

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

The screenshot shows a GitHub repository page for 'fpv-tutorial-SS23'. The repository is public and was generated from 'ManuelLerchner/markdown-script'. The 'Code' tab is selected, showing a list of recent commits from 'github-actions[bot]'. The commits include updates to PDFs, documentation, and shell scripts. Below the commits, there's a preview of the README.md file, which contains the title 'FPV Tutorial - SS23' and two status indicators: 'Rerender PDFs' and 'Deploy static content to Pages', both showing 'passing'. The 'About' section states: 'Materialien für Manuel's FPV-Tutorium im Sommersemester 2023.' A note at the bottom says: 'Die Materialien sind privat erstellt und können Fehler enthalten. Im Zweifelsfall haben immer die offiziellen'. The repository has 32 commits, 1 branch, and 0 tags. It has 0 stars, 1 watching, and 0 forks. The 'Languages' chart shows OCaml at 61.5% and Shell at 38.5%. The 'Environments' section shows 'github-pages' as active.

ManuelLerchner / fpv-tutorial-SS23 · Zellen 334 · Public
generated from ManuelLerchner/markdown-script

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master · 1 branch · 0 tags

github-actions[bot] Update PDFs · 0778f55 · 2 weeks ago · 32 commits

- .github/workflows fix · 2 weeks ago
- docs Update PDFs · 2 weeks ago
- md add slide template · 2 weeks ago
- ocaml clean up project · 2 weeks ago
- slides clean up project · 2 weeks ago
- .gitignore improve rendering · 2 weeks ago
- README.md add slide template · 2 weeks ago
- render.sh initial commit · last month

README.md

FPV Tutorial - SS23

Rerender PDFs passing Deploy static content to Pages passing

About

Materialien für Manuel's FPV-Tutorium im Sommersemester 2023.

Die Materialien sind privat erstellt und können Fehler enthalten. Im Zweifelsfall haben immer die offiziellen

Pin Unwatch 1 Fork

About

Materialien für das FPV-Tutorium im Sommersemester 2023

[manuellerchner.github.io/fpv-tutorial-SS...](#)

Readme
0 stars
1 watching
0 forks

Environments

github-pages Active

Languages

OCaml 61.5% Shell 38.5%

Quiz

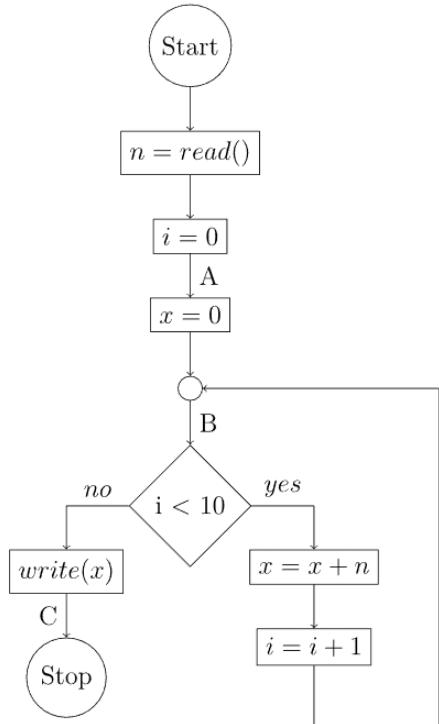
The screenshot shows the Artemis 6.1.3 interface. At the top, there is a navigation bar with icons for notifications, sun, user ge47wer, and a menu. Below the navigation bar, the course path is displayed: Courses > Funktionale Programmierung und Verifikation (Sommersemester 2023) > Exercises > Week 02 Quiz. The main content area shows a dark banner for "Week 02 Quiz" with a green "Quiz" button. Below the banner, it says "Points: 20". A message indicates "The quiz is not active." with three small interactive icons. At the bottom, there is a "Communication" section with a search bar, filters for "Unresolved", "Own", and "Reacted" posts, and a date filter. It displays the message "No posts found." with a blue "+" button.

Passwort:

T01: Recap Implications

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

T02: Assertions



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) $x = i * n \wedge i = 10$

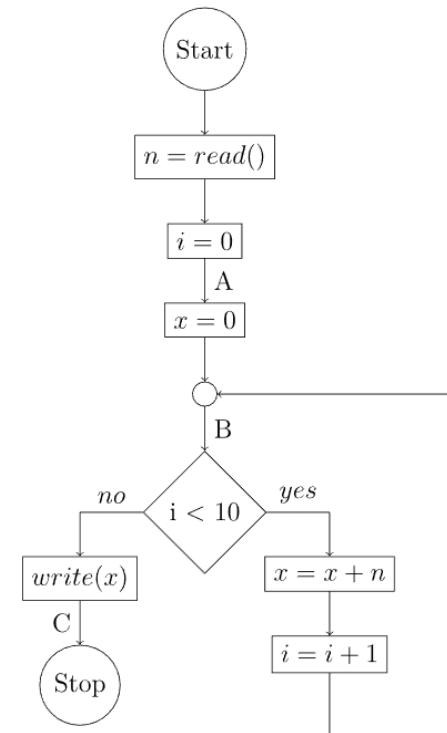
T03: The Strong and the Weak

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

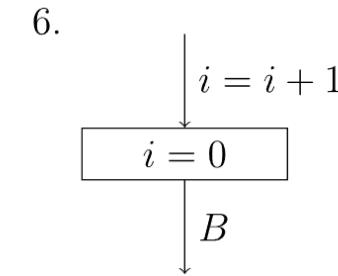
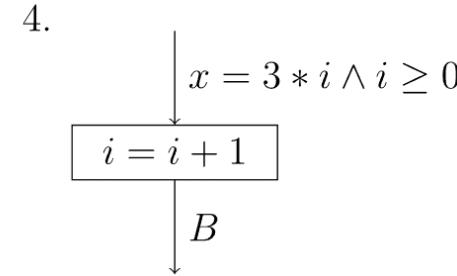
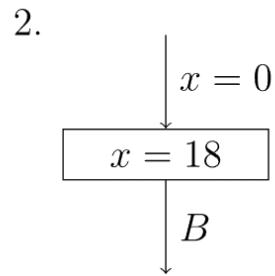
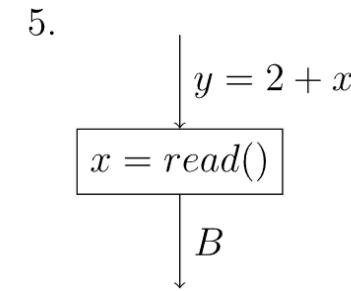
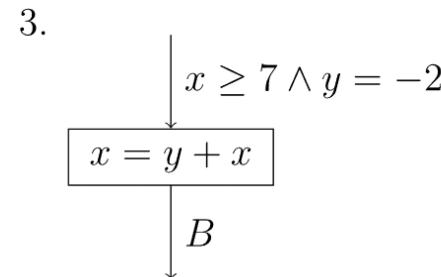
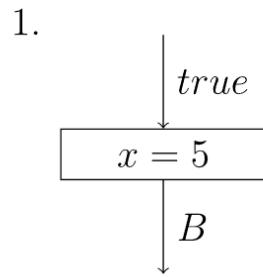
- o a) $i \geq 0$ ✓
- o b) $i = 10$ ✓
- o c) $i > 0$ ✓
- o d) $x \neq n$ ✗
- o e) $x = 10n$ ✓
- o f) $x = i * n \wedge i = 10$ ✓

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 ?

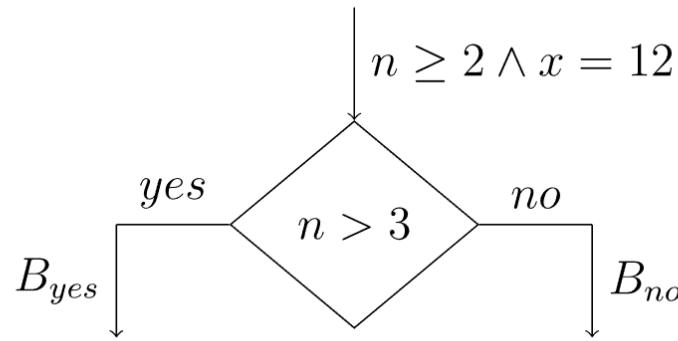


T04: Strongest Postconditions 1

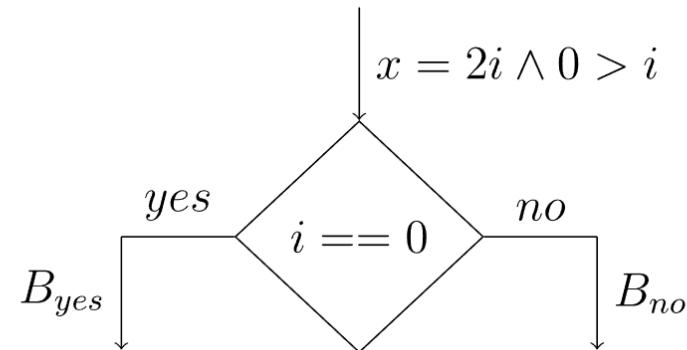


T04: Strongest Postconditions 2

7.

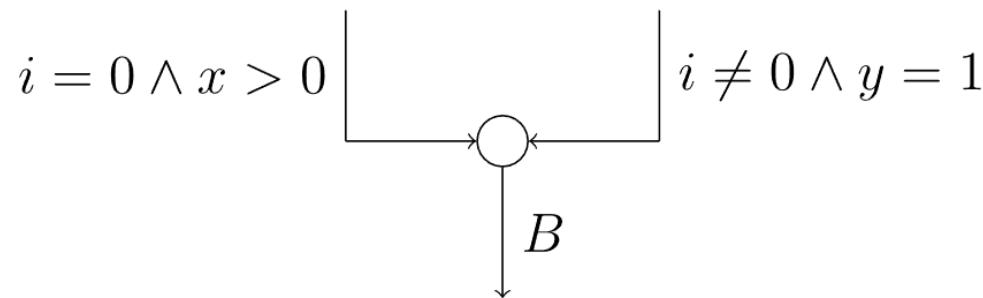


8.



T04: Strongest Postconditions 3

9.



FPV Tutorübung

Woche 2

Preconditions, Postconditions and Local Consistency

Manuel Lerchner

03.05.2023

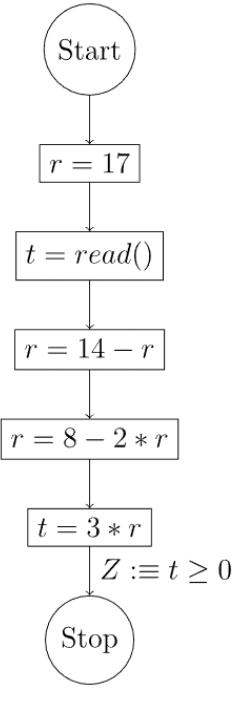
Quiz

The screenshot shows the Artemis 6.1.3 interface. At the top, there are navigation icons for notifications, brightness, user ge47wer, and a menu. Below the header, the breadcrumb navigation shows: Courses > Funktionale Programmierung und Verifikation (Sommersemester 2023) > Exercises > Week 02 Quiz. The main content area displays a dark card for "Week 02 Quiz" with a green "Quiz" button. It shows "Points: 20". Below this, a message says "The quiz is not active." with three small interactive icons. A "Communication" section follows, featuring a search bar, filter options for "Unresolved", "Own", and "Reacted" posts, and a date range selector. The message "No posts found." is displayed, along with a blue "+" button.

