# FPV Tutorübung

## Woche 1
## Implications, Assertions and Conditions

Manuel Lerchner

20.04.2023

# Organisatorisches

## Grade Bonus

- Successful participation ($\geq 70\%$) in quizzes and programming tasks will lead to a bonus of 0.3 in the final exam, provided that you passed the exam.
- Programming homework and quizzes are to be submitted individually.
- Discussing solutions before the end of the week is considered plagiarism.
- Plagiarism will not be tolerated and will (at the very least) lead to exclusion from the bonus system

## Changes

- Manual correction of homework not possible. However, non-programming exercises remain crucial for the exam
- 20% of the exam will be Single-Choice
- To receive points in the exam, your code needs to compile
- We currently anticipate an in-person exam using Artemis

# Materialien

# Quiz



Passwort:

_____

# T01: Recap Implications

1. $x = 1 \implies 0 < x$
2. $x < 6 \implies x = 3$
3. $x > 0 \implies x \geq 0$
4. $x = -2 \implies x < -1 \lor x > 1$
5. $x = 0 \lor x = 7 \implies 4 \neq x$
6. $x = 1 \implies x \leq 3 \land y > 0$
7. $x < 8 \land y = x \implies y \neq 12$
8. $x = 1 \lor y = 1 \implies x > 0$
9. $x \neq 5 \implies false$
10. $true \implies x \neq y$
11. $false \implies x = 1$
12. $x \geq 1 \implies 2x + 3 = 5$
13. $A \land x = y \implies A$
14. $B \implies A \lor B$
15. $A \implies (B \implies A)$
16. $(A \implies B) \implies A$

# T02: Assertions

Start

$n = read()$

$i = 0$

A

$x = 0$

B

$i < 10$

no

yes

$write(x)$

C

Stop

$x = x + n$

$i = i + 1$

1. Which of the following assertions hold at point $A$?
   - a) $i \geq 0$
   - b) $x = 0$
   - c) $i \leq 10 \wedge x \neq 0$
   - d) $true$
   - e) $i = 0$
   - f) $x = i$

2. Which of the following assertions hold at point $B$?
   - a) $x = 0 \wedge i = 0$
   - b) $x = i$
   - c) $i < x$
   - d) $0 \leq i \leq 10$
   - e) $i \geq 0 \wedge x \geq 0$
   - f) $n = 1 \implies x = i$

3. Which of the following assertions hold at point $C$?
   - a) $i \geq 0$
   - b) $i = 10$
   - c) $i > 0$
   - d) $x \neq n$
   - e) $x = 10n$
   - f) $x = i * n \wedge i = 10$

# T03: The Strong and the Weak

3. Which of the following assertions hold at point $C$?

- a) $i \geq 0$ ✓
- b) $i = 10$ ✓
- c) $i > 0$ ✓
- d) $x \neq n$ ✗
- e) $x = 10n$ ✓
- f) $x = i * n \land i = 10$ ✓

Again consider the assertions that hold at point $C$ of assignment 2. Discuss the following questions:

1. When annotating the control flow graph, can you say that one of the given assertions is "better" than the others?
2. Can you arrange the given assertions in a meaningful order?
3. How can you define a *stronger than* relation formally?
4. How do $true$ and $false$ fit in and what is their meaning as an assertion?
5. What are the strongest assertions that still hold at $A$, $B$ and $C$?

# T04: Strongest Postconditions 1

1.

$$true$$

$$\boxed{x = 5}$$

$$B$$

2.

$$x = 0 \wedge y = 2$$

$$\boxed{x = 18}$$

$$B$$

3.

$$x \geq 7 \wedge y = -2$$

$$\boxed{x = y + x}$$

$$B$$

4.

$$x = 3 * i \wedge i \geq 0$$

$$\boxed{i = i + 1}$$

$$B$$

5.

$$y = 2 + x$$

$$\boxed{x = read()}$$

$$B$$

6.

$$i = i + 1$$

$$\boxed{i = 0}$$

$$B$$

# T04: Strongest Postconditions 2

7.


$$n \geq 2 \wedge x = 12$$

$yes$    $n > 3$    $no$

$B_{yes}$      $B_{no}$

8.


$$x = 2i \wedge 0 > i$$

$yes$    $i == 0$    $no$

$B_{yes}$      $B_{no}$
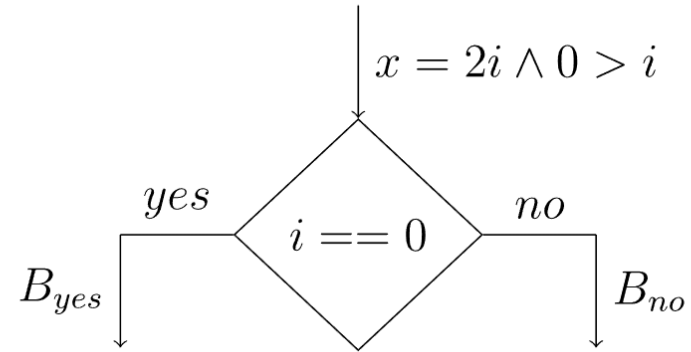
# T04: Strongest Postconditions 3

9.

$$i = 0 \wedge x > 0 \qquad\qquad i \neq 0 \wedge y = 1$$

$$B$$

# FPV Tutorübung

## Woche 2
## Preconditions, Postconditions and Local Consistency

Manuel Lerchner

03.05.2023

# Quiz



Passwort:

_____

# T01: From Post- to Preconditions

1.



$r = 17$

$t = read()$

$r = 14 - r$

$r = 8 - 2 * r$

$t = 3 * r$

$Z :\equiv t \geq 0$

2.



$y = read()$

$x = 42 - y$

$y = 37$

$y = x - y$

$x = y * y + 4$

$write(x)$

$Z :\equiv x = 29$

3.



$n = read()$

$n < 0$    yes    no

$k = -1$      $k = 1$

$n = 1 + n * k$

$Z :\equiv n \geq 1$

1. For each of these graphs show whether the assertion $Z$ holds...
   (a) ...using strongest postconditions and
   (b) ...using weakest preconditions.
2. Discuss advantages and disadvantages of either approach.

