Functional Programming and Verification

Prof. Dr. H. Seidl, N. Hartmann, R. Vogler — WS 2018/19

Exercise Sheet 2 — Deadline: 04.11.2018

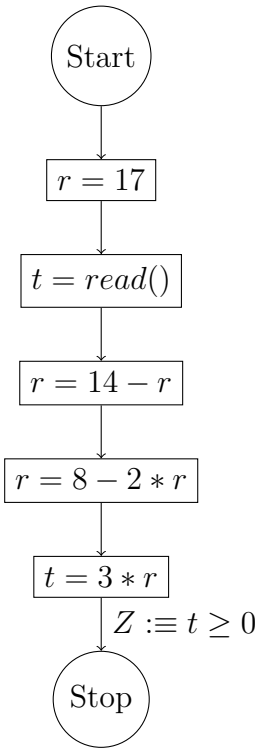
General Information

Detailed information about the lecture, tutorials and homework assignments can be found on the lecture website[1]. Solutions have to be submitted to Moodle[2]. Upload your solution as a single PDF file. Make sure it is readable, blurred images will be rejected. Use Piazza[3] to ask questions and discuss with your fellow students.

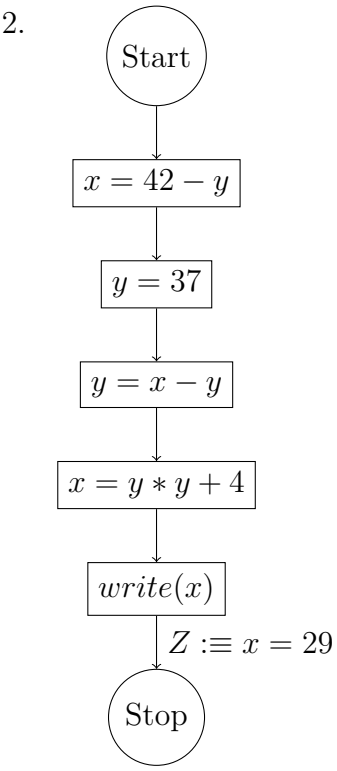
Assignment 2.1 (L) From Post- to Preconditions

Consider these control flow graphs:

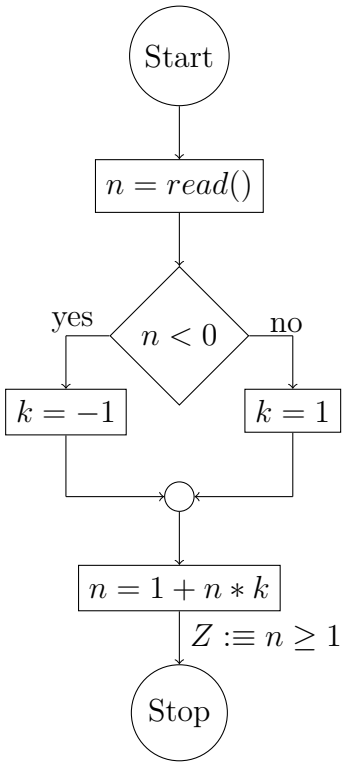
1.



2.



3.



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

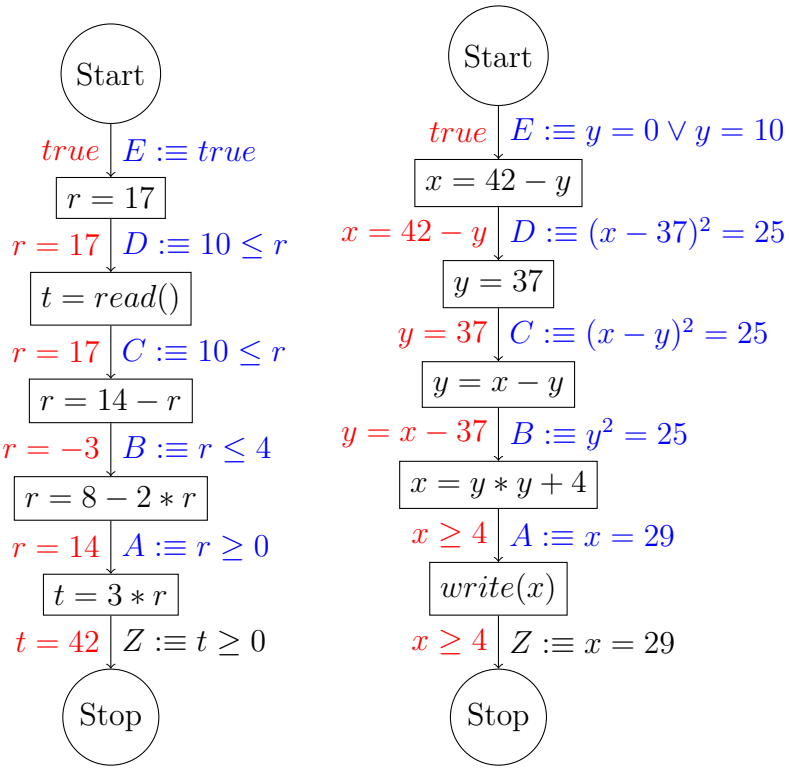
[1] https://www.in.tum.de/i02/lehre/wintersemester-1819/vorlesungen/functional-programming-and-verification/

