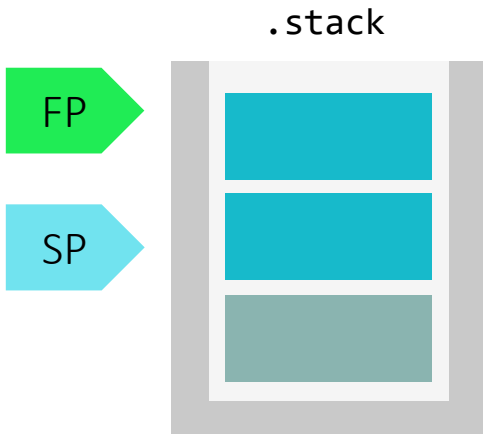
Einführung in die Programmierung



Tutorium 9

Wintersemester 2025/2026

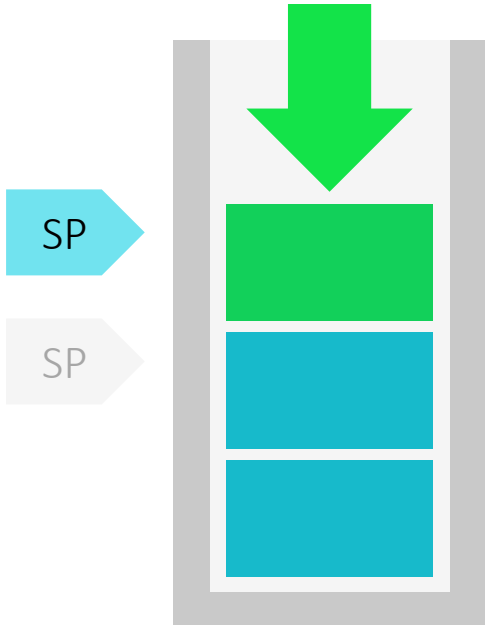
Arbeitsgruppe Systemsoftware
Angewandte Informatik 12

technische universität dortmund

Arbeitsgruppe Systemsoftware

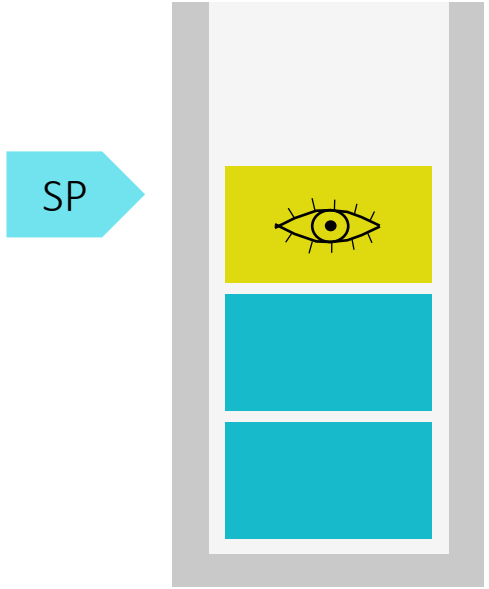
Implementieren Sie die Datenstruktur Stack als C++-Klasse, um darauf aufbauend einen Push-Down-Automaten (PDA) zu programmieren, der mathematische Ausdrücke in der klammerfreien Łukasiewicz-Präfixnotation auswertet.

Elementare Operationen



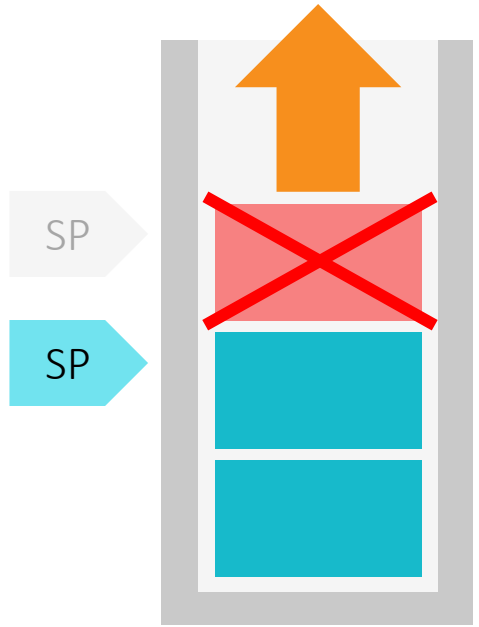
push(…)