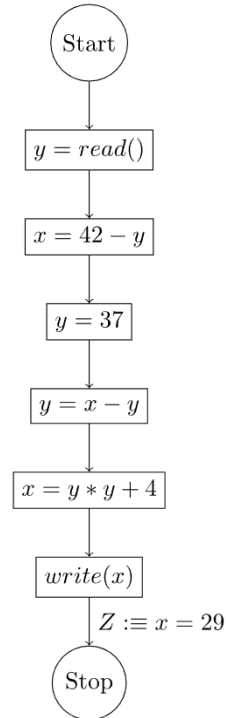# T01: From Post- to Preconditions 1
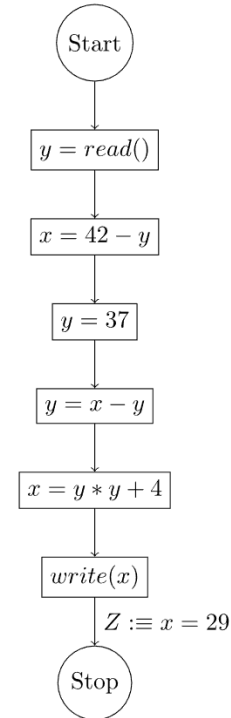
Post-Condition:



Pre-Condition:

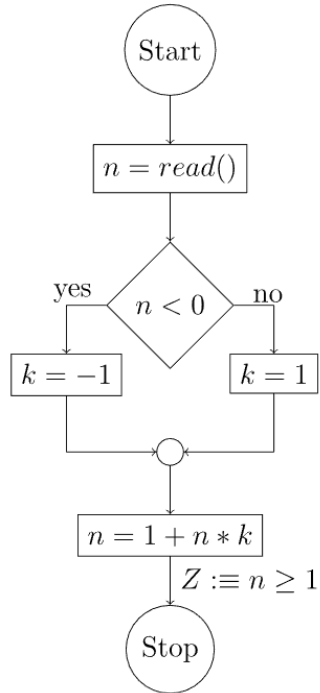# T01: From Post- to Preconditions 2

Post-Condition:



Pre-Condition:

# T01: From Post- to Preconditions 3
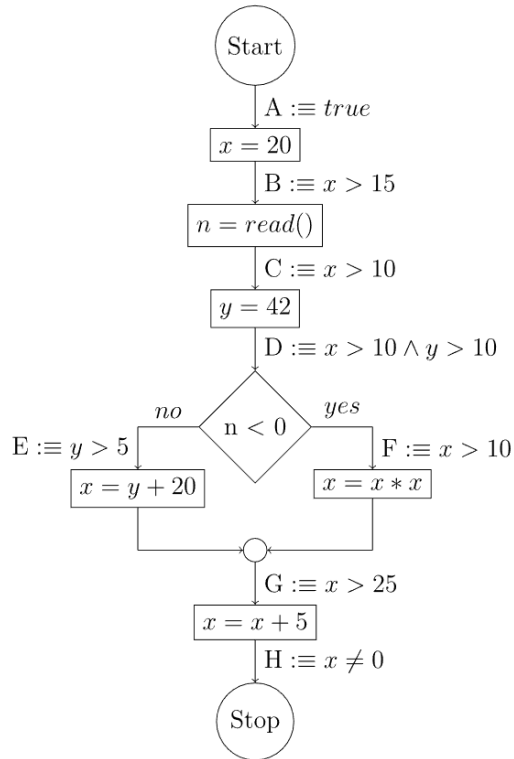
Post-Condition:



Pre-Condition:

# T02: Local Consistency



Start

A $:\equiv true$

$x = 20$

B $:\equiv x > 15$

$n = read()$

C $:\equiv x > 10$

$y = 42$

D $:\equiv x > 10 \wedge y > 10$

$n < 0$

$no$     $yes$

E $:\equiv y > 5$          F $:\equiv x > 10$

$x = y + 20$          $x = x * x$

G $:\equiv x > 25$

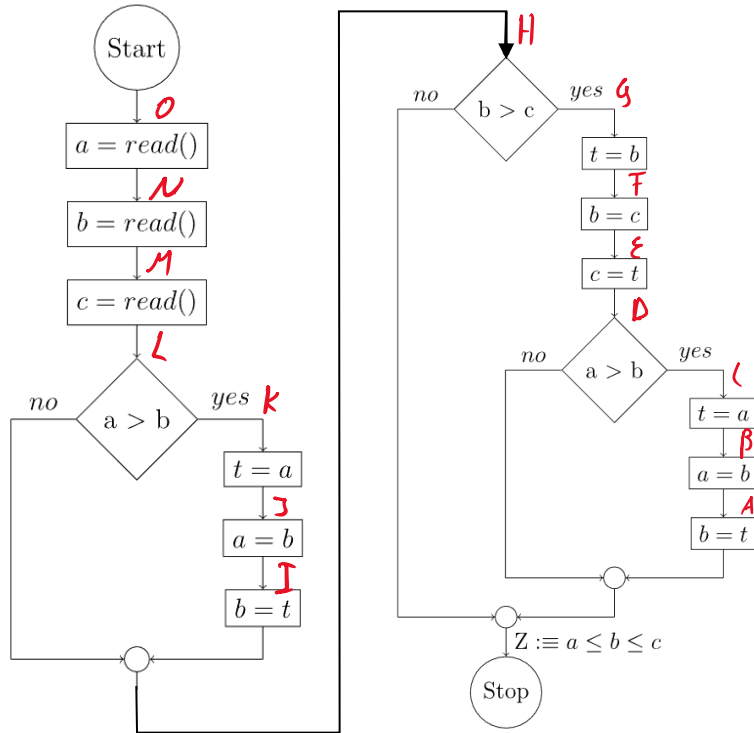$x = x + 5$

H $:\equiv x \neq 0$

Stop

Check whether the annotated assertions prove that the program computes an $x \neq 0$ and discuss why this is the case.