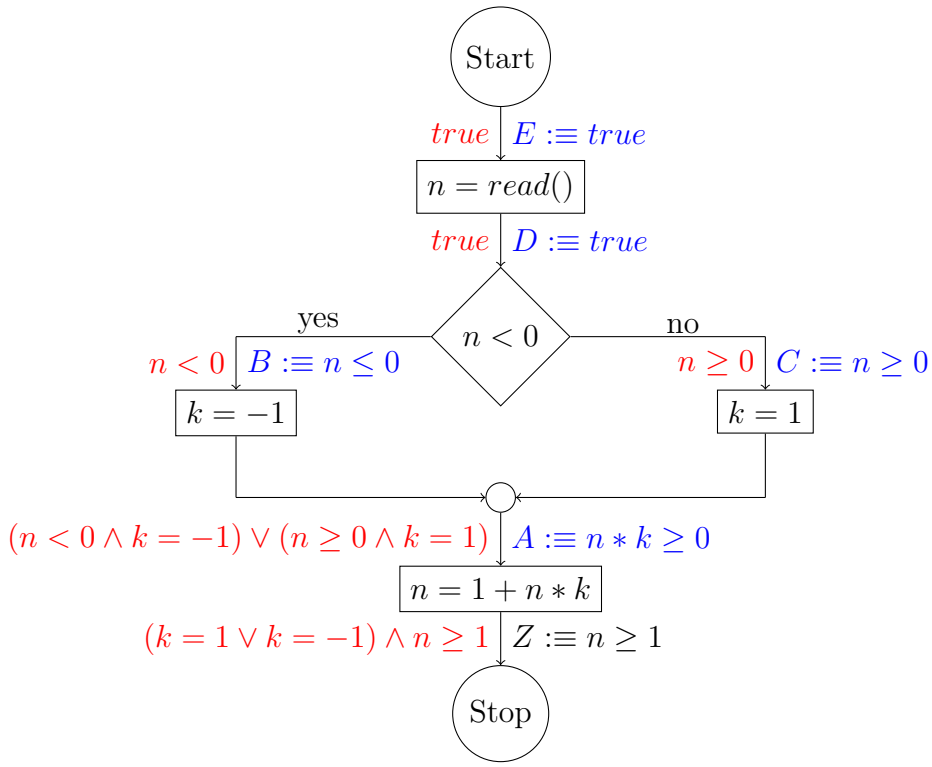
[2] https://www.moodle.tum.de/course/view.php?id=44932

[3] https://piazza.com/tum.de/fall2018/in0003/home

Suggested Solution 2.1

1. The strongest postconditions and weakest preconditions are annotated at the left and right side of each edge respectively:





(a)
- Once strongest postconditions are annotated, it is easy to see that Z holds, since $r = 14 \implies r \geq 0$.
- We compute weakest preconditions as follows:

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{t = 3 * r}]\!](Z)\\
\equiv\ &\mathsf{WP}[\![\texttt{t = 3 * r}]\!](t \geq 0)\\
\equiv\ &3 * r \geq 0\\
\equiv\ &r \geq 0 \quad \equiv: A
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{r = 8 - 2 * r}]\!](A)\\
\equiv\ &\mathsf{WP}[\![\texttt{r = 8 - 2 * r}]\!](r \geq 0)\\
\equiv\ &8 - 2 * r \geq 0\\
\equiv\ &-2 * r \geq -8\\
\equiv\ &r \leq 4 \quad \equiv: B
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{r = 14 - r}]\!](B)\\
\equiv\ &\mathsf{WP}[\![\texttt{r = 14 - r}]\!](r \leq 4)\\
\equiv\ &14 - r \leq 4\\
\equiv\ &10 \leq r \quad \equiv: C
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{t = read()}]\!](C)\\
\equiv\ &\mathsf{WP}[\![\texttt{t = read()}]\!](10 \leq r)\\
\equiv\ &\forall t.\ 10 \leq r\\
\equiv\ &10 \leq r \quad \equiv: D
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{r = 17}]\!](D)\\
\equiv\ &\mathsf{WP}[\![\texttt{r = 17}]\!](10 \leq r)\\
\equiv\ &10 \leq 17\\
\equiv\ &\mathit{true} \quad \equiv: E
\end{aligned}
$$

true is the precondition to satisfy Z and, since true always holds, so does Z.