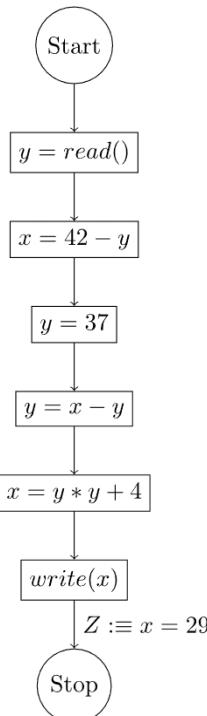
Passwort:

T01: From Post- to Preconditions

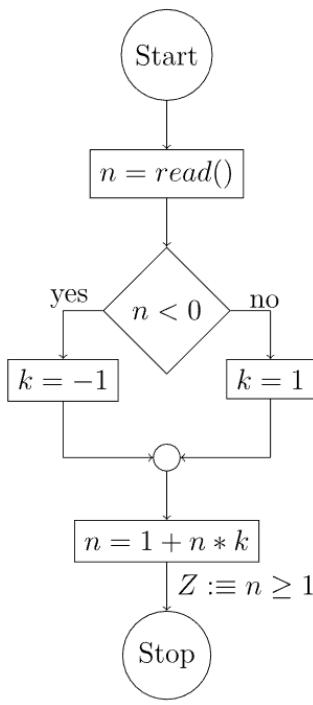
1.



2.



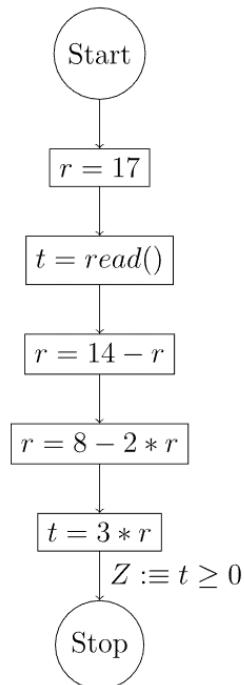
3.



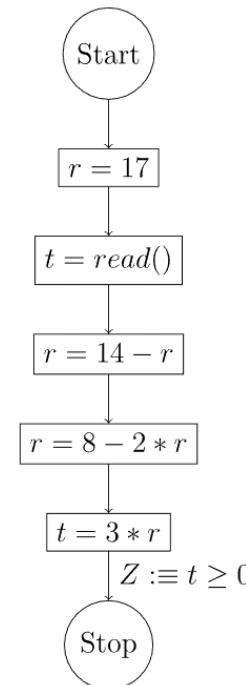
1. For each of these graphs show whether the assertion Z holds...
 - (a) ...using strongest postconditions and
 - (b) ...using weakest preconditions.
2. Discuss advantages and disadvantages of either approach.

T01: From Post- to Preconditions 1

Post-Condition:

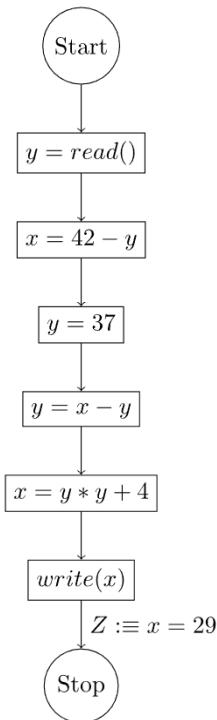


Pre-Condition:

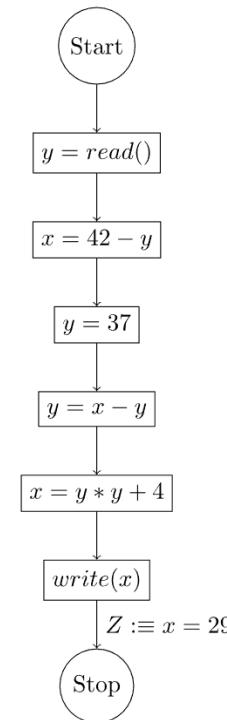


T01: From Post- to Preconditions 2

Post-Condition:

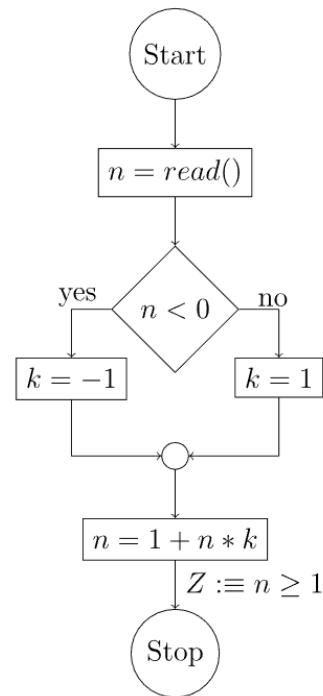


Pre-Condition:

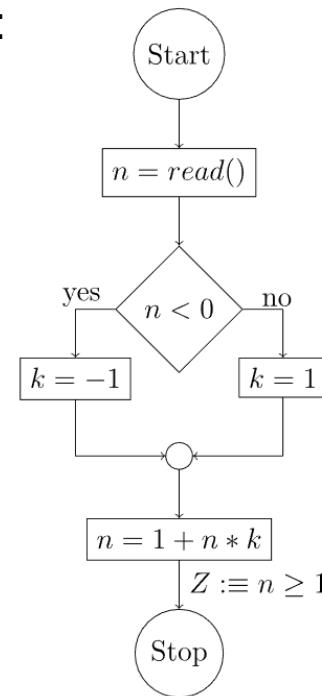


T01: From Post- to Preconditions 3

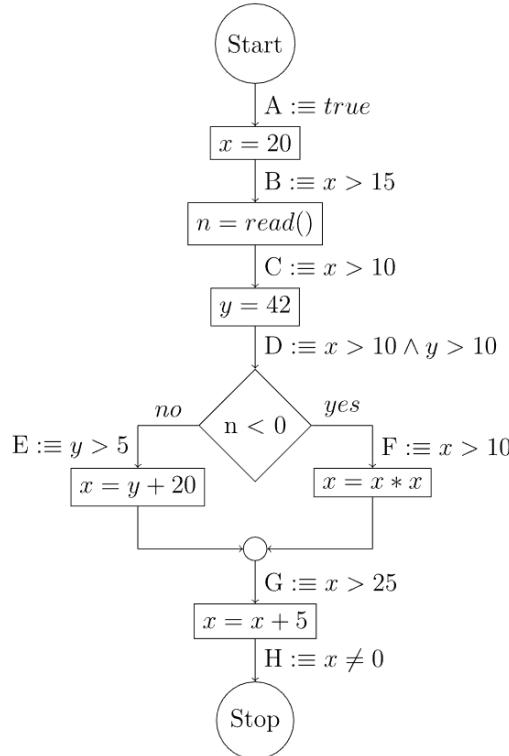
Post-Condition:



Pre-Condition:

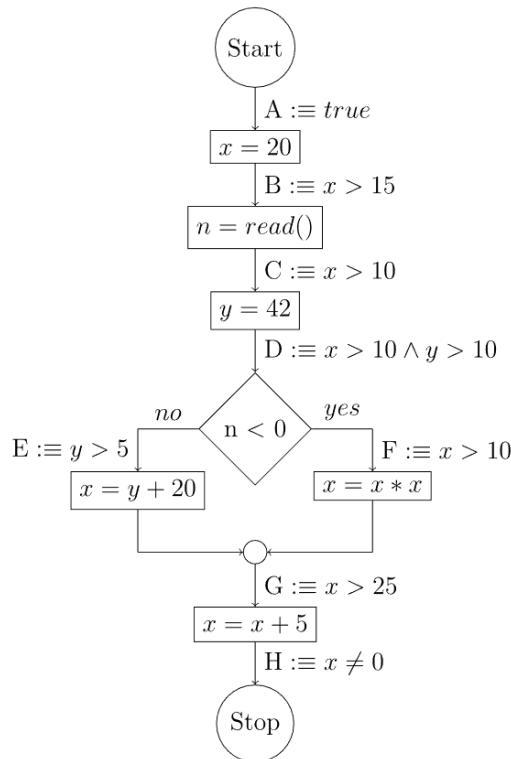


T02: Local Consistency

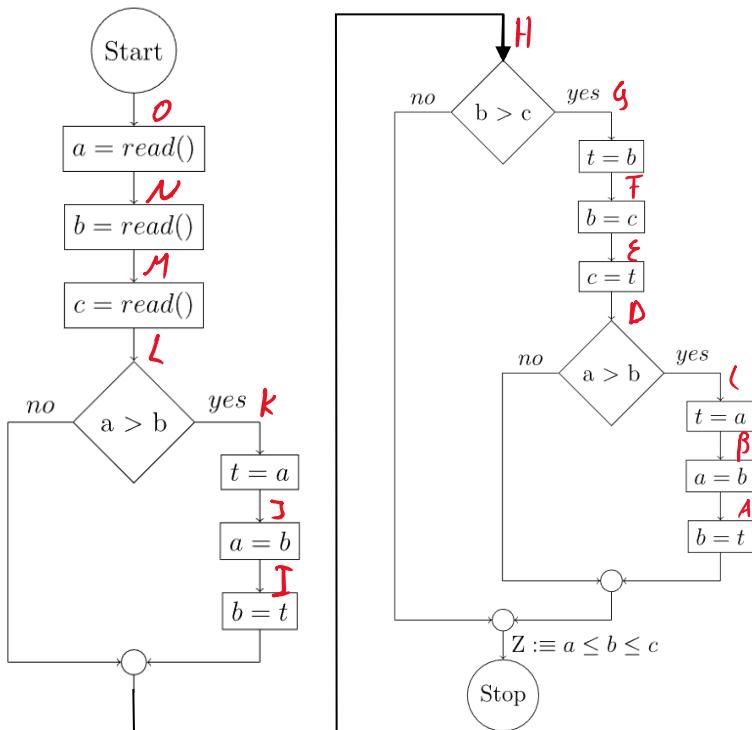


Check whether the annotated assertions prove that the program computes an $x \neq 0$ and discuss why this is the case.

