

# **III.1. Grundelemente der Programmierung**

- 1. Erste Schritte**
- 2. Einfache Datentypen**
- 3. Anweisungen und Kontrollstrukturen**
- 4. Verifikation**
- 5. Reihungen (Arrays)**

# 4. Verifikation

---

## ■ Spezifikation: Angabe, **was** ein Programm tun soll

- natürliche Sprache
- grafische Sprachen (UML, ...)
- logische Sprachen (Z, VDM, ...)

## ■ Testen: Überprüfung für endlich viele Eingaben

→ keine 100% Sicherheit

## ■ Verifikation: Mathematischer Beweis der Korrektheit

- Terminierung: Hält Programm immer an?
- Partielle Korrektheit: Falls Programm anhält, erfüllt es Spezifikation?
- Totale Korrektheit: Terminierung & Partielle Korrektheit

→ Semantik der Programmiersprache

# Fakultät

```
void main () {  
  
    int n = Integer.parseInt(IO.readln("Gib Zahl ein: ")), i, res;  
    <true>  
    <n=n>  
    i = n;  
    <i=n>  
    <i=n ∧ 1=1>  
    res = 1;  
    <i=n ∧ res=1>  
    <i! * res = n!>  
    while (i > 1) {  
        <i! * res = n! ∧ i>1>  
        <(i-1)! * (res * i) = n!>  
        res = res * i;  
        <(i-1)! * res = n!>  
        i = i - 1;  
        <i! * res = n!>  
    }  
    <i! * res = n! ∧ i>1>  
    <res = n!>  
    IO.println("Fakultaet ist " + res);  
}
```

## Programm P

### ■ Spezifikation:

Programm berechnet (in `res`) Fakultät von `n`

### ■ Terminierung:

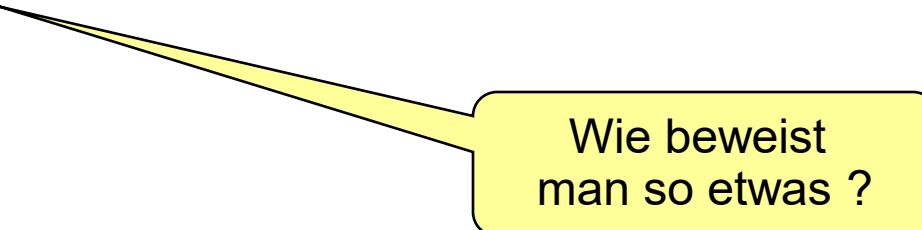
Programm hält an, weil `i` in jedem Schleifendurchlauf kleiner wird

### ■ Partielle Korrektheit:

Nach Ausführung ist `res = n!`

### ■ Totale Korrektheit

```
i = n;  
  
res = 1;  
  
while (i > 1) {  
  
    res = res * i;  
  
    i = i - 1;  
}
```



Wie beweist  
man so etwas ?

- Verifikation nötig bei sicherheitskritischen Anwendungen
- hilft für Programmentwurf und Programmierstil

# Partielle Korrektheit: Hoare-Kalkül

---

## ■ Spezifikation (zur partiellen Korrektheit)

$\langle \varphi \rangle \text{ P } \langle \psi \rangle$

Wenn vor Ausführung von P **Vorbedingung**  $\varphi$  gilt

und Ausführung von P terminiert,

dann gilt hinterher **Nachbedingung**  $\psi$ .

■ Bsp:    `< true > P < res = n! >`

■ Partielle Korrektheit ist *semantische* Aussage

Hoare-Kalkül: 7 Regeln zur Herleitung von Korrektheitsaussagen

# Zuweisungsregel

---

$\langle \varphi [x/t] \rangle \quad x = t; \quad \langle \varphi \rangle$

$x$  ist Variable,  $t$  ist Ausdruck (ohne Seiteneffekte),  
 $\varphi [x/t]$  ist  $\varphi$  mit allen  $x$  ersetzt durch  $t$

Bsp:  $\langle 5 = 5 \rangle \quad x = 5; \quad \langle x = 5 \rangle$

$\langle 5 = 5 \rangle$

$x = 5;$

$\langle x = 5 \rangle$

# Konsequenzregel 1 (Stärkere Vorbedingung)

---

$$\frac{<\varphi> \text{ P } <\psi> \quad \alpha \Rightarrow \varphi}{<\alpha> \text{ P } <\psi>}$$

Bsp: **<true>** **x = 5**; **<x = 5>**, denn:

$$\frac{<5 = 5> \text{ x = 5; } <\mathbf{x} = 5> \quad \mathbf{true} \Rightarrow 5 = 5}{<\mathbf{true}> \text{ x = 5; } <\mathbf{x} = 5>}$$

**<true>**  
**<5 = 5>**  
**x = 5;**  
**<x = 5>**

# Konsequenzregel 2 (Schwächere Nachbedg.)

$$\frac{\langle \varphi \rangle \text{ P } \langle \psi \rangle \quad \psi \Rightarrow \beta}{\langle \varphi \rangle \text{ P } \langle \beta \rangle}$$