# T02: Local Consistency (Extra Space)

# T03: Trouble Sort



1. Annotate each program point in the following control flow diagram with a suitable assertion, then show that your annotations are locally consistent and prove that $Z$ holds at the given program point.
2. Discuss the drawbacks of annotating each program point with an assertion before applying weakest preconditions, and discuss how you could optimize the approach to proving that $Z$ holds.

# T03: Trouble Sort (Extra Space)
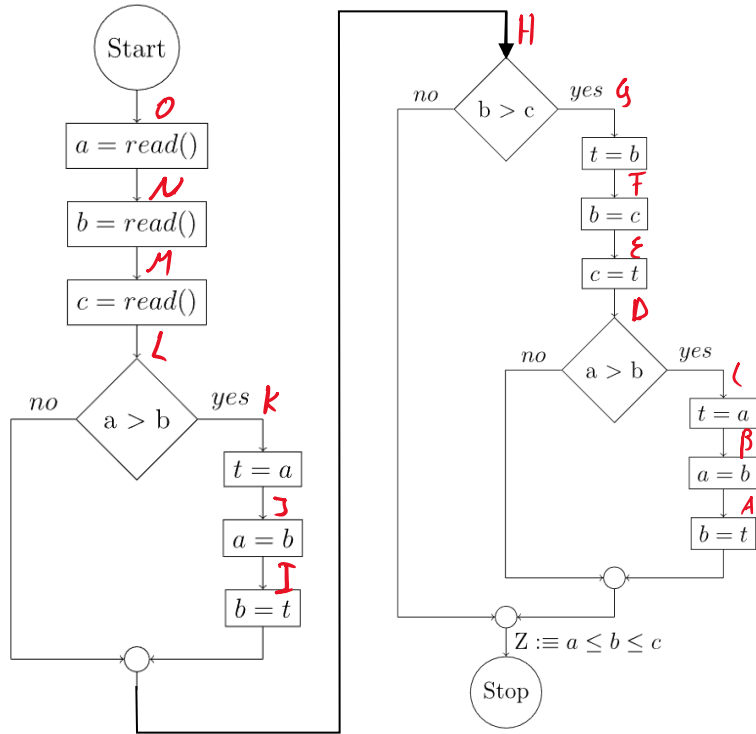
# FPV Tutorübung

## Woche 3
## MiniJava 2.0, Loop Invariants

Manuel Lerchner

09.05.2023

# Quiz



Passwort:

_____

# T01: MiniJava 2.0

In the lecture, the weakest precondition operator has been defined for all statements of MiniJava. In this assignment, we consider an extension of the MiniJava language, which provides four new statements:

1. `rand x`:

   Assigns a random value to variable $x$,

2. `x = either` $e_0, \ldots, e_k$:

   Assigns one of the values of the expressions $e_0, \ldots, e_k$ to variable `x` non-deterministically,

3. `x = e in a, b`:

   Assigns the value $1$ to variable `x`, if the value of expression `e` is in the range $[a, b]$ and $0$ if `e` is not in the range or the range is empty $(a > b)$,

4. `stop`:
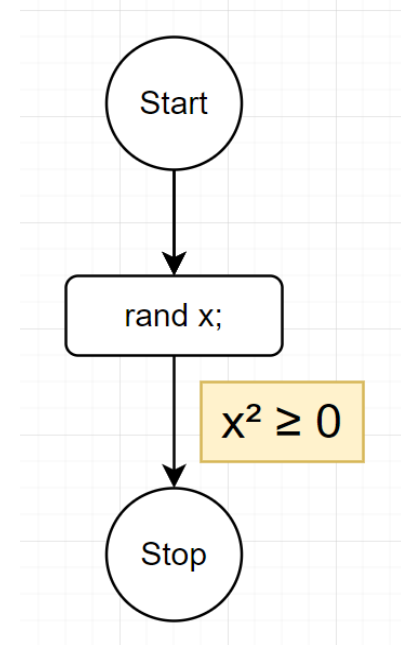
   Immediately stops the program.

Define the weakest precondition operator $\mathbf{WP}[\![\ldots]\!](B)$ for each of these statements. (In terms of B)

## Beispiele zum Testen:
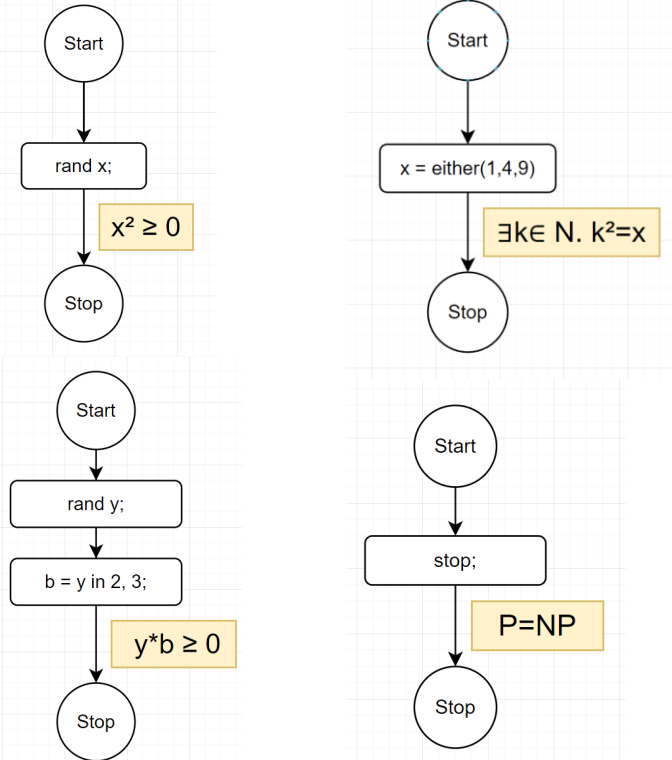
# T01: MiniJava 2.0
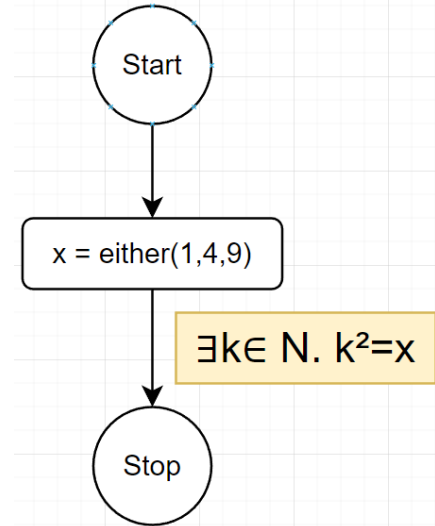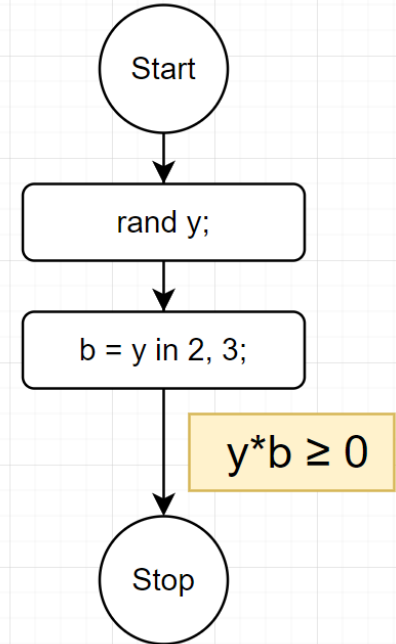
WP[rand x;](B) =