T02: Local Consistency (Extra Space)

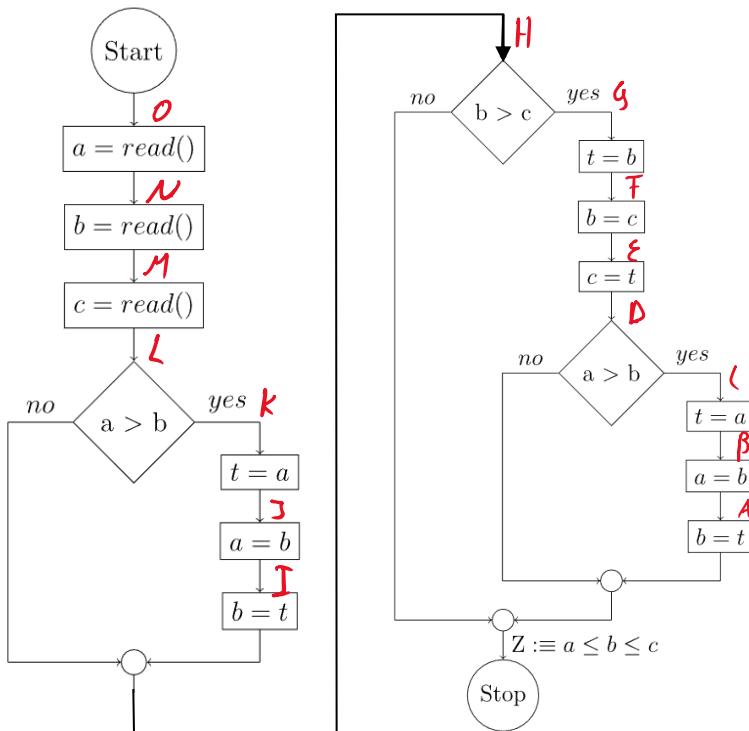


T03: Trouble Sort



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

The screenshot shows the Artemis 6.1.6 interface. At the top, it says "Artemis 6.1.6". Below that, a breadcrumb navigation shows "Courses > Funktionale Programmierung und Verifikation (Sommersemester 2023) >". The main content area displays a green checkmark icon followed by "Week 03 Quiz" and a green "Quiz" button. Below this, the text "Points: 20" is shown. A blue "Open quiz" button is at the bottom.

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, \dots, e_k` :

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

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

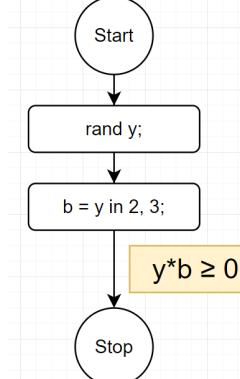
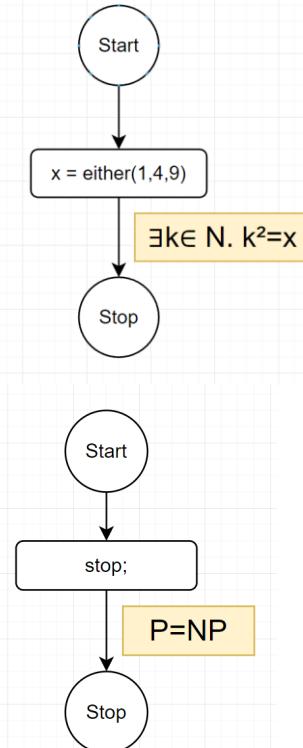
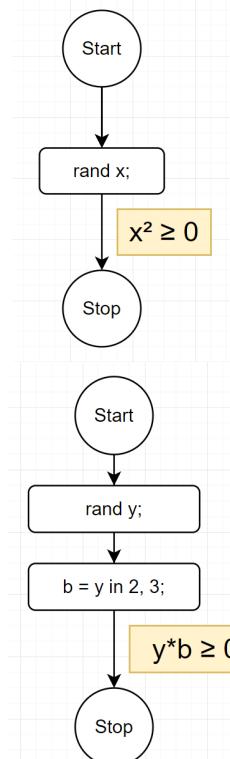
Assigns the value 1 to variable x , if the value of expression e is in the range $[a, b]$ and 0 if e is not in the range or the range is empty ($a > b$),

4. `stop`:

Immediately stops the program.

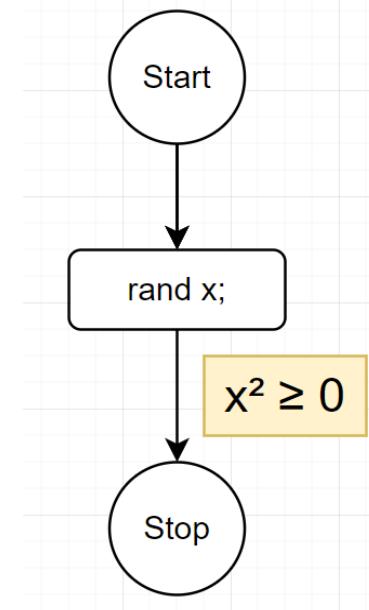
Define the weakest precondition operator $\text{WP}[\dots](B)$ for each of these statements. (In terms of B)

Beispiele zum Testen:



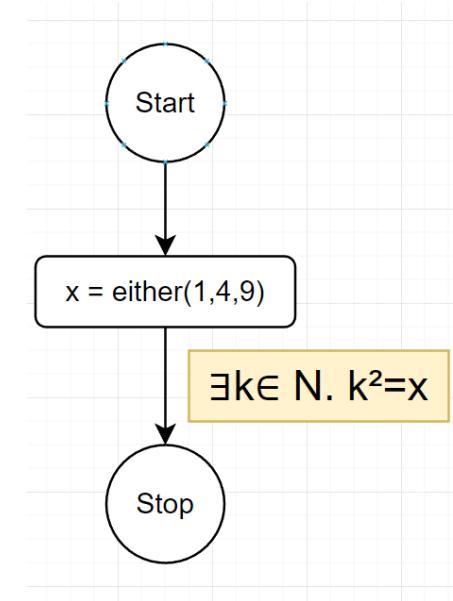
T01: MiniJava 2.0

WP[rand x;](B) =



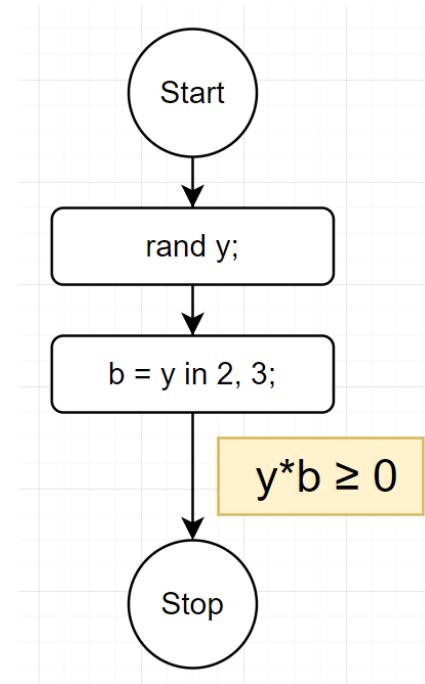
T01: MiniJava 2.0

$\text{WP}[x = \text{either } e_0, e_1 \dots e_k](B) =$



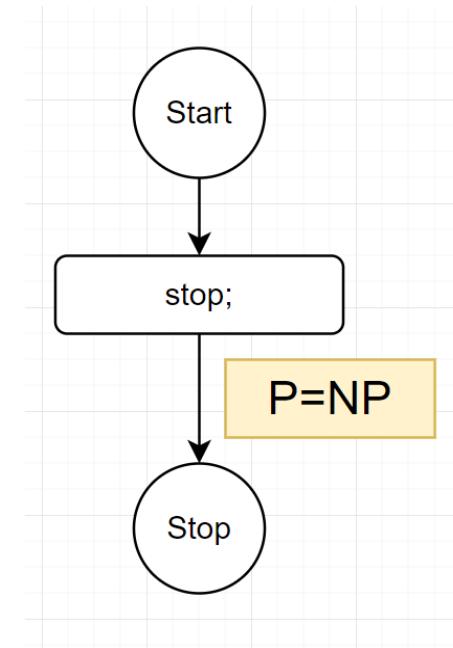
T01: MiniJava 2.0

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



T01: MiniJava 2.0

WP[stop](B) =



T02: Loop Invariants

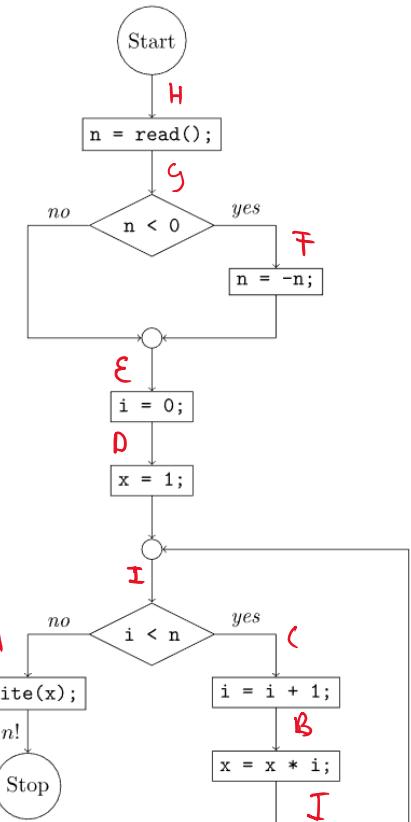
1. Discuss the problem that arises when computing weakest preconditions to prove Z .
2. How can you use weakest preconditions to prove Z anyway?
3. Try proving Z using the loop invariants $x \geq 0$ and $i = 0 \wedge x = 1 \wedge n = 0$ at

the end of the loop body and in particular discuss these questions:

- o a) How has a useful loop invariant be related to Z ?
- o b) What happens if the loop invariant is chosen too strong?
- o c) What happens if the loop invariant is chosen too weak?
- o 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.