Element hinzufügen
Stapelzeiger inkrementieren



peek()

Element abrufen
Stapelzeiger unverändert lassen



pop()

Zuletzt hinzugefügtes Element entfernen
Stapelzeiger dekrementieren

Łukasiewicz-Push-Down-Automat | Code

```
...
62  std::string replaceBrackets(const std::string& input) {
63      std::string result;
64      //TODO1
65      for (char ch : input) {
66          if (ch == '(' || ch == ')' || ch == ',') {
67              result += ' ';
68          } else {
69              result += ch;
70          }
71      }
72      return result;
73  }
```

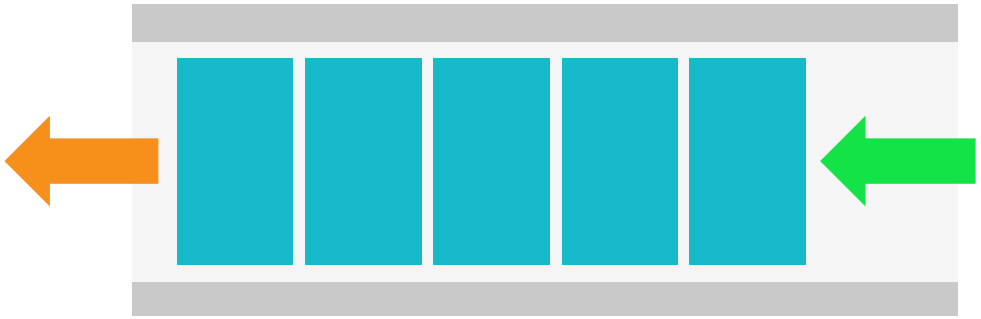
Łukasiewicz-Push-Down-Automat | Code

```
...
80      //TODO2
81      while (!reverseTokens.empty()) {
82          //TODO3
83          result = 0;
84          token = reverseTokens.peek();
85          reverseTokens.pop();
```

```
...
86      //TODO4
87      if (token == "+" || token == "-" || token == "*" || token == "/") {
88          if (stack.len() < 2) throw std::runtime_error("Not enough operands.");
89
90          //TODO5
91          b = stack.peek(); stack.pop();
92          a = stack.peek(); stack.pop();
93
94          if (token == "+") { result = a + b; }
95          else if (token == "-") { result = b - a; }
96          else if (token == "*") { result = a * b; }
97          else if (token == "/") {
98              if (b == 0) throw std::runtime_error("Division by zero.");
99              result = b / a;
100         }
101         stack.push(result);
...
```

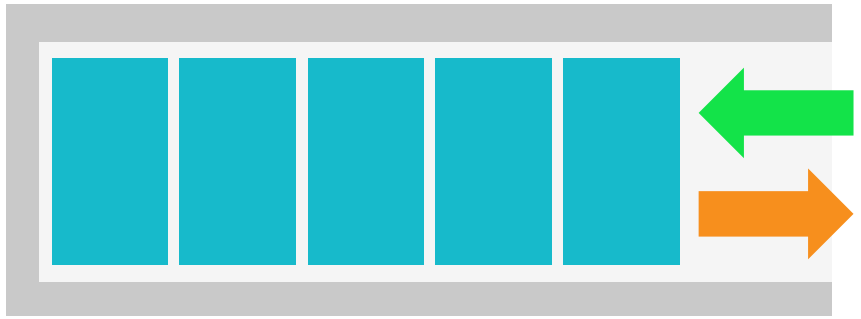
```
...
102         } else {
103             try {
104                 //TODO6
105                 number = std::stod(token);
106                 stack.push(number);
107             } catch (...) {
108                 throw std::runtime_error("Invalid number format: " + token);
109             }
110         }
111     }
...
```

```
...
112  //TODO7
113  if (stack.empty()) throw std::runtime_error
     ("Invalid expression: Stack is empty at the end.");
114  //TODO8
115  result = stack.peek();
116  stack.pop();
117
118  //TODO9
119  if (!stack.empty()) throw std::runtime_error
     ("Invalid expression: Stack is not empty at the end.");
120  return result;
...
```



