
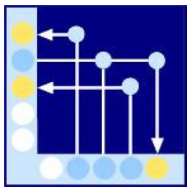

# Advanced Programming with MOSTflexiPL

Lecture in Wintersemester 2025/2026
Prof. Dr. habil. Christian Heinlein

## 3. Task Sheet (November 18, 2025)

## Aufgabe 5: Rotation of Variable Values

Define an operator `rotate` that, depending on the way it is invoked, rotates the values of any number of `int` variables either left or right by one position, for example:

```
a : int?; a =! 1;
b : int?; b =! 2;
c : int?; c =! 3;
rotate a b c left;
print only ?a; print only ?b; print ?c;          $$ 231
rotate a b c right;
print only ?a; print only ?b; print ?c          $$ 123
```

## Aufgabe 6: Minimum and Maximum

Define an operator that, depending on the way it is invoked, returns either the minimum or