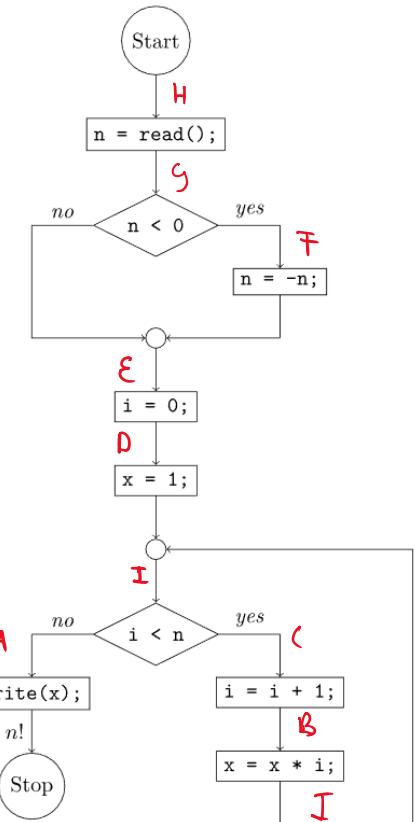
A program computes the factorial of its input:



T02: Loop Invariants 1

3. Try proving Z using the 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:

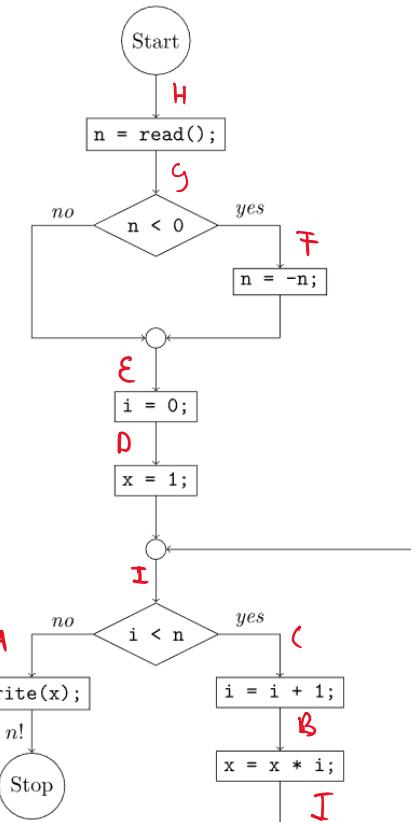
A program computes the factorial of its input:



T02: Loop Invariants 2

3. Try proving Z using the 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:

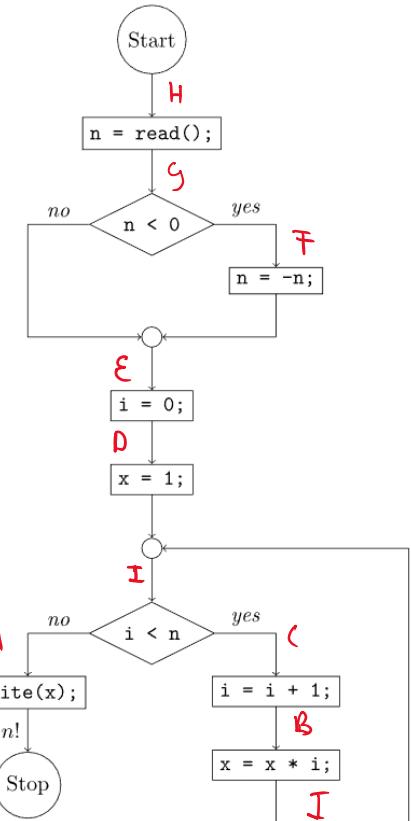
A program computes the factorial of its input:



T02: Loop Invariants 3

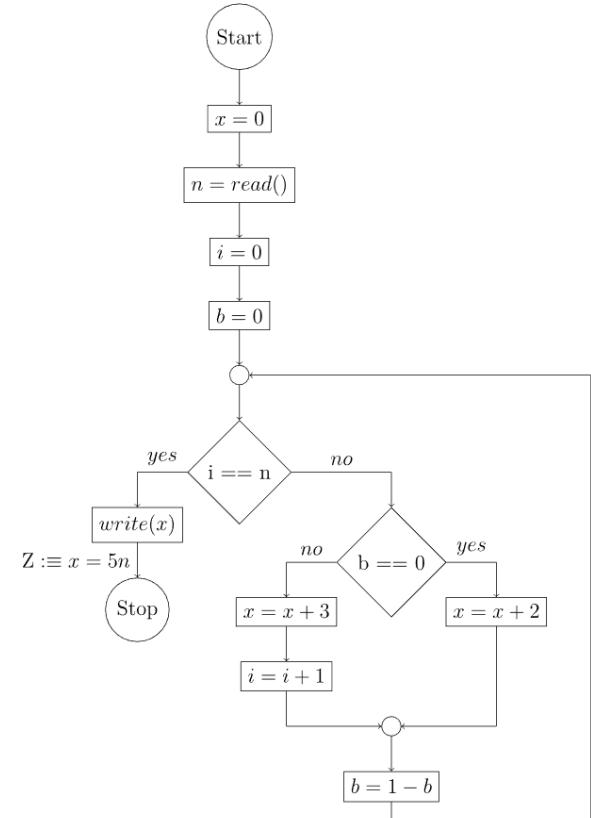
4. Retry proving Z using the loop invariant $x = i!$ (again at the end of the loop body)
and improve this invariant until the proof succeeds.

A program computes the factorial of its input:



T03: Two b, or Not Two b

Prove Z using weakest preconditions.



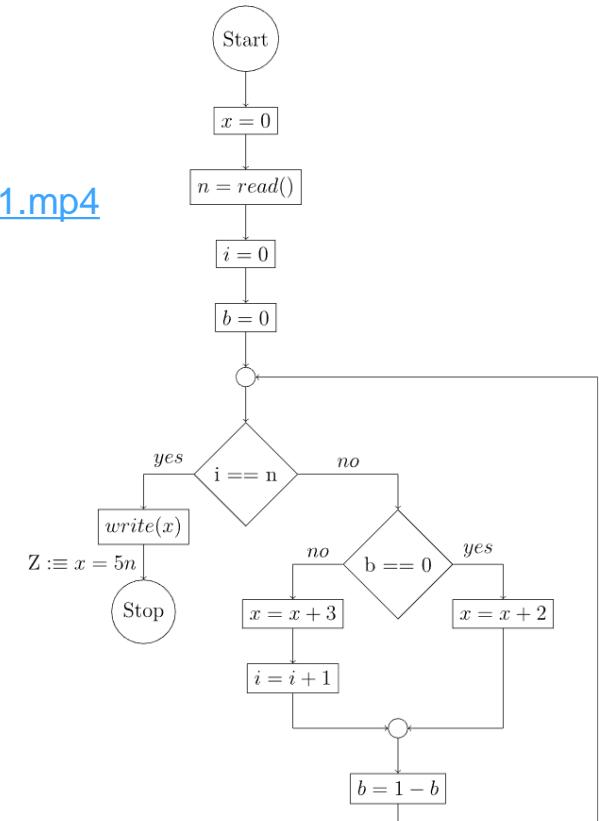
T03: Two b, or Not Two b

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



Tipps für Loop Invarianten

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

Tipp 1

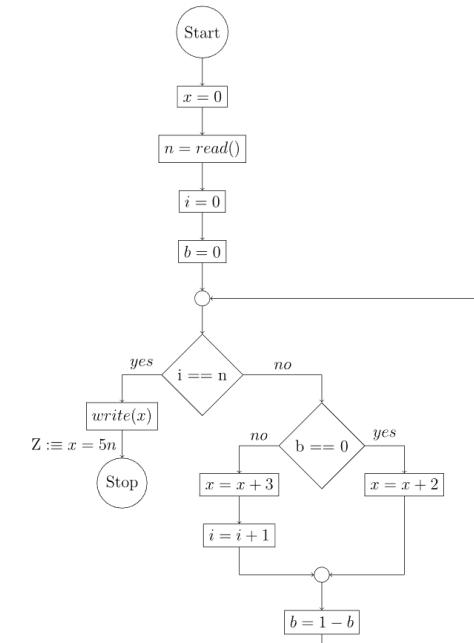
Wir benötigen eine Aussage über den Wert der Variablen, über die wir etwas beweisen wollen (x) in der Schleifeninvariante. Die Aussage muss dabei mindestens so präzise ($\neq, \geq, \leq, =$) sein, wie die Aussage, die wir beweisen wollen.

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.



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

FPV Tutorübung

Woche 4

Loop Invariants and Termination proofs

Manuel Lerchner

15.05.2023

Quiz

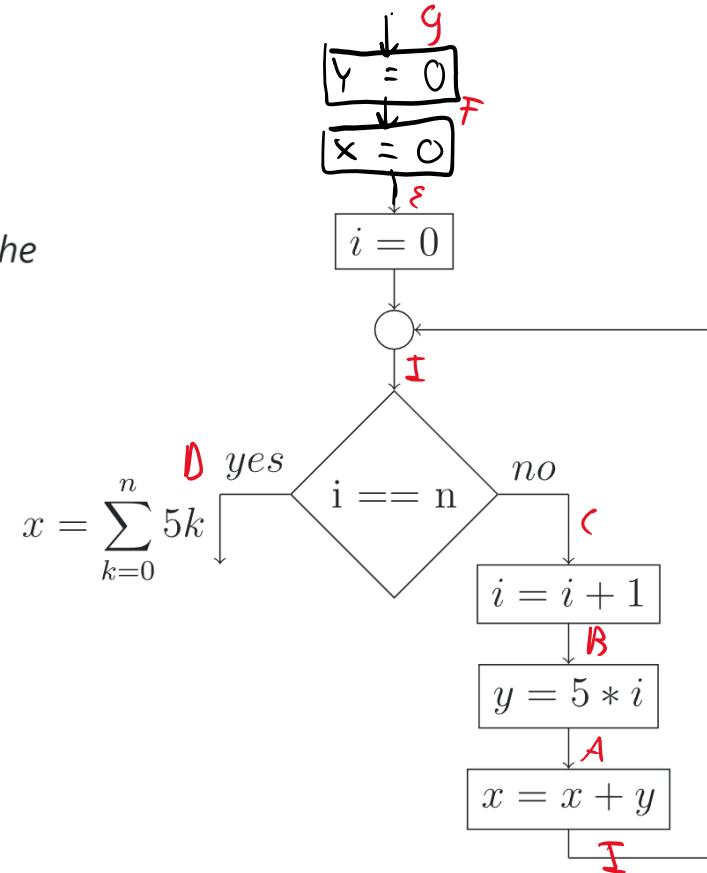
The screenshot shows the Artemis 6.1.7 interface. At the top, there's a dark header bar with the Artemis logo and the text "Artemis 6.1.7". Below it, a light blue navigation bar shows the path "Courses > Funktionale Programmierung und Verifikation (Sommersemester 2)". The main content area has a dark background. It displays a green checkmark icon followed by the text "Week 04 Quiz" and a green button labeled "Quiz". Below this, the text "Points: 23" is shown. At the bottom left, there's a blue button with a play icon and the text "Open quiz".