Queue

First in first out



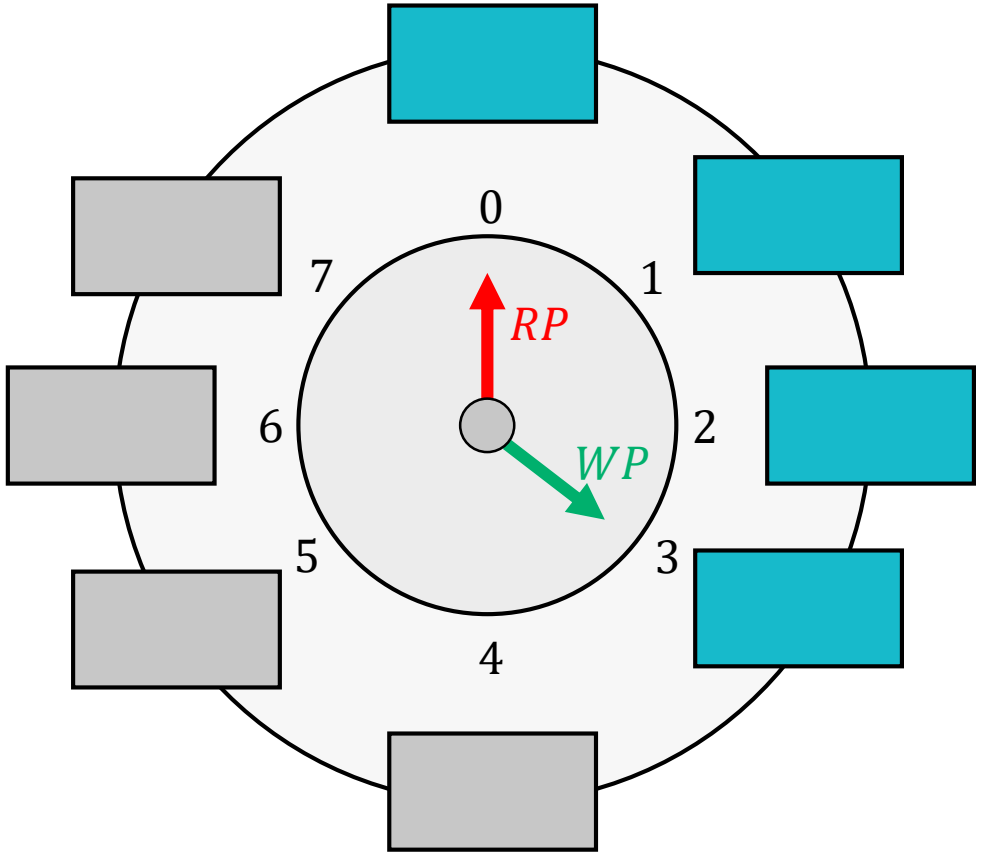
Stack

Last in first out

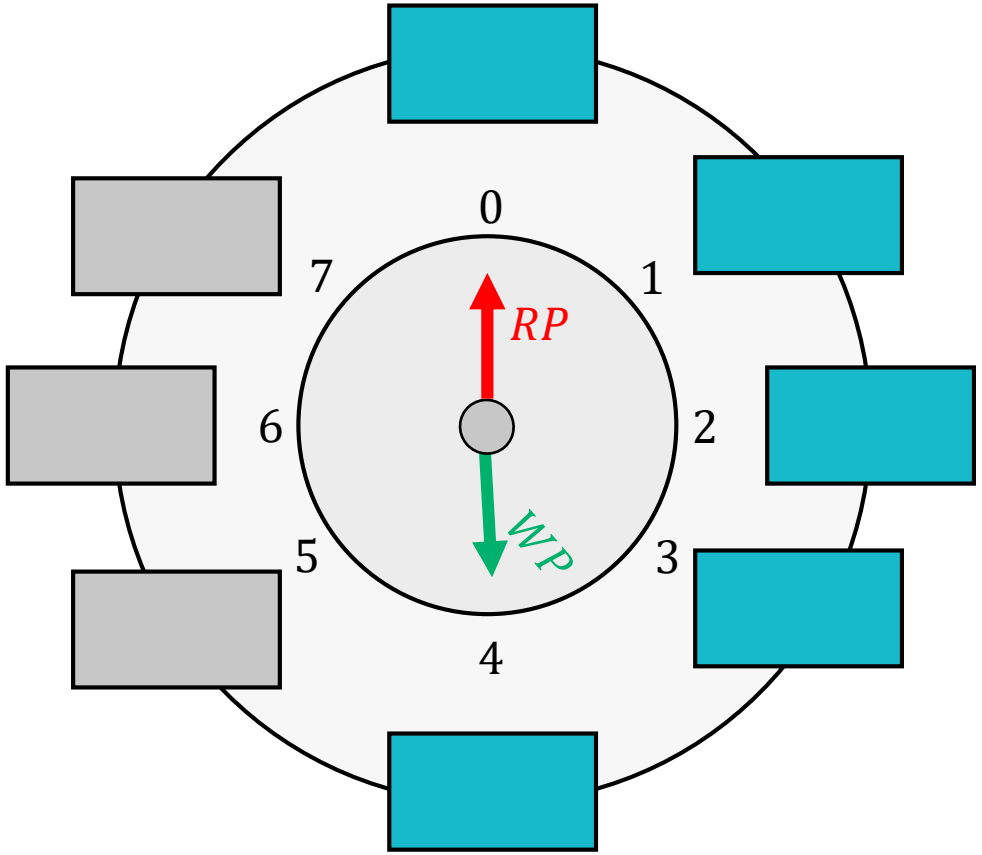
- **Ziel**: Elemente sollen im Speicher ihre Position nicht ändern
- Ein **Lesezeiger** RP zeigt auf das vorderste und ein **Schreibzeiger** WP auf das hinterste Element der Queue
- Wird ein Element hinzugefügt, muss WP inkrementiert werden