# T01: MiniJava 2.0
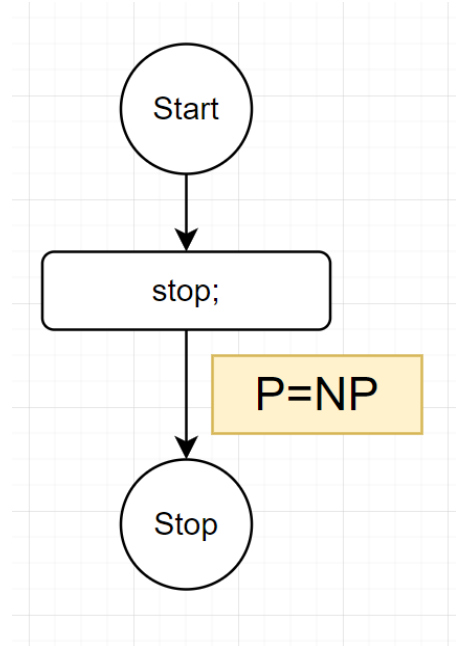
$WP[x = \text{either } e_0, e_1 \ldots e_k](B) =$



Start

x = either(1,4,9)

$\exists k \in \mathbb{N}.\ k^2 = x$

Stop

# T01: MiniJava 2.0

WP[x e in a, b](B) =

# T01: MiniJava 2.0

WP[stop](B) =

# T02: Loop Invariants

1. Discuss the problem that arises when computing weakest preconditions to prove $Z$.
2. How can you use weakest preconditions to prove $Z$ anyway?
3. Try proving $Z$ using the the loop invariants $x \geq 0$ and $i = 0 \land x = 1 \land n = 0$ at **the end of the loop body** and in particular discuss these questions:
   - a) How has a useful loop invariant be related to $Z$?
   - b) What happens if the loop invariant is chosen too strong?
   - c) What happens if the loop invariant is chosen too weak?
   - d) Can you give a meaningful lower and upper bound for useful loop invariants?
4. Retry proving $Z$ using the loop invariant $x = i!$ (again at the end of the loop body) and improve this invariant until the proof succeeds.

A program computes the factorial of its input:
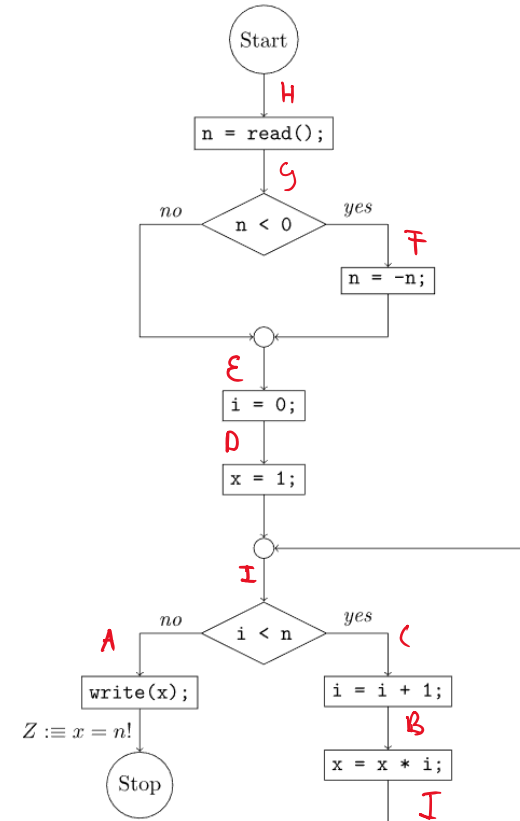
# T02: Loop Invariants 1

3. Try proving $Z$ using the the loop invariants $x \geq 0$ and $i = 0 \wedge x = 1 \wedge n = 0$ at **the end of the loop body** and in particular discuss these questions:

A program computes the factorial of its input:

# T02: Loop Invariants 2

3. Try proving $Z$ using the the loop invariants $x \geq 0$ and $i = 0 \land x = 1 \land n = 0$ at **the end of the loop body** and in particular discuss these questions:

A program computes the factorial of its input:



$$Z :\equiv x = n!$$
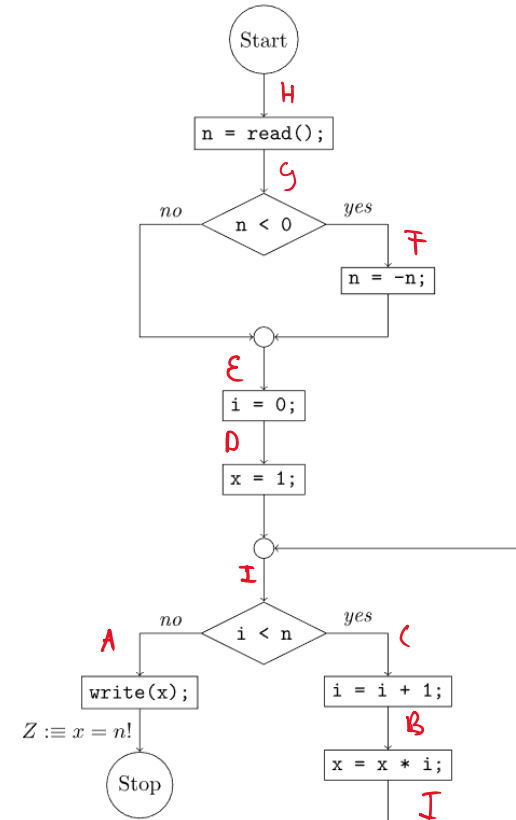
# T02: Loop Invariants 3

4. Retry proving $Z$ using the loop invariant $x = i!$ (again at the end of the loop body) and improve this invariant until the proof succeeds.
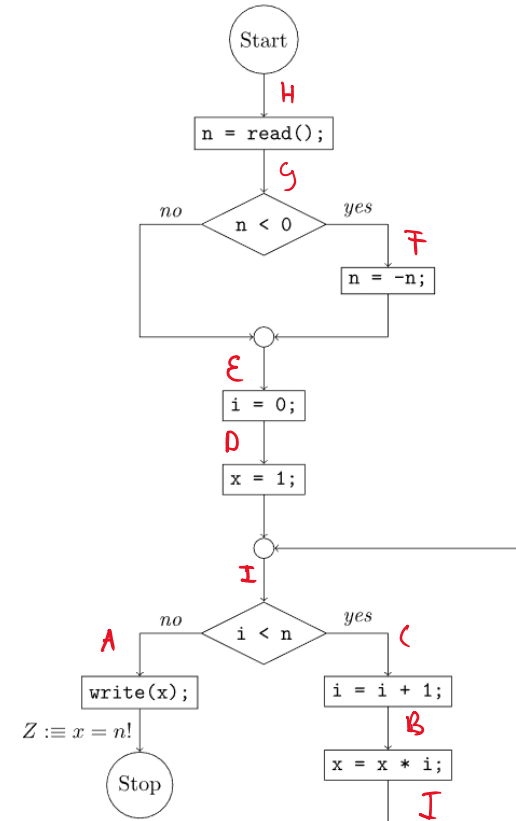


A program computes the factorial of its input:

$Z :\equiv x = n!$
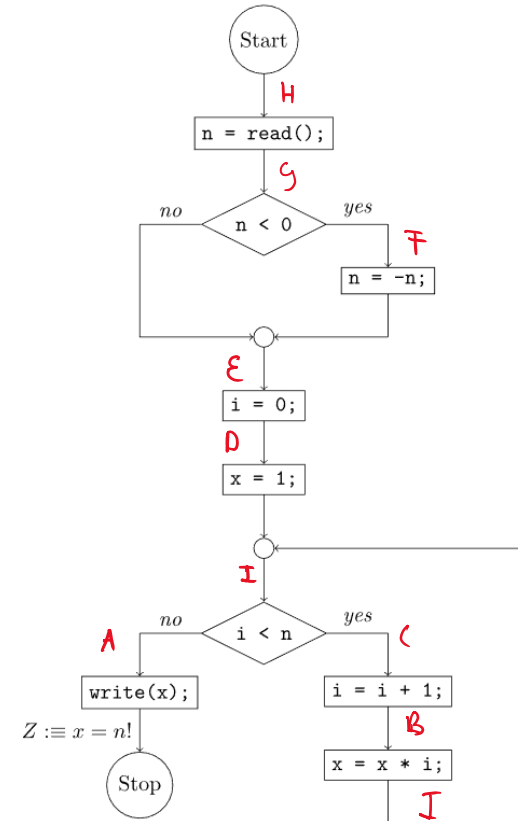
# T03: Two b, or Not Two b

Prove $Z$ using weakest preconditions.
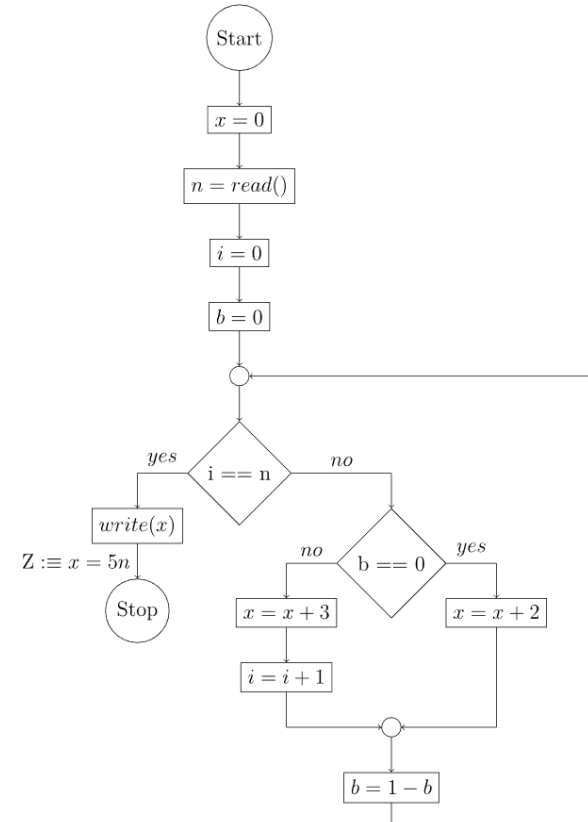
# T03: Two b, or Not Two b

Tipps zum finden von Loop Invarianten:

https://ttt.in.tum.de/recordings/Info2_2017_11_24-1/Info2_2017_11_24-1.mp4

Beispieltrace: n=3

| Variable \ Schleifendurchgang | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| x | 0 | 2 | 5 | 7 | 10 | 12 | 15 |
| i | 0 | 0 | 1 | 1 | 2 | 2 | 3 |
| b | 0 | 1 | 0 | 1 | 0 | 1 | 0 |