(b)
- Z does not hold, because $x \geq 4$ does not imply $x = 29$.
- We show the same using weakest preconditions:

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{write(x)}]\!](Z)\\
\equiv\ &\mathsf{WP}[\![\texttt{write(x)}]\!](x = 29)\\
\equiv\ &x = 29 \quad \equiv: A
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = y * y + 4}]\!](A)\\
\equiv\ &\mathsf{WP}[\![\texttt{x = y * y + 4}]\!](x = 29)\\
\equiv\ &y^2 + 4 = 29\\
\equiv\ &y^2 = 25 \quad \equiv: B
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{y = x - y}]\!](B)\\
\equiv\ &\mathsf{WP}[\![\texttt{y = x - y}]\!](y^2 = 25)\\
\equiv\ &(x - y)^2 = 25 \quad \equiv: C
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{y = 37}]\!](C)\\
\equiv\ &\mathsf{WP}[\![\texttt{y = 37}]\!]((x - y)^2 = 25)\\
\equiv\ &(x - 37)^2 = 25 \quad \equiv: D
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = 42 - y}]\!](D)\\
\equiv\ &\mathsf{WP}[\![\texttt{x = 42 - y}]\!]((x - 37)^2 = 25)\\
\equiv\ &(42 - y - 37)^2 = 25\\
\equiv\ &(5 - y)^2 = 25\\
\equiv\ &5 - y = 5 \vee 5 - y = -5\\
\equiv\ &y = 0 \vee y = 10 \quad \equiv: E
\end{aligned}
$$

For Z to hold, y must be 0 or 10 at the start of the program. If we would use a language that zero-initializes all variables, Z would always hold. However, since this is not the case for MiniJava, the program is not guaranteed to compute Z.

(c)
- Strongest Postcondition $(k = 1 \vee k = -1) \wedge n \geq 1 \implies n \geq 1$ and thus Z holds.
- We compute weakest preconditions:

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{n = 1 + n * k}]\!](Z)\\
\equiv\ &\mathsf{WP}[\![\texttt{n = 1 + n * k}]\!](n \geq 1)\\
\equiv\ &1 + n * k \geq 1\\
\equiv\ &n * k \geq 0 \quad \equiv\!: A
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{k = -1}]\!](A)\\
\equiv\ &\mathsf{WP}[\![\texttt{k = -1}]\!](n * k \geq 0)\\
\equiv\ &-n \geq 0\\
\equiv\ &n \leq 0 \quad \equiv\!: B
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{k = 1}]\!](A)\\
\equiv\ &\mathsf{WP}[\![\texttt{k = 1}]\!](n * k \geq 0)\\
\equiv\ &n \geq 0 \quad \equiv\!: C
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{n < 0}]\!](C, B)\\
\equiv\ &\mathsf{WP}[\![\texttt{n < 0}]\!](n \geq 0, n \leq 0)\\
\equiv\ &(\neg(n < 0) \implies n \geq 0) \wedge (n < 0 \implies n \leq 0)\\
\equiv\ &(n \geq 0 \implies n \geq 0) \wedge (n < 0 \implies n \leq 0)\\
\equiv\ &\mathit{true} \quad \equiv\!: D
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{n = read()}]\!](D)\\
\equiv\ &\mathsf{WP}[\![\texttt{n = read()}]\!](\mathit{true})\\
\equiv\ &\forall n.\ \mathit{true}\\
\equiv\ &\mathit{true} \quad \equiv\!: E
\end{aligned}
$$

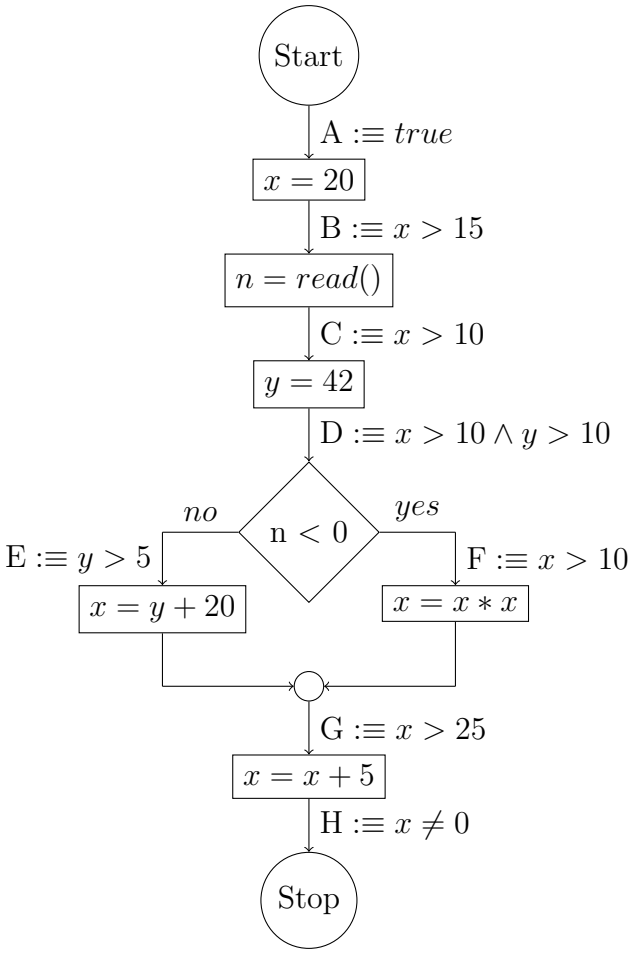
Again, *true* at the start node proves that Z holds for all runs of the program.