  → Element wird an die Stelle WP geschrieben
  (‚Nummer ziehen‘ wie bei einer Behörde)

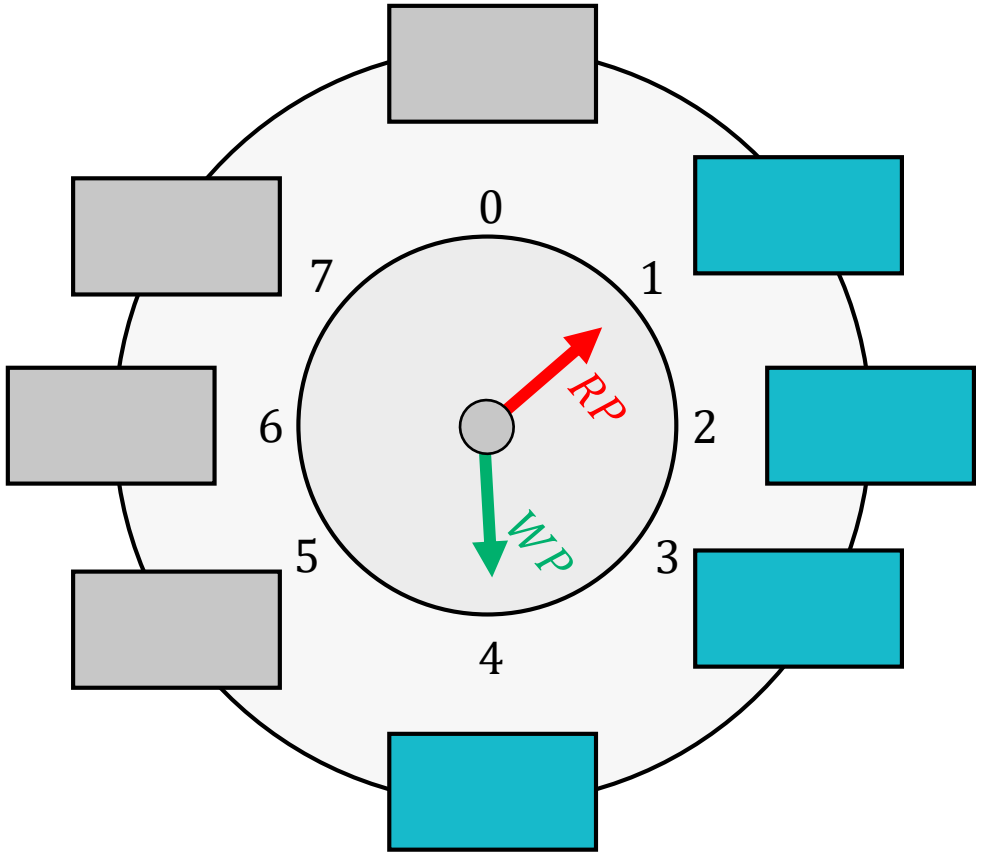
- Soll ein Element gelesen und entfernt werden, wird an der Stelle RP gelesen und RP darauf dekrementiert
- Bei beiden Operationen werden ausschließlich die Zeiger aber nicht die Elemente selbst bewegt


