3.2.1: Die sechs Regeln von Hoare:

Für alle Zusicherungen A, B, C, für alle Wertzuweisungen x:=ß ,
für alle Booleschen Ausdrücke b und für alle Anweisungen c, d:

1 $\dfrac{\text{true}}{\{A\}\ \text{skip}\ \{A\}}$, wobei skip die leere Anweisung (null) ist.

2 $\dfrac{\text{true}}{\{A'\}\ \mathbf{x:=ß}\ \{A\}}$, wobei A' aus A entsteht, indem man in A alle freien Vorkommen von x durch ß ersetzt.

(außer: x war zuvor undefiniert;
wir könnten stets verlangen,
dass jede Variable bei ihrer
Deklaration initialisiert wird)

3 $\dfrac{\{A\}\ \mathbf{c}\ \{B\}\ \wedge\ \{B\}\ \mathbf{d}\ \{C\}}{\{A\}\ \mathbf{c;d}\ \{C\}}$

Für alle Zusicherungen A, A", B, B", für alle Booleschen Ausdrücke b und für alle Anweisungen c, d:

4

$$\frac{\{A \wedge b\}\ \mathbf{c}\ \{B\} \ \wedge\ \{A \wedge \underline{\text{not}}(b)\}\ \mathbf{d}\ \{B\}}{\{A\}\ \mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ d\ \mathbf{end\ if}\ \{B\}}$$

5

$$\frac{\{A \wedge b\}\ \mathbf{c}\ \{A\}}{\{A\}\ \mathbf{while}\ b\ \mathbf{loop}\ c\ \mathbf{end\ loop}\ \{A \wedge \underline{\text{not}}(b)\}}$$

Ein solches A heißt Schleifen-invariante.

6

$$\frac{A \Rightarrow A'' \ \wedge\ \{A''\}\ \mathbf{c}\ \{B''\} \ \wedge\ B'' \Rightarrow B}{\{A\}\ \mathbf{c}\ \{B\}}$$

Konsequenzregel

Aufgabe 1 (Axiomatische Semantik)

```
Get(x);
y:=0;
while (y<x) loop
  y:=y+1;
end loop;
```

Zeigen Sie unter Verwendung der Schleifeninvariante {y<=x}, dass

{x<y} <Schleife> {x=y}

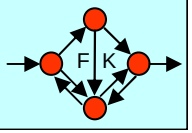
gilt (x und y sind vom Typ *natural*).

Aufgabe 2

Gegeben ist folgendes kleines Programmfragment (x und y sind vom Typ *integer*):

```
x:=x-y;
y:=x+y;
x:=y-x;
```

Beschreiben Sie umgangssprachlich die Funktion dieses Fragments. Spezifizieren Sie nun formal die Vor- und Nachbedingung. Weisen Sie die Korrektheit des Fragments nach.

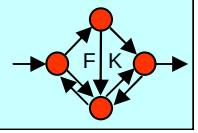


Aufgabe 3 (Axiomatische Semantik)

```
     Get(n);
     i:=0;
     Erg:=0
     while (i<n) loop
(1)    Erg:=Erg+i;
(2)    i:=i+1;
     end loop;
```

Finden Sie eine Schleifeninvariante, mit deren Hilfe Sie die realisierte Abbildung beschreiben können (`n` ist vom Typ *natural*).

Vertauschen Sie nun die Zeilen (1) und (2). Was ändert sich? Was passiert, wenn man `Erg` mit 1 initialisiert?



Aufgabe 4

Gegeben seien folgende Programmstücke (x und y sind vom Typ *natural*).

```
Get(x);
Erg:=0;
while (((Erg + 1)* (Erg + 1))<= x) loop
   Erg := Erg + 1;
end loop;
```

```
--
```

```
Get(x); Get(y); Erg:=0;
while (x>=y) loop
   x:=x-y; Erg:=Erg+1;
end loop;
```

```
--
```

```
Get(x);
y:=x;
while (y>0) loop
   y:=y-1;
   x:=x*y;
end loop;
```

Was berechnen diese Programme? Zeigen Sie Ihre Vermutung mittels der Hoareschen Regeln.

```
x, y, z: Natural;
{x=⊥ ∧ y=⊥ ∧ z=⊥}
begin Get (x); Get (y);
{a∈ IN0 ∧ a≥0 ∧ x=a ∧ b∈ IN0 ∧ b≥0 ∧ y=b ∧ z=⊥}
z:=0;
{a∈ IN0 ∧ a≥0 ∧ x=a ∧ b∈ IN0 ∧ b≥0 ∧ y=b ∧ z=0}
while y > 0 loop
        if (y mod 2 = 0) then
                y:=y div 2;
                x:=x+x;
        else
                y:=y-1;
                z:=z+x;
        end if;
end loop;
Put(z); end;
```

Wir betrachten nun nur die while-Schleife.