2. Advantages and disadvantages of these approaches are, among others:

   - Strongest Postconditions:
     + Gives the most precise information about the program state
     + Very natural way to think about the program, since it respects program execution order
     - Formulas can contain lots of details that are not required to prove a particular assertion
     - Computation rules can be very hard to define, e.g. for assignment
   - Weakest Preconditions:
     + Formulas are reduced to the essential parts to prove a particular goal
     + Computation rules relatively easy to define
     + WP at start node provides information about the runs under which the assertion does not hold
     + Can be used to prove other properties (e.g. termination)

Assignment 2.2 (L) Local Consistency

In the following control flow graph assertions are annotated to all the edges.



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

Suggested Solution 2.2

We recap: In order for an assertion B to hold after a statement s, the weakest precondition $\mathsf{WP}[\![\mathtt{s}]\!](B)$ has to be satisfied before the statement. The important point here is that more than $\mathsf{WP}[\![\mathtt{s}]\!](B)$ (a stronger assertion) may hold before s. In other words: Assume the assertion A holds before the statement s, than we do not require $A \equiv \mathsf{WP}[\![\mathtt{s}]\!](B)$, but $A \implies \mathsf{WP}[\![\mathtt{s}]\!](B)$. If this latter condition is satisfied, we say that the annotated assertions are locally consistent. To prove that H holds for all executions of the program, we have to check that

- all annotated assertions are locally consistent and
- *true* is annotated at the start node.

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = x + 5}]\!](H)\\
\equiv\ &\mathsf{WP}[\![\texttt{x = x + 5}]\!](x \neq 0)\\
\equiv\ &x + 5 \neq 0\\
\Longleftarrow\ &x > 25 \quad \equiv G
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = x * x}]\!](G)\\
\equiv\ &\mathsf{WP}[\![\texttt{x = x * x}]\!](x > 25)\\
\equiv\ &x * x > 25\\
\equiv\ &|x| > 5\\
\Longleftarrow\ &x > 10 \quad \equiv F
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = y + 20}]\!](G)\\
\equiv\ &\mathsf{WP}[\![\texttt{x = y + 20}]\!](x > 25)\\
\equiv\ &y + 20 > 25\\
\equiv\ &y > 5 \quad \equiv E
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{n < 0}]\!](E, F)\\
\equiv\ &\mathsf{WP}[\![\texttt{n < 0}]\!](y > 5, x > 10)\\
\equiv\ &(n \geq 0 \wedge y > 5) \vee (n < 0 \wedge x > 10)\\
\Longleftarrow\ &x > 10 \wedge y > 10 \quad \equiv D
\end{aligned}
$$

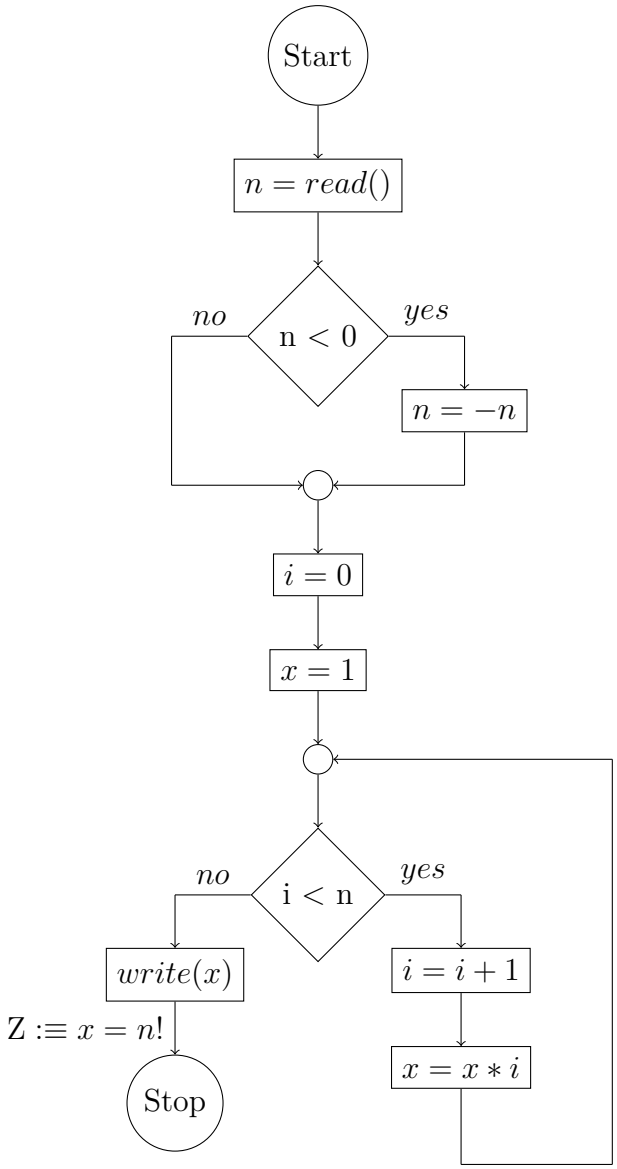
$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{y = 42}]\!](D)\\
\equiv\ &\mathsf{WP}[\![\texttt{y = 42}]\!](x > 10 \wedge y > 10)\\
\equiv\ &x > 10 \wedge 42 > 10\\
\equiv\ &x > 10 \quad \equiv C
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{n = read()}]\!](C)\\
\equiv\ &\mathsf{WP}[\![\texttt{n = read()}]\!](x > 10)\\
\equiv\ &\forall n.x > 10\\
\equiv\ &x > 10\\
\Longleftarrow\ &x > 15 \quad \equiv B
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = 20}]\!](B)\\
\equiv\ &\mathsf{WP}[\![\texttt{x = 20}]\!](x > 15)\\
\equiv\ &20 > 15\\
\equiv\ &\mathit{true} \quad \equiv A
\end{aligned}
$$