0
1
2
3
4
5
6
7
RP
WP

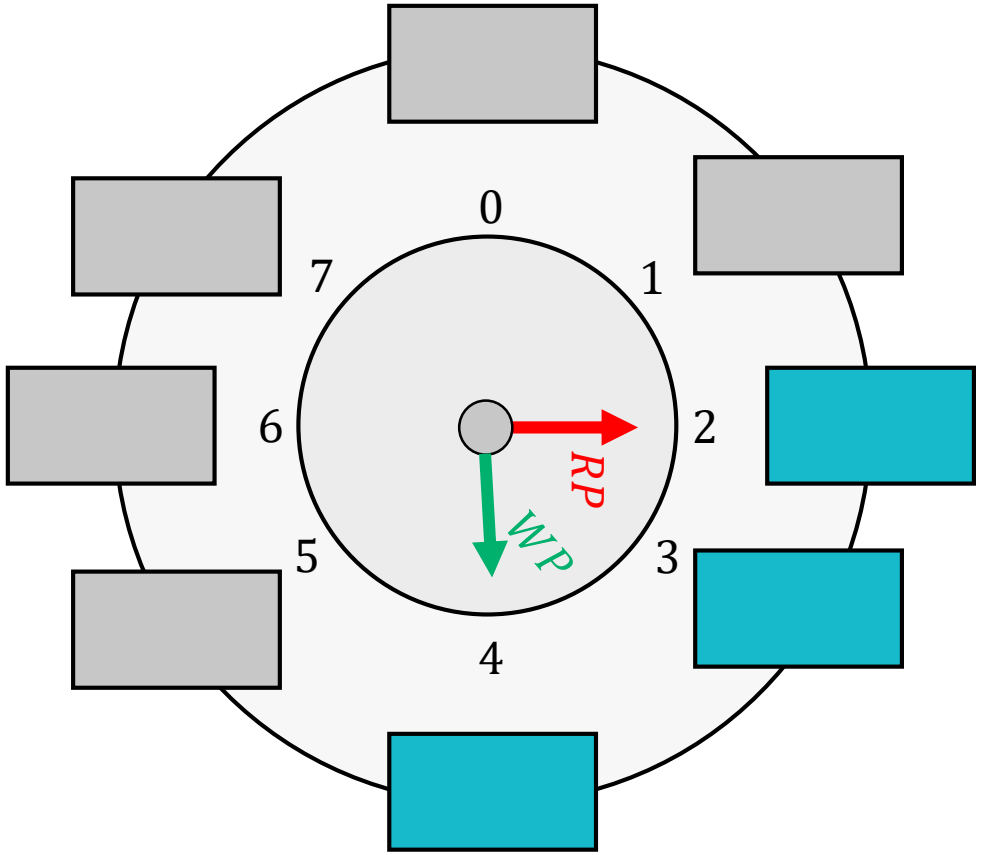
Element hinzufügen



Element entfernen



Weiteres Element entfernen



```
template <typename T> class Queue {
private:
    static constexpr int maxSize = MAXSIZE;
    T data[maxSize];
    int wp;     // Write Pointer (WP)
    int rp;     // Read Pointer  (RP)
    int count;  // Elementcount

    void error(const char* s) {
        std::cerr << s << std::endl;
        exit(1);
    }

...
```

```
public:
    Queue() : wp(0), rp(0), count(0) {}

    void append(T x) {
        if (full()) { error("Queue is full!"); }
        data[wp] = x;
        wp = (wp + 1) % maxSize;
        count++;
    }

    void pop() {
        if (empty()) { error("Queue is empty!"); }
        rp = (rp + 1) % maxSize;
        count--;
    }

    T peek() {
        if (empty()) { error("Queue is empty!"); }
        return data[rp];
    }
```

```
...
    bool empty() { return (count == 0); }

    bool full() { return (count == maxSize); }

    int len() const { return count; }

    friend std::ostream& operator<<(...) {

        int idx = 0;

        for (int i = 0; i < q.count; ++i) {
            idx = (q.rp + i) % maxSize;
            os << q.data[idx];
            if (i < q.count - 1) { os << ", " };
        }

        return os;
    }
};
```

Warteschlangen/Ringspeicher können auch als Hardware implementiert werden