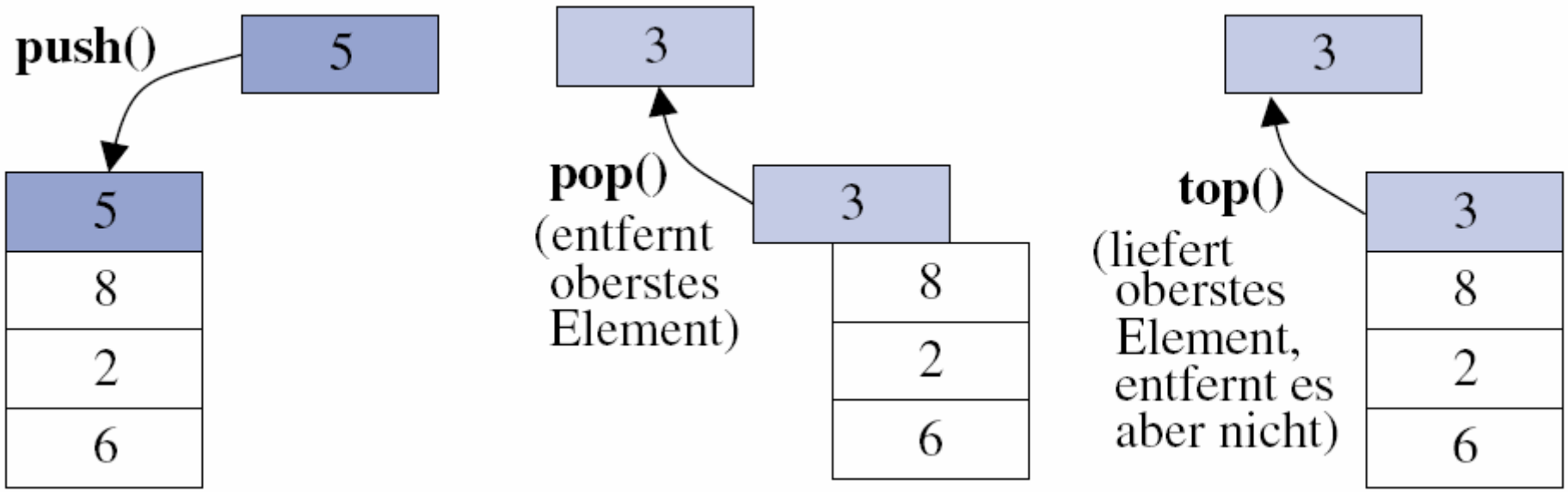
Stack (Stapel)



push(): legt ein Element an oberster Stelle auf den Stack

pop(): entfernt das oberste Element aus dem Stack

top(): liefert oberstes Stack-Element, entfernt es aber nicht

Realisierung eines Stacks als Array (in C)

```c
#include <stdio.h> /* stack.c */
#include <stdlib.h>
static int  *stack = NULL,
            sp, unten, oben;
void stackinit(int n) {
  stack = (int*)malloc(n*sizeof(int));
  sp = unten = 0;
  oben = n-1;
}
int isEmpty() { return sp <= unten; }
int isFull()  { return sp  > oben; }
void push(int wert) {
  if (stack == NULL) {
    printf("Stack fehlt\n"); return;
  }
  if (isFull()) {
    printf("Stack voll\n"); return;
  }
  stack[sp++] = wert;
}
```

```c
void pop() {
  if (stack == NULL) {
    printf("Stack fehlt\n"); return;
  }
  if (isEmpty()) {
    printf("Stack leer\n"); return;
  }
  --sp;
}
int top() {
  if (stack == NULL) {
    printf("Stack fehlt\n"); return -1
  }
  if (isEmpty()) {
    printf("Stack leer\n"); return -1;
  }
  return stack[sp-1];
}
int popTop() {
  int wert = top();
  pop();
  return wert;
}
```