Assignment 2.3 (L) Loop Invariants

A program computes the factorial of its input:



Perform the following tasks:

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 $i = 0 \wedge x = 1 \wedge n = 0$ 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 $x = i!$ and improve this invariant until the proof succeeds.

Suggested Solution 2.3

1. Due to the loop in the program, there is a cyclic dependency on the weakest preconditios, so we cannot compute any of the assertions inside the loop.

2. In the previous assignment, we have seen, that we do not necessarily need to compute the assertions. Instead, we can build a successful proof by chosing an assertion manually and show that it is locally consistent. We typically use the loop's join point for such an assertion. We call this assertion the *loop invariant*. The loop invariant then serves as a starting point for our WP computations.

3. We try to prove with the invariant $I :\equiv x \geq 0$:

$$\begin{aligned} &\mathsf{WP}[\![\texttt{write(x)}]\!](Z) \\ \equiv\ &\mathsf{WP}[\![\texttt{write(x)}]\!](x = n!) \\ \equiv\ &x = n! \quad :\equiv A \end{aligned} \qquad \begin{aligned} &\mathsf{WP}[\![\texttt{x = x * i}]\!](I) \\ \equiv\ &\mathsf{WP}[\![\texttt{x = x * i}]\!](x \geq 0) \\ \equiv\ &x * i \geq 0 \quad \equiv: B \end{aligned}$$

$$\begin{aligned} &\mathsf{WP}[\![\texttt{i = i + 1}]\!](B) \\ \equiv\ &\mathsf{WP}[\![\texttt{i = i + 1}]\!](x * i \geq 0) \\ \equiv\ &x(i+1) \geq 0 \quad \equiv: C \end{aligned} \qquad \begin{aligned} &\mathsf{WP}[\![\texttt{i < n}]\!](A, C) \\ \equiv\ &\mathsf{WP}[\![\texttt{i < n}]\!](x = n!, x(i+1) \geq 0) \\ \equiv\ &(i \geq n \wedge x = n!) \vee (i < n \wedge x(i+1) \geq 0) \not\Longleftarrow I \end{aligned}$$

The proof fails, because local consistency cannot be shown at the loop branch. In order to show, that x has exactly the value $n!$ when the loop is left, we need very precise information about the value of x inside the loop (obviously, $x \geq ...$ can never imply $x = ...$). Thus, we can answer:

(a) Typically, a useful loop invariant is directly related to the goal to prove. If one tries to show a particular value of a variable at the end of the program, the invariant has to express precise information about this variable as well.

(c) If the loop invariant is too weak, it is impossible to show local consistency at some point.

(d) It is difficult to define a lower bound for the strength of the loop invariant, however, usually the loop invariant must not be less expressive about the proof goal's variables than the goal itself.

We try again with the invariant $I :\equiv i = 0 \wedge x = 1 \wedge n = 0$:

$$\begin{aligned} &\mathsf{WP}[\![\texttt{write(x)}]\!](Z) \\ \equiv\ &\mathsf{WP}[\![\texttt{write(x)}]\!](x = n!) \\ \equiv\ &x = n! \quad :\equiv A \end{aligned} \qquad \begin{aligned} &\mathsf{WP}[\![\texttt{x = x * i}]\!](I) \\ \equiv\ &\mathsf{WP}[\![\texttt{x = x * i}]\!](i = 0 \wedge x = 1 \wedge n = 0) \\ \equiv\ &i = 0 \wedge x * i = 1 \wedge n = 0 \\ \equiv\ &false \quad \equiv: B \end{aligned}$$

