60*. Techn. Ber. 35. Mathematisch Centrum, Amsterdam. URL: http://www.cs.utexas.edu/users/EWD/MCReps/MR35.PDF

Operatoren

Operator	Argumente	Assoziativität	Präzedenz
+	2	links	1
-	2	links	1
*	2	links	2
/	2	links	2
neg	1		3
^	2	rechts	4

Regeln

Wir haben:

- **Eingabe** als Array
- **Stack**
- **Ausgabe** als Array

Regeln

Wir haben:

- **Eingabe** als Array
- **Stack**
- **Ausgabe** als Array

Da in unserem Fall Assoziativität keine große Rolle spielt, gilt:

$$\begin{array}{lcl} \text{Zahl} & \rightarrow & \text{Ausgabe} \\ \text{Operator} & \rightarrow & \text{Stack} \\ \text{Klammer} & \rightarrow & \text{Stack} \end{array}$$

Regeln

Wir haben:

- **Eingabe** als Array
- **Stack**
- **Ausgabe** als Array

Da in unserem Fall Assoziativität keine große Rolle spielt, gilt:

$$\begin{array}{lcl} \text{Zahl} & \rightarrow & \text{Ausgabe} \\ \text{Operator} & \rightarrow & \text{Stack} \\ \text{Klammer} & \rightarrow & \text{Stack} \end{array}$$

- Ein Operator darf nur auf den Stack gelegt werden, wenn die Präzedenz wirklich höher ist als die Präzedenz des obersten Elements auf dem Stack – sonst solange `pop()` aufrufen

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe:

1	+	2	*	4	-	8

Stack: `leer`

Ausgabe: `leer`

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `+` `2` `*` `4` `-` `8`

Stack: `leer`

Ausgabe: `1`

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `2` `*` `4` `-` `8`

Stack: `+`

Ausgabe: `1`

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `*` `4` `-` `8`

Stack: `+`

Ausgabe: `1` `2`

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: | 4 | - | 8 |

Stack: | + | * |

Ausgabe: | 1 | 2 |

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: | - | 8 |

Stack: | + | * |

Ausgabe: | 1 | 2 | 4 |

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `8`

Stack: `-`

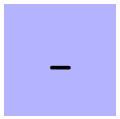
Ausgabe: `1` `2` `4` `*` `+`

1. Beispiel

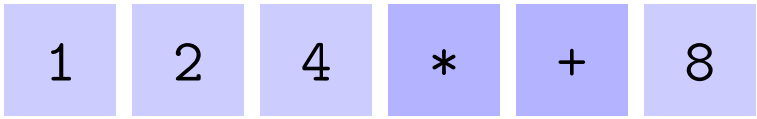
$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack:



Ausgabe:



1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

1	2	4	*	+	8	-

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

1	2	4	*	+	8	-

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

1	2	4	*	+	8	-

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

1	8	+	8	-

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

1	8	+	8	-

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack: `leer`

Ausgabe: `9` `8` `-`

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack: `leer`

Ausgabe: `9` `8` `-`

1. Beispiel

$$1 + 2 \cdot 4 - 8 = 1 + 8 - 8 = 1$$

Eingabe: `leer`

Stack: `leer`

Ausgabe: `1`

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe:

3	*	(	2	+	4	)	/	2	^	2

Stack: `leer`

Ausgabe: `leer`

2. Beispiel

$$\frac{3 \cdot (2+4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe:

*	(	2	+	4	)	/	2	^	2

Stack: `leer`

Ausgabe:

3

2. Beispiel

$$\frac{3 \cdot (2+4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe:

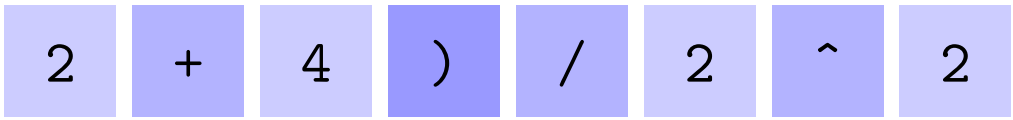
Stack:

Ausgabe:

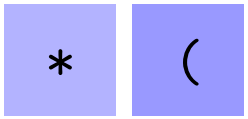
2. Beispiel

$$\frac{3 \cdot (2+4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe:



Stack:



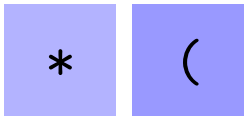
Ausgabe:

2. Beispiel

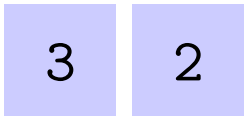
$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe:

Stack:



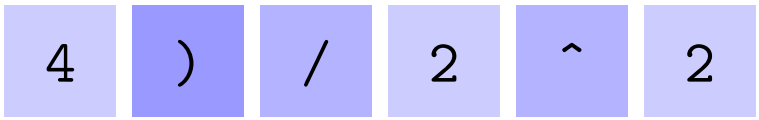
Ausgabe:



2. Beispiel

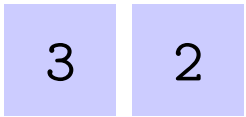
$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe:



Stack:

Ausgabe:



2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe:

Stack:

Ausgabe:

2. Beispiel

$$\frac{3 \cdot (2+4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: |) | / | 2 | ^ | 2 |

Stack: | * | (|

Ausgabe: | 3 | 2 | 4 | + |

2. Beispiel

$$\frac{3 \cdot (2+4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: | / | 2 | ^ | 2 |

Stack: | * |

Ausgabe: | 3 | 2 | 4 | + |

2. Beispiel

$$\frac{3 \cdot (2+4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: | 2 | ^ | 2 |

Stack: | / |

Ausgabe: | 3 | 2 | 4 | + | * |

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: | ^ | 2 |

Stack: | / |

Ausgabe: | 3 | 2 | 4 | + | * | 2 |

2. Beispiel

$$\frac{3 \cdot (2+4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: 2

Stack: / ^

Ausgabe: 3 2 4 + * 2

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack:

`/`	`^`

Ausgabe:

3	2	4	+	*	2	2

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack:

/

Ausgabe:

3	2	4	+	*	2	2	^

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

3	2	4	+	*	2	2	^	/

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

3	2	4	+	*	2	2	^	/

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

3	2	4	+	*	2	2	^	/

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

3	6	*	2	2	^	/

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

3	6	*	2	2	^	/

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

`18`	`2`	`2`	`^`	`/`

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe:

18	2	2	^	/

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe: `18` `4` `/`

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe: `18` `4` `/`

2. Beispiel

$$\frac{3 \cdot (2 + 4)}{2^2} = \frac{18}{4} = 4{,}5$$

Eingabe: `leer`

Stack: `leer`

Ausgabe: `4,5`

Laufzeit

- Jeder Operator und jeder Operand wird genau einmal in die Ausgabeliste gelegt und ein Operator wird höchstens einmal auf den Stack gelegt

Laufzeit

- Jeder Operator und jeder Operand wird genau einmal in die Ausgabeliste gelegt und ein Operator wird höchstens einmal auf den Stack gelegt