Passwort:

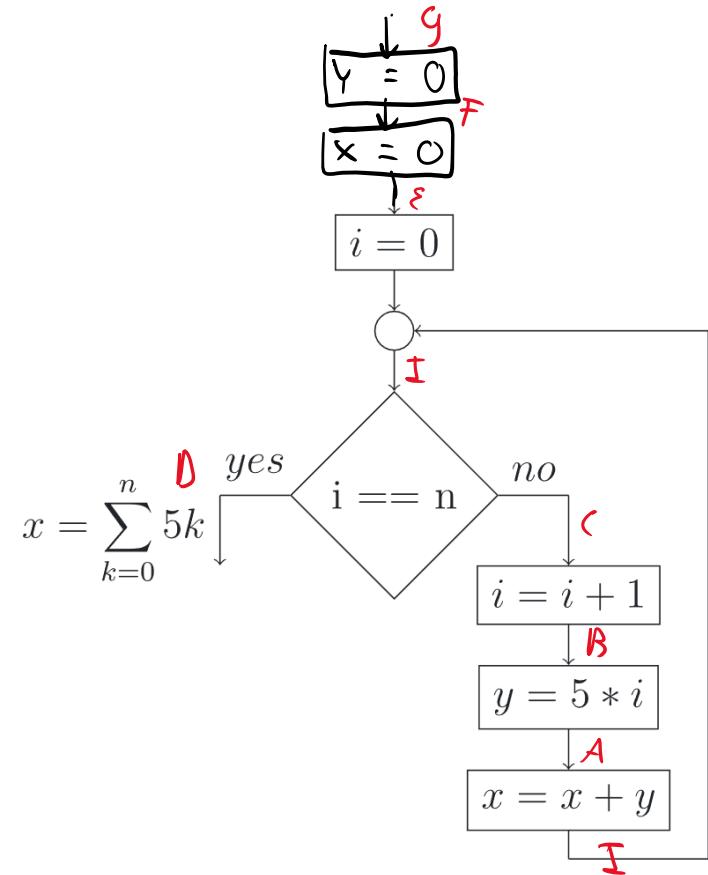
T01: Loop Invariants

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.: $\sum_{k=0}^{-1} (\dots) = 0$.



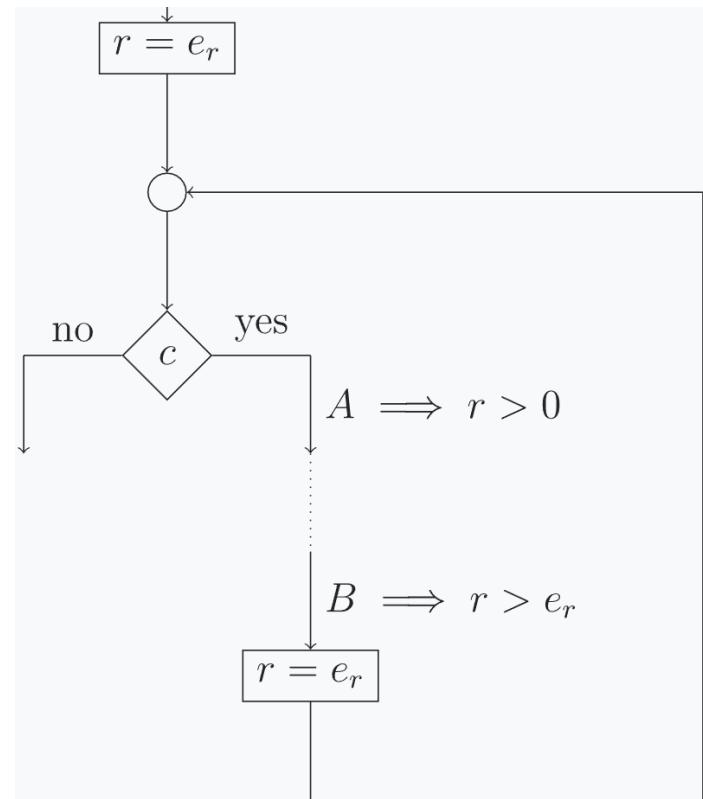
T01: Loop Invariants



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?

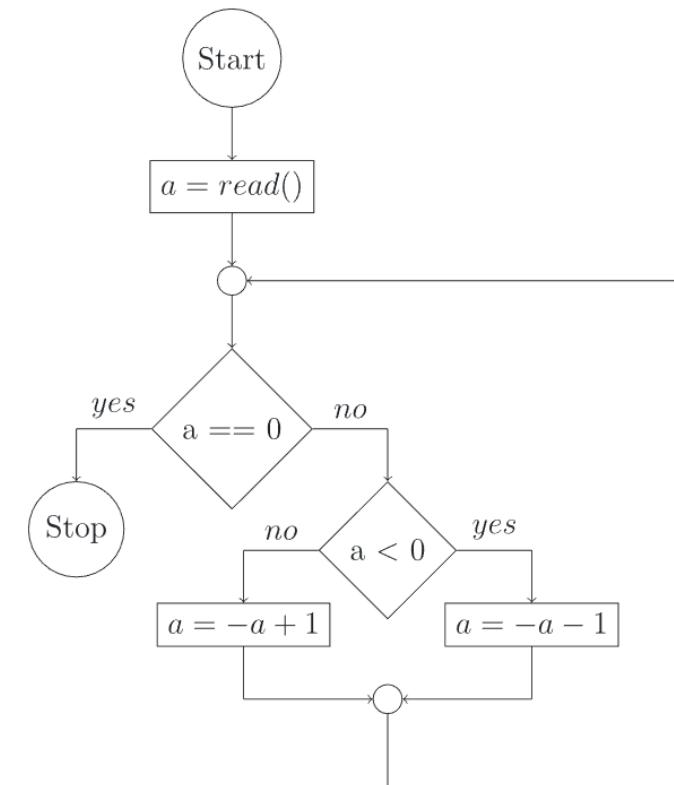


T03: A Wavy Approach

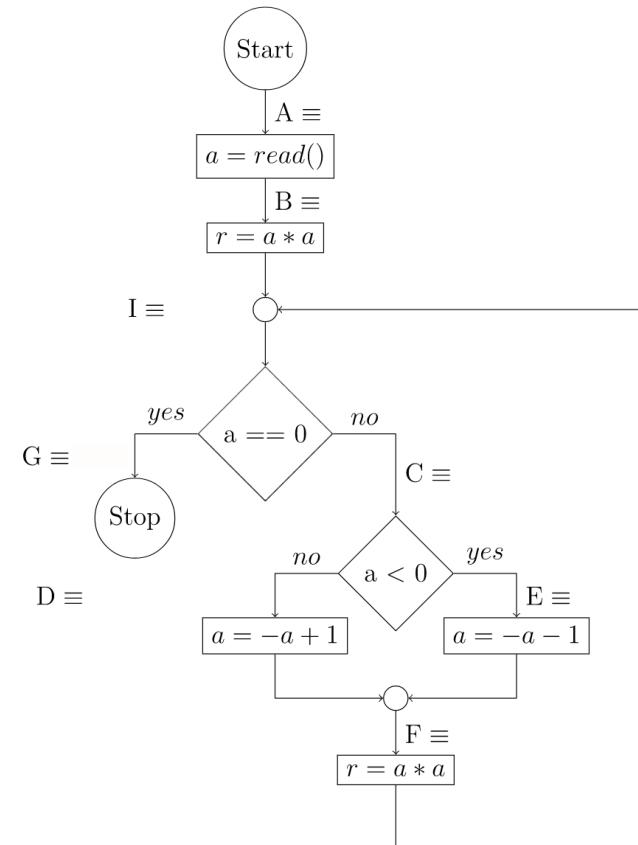
Prove termination of the following program:

Todos:

1. Schleife verstehen
2. Variable r definieren / finden
 - $r \geq 0$ 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

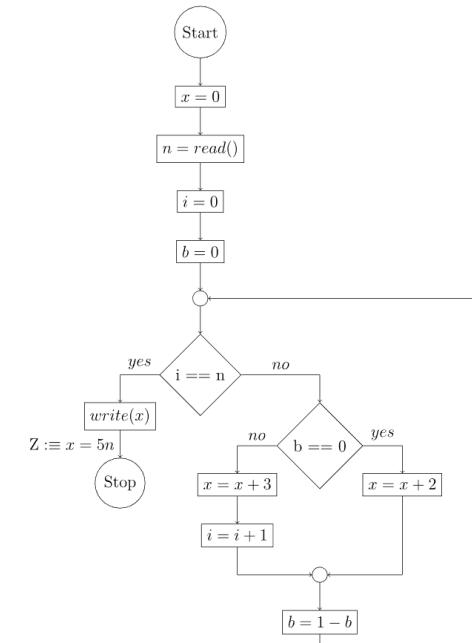
Wir benötigen eine Aussage über den Wert der Variablen, über die wir etwas beweisen wollen (x) in der Schleifeninvariante. Die Aussage muss dabei mindestens so präzise ($\neq, \geq, \leq, =$) sein, wie die Aussage, die wir beweisen wollen.

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.



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

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.

FPV Tutorübung

Woche 6

Ocaml: List-Module, Binary Search Trees

Manuel Lerchner

31.05.2023

T01: Explicit Type Annotation

In OCaml, types are inferred automatically, so there is no need to write them down explicitly. However, types can be annotated by the programmer. Discuss:

1. In the following expression, annotate the types of all subexpressions:

```
let f = fun x y -> x, [y]
```

2. When can explicitly annotated types be helpful?

T02: The List Module

Check the documentation of the OCaml [List module](#) [here](#) and find out what the following functions do. Then implement them yourself. Make sure your implementations have the same type. In cases where the standard functions throw exceptions, you may just [failwith "invalid"](#).

1. **✗ hd** [0 of 1 tests passing](#)

Implement the function `hd`

2. **✗ tl** [0 of 1 tests passing](#)

Implement the function `tl`

3. **✗ length** [0 of 1 tests passing](#)

Implement the function `length`

4. **✗ append** [0 of 1 tests passing](#)

Implement the function `append`

5. **✗ rev** [0 of 1 tests passing](#)

Implement the function `rev`

6. **✗ nth** [0 of 1 tests passing](#)

Implement the function `nth`



T03: Binary Search Tree 1

In this assignment, a collection to organize integers shall be implemented via binary search trees.

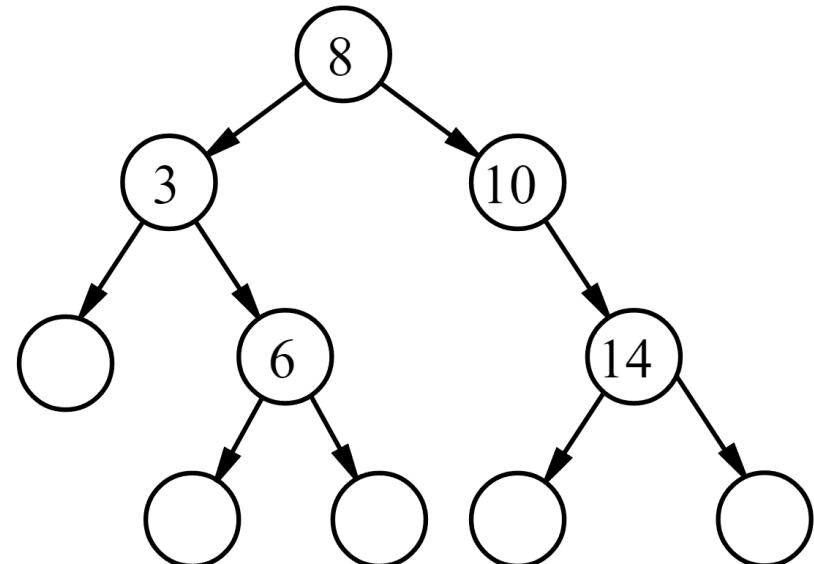
1. Define a suitable data type `tree` for binary trees that store integers. Each node in the binary tree should either be

- an inner node which stores a value of type `int` and has a left and a right child of type `tree`, or
- a leaf node and contain no value.