Realisierung eines Stacks als Array (in Java)

```
class Stack {
  private int[] stack;
  private int sp, unten, oben;
  Stack(int n) {
     stack = new int[n];
     sp = unten = 0;
     oben = n−1;
  }
  boolean isEmpty() { return sp <= unten; }
  boolean isFull()  { return sp  > oben; }

  void push(int wert) throws StackFehler {
     if (stack == null)
        throw new StackFehler("Stack fehlt");
     if (isFull())
        throw new StackFehler("Stack voll");
     stack[sp++] = wert;
  }
```

```
  void pop() throws StackFehler {
     if (stack == null)
        throw new StackFehler("Stack fehlt");
     if (isEmpty())
        throw new StackFehler("Stack leer");
     −−sp;
  }
  int top() throws StackFehler {
     if (stack == null)
        throw new StackFehler("Stack fehlt");
     if (isEmpty())
        throw new StackFehler("Stack leer");
     return stack[sp−1];
  }
  int popTop() throws StackFehler {
     int wert = top();
     pop();
     return wert;
  }
```

Realisierung eines Stacks als verkettete Liste (in C)

```
struct elem {
    int          zahl;
    struct elem *next;
};
static struct elem *liste = NULL;
int isEmpty() { return liste==NULL;}
void push(int wert) {
  struct elem *neu =
    (struct elem *)malloc(sizeof *neu)
  neu->zahl = wert;
  neu->next = liste;
  liste  = neu;
}
```

```
void pop() {
  if (isEmpty()) {
     printf("Stack leer\n");
     return;
  }
  liste = liste->next;
}
int top() {
  if (isEmpty()) {
     printf("Stack leer\n");
     return -1;
  }
  return liste->zahl;
}
int popTop() {
  int wert = top();
  pop();
  return wert;
}
```

Realisierung eines Stacks als verkettete Liste (in Java)

```
class Elem {
   int   zahl;
   Elem next = null;
   Elem(int z) {  zahl = z;  }
}
class StackListe {
    private Elem liste = null;
    boolean isEmpty() { return liste == null;

    void push(int wert) {
        Elem neu = new Elem(wert);
        neu.next = liste;
        liste  = neu;
    }
```

```
    void pop() throws StackFehler {
        if  (isEmpty())
            throw new StackFehler("Stack leer");
        liste  =  liste .next;
    }
    int  top()  throws StackFehler {
        if  (isEmpty())
            throw new StackFehler("Stack leer");
        return  liste .zahl;
    }
    int  popTop() throws StackFehler {
        int  wert = top();
        pop();
        return wert;
    }
}
```