$$\begin{aligned} &\mathsf{WP}[\![\texttt{i = i + 1}]\!](B) \\ \equiv\ &\mathsf{WP}[\![\texttt{i = i + 1}]\!](false) \\ \equiv\ &false \quad \equiv: C \end{aligned} \qquad \begin{aligned} &\mathsf{WP}[\![\texttt{i < n}]\!](A, C) \\ \equiv\ &\mathsf{WP}[\![\texttt{i < n}]\!](x = n!, false) \\ \equiv\ &(i \geq n \wedge x = n!) \vee (i < n \wedge false) \\ \equiv\ &i \geq n \wedge x = n! \quad \Longleftarrow I \end{aligned}$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = 1}]\!](I)\\
\equiv\ &\mathsf{WP}[\![\texttt{x = 1}]\!](i = 0 \wedge x = 1 \wedge n = 0)\\
\equiv\ &i = 0 \wedge n = 0 \quad \equiv: D
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{i = 0}]\!](D)\\
\equiv\ &\mathsf{WP}[\![\texttt{i = 0}]\!](i = 0 \wedge n = 0)\\
\equiv\ &n = 0 \quad \equiv: E
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{n = -n}]\!](E)\\
\equiv\ &\mathsf{WP}[\![\texttt{n = -n}]\!](n = 0)\\
\equiv\ &n = 0 \quad \equiv: F
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{n < 0}]\!](E, F)\\
\equiv\ &\mathsf{WP}[\![\texttt{n < 0}]\!](n = 0, n = 0)\\
\equiv\ &(n \geq 0 \wedge n = 0) \vee (n < 0 \wedge n = 0)\\
\equiv\ &n = 0 \quad \equiv: G
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{n = read()}]\!](G)\\
\equiv\ &\mathsf{WP}[\![\texttt{n = read()}]\!](n = 0)\\
\equiv\ &\forall n.n = 0\\
\equiv\ &\mathit{false} \quad \equiv: H
\end{aligned}
$$

Now, we managed to show local consistency, however, we failed to show that Z holds for all executions of the program. In fact, since we got the precondition false, we merely showed that Z is not violated as long as we do not execute the program, which is completely useless. It is particularly interesting, that, after the `read` statement, we require $n = 0$, which shows that Z indeed holds if 0 is read from the user. Checking the invariant again, notice, that $i = 0 \wedge x = 1 \wedge n = 0$ does not even hold at the respective program point. We learned:

(b) If the loop invariant is too strong, usually, local consistency can be shown, however, one fails to show that it holds for all program executions (one fails to show *true* at the start node).

(d) If an invariant is chosen that is stronger than what actually holds at the join point (strongest postcondition), the proof is quite likely to fail. Thus, the strongest assertion that holds at the join point (e.g. computed using strongest postcondition) is a very natural upper bound for the strength of the loop invariant.

4. When looking at the program closely, one can notice that after i iterations, the program computed $x = i!$, so we try proving Z using the loop invariant $I :\equiv x = i!$:

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{write(x)}]\!](Z)\\
\equiv\ &\mathsf{WP}[\![\texttt{write(x)}]\!](x = n!)\\
\equiv\ &x = n! \quad :\equiv A
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = x * i}]\!](I)\\
\equiv\ &\mathsf{WP}[\![\texttt{x = x * i}]\!](x = i!)\\
\equiv\ &x * i = i!\\
\equiv\ &x = (i-1)! \quad :\equiv B
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{i = i +1}]\!](B)\\
\equiv\ &\mathsf{WP}[\![\texttt{i = i + 1}]\!](x = (i-1)!)\\
\equiv\ &x = i! \quad \equiv: C
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{i < n}]\!](A, C)\\
\equiv\ &\mathsf{WP}[\![\texttt{i < n}]\!](x = n!, x = i!)\\
\equiv\ &(i \geq n \wedge x = n!) \vee (i < n \wedge x = i!)\\
\Longleftarrow\ &(i = n \wedge x = n!) \vee (i < n \wedge x = i!)\\
\equiv\ &(i = n \wedge x = i!) \vee (i < n \wedge x = i!)\\
\equiv\ &i \leq n \wedge x = i! \quad \not\Longleftarrow I
\end{aligned}
$$

We fail to show local consistency with this invariant, however, we recognize that if I itself contained the term $i \leq n$, it might work, so we restart the proof with $I :\equiv x = i! \wedge i \leq n$:

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = x * i}]\!](I) \\
\equiv\ &\mathsf{WP}[\![\texttt{x = x * i}]\!](x = i! \wedge i \leq n) \\
\equiv\ &x * i = i! \wedge i \leq n \\
\equiv\ &x = (i-1)! \wedge i \leq n \quad :\equiv B
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{i = i + 1}]\!](B) \\
\equiv\ &\mathsf{WP}[\![\texttt{i = i + 1}]\!](x = (i-1)! \wedge i \leq n) \\
\equiv\ &x = i! \wedge i + 1 \leq n \\
\equiv\ &x = i! \wedge i < n \quad :\equiv C
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{i < n}]\!](A, C) \\
\equiv\ &\mathsf{WP}[\![\texttt{i < n}]\!](x = n!, x = i! \wedge i < n) \\
\equiv\ &(i \geq n \wedge x = n!) \vee (i < n \wedge x = i! \wedge i < n) \\
\Longleftarrow\ &(i = n \wedge x = n!) \vee (i < n \wedge x = i!) \\
\equiv\ &(i = n \wedge x = i!) \vee (i < n \wedge x = i!) \\
\equiv\ &x = i! \wedge i \leq n \quad \equiv I
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{x = 1}]\!](I) \\
\equiv\ &\mathsf{WP}[\![\texttt{x = 1}]\!](x = i! \wedge i \leq n) \\
\equiv\ &1 = i! \wedge i \leq n \quad \equiv\!: D
\end{aligned}
$$