Bsp:  $\langle \text{true} \rangle \quad x = 5; \quad \langle x \geq 5 \rangle$ , denn:

$$\frac{\langle \text{true} \rangle \ x = 5; \ \langle x = 5 \rangle \quad x = 5 \Rightarrow x \geq 5}{\langle \text{true} \rangle \quad x = 5; \quad \langle x \geq 5 \rangle}$$

$\langle \text{true} \rangle$   
 $\langle 5 = 5 \rangle$   
 $x = 5;$   
 $\langle x = 5 \rangle$   
 $\langle x \geq 5 \rangle$

# Sequenzregel

$$\frac{<\varphi> \text{ P } <\psi> \quad <\psi> \text{ Q } <\beta>}{<\varphi> \text{ P Q } <\beta>}$$

Bsp: <true>

```
x = 5;  
res = x * x + 6;  
<res = 31>
```

```
<true>  
<5 = 5>  
  
x = 5;  
  
<x = 5>  
<x * x + 6 = 31>  
  
res = x * x + 6;  
  
<res = 31>
```

# Bedingungsregel 1

$$\frac{\langle \varphi \wedge B \rangle \quad P \quad \langle \psi \rangle \quad \varphi \wedge \neg B \Rightarrow \psi}{\langle \varphi \rangle \text{ if } (B) \quad \{P\} \quad \langle \psi \rangle}$$

Bsp: `<true>`

```
res = y;  
if (x > y) res = x;  
<res = max(x,y)>
```

denn: `<res = y ∧ x > y >`  
`<x = max(x,y)>`  
`res = x;`  
`<res = max(x,y)>`

und `<res = y ∧ ¬x > y >`  
 $\Rightarrow$  `<res = max(x,y)>`

```
<true>  
<y = y>  
res = y;  
<res = y>  
  
if (x > y) {  
    <res = y ∧ x > y >  
    <x = max(x,y)>  
    res = x;  
    <res = max(x,y)> }  
  
<res = max(x,y)>
```

# Bedingungsregel 2

$\langle \varphi \wedge B \rangle \quad P \quad \langle \psi \rangle$

$\langle \varphi \wedge \neg B \rangle \quad Q \quad \langle \psi \rangle$

$\langle \varphi \rangle \text{ if } (B) \{ P \} \text{ else } \{ Q \} \langle \psi \rangle$

Bsp:  $\langle \text{true} \rangle$

```
if (x < 0)
    res = -x;
else
    res = x;
<res = |x|>
```

denn:  $\langle \text{true} \rangle$

```
if (x < 0) {
    <true  $\wedge$  x < 0>
    <-x = |x|>

    res = -x;

    <res = |x|> }

else {
    <true  $\wedge$   $\neg$  x < 0>
    <x = |x|>

    res = x;

    <res = |x|> }

<res = |x|>
```

# Schleifenregel

$$\frac{<\varphi \wedge B> \quad P \quad <\varphi>}{<\varphi> \text{ while } (B) \{P\} \quad <\varphi \wedge \neg B>}$$

```
<true>
    i = n; res = 1;
<i = n \wedge res = 1>
< \varphi >
    while (i > 1) {res = res * i; i = i - 1; }
< \varphi \wedge \neg i > 1>
<res = n! >
```

# Schleifenregel

$$\frac{<\varphi \wedge B> \quad P \quad <\varphi>}{<\varphi> \text{ while } (B) \{P\} <\varphi \wedge \neg B>}$$

```
<true>
    i = n; res = 1;
<i = n ∧ res = 1>
<i! * res = n!>
    while (i > 1) {res = res * i; i = i - 1; }
<i! * res = n! ∧ ¬ i > 1>
<res = n! >
```

$\varphi$  ist Schleifen-invariante

denn:  $<i! * res = n! \wedge i > 1>$   
 $<(i-1)! * (res * i) = n!>$   
     $res = res * i;$   
     $i = i - 1;$   
 $<i! * res = n!>$

# Hoare-Kalkül

## ■ Zuweisungsregel

$$\boxed{\textcolor{red}{<\varphi [x/t]>} \quad x = t; \quad <\varphi>}$$

## ■ Konsequenzregeln

$$\boxed{\begin{array}{c} <\varphi> \quad P \quad <\psi> \\ \hline & \alpha \Rightarrow \varphi \\ & <\alpha> \quad P \quad <\psi> \end{array}}$$

$$\boxed{\begin{array}{c} <\varphi> \quad P \quad <\psi> \\ \hline & \psi \Rightarrow \beta \\ & <\varphi> \quad P \quad <\beta> \end{array}}$$

## ■ Sequenzregel

$$\boxed{\begin{array}{c} <\varphi> \quad P \quad <\psi> \quad <\psi> \quad Q \quad <\beta> \\ \hline & <\varphi> \quad P \quad Q \quad <\beta> \end{array}}$$