Since you are free to define your `tree` type however you wish (the type `tree` is said to be *abstract*), we need to define functions for creating trees.

2. Define functions `node` and `leaf`, which should create inner nodes and leaves, respectively:

- `node v l r` should create an inner node with the value `v`, left child `l` and right child `r`,
- `leaf ()` should create an leaf node of your `tree` type.

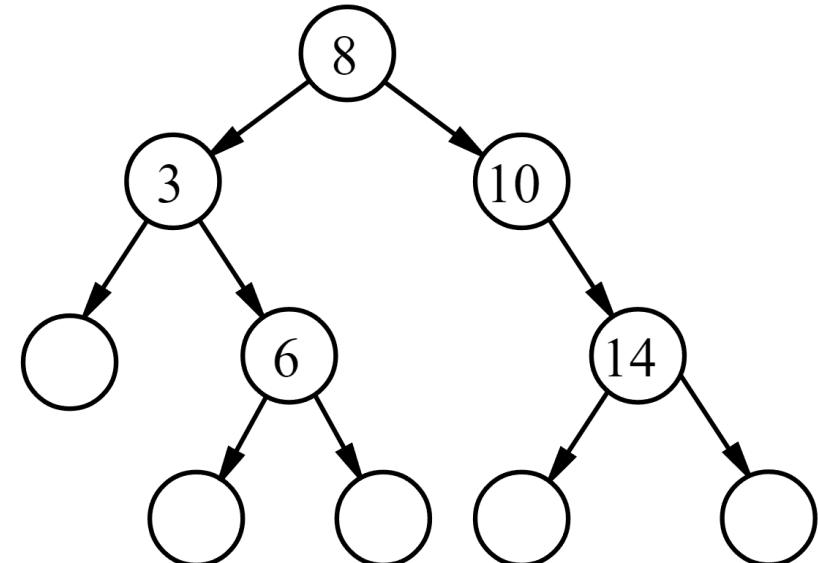


T03: Binary Search Tree 2

Similarly, we need a function that allows us to inspect the structure of your tree, specifically to access the children of a node. The OCaml [Option type \(API documentation for Option\)](#) allows us to cleanly distinguish between the presence or absence of a value. Here, we will use it to distinguish between inner nodes, which have children, and leaf nodes, which do not. Using the [Option type](#) and returning [None](#) instead of raising an exception is (often) good functional programming style!

3. Define the function [inspect](#), which allows us to access the children of a node:

- `inspect n`, where `n` is an inner node of your `tree` type with the value `v` and children `l` and `r`, should return `Some (v, l, r)`. Here, `Some` is used to indicate the *presence* of a value and children.
- `inspect l`, where `l` is a leaf of your `tree` type (with no children), should return `None`. Here, `None` is used to indicate the *absence* of a value and children.



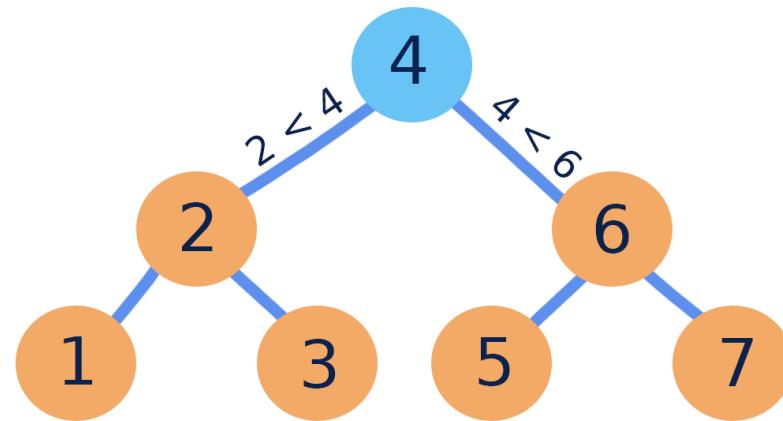
T03: Binary Search Tree 3

4. Define a binary tree `t1` which contains the values

`8, 12, 42, 1, 6, 9, 8`. To construct the tree, start
with an empty tree, then insert the given values
in order.

T03: Binary Search Tree 4

5. Implement a function `to_list : tree -> int list` that returns an ordered list of all values in the tree.

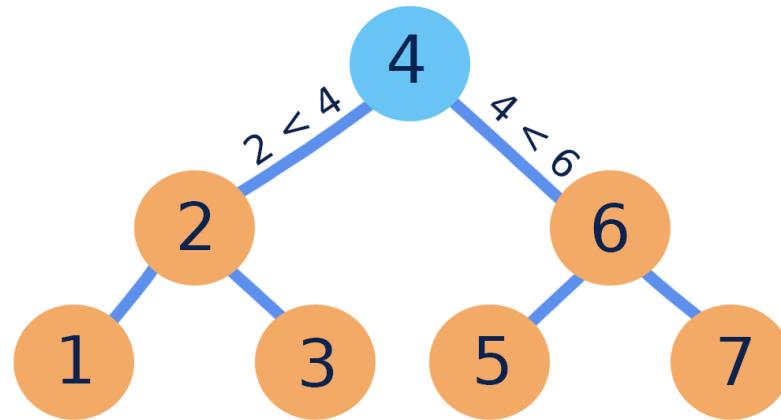


In Order Traversal: [1 2 3 4 5 6 7]

T03: Binary Search Tree 5

6. Implement a function `insert : int -> tree -> tree` which inserts a value into the tree. If the value exists already, the tree is not modified.

8

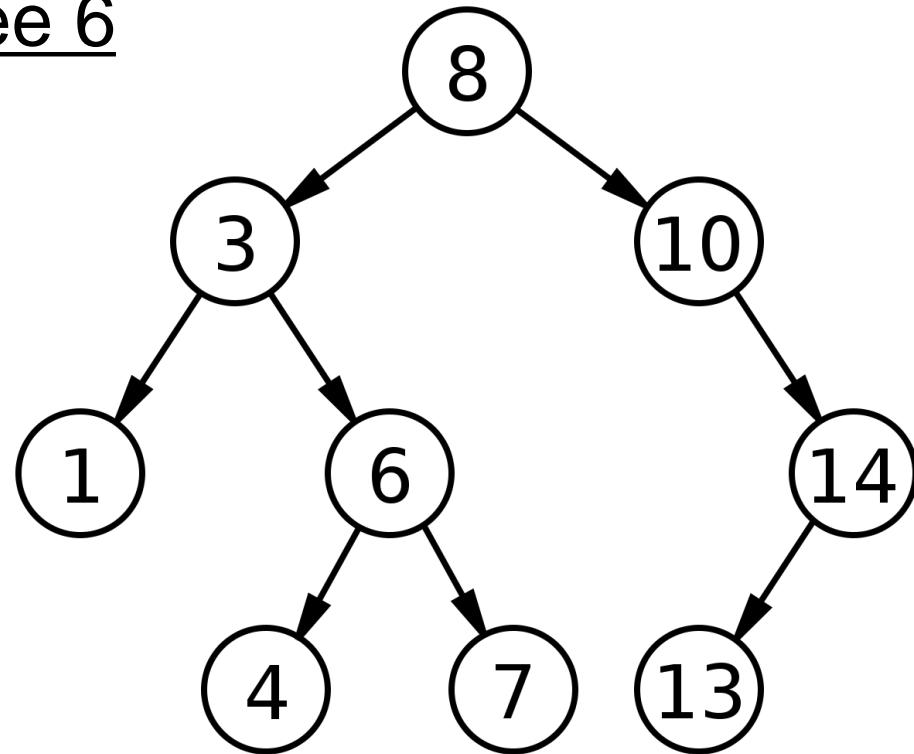


T03: Binary Search Tree 6

7. Implement a function `remove : int -> tree -> tree` to remove a value (if it exists) from the tree.

Upon removing a value from an inner node:

- if either child of the inner node is empty, replace the entire inner node with the other child, otherwise
- if neither child of the inner node is empty, the value should be replaced with the largest value from the *left* subtree.



FPV Tutorübung

Woche 7

OCaml: List-Module 2, Mappings, Operator Functions

Manuel Lerchner

08.06.2023

T01: List Module Part 2

- Use functions from the List-Module!

Implement the following functions without defining any recursive functions yourself:

1. **☒ float_list** 0 von 1 Tests bestanden

Implement the function `float_list : int list -> float list` that converts all ints in the list to floats.

2. **☒ to_string** 0 von 1 Tests bestanden

Implement the function `to_string : int list -> string` that builds a string representation of the given list. E.g.: "[0;42;123;420;1;]"

3. **☒ part_even** 0 von 1 Tests bestanden

Implement the function `part_even : int list -> int list` that partitions all even values to the front of the list.

4. **☒ squaresum** 0 von 1 Tests bestanden

Implement the function `squaresum : int list -> int` that computes $\sum_{i=1}^n x_i^2$ for a list $[x_1, \dots, x_n]$.