Start

$x = 0$

$n = read()$

$i = 0$

$b = 0$

$i == n$ — yes / no

yes: $write(x)$

$Z :\equiv x = 5n$

Stop

$b == 0$ — no / yes

no: $x = x + 3$

$i = i + 1$

yes: $x = x + 2$

$b = 1 - b$

# Tipps für Loop Invarianten

https://ttt.in.tum.de/recordings/Info2_2017_11_24-1/Info2_2017_11_24-1.mp4
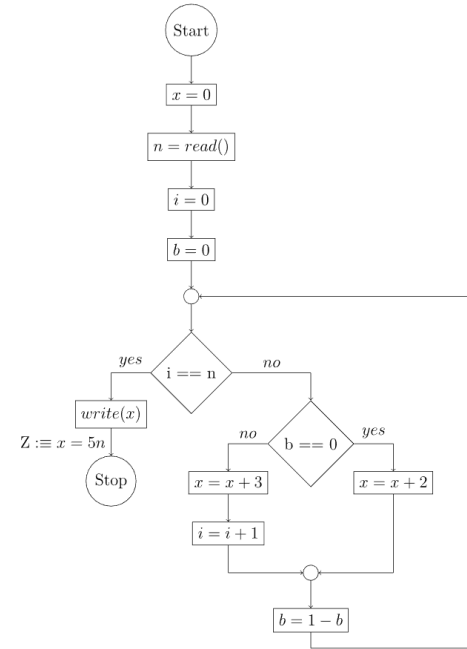
### Tipp 1

Wir benötigen eine Aussage über den Wert der Variablen, über die wir etwas beweisen wollen ($x$) in der Schleifeninvariante. Die Aussage muss dabei mindestens so präzise ($\neq, \geq, \leq, =$) sein, wie die Aussage, die wir beweisen wollen.

### Tipp 2

Variablen, die an der Berechnung von $x$ beteiligt sind **und** Werte von einer Schleifeniteration in die nächste transportieren ("loop-carried"), müssen in die Schleifeninvariante aufgenommen werden.

### Tipp 3

Die Schleife zu verstehen ist unerlässlich. Eine Tabelle für einige Schleifendurchläufe kann helfen die Zusammenhänge der Variablen (insbesondere mit dem Schleifenzähler $i$) aufzudecken. Oft lassen sich mit einer Tabelle, in der man die einzelnen Berechnungsschritte notiert, diese Zusammenhänge deutlich leichter erkennen, als mit einer Tabelle, die nur konkrete Werte enthält.

$$I :\equiv x = 5i + 2b \land b \in \{0,1\} \land (i = n \implies b = 0)$$

# FPV Tutorübung

## Woche 4
## Loop Invariants and Termination proofs

Manuel Lerchner

15.05.2023

# Quiz



Passwort:

_____

# T01: Loop Invariants

Find a suitable loop invariant and prove it locally consistent.

*Note: We follow the standard practice that the empty sum, where the number of terms is zero, is 0, e.g.:* $\sum_{k=0}^{-1} (\dots) = 0.$



$$x = \sum_{k=0}^{n} 5k$$
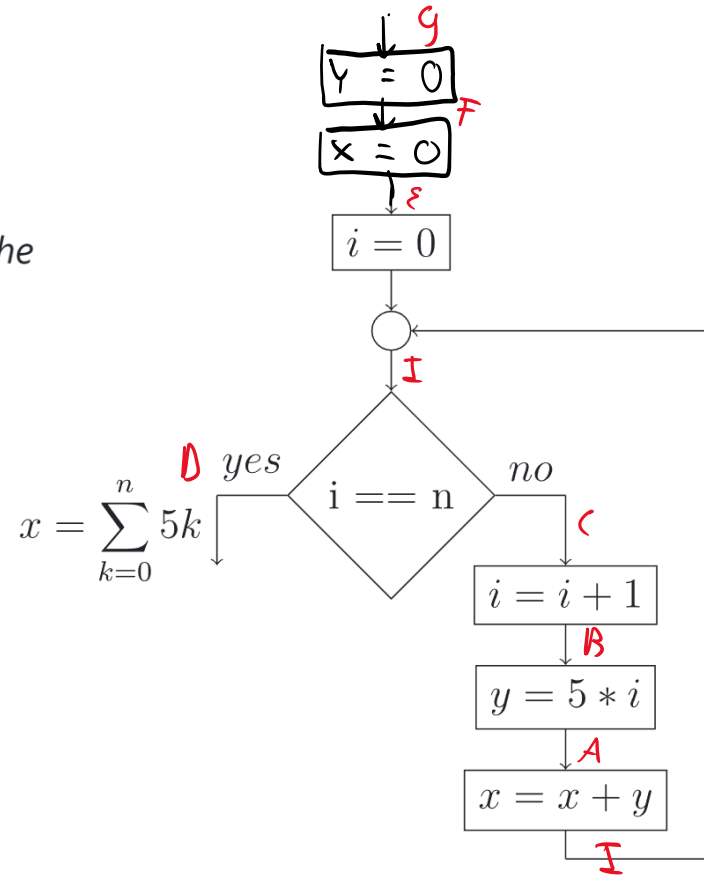
# T01: Loop Invariants
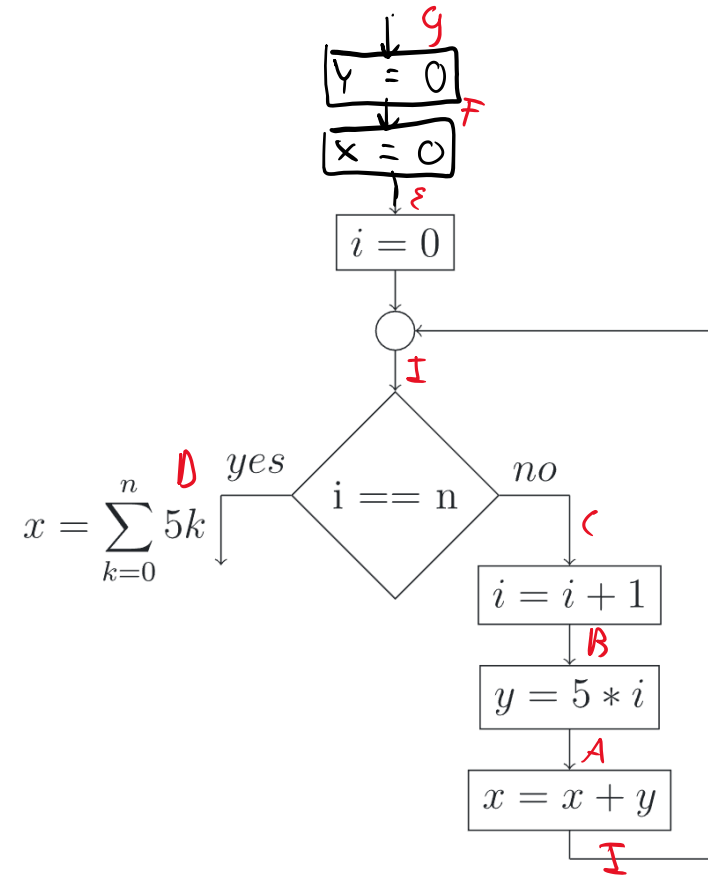
# T02: Termination