## ■ Bedingungsregeln

$$\boxed{\begin{array}{c} <\varphi \wedge B> \quad P \quad <\psi> \quad \varphi \wedge \neg B \Rightarrow \psi \\ \hline & <\varphi> \text{ if } (B) \{P\} \quad <\psi> \end{array}}$$

$$\boxed{\begin{array}{c} <\varphi \wedge B> \quad P \quad <\psi> \quad <\varphi \wedge \neg B> \quad Q \quad <\psi> \\ \hline & <\varphi> \text{ if } (B) \{P\} \text{ else } \{Q\} \quad <\psi> \end{array}}$$

## ■ Schleifenregel

$$\boxed{\begin{array}{c} <\varphi \wedge B> \quad P \quad <\varphi> \\ \hline & <\varphi> \text{ while } (B) \{P\} \quad <\varphi \wedge \neg B> \end{array}}$$

# Fakultät mit Assertions

---

```
void main () {  
  
    int n = Integer.parseInt(IO.readln("Gib Zahl ein: ")), i, res;  
  
    assert true;  
    assert n == n;  
  
    i = n;  
  
    assert i == n;  
    assert i == n && 1 == 1;  
  
    res = 1;  
  
    assert i == n && res == 1;  
    assert fac(i) * res == fac(n);  
  
    while (i > 1) {  
        assert fac(i) * res == fac(n) && i > 1;  
        assert fac(i-1) * (res * i) == fac(n);  
  
        res = res * i;  
  
        assert fac(i-1) * res == fac(n);  
  
        i = i - 1;  
  
        assert fac(i) * res == fac(n);  
    }  
    assert fac(i) * res == fac(n) && !(i > 1);  
    assert res == fac(n);  
  
    IO.println("Fakultaet ist " + res);  
}
```

# Terminierung

Für jede Schleife **while** (**B**) {**P**} finde einen **int**-Ausdruck **V** (*Variante* der Schleife), so dass:

$$\mathbf{B} \Rightarrow \mathbf{V} \geq 0 \quad \text{und} \quad \langle \mathbf{V} = m \wedge \mathbf{B} \rangle \quad \mathbf{P} \quad \langle \mathbf{V} < m \rangle$$

```
while (i > 1) {res = res * i; i = i - 1; }
```

Variante ist **i**,

denn:  $i > 1 \Rightarrow i \geq 0$

$$\begin{aligned} &\langle i = m \wedge i > 1 \rangle \\ &\langle i-1 < m \rangle \\ &\quad \text{res} = \text{res} * i; i = i - 1; \\ &\langle i < m \rangle \end{aligned}$$

# Verifikation der Addition

```
void main () {  
  
    int a = Integer.parseInt(IO.readln("Gib erste Zahl ein: ")),  
        b = Integer.parseInt(IO.readln("Gib zweite Zahl ein: ")), x, res;  
  
    x = a;  
  
    res = b;  
  
    while (x > 0) {  
  
        x = x - 1;  
  
        res = res + 1;  
    }  
  
    IO.println(a + " + " + b + " = " + res);  
}
```

Vorbedingung:  $a \geq 0$

Nachbedingung:  $res = a + b$

# Verifikation der Addition

```
void main () {  
  
    int a = Integer.parseInt(IO.readln("Gib erste Zahl ein: ")),  
        b = Integer.parseInt(IO.readln("Gib zweite Zahl ein: ")), x, res;  
  
    x = a;  
  
    res = b;  
    //Invariante: x ≥ 0 ∧ x + res = a + b  
    //Variante: x  
    while (x > 0) {  
  
        x = x - 1;  
  
        res = res + 1;  
    }  
  
    IO.println(a + " + " + b + " = " + res);  
}
```

Vorbedingung:  $a \geq 0$

Nachbedingung:  $res = a + b$

# Verifikation der Subtraktion

```
void main () {  
  
    int x = Integer.parseInt(IO.readln("Gib erste Zahl ein: ")),  
        y = Integer.parseInt(IO.readln("Gib zweite Zahl ein:")), z, res;  
  
    z = y;  
  
    res = 0;  
  
    while (x > z) {  
  
        z = z + 1;  
  
        res = res + 1;  
  
    }  
  
    IO.println(x + " - " + y + " = " + res);  
}
```

Vorbedingung:  $x \geq y$

Nachbedingung:  $res = x - y$

# Verifikation der Subtraktion

```
void main () {  
  
    int x = Integer.parseInt(IO.readln("Gib erste Zahl ein: ")),  
        y = Integer.parseInt(IO.readln("Gib zweite Zahl ein:")), z, res;  
  
    z = y;  
  
    res = 0;  
    //Invariant: x ≥ z ∧ res = z - y  
    //Variante: x - z  
    while (x > z) {  
  
        z = z + 1;  
  
        res = res + 1;  
  
    }  
  
    IO.println(x + " - " + y + " = " + res);  
}
```

Vorbedingung:  $x \geq y$

Nachbedingung:  $res = x - y$