T01: List Module Part 2

- Selected Functions from the List-Module
 - `List.map ('a -> 'b) -> 'a list -> 'b list`
 - `List.fold_left ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`
 - `List.find_opt ('a -> bool) -> 'a list -> 'a option`
 - `List.filter ('a -> bool) -> 'a list -> 'a list`

T01: List Module Part 2

- `List.map ('a -> 'b) -> 'a list -> 'b list`



T01: List Module Part 2

- `List.fold_left ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`

1	2	-3	4	5	8
---	---	----	---	---	---

T01: List Module Part 2

- `List.find_opt ('a -> bool) -> 'a list -> 'a option`

1	2	-3	4	5	8
---	---	----	---	---	---

T01: List Module Part 2

- `List.filter` ('a -> bool) -> 'a list -> 'a list

1	2	-3	4	5	8
---	---	----	---	---	---

T02: Mappings

Idea: Create a Dictionary-Datastructure

```
age_dictionary = {
    "John": 25,
    "Mary": 20,
    "Tom": 30
}
```

1. Implement these functions to work with mappings based on associative lists:

1. ~~(X)~~ is_empty 0 of 1 tests passing

is_empty : ('k * 'v) list -> bool

2. ~~(X)~~ get 0 of 1 tests passing

get : 'k -> ('k * 'v) list -> 'v option

If the key is mapped to multiple values, return the first such value

3. ~~(X)~~ put 0 of 1 tests passing

put : 'k -> 'v -> ('k * 'v) list -> ('k * 'v) list

If the key is already mapped to one or more values, remove those pairs first

4. ~~(X)~~ contains_key 0 of 1 tests passing

contains_key : 'k -> ('k * 'v) list -> bool

5. ~~(X)~~ remove 0 of 1 tests passing

remove : 'k -> ('k * 'v) list -> ('k * 'v) list

If the key is mapped to multiple values, remove all such values

6. ~~(X)~~ keys 0 of 1 tests passing

keys : ('k * 'v) list -> 'k list

7. ~~(X)~~ values 0 of 1 tests passing

values : ('k * 'v) list -> 'v list

T02: Mappings

- How to store dictionaries?
 - Association Lists
 - Functional mapping

```
assoc_list = [  
    ("John", 25),  
    ("Mary", 20),  
    ("Tom", 30)  
]
```

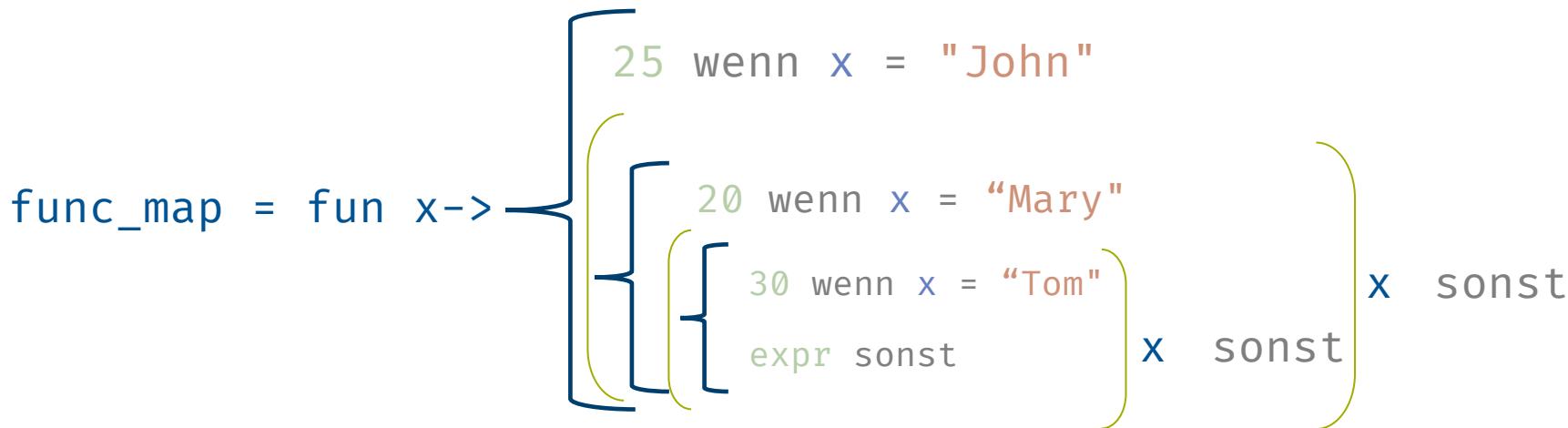
```
func_map = fun x->
```



```
25 wenn x = "John"  
20 wenn x = "Mary"  
30 wenn x = "Tom"  
expr sonst
```

T02: Functional Mappings

- Every layer saves **exactly** one datapoint
 - If the parameter matches the datapoint -> return its value
 - Else delegate to sub-function



T03: Operator Functions

In OCaml, infix notation of operators is just syntactic sugar for a call to the corresponding function. For example, the binary addition `+` merely calls the function `(+) : int -> int -> int`.

1. Discuss why this is a very useful feature.

Note: This is a tutorial exercise, you do not need to submit anything for this exercise.

FPV Tutorübung

Woche 8

OCaml: Tail Recursion, Lazy Lists, Partial Application

Manuel Lerchner

14.06.2023

T01: Tail Recursion 1

a)

```
let rec f a = match a with
| [] -> a
| x::xs -> (x + 1)::f xs
```

b)

```
let rec g a b =
  if a = b then 0
  else if a < b then g (a + 1) b
  else g (a - 1) b
```

c)

```
let rec h a b c =
  if b then h a (not b) (c * 2)
  else if c > 1000 then a
  else h (a + 2) (not b) c * 2
```

d)

```
let rec i a = function
| [] -> a
| x::xs -> i (i (x,x) [x]) xs
```

1. Decide which of the following functions are implemented tail recursively:

T01: Tail Recursion 2

2. Write tail recursive versions of the following functions (without changing their types). In addition to the definition from the lecture, all functions must use constant stack space ($\mathcal{O}(1)$) in the size of its input). In particular, all the helper functions used need to be tail-recursive and use constant stack space too! If you use a library function, check that the documentation (e.g. for [List](#)) marks it as tail-recursive, or when in doubt implement a tail-recursive version yourself!

- Tipp: Use accumulator variables and helper functions

a)

```
let rec fac n =
  if n = 0 then 1
  else n * fac (n - 1)
```

b)

```
let rec remove a = function
| [] -> []
| x::xs -> if x = a then remove a xs else x::remove a xs
```

c)

```
let rec partition p l = match l with
| [] -> [],[]
| x::xs ->
  let a,b = partition p xs in
  if p x then x::a,b else a,x::b
```

T02: Lazy List Idea

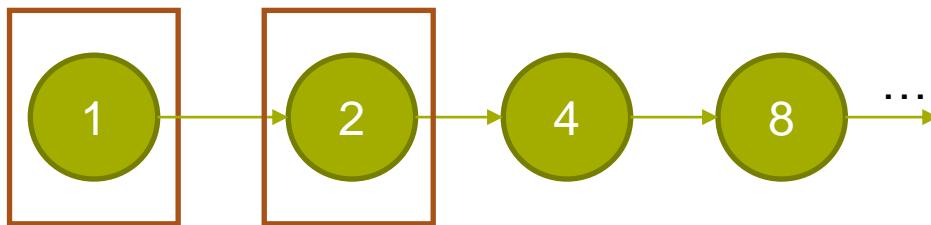
```
2 # Classical: Calculate all values at once and return them as a list
3 def powers_of_two(n):
4     return [2 ** i for i in range(n)]
5
6
7 my_powers = powers_of_two(10)
8
9 print(my_powers)
10 # [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
11
```

```
13 # Generator: Calculate the values on demand, one by one
14 def powers_of_two_generator(i):
15     while True:
16         yield 2 ** i
17         i += 1
18
19
20 generator = powers_of_two_generator(0)
21
22
23 for i in range(10):
24     print(next(generator))
25
26 # 1
27 # 2
28 # 4
29 # 8
30 # 16
31 # 32
32 # 64
33 # 128
34 # 256
35 # 512
36 # ... and so on
```

T02: Lazy List

Infinite data structures (e.g. lists) can be realized using the concept of **lazy evaluation**. Instead of constructing the entire data structure immediately, we only construct a small part and keep us a means to construct more on demand.

1, fun ()->n n



```
type 'a llist = Cons of 'a * (unit -> 'a llist)
```

```
int -> int llist
let rec powers_of_2 i =
| Cons (pow 2 i, fun () -> powers_of_2 (i + 1))
```

T02: Lazy List

1. **lnat** 0 von 1 Tests bestanden

Implement the function `lnat : int -> int llist`
that constructs the list of all natural numbers
starting at the given argument.

2. **lfib** 0 von 1 Tests bestanden

Implement the function `lfib : unit -> int llist`
that constructs a list containing the Fibonacci
sequence.

3. **ltake** 0 von 1 Tests bestanden

Implement the function `ltake : int -> 'a llist
-> 'a list` that returns the first n elements of the
list.

4. **lfilter** 0 von 1 Tests bestanden

Implement the function `lfilter : ('a -> bool) -> 'a llist -> 'a llist` to filter those elements
from the list that do not satisfy the given predicate.

```
type 'a llist = Cons of 'a * (unit -> 'a llist)
```

```
int -> int llist
let rec powers_of_2 i =
| Cons (pow 2 i, fun () -> powers_of_2 (i + 1))
```

T02: Lazy List

```
1  type llist<T> = [T, () => llist<T>];
2
3  ✓ function fibonacci_generator(): llist<number> {
4    ✓ function fib_step(a: number, b: number): llist<number> {
5      return [a, () => fib_step(b, a + b)];
6    }
7
8    return fib_step(0, 1);
9  }
10
11 let fibonacci_numbers = fibonacci_generator();
12
13 ✓ for (let i = 0; i < 10; i++) {
14   let [value, next_generator] = fibonacci_numbers;
15
16   console.log(value);
17
18   fibonacci_numbers = next_generator();
19 }
20
21 ✓ // [0 1]
22 // 0 [1 1]
23 // 0 1 [1 2]
24 // 0 1 1 [2 3]
25 // 0 1 1 2 [... ]
26
```

```
type 'a llist = Cons of 'a * (unit -> 'a llist)
```

T03: Partial Application

Types of (apparently) n -ary functions are denoted as `arg_1 -> ... -> arg_n -> ret` in OCaml.

1. Discuss, why this notation is indeed meaningful.
2. Give the types of these expressions and discuss to what they evaluate:

```
let a (* : todo *) = (fun a b -> (+) b)

let b (* : todo *) = (fun a b -> List.fold_left b 1 (List.map ( * ) a))

let c (* : todo *) = (fun a b c -> c (a + b)) 3

let d (* : todo *) = (fun a b c -> b (c a) :: [a]) "x"

let e (* : todo *) = (let x = List.map in x (<))
```

T03: Partial Application

Types of (apparently) n -ary functions are denoted as `arg_1 -> ... -> arg_n -> ret` in OCaml.