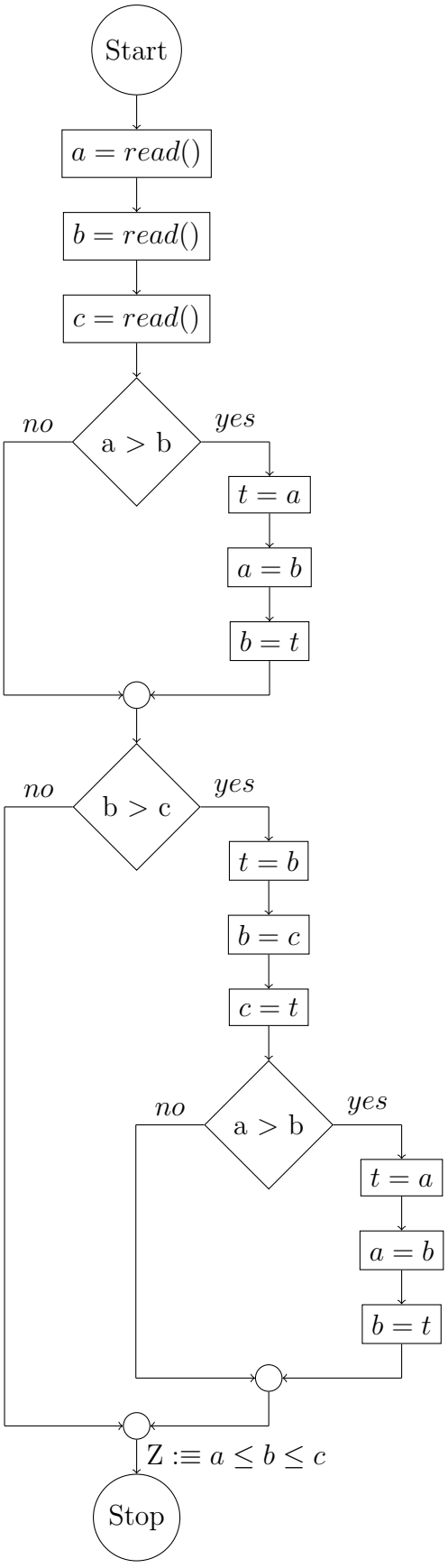
$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{i = 0}]\!](D) \\
\equiv\ &\mathsf{WP}[\![\texttt{i = 0}]\!](1 = i! \wedge i \leq n) \\
\equiv\ &0 \leq n \quad \equiv\!: E
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{n = -n}]\!](E) \\
\equiv\ &\mathsf{WP}[\![\texttt{n = -n}]\!](0 \leq n) \\
\equiv\ &0 \geq n \quad \equiv\!: F
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{WP}[\![\texttt{n < 0}]\!](E, F) \\
\equiv\ &\mathsf{WP}[\![\texttt{n < 0}]\!](0 \leq n, 0 \geq n) \\
\equiv\ &(n \geq 0 \wedge 0 \leq n) \vee (n < 0 \wedge 0 \geq n) \\
\equiv\ &\mathit{true} \quad :\equiv G
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{WP}[\![\texttt{n = read()}]\!](G) \\
\equiv\ &\mathsf{WP}[\![\texttt{n = read()}]\!](\mathit{true}) \\
\equiv\ &\mathit{true} \quad :\equiv H
\end{aligned}
$$