$\{a \in \mathbb{N}_0 \wedge a \geq 0 \wedge x = a \wedge b \in \mathbb{N}_0 \wedge b \geq 0 \wedge y = b \wedge z = 0\} \Rightarrow$
$\{x \geq 0 \wedge y \geq 0 \wedge z \geq 0 \wedge z + x \cdot y = a \cdot b\}$

```
while y > 0 loop
```
$\{x \geq 0 \wedge y > 0 \wedge z \geq 0 \wedge z + x \cdot y = a \cdot b\}$
```
        if (y mod 2 = 0) then
                y := y div 2;
                x := x + x;
        else
                y:=y-1;
                z:=z+x;
        end if;
```
$\{x \geq 0 \wedge y \geq 0 \wedge z \geq 0 \wedge z + x \cdot y = a \cdot b\}$
```
end loop;
```
$\{x \geq 0 \wedge y \geq 0 \wedge z \geq 0 \wedge z + x \cdot y = a \cdot b \wedge y \leq 0\}$
$\Rightarrow \{y = 0 \wedge z = a \cdot b\}$
```
Put(z);
```

Aufgabe 5

```
function exp(x,y:natural):natural;
var Erg:natural;
begin Erg:=0;
  while y>0 loop
    if   y mod 2 = 0 then
      y:=y div 2;
      x:=x*x;
    elsif
      y:=y-1;
      Erg:=Erg*x;
    end if;
  end loop;
  return(Erg);
end;
```

Beweisen Sie mit geeigneten Zusicherungen und den Hoareschen Regeln, dass die Funktion exp x^y berechnet.

Aufgabe 6 (ehemalige Prüfungsaufgabe)

Das folgende Programm `Aufgabe6` bestimmt die Summe aller geraden Zahlen von 2 bis X. Sie sollen zeigen, dass das Programm `Aufgabe6` bezüglich der Spezifikation

{X>= 0 und X gerade} `<Aufgabe6>` **{SUM = (X*X + 2*X)/4}**

partiell korrekt ist. Partiell korrekt heißt: Korrekt, unter der Voraussetzung, dass das Programm terminiert.

```
program Aufgabe6 is
sum, x, z : integer;
begin
   Get(x);
   sum := 0;
   z := 2;
   while z < (X+2) loop
      sum := sum + z;
      z := z + 2;
   end;
end.
```

Aufgabe 7 (weakest Precondition)

Gegeben seien die folgenden Fallunterscheidungen und deren Nachbedingung. Bestimmen Sie die Vorbedingung für die gegebenen Fallunterscheidungen.

a.
```
z = x; if (z < y)
  z = y;
```
$\{z = max(x, y)\}$

b.
```
if (x < y)
  a = x;
else
  a = y;
```
$\{a = min(x, y)\}$

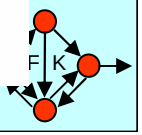
Aufgabe 8 (weakest Precondition)

Bestimmen Sie die *weakest Precondition* für die folgenden Wertzuweisungen bei angegebener Nachbedingung:

a. $wp($`a = a + 1;`$, a < b)$

b.
```
n = n + m;
```
$\{ n > 0 \}$

c. $wp($`n = m + n;`$, a > 0)$



Aufgabe 9

Leiten Sie die *schwächste Vorbedingung* (*weakest precondition)* wp aus der entsprechenden *Nachbedingung* (*postcondition*) für folgendes Programmstück ab:

```
a, b: INTEGER

a := 2 * b - 2;

if a < 16 then

  b := 2 * a - 16;

else

  b := a+8;

end if;

b := b + 4;

{ Q ≡ 8 ≤ b ≤ 20 }
```