1. Discuss, why this notation is indeed meaningful.
2. Give the types of these expressions and discuss to what they evaluate:

```
let a (* : todo *) = (fun a b -> (+) b)

let b (* : todo *) = (fun a b -> List.fold_left b 1 (List.map ( * ) a))

let c (* : todo *) = (fun a b c -> c (a + b)) 3

let d (* : todo *) = (fun a b c -> b (c a) :: [a]) "x"

let e (* : todo *) = (let x = List.map in x (<))
```

T03: Partial Application

```
let a (* : todo *) = (fun a b -> (+) b)
```

T03: Partial Application

```
let b (* : todo *) = (fun a b -> List.fold_left b 1 (List.map (* ) a))
```

T03: Partial Application

```
let c (* : todo *) = (fun a b c -> c (a + b)) 3
```

FPV Tutorübung

Woche 9

OCaml: Side Effects, Exceptions and Files

Manuel Lerchner

19.06.2023

T01: Students In Students Out

```

10
11 (* define demo database *)
12 let db : database =
13 [
14   {
15     first_name = "John";
16     last_name = "Doe";
17     id = 0;
18     semester = 1;
19     grades = [ (0, 4.0); (1, 3.0); (2, 3.7) ];
20   };
21   {
22     first_name = "Jane";
23     last_name = "Doe";
24     id = 1;
25     semester = 2;
26     grades = [ (0, 3.0); (1, 3.5); (2, 3.7) ];
27   };
28   {
29     first_name = "Manuel";
30     last_name = "Lerchner";
31     id = 1;
32     semester = 2;
33     grades = []
34   };
]

```



John;Doe;0;1;3
 0;4.
 1;3.
 2;3.7
 Jane;Doe;1;2;3
 0;3.
 1;3.5
 2;3.7
 Manuel;Lerchner;1;2;0

student_database.txt

```

type student = {
  first_name : string;
  last_name : string;
  id : int;
  semester : int;
  grades : (int * float) list
}

type database = student list

```

Now, we define a file format to store students that, for each student, contains a line

first_name;last_name;id;semester;grade_count

followed by a number of lines

course;grade

with grades.

T01: Students In Students Out

```
John;Doe;0;1;3  
0;4.  
1;3.  
2;3.7  
Jane;Doe;1;2;3  
0;3.  
1;3.5  
2;3.7  
Manuel;Lerchner;1;2;0
```

student_database.txt



```
12 let db : database =  
13 [  
14 {  
15   first_name = "John";  
16   last_name = "Doe";  
17   id = 0;  
18   semester = 1;  
19   grades = [ (0, 4.0); (1, 3.0); (2, 3.7) ];  
20 };  
21 {  
22   first_name = "Jane";  
23   last_name = "Doe";  
24   id = 1;  
25   semester = 2;  
26   grades = [ (0, 3.0); (1, 3.5); (2, 3.7) ];  
27 };  
28 {  
29   first_name = "Manuel";  
30   last_name = "Lerchner";  
31   id = 1;  
32   semester = 2;  
33   grades = []  
34 };
```

T01: Students In Students Out

- File I/O
 - `open_in`
 - `open_out`
 - `close_in`
 - `close_out`
 - `input_line`
 - `output_string`
 - Exceptions
 - `try expr with exn -> expr`
 - Other helpful functions
 - `String.split_on_char`
 - `String.concat`
 - `List.iter`
1. **✗ `store_db` 0 von 1 Tests bestanden**
Implement a function `store_db : string -> database -> unit` to store the students in the given file.
 2. **✗ `load_db` 0 von 1 Tests bestanden**
Implement a function `load_db : string -> database` to read the students back out from the given file.
Throw an exception `Corrupt_database_file` if something is wrong with the file.
 3. **✗ Round Trip 0 von 1 Tests bestanden**
It should be possible to round trip a database through a file, even if you don't implement the exact format described above.
- Now, we define a file format to store students that, for each student, contains a line
first_name;last_name;id;semester;grade_count
followed by a number of lines
course;grade
with grades.

T02: (Delayed) Evaluation, Side-effects, Pure Functions

Discuss this difference between the following two expressions:

```
let x = print_endline "foo" in x, x
```

```
let x () = print_endline "foo" in x (), x ()
```

1. What are side-effects? Give some examples.
2. What are pure functions? What are their benefits?
3. Why does delaying evaluation only make sense in case of side-effects or in presence of non-terminating expressions?
4. Why do we want to use `()` instead of some unused variable or the discard `_`?

FPV Tutorübung

Woche 10
OCaml: Modules

Manuel Lerchner

28.06.2023

T01: A Multitude of Map Modules

1. **StringPrintable** [2 von 2 Tests bestanden](#)

Implement a module `StringPrintable` of signature `Printable`.

2. **OrderedPrintable** [1 von 1 Tests bestanden](#)

Define a signature `OrderedPrintable` that extends the `Printable` signature by a `compare` function with the usual type.

3. **IntOrderedPrintable** [3 von 3 Tests bestanden](#)

Additionally to the `StringPrintable` now implement an `IntOrderedPrintable` module.

4. **BinaryTreeMap** [1 von 1 Tests bestanden](#)

Implement a functor `BinaryTreeMap` that realizes the `Map` signature and uses a binary tree to store *key-value*-pairs. The functor takes an ordered key printable and a value printable as arguments.

5. **IntIntMap** [5 von 5 Tests bestanden](#)

Use the `BinaryTreeMap` functor to define the module `IntIntMap` for *int-to-int* maps.

6. **IntStringMap** [5 von 5 Tests bestanden](#)

Use the `BinaryTreeMap` functor to define the module `IntStringMap` for *int-to-string* maps.

We model domains of printable values using modules with signature

```
module type Printable = sig
  type t
  val to_string : t -> string
end
```

```
module type Map = sig
  type key
  type value
  type t
  val empty : t
  val set : key -> value -> t -> t
  val get : key -> t -> value
  val get_opt : key -> t -> value option
  val to_list : t -> (key * value) list
  val to_string : t -> string
end
```

Where the functions from `Map` have the following semantics:

- `set key value m` updates the mapping, such that `key` is now mapped to `value`
- `get key m` retrieves the value for the given key and throws a `Not_found` exception if no such key exists in the map.
- `get_opt key m` retrieves the value for the given key or `None` if the key does not exist.
- `to_string m` produces a string representation for the mapping, e.g.: "{ 1 -> \"x\", 5 -> \"y\" }"
- `to_list m` returns a list containing all key-value tuples in the given mapping.

OCaml vs Java: Module Type

- Module types sind ähnlich zu Interfaces in Java
 - Kapselung von zusammengehörigen Daten / Funktionen

```
module type Animal = sig
  unit -> string
  val make_sound : unit → string
end
```



```
public interface Animal {
  public String makeSound();
}
```

OCaml vs Java: Module

- Modules sind wie Klassen in Java, sie können Module types *implementieren*
- Typisierung entspricht *implements*

```
module Cat : Animal = struct
  unit -> string
  | let make_sound () = "Miau"
end

module Dog : Animal = struct
  unit -> string
  | let make_sound () = "Woof"
end
```



```
class Cat implements Animal {
  @Override
  public String makeSound() {
    return "Miau";
  }
}

class Dog implements Animal {
  @Override
  public String makeSound() {
    return "Woof";
}
```

OCaml vs Java: Module Type with generic type

- Modules types mit eigenem Datentyp entsprechen generischen Interfaces
 - Mit zusätzlich get() Funktion

```
module type ListElement = sig
| type t
| unit -> t
| val get : unit → t
|
end
```



```
interface ListElement<T> {
|   T get();
}
```

OCaml vs Java: Include Keyword

- The *include* keyword is similar to the *extend* keyword in Java

```
module type Animal = sig
  unit -> string
  | val make_sound : unit → string
end
```

```
module type Mammal = sig
  | include Animal
  unit -> string
  | val give_birth : unit → string
end
```



```
public interface Animal {
  | public String makeSound();
}

interface Mammal extends Animal {
  | public String giveBirth();
}
```

OCaml vs Java: Functors

- Functors are Similar to Generic classes, where the generic type has a *constraint*

```
module type Serializable = sig
| type t
| t -> string
| val serialize : t → string
end
```

```
module ListSerializer (T : Serializable ) = struct
| T.t list -> string
| let serialize_list (l:T.t list) =
| | List.fold_left (fun acc x → acc ^ T.serialize x ^ "\n") "" l
end
```



```
interface Serializable<T> {
| String serialize(T t);
| T get();
}

class ListSerializer<T extends Serializable> {
| public String serialize(T[] list) {
| | String result = "";
| | for (T t : list) {
| | | result += t.serialize(t.get()) + "\n";
| }
| | return result;
| }
```

OCaml vs Java: Functor Example

```

module MyInteger : Serializable with type t = int = struct
| type t = int
| t-> string
| let serialize x = string_of_int x
end

module ListSerializer (T : Serializable ) = struct
| T.t list -> string
| let serialize_list (l:T.t list) =
| | List.fold_left (fun acc x -> acc ^ T.serialize x ^ "\n") "" l
end

module IntListSerializer = ListSerializer(MyInteger)

unit
let res =  print_string (IntListSerializer.serialize_list [1;2;3])
(*
1
2
3
*)

```

Nicht ganz, ListSerializer
hat falsche Signatur



```

class MyInteger implements Serializable<Integer> {
    Integer i;
}

public MyInteger(Integer i) {
    this.i = i;
}

public String serialize(Integer i) {
    return i.toString();
}

public Integer get() {
    return i;
}

class ListSerializer<T extends Serializable> { 
    public String serialize(T[] list) {
        String result = "";
        for (T t : list) {
            result += t.serialize(t.get()) + "\n";
        }
        return result;
    }
}

class IntListSerializer extends ListSerializer<MyInteger> {
}

IntListSerializer serializer = new IntListSerializer();

System.out.println(serializer.serialize(
    new MyInteger[] [
        new MyInteger(1),
        new MyInteger(2),
        new MyInteger(3)
    ]);
// 1
// 2
// 3

```

Sharing Constraints

```
module type Inc = sig
| type t
| t -> t
| val inc : t → t
end
```

```
module HiddenIntInc : Inc = struct
| type t = int
| t -> t
| let inc x = x + 1
end
```



```
interface Inc<T> {
| T inc(T t);
}
```

```
class HiddenIntInc implements Inc {
| public Integer inc(Integer t) {
| | return t + 1;
| }
}
```

```
utop # HiddenIntInc.inc;;
- : HiddenIntInc.t -> HiddenIntInc.t = <fun>
-( 17:23:37 )-< command 12 >
utop # HiddenIntInc.inc 4;;
Error: This expression has type int but an expression was expected of type
      HiddenIntInc.t
-( 17:23:46 )-< command 13 >
```

Sharing Constraints

```
module type Inc = sig
| type t
| t -> t
| val inc : t → t
end
|
module ExposedIntInc : Inc with type t = int = struct
| type t = int
| t -> t
| let inc x = x + 1
end
```

```
interface Inc<T> {
| T inc(T t);
}
class ExposedIntInc implements Inc<Integer> {
| public Integer inc(Integer t) {
| | return t + 1;
| }
}
```

```
utop # ExposedIntInc.inc;;
- : int -> int = <fun>
-( 17:25:08 )-< command 15 >—
utop # ExposedIntInc.inc 4;;
- : int = 5
-( 17:25:12 )-< command 16 >—
utop #
```



Summary

- Ähnlichkeiten zwischen OCaml und Java
 - module type == Interface
 - module == Klasse
 - typisiertes Module == Klasse die Interface implementiert
 - Include keyword in Module type == Interface extended anderes Interface
 - Functor == generische Klasse mit Constraint auf Generic
 - Wird „ausgeführt“ indem der generische Typ „festgelegt“ wird