In the lecture, you have learned how to prove termination of a MiniJava program. Discuss these questions:

1. How can you decide whether a termination proof is required at all?
2. What is the basic idea of the termination proof?
3. How is the program to be modified?
4. What has to be proven?
5. How is the loop invariant influenced?
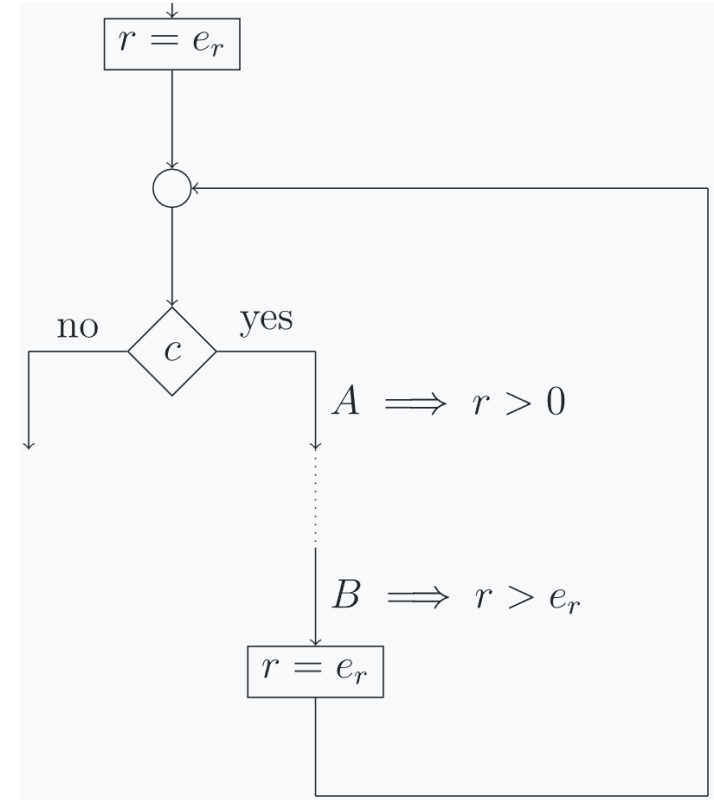
# T03: A Wavy Approach

Prove termination of the following program:

**Todos:**

1. Schleife verstehen
2. Variable r definieren / finden
   - r ≥ 0 in jedem Durchgang
   - r wird strikt kleiner
3. Neue Variable und Assertions einfügen
   - Am Ende „true" Assertion!
4. Local-Consistency zeigen

# T03: A Wavy Approach

# Tipps für Loop Invarianten

https://ttt.in.tum.de/recordings/Info2_2017_11_24-1/Info2_2017_11_24-1.mp4

**Tipp 1**

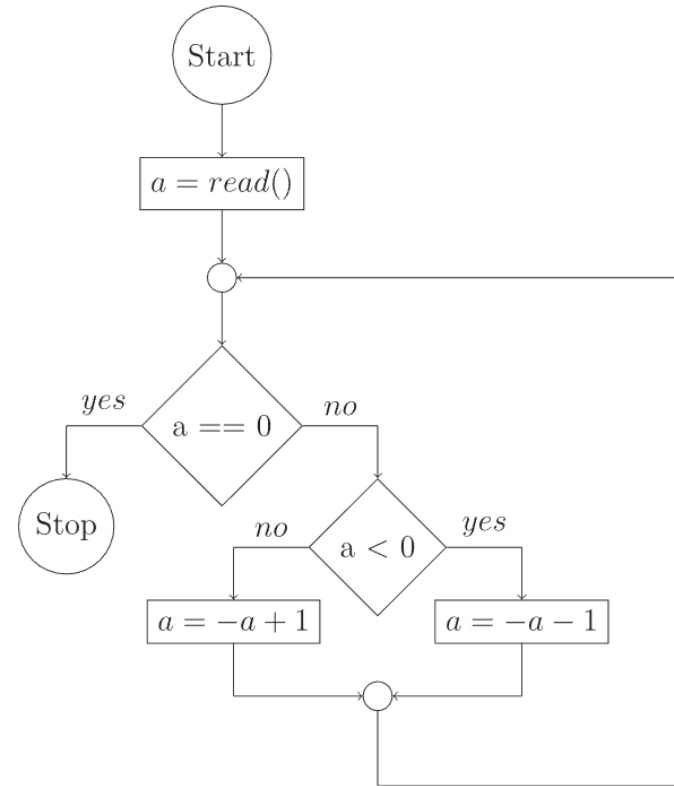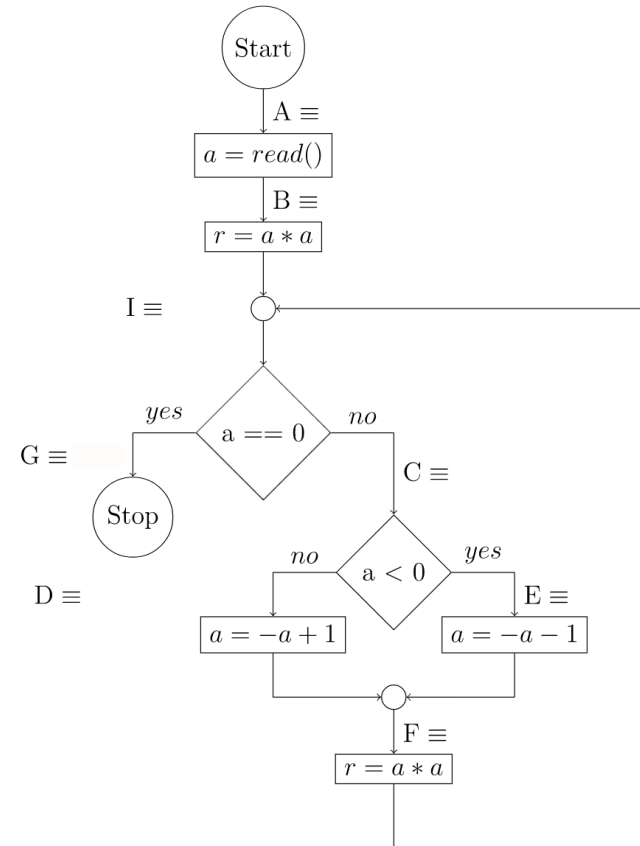Wir benötigen eine Aussage über den Wert der Variablen, über die wir etwas beweisen wollen ($x$) in der Schleifeninvariante. Die Aussage muss dabei mindestens so präzise ($\neq, \geq, \leq, =$) sein, wie die Aussage, die wir beweisen wollen.