Umgekehrte polnische Notation

Infix-Schreibweise: **(4 + 5) * (3 - 1) / 9**

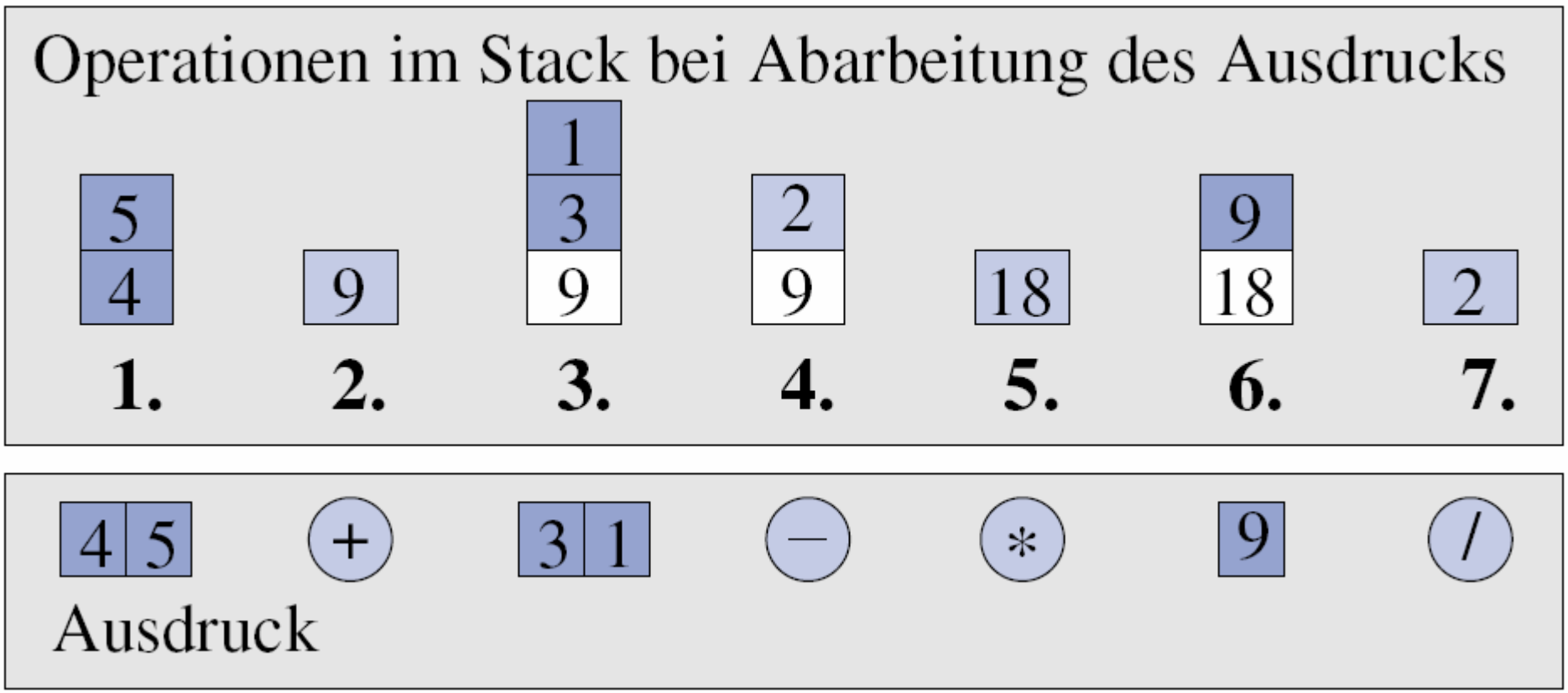
Umgekehrte polnische Notation: **4 5 + 3 1 - * 9 /**

Vorteil einer solchen Postfix-Notation

- → keinerlei Prioritäten für Operatoren zu berücksichtigen
- → es reicht, Operanden in einem Stack an jeweils oberster Stelle abzulegen und beim Lesen eines
  - ➢ binären Operators: beide obersten Stack-Elemente entfernen, diese mit Operator verknüpfen und Ergebnis als neues oberstes Elem. im Stack ablegen
  - ➢ unären Operators: oberstes Stack-Element mit diesem Operator behandeln

Umgekehrte polnische Notation

Stackoperationen bei Abarbeitung von 4 5 + 3 1 - * 9 /



Transformieren eines Infix-Ausdrucks in einen Postfix-Ausdruck (C)

```
fgets(ausdr, 1000, stdin);
while (ausdr[i] != '\n') {
  z = ausdr[i];
  if (z == ')')
     printf("%c ", (char)popTop());
  else if (z == '+' || z == '-' ||
           z == '*' || z == '/')
     push(z);
  else if (isdigit(z)) {
     do {
        printf("%c", z);
        z = ausdr[++i];
     } while (isdigit(z));
     --i;
     printf(" ");
  } else if (z != '(')
     printf(" ");
  i++;
}
```

((4+5)*(3-1))/9
4 5 + 3 1 − ∗ 9 /

12*(((10+5)*(3+1))+120)
12 10 5 + 3 1 + ∗ 120 + ∗

```
while (!isEmpty())
   printf("%c ", (char)popTop());
```