$$\mathcal{O}(n)$$

Jedes Token wird genau einmal verarbeitet

Implementierung in Java

```
double calculator(String expression) {
    List<Token> tokens = tokenize(expression);
    List<Token> rpn = shuntingYard(tokens);
    double result = calculate(rpn);

    return result;
}
```

```
abstract class Token {
    abstract int getPrecedence();
    abstract boolean isLeftAssociative();
    abstract void evaluate(Stack<Double> s);
    abstract String toString();
}
```

```
class Num extends Token {
    double value;

    Num(String val) {
        this.value = Double.valueOf(val);
    }

    int getPrecedence() {
        return 0;
    }

    boolean isLeftAssociative() {
        return true;
    }

    String toString() {
        return String.valueOf(value);
    }

    void evaluate(Stack<Double> s) {
        s.push(this.value);
    }
}
```

```
class Power extends Token {
    int getPrecedence() {
        return 4;
    }

    boolean isLeftAssociative() {
        return false;
    }

    String toString() {
        return "^";
    }

    void evaluate(Stack<Double> s) {
        double b = s.pop();
        double a = s.pop();
        s.push(Math.pow(a, b));
    }
}
```

```
List<Token> shuntingYard(List<Token> input) {
    List<Token> output = new ArrayList<>();
    Stack<Token> opStack = new Stack<>();
```

```
List<Token> shuntingYard(List<Token> input) {
    List<Token> output = new ArrayList<>();
    Stack<Token> opStack = new Stack<>();

    for (Token token : input) {
        if (token instanceof Num) {
            output.add(token);
        }
```

```
List<Token> shuntingYard(List<Token> input) {
    List<Token> output = new ArrayList<>();
    Stack<Token> opStack = new Stack<>();

    for (Token token : input) {
        if (token instanceof Num) {
            output.add(token);
        }  else if (token instanceof LParan) {
            opStack.push(token);
        }
```

```
List<Token> shuntingYard(List<Token> input) {
    List<Token> output = new ArrayList<>();
    Stack<Token> opStack = new Stack<>();

    for (Token token : input) {
        if (token instanceof Num) {
            output.add(token);
        }  else if (token instanceof LParan) {
            opStack.push(token);
        }  else if (token instanceof RParan) {
            popUntilOpenParan(opStack, output);
            opStack.pop();
        }
```

```
List<Token> shuntingYard(List<Token> input) {
    List<Token> output = new ArrayList<>();
    Stack<Token> opStack = new Stack<>();

    for (Token token : input) {
        if (token instanceof Num) {
            output.add(token);
        } else if (token instanceof LParan) {
            opStack.push(token);
        } else if (token instanceof RParan) {
            popUntilOpenParan(opStack, output);
            opStack.pop();
        } else {
            popHigherPrecedenceOps(opStack, output, token);
            opStack.push(token);
        }
    }
```

```java
List<Token> shuntingYard(List<Token> input) {
    List<Token> output = new ArrayList<>();
    Stack<Token> opStack = new Stack<>();

    for (Token token : input) {
        if (token instanceof Num) {
            output.add(token);
        } else if (token instanceof LParan) {
            opStack.push(token);
        } else if (token instanceof RParan) {
            popUntilOpenParan(opStack, output);
            opStack.pop();
        } else {
            popHigherPrecedenceOps(opStack, output, token);
            opStack.push(token);
        }
    }

    while (!opStack.isEmpty()) {
        output.add(opStack.pop());
    }
```

```
List<Token> shuntingYard(List<Token> input) {
    List<Token> output = new ArrayList<>();
    Stack<Token> opStack = new Stack<>();

    for (Token token : input) {
        if (token instanceof Num) {
            output.add(token);
        } else if (token instanceof LParan) {
            opStack.push(token);
        } else if (token instanceof RParan) {
            popUntilOpenParan(opStack, output);
            opStack.pop();
        } else {
            popHigherPrecedenceOps(opStack, output, token);
            opStack.push(token);
        }
    }

    while (!opStack.isEmpty()) {
        output.add(opStack.pop());
    }

    return output;
}
```

```
void popUntilOpenParan(Stack<Token> opStack, List<Token> output) {
    while (!(opStack.isEmpty() || opStack.peek() instanceof LParan)) {
        output.add(opStack.pop());
    }
}
```

```java
void popUntilOpenParan(Stack<Token> opStack, List<Token> output) {
    while (!(opStack.isEmpty() || opStack.peek() instanceof LParan)) {
        output.add(opStack.pop());
    }
}

void popHigherPrecedenceOps(Stack<Token> opStack, List<Token> output, Token currTok) {
    while (!(opStack.isEmpty() || opStack.peek() instanceof LParan)) {
        Token stackTop = opStack.peek();
        int stackPrec = stackTop.getPrecedence();
        int currPrec = currTok.getPrecedence();

        if (stackPrec > currPrec || (stackPrec == currPrec && currTok.isLeftAssociative())) {
            output.add(opStack.pop());
        } else {
            break;
        }
    }
}
```

Danke für Ihre Aufmerksamkeit!

Danke für Ihre Aufmerksamkeit!

Fragen? :-)