□

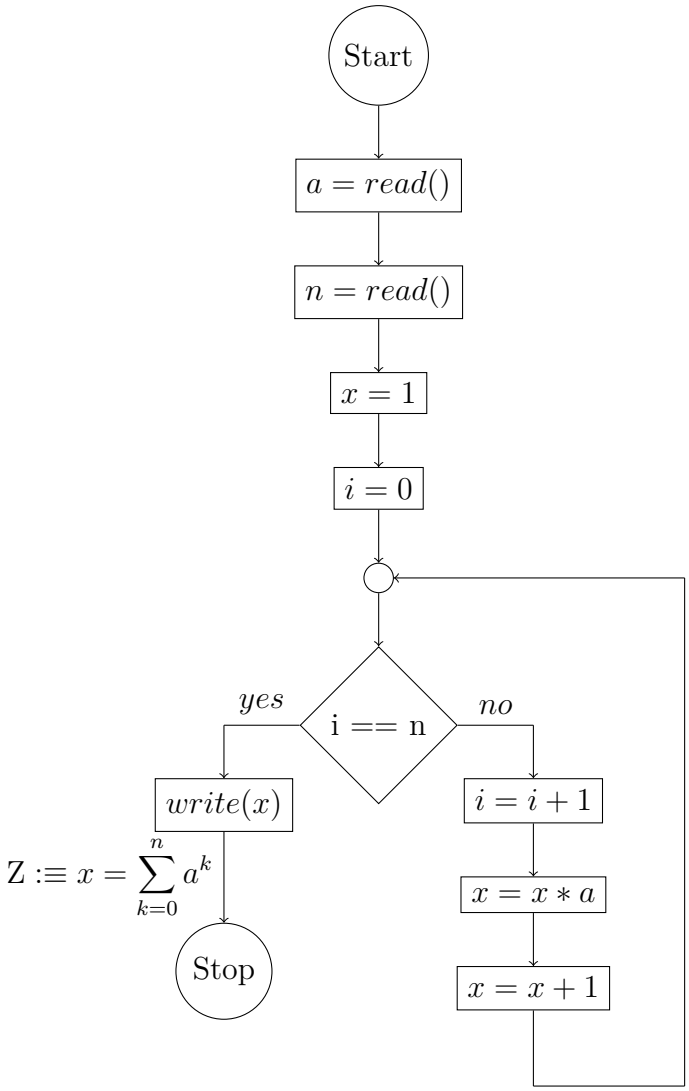
Assignment 2.4 (H) Trouble Sort [4.5 Points]

Prove Z using weakest preconditions:



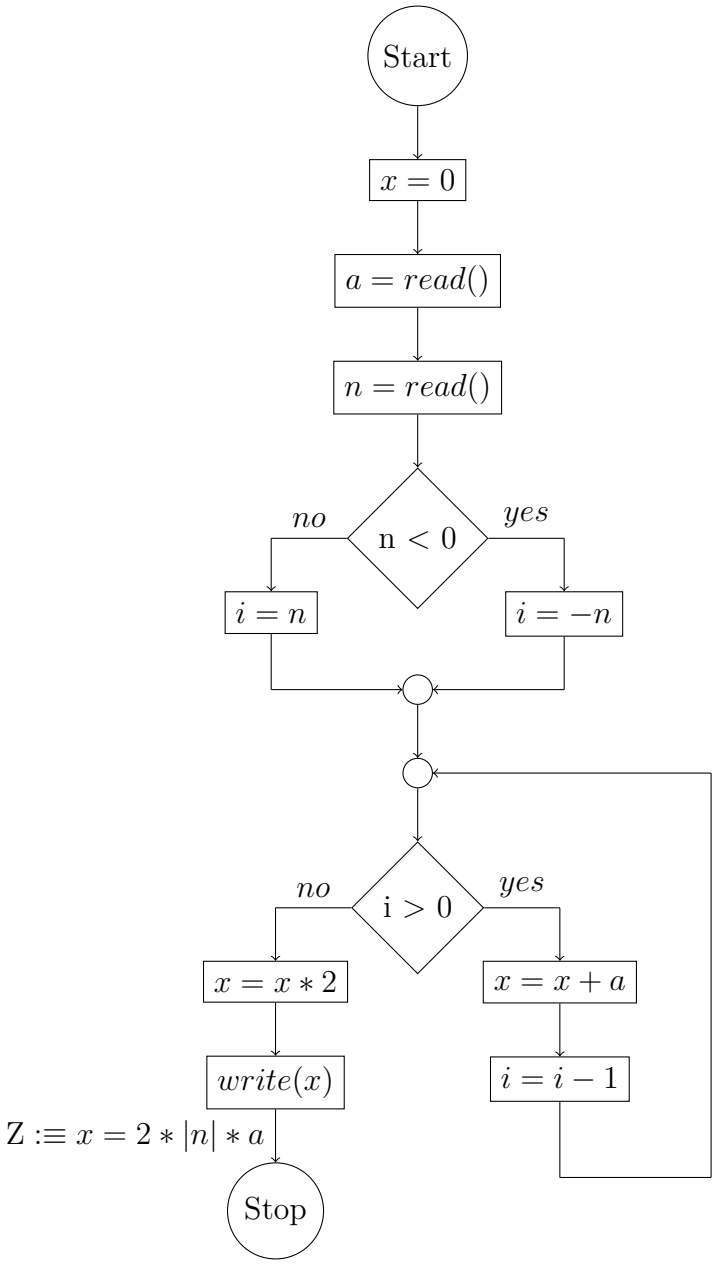
Assignment 2.5 (H) Powerful Sums [5 Points]

Prove Z using weakest preconditions:



Assignment 2.6 (H) Twona [6.5 Points]

Prove Z using weakest preconditions:



Assignment 2.7 (H) MiniJava 2.0 [4 Points]

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 $\mathsf{WP}[\![\ ]\!](B)$ for each of these statements.