Transformieren eines Infix-Ausdrucks in einen Postfix-Ausdruck (Java)

```
String ausdr = ein.readLine("");
StackListe stack = new StackListe();
 for (int i=0; i<ausdr.length(); i++) {
    char z = ausdr.charAt(i);
    if (z == ')')
       try { System.out.print(
                 (char)stack.popTop()+" ");
       } catch (StackFehler f) {
          System.out.println(f);
       }
    else if (z == '+' || z == '-' ||
             z == '*' || z == '/')
       stack.push(z);
    else if (Character.isDigit(z)) {
       do {
          System.out.print(z);
          if (++i >= ausdr.length())
             break;
          z = ausdr.charAt(i);
       } while (Character.isDigit(z))
       i--;
       System.out.print(" ");
    } else if (z != '(')
       System.out.print(" ");
 }
 while (!stack.isEmpty())
    try { System.out.print(
              (char)stack.popTop()+" ");
    } catch (StackFehler f) {
       System.out.println(f);
    }
 }
```

((4+5)*(3-1))/9

4 5 + 3 1 − ∗ 9 /

12*(((10+5)*(3+1))+120)

12 10 5 + 3 1 + ∗ 120 + ∗

Berechnen eines Postfix-Ausdrucks (C)

```
fgets(ausdr, 1000, stdin);
while (ausdr[i] != '\n') {
  z = ausdr[i];
  if (z == '+' || z == '-' ||
      z == '*' || z == '/') {
    op2 = popTop();
    op1 = popTop();
    if (z == '+')
      push(op1 + op2);
    else if (z == '-')
      push(op1 - op2);
    else if (z == '*')
      push(op1 * op2);
    else if (z == '/')
      push(op1 / op2);
  } else if (isdigit(z)) {
    int wert = 0;
    do {
      wert = wert*10 + z-'0';
      z = ausdr[++i];
    } while (isdigit(z));
    push(wert);
    --i;
  }
  i++;
}
printf("= %d\n", popTop());
```

12 10 5 + 3 1 + * 120 + *
= 2160

4 5 + 3 1 - * 9 /
= 2

Berechnen eines Postfix-Ausdrucks (Java)

```
String ausdr = ein.readLine("");
StackListe stack = new StackListe();
for (int i=0; i<ausdr.length(); i++) {
    char z = ausdr.charAt(i);
    if (z=='+' || z=='−' || z=='∗' || z=='/') {
      try { op2 = stack.popTop();
            op1 = stack.popTop();
      } catch (StackFehler f) {
          System.out.println(f); }
      if (z == '+')
stack.push(op1 + op2);
      else if (z == '−') stack.push(op1 − op2);
      else if (z == '∗') stack.push(op1 ∗ op2);
      else if (z == '/') stack.push(op1 / op2);
```

12 10 5 + 3 1 + * 120 + *
= 2160

4 5 + 3 1 - * 9 /
= 2

```
    } else if (Character.isDigit(z)) {
      int wert = 0;
      do {
          wert = wert∗10 + z−'0';
          if (++i >= ausdr.length())
              break;
          z = ausdr.charAt(i);
      } while (Character.isDigit(z));
      stack.push(wert);
      −−i;
    }
}
try { System.out.println(
        "= "+stack.popTop()+" ");
} catch (StackFehler f) {
    System.out.println(f); }
```

Umgekehrte polnische Notation

Man kann die Standardausgabe der beiden vorherigen Programme intopostfix.c und Intopostfix.java auch über eine Pipe (Zeichen |) direkt in die Standardeingabe der Programme postfix.c und Postfix.java umlenken, so dass man sich auch Infix-Ausdrücke berechnen lassen kann.

./intopostfix | ./postfix **oder**
java Intopostfix | java Postfix

$12*(((10+5)*(3+1))+120)$
$= 2160$

$((12*10)+5)*((3+1)+120)$
$= 15500$