**Tipp 2**

Variablen, die an der Berechnung von $x$ beteiligt sind **und** Werte von einer Schleifeniteration in die nächste transportieren ("loop-carried"), müssen in die Schleifeninvariante aufgenommen werden.

**Tipp 3**

Die Schleife zu verstehen ist unerlässlich. Eine Tabelle für einige Schleifendurchläufe kann helfen die Zusammenhänge der Variablen (insbesondere mit dem Schleifenzähler $i$) aufzudecken. Oft lassen sich mit einer Tabelle, in der man die einzelnen Berechnungsschritte notiert, diese Zusammenhänge deutlich leichter erkennen, als mit einer Tabelle, die nur konkrete Werte enthält.



$$I :\equiv x = 5i + 2b \wedge b \in \{0, 1\} \wedge (i = n \implies b = 0)$$

# FPV Tutorübung

## Woche 5

## Ocaml

Manuel Lerchner

22.05.2023

# T01: Expressions

So far, you learned about the following types of expressions:

- Constants
- Variables
- Unary operators
- Binary operators
- Tuples
- Records
- Lists
- If-then-else
- Pattern matching
- Function definition
- Function application
- Variable binding

1. For each of the aforementioned types of expressions, give the general structure and two concrete examples with different subexpressions.

# T01: Expressions

- Constants:

- Variables:

- Unary Operator:

- Binary Operator:

- Tuples:

# T01: Expressions

- Records (definition):

- Records (access):

- Lists:

- if-then-else:

# T01: Expressions

- Pattern Matching:

- Function Definition :

- Function Application :

- Variable Binding:

# T01: Expressions

2. For the following expressions, list all contained subexpressions and give their corresponding types. Then evaluate the expressions:

```
(* a *) let a = fun x y -> x + 2 in a 3 8 :: []
```

```
(* a *) let a = fun x y -> x + 2 in a 3 8 :: []
```

```
(* b *) ((fun x -> x::[]) (9 - 5), true, ('a', 7))
```

# T01: Expressions

```
(* a *) let a = fun x y -> x + 2 in a 3 8 :: []
```

# T01: Expressions

```
(* b *) ((fun x -> x::[]) (9 - 5), true, ('a', 7))
```

# T02: What's the Point

Using what you learned about tuple types in the lecture, implement functionality for computing with three-dimensional vectors.

1. ⊗ **Define a suitable data type for your point.** 0 of 1 tests passing
   The type `vector3` should be a tuple of 3 float values.
2. ❓ **Define three points** 0 of 1 tests passing
   The points `p1`, `p2` and `p3` should all be different, but their exact values don't matter. Use them, along with other points, to test your functions.
3. ❓ **string_of_vector3** 0 of 1 tests passing
   Implement a function `string_of_vector3 : vector3 -> string` to convert a vector into a human-readable representation.
   For example, the string for the zero vector should be: `(0.,0.,0.)`.
   **Hint**: use `string_of_float` to convert components.
4. ❓ **vector3_add** 0 of 1 tests passing
   Write a function `vector3_add : vector3 -> vector3 -> vector3` that adds two vectors component-wise.
5. ❓ **vector3_max** 0 of 1 tests passing
   Write a function `vector3_max : vector3 -> vector3 -> vector3` that returns the larger argument vector (the vector with the greater magnitude).
6. ❓ **combine** 0 of 1 tests passing
   Write a function `combine : vector3 -> vector3 -> vector3 -> string` that adds its first argument to the larger of the other two arguments and returns the result as a string.

# T03: Student Database

In this assignment, you have to manage the students of a university.

1. **?** **Type** No results
   First you need to define some types.
   - Define a data type for a `student`.
     A student should be represented as a record of the students `first_name`, `last_name`, identification number `id`, number of the current `semester` as well as the list of `grades` received in different courses.
     The grades should be a pair of the course number and the grade value, a floating point number.
   - To actually manage student you need a `database` which shall be represented as a list of students.
2. **?** **insert** No results
   Write a function `insert : student -> database -> database` that inserts a student into the database.
3. **?** **find_by_id** No results
   Write a function `find_by_id : int -> database -> student list` that returns a list with the (first) student with the given id (either a single student or an empty list, if no such student exists).
4. **?** **find_by_last_name** No results
   Implement a function `find_by_last_name : string -> database -> student list` to find all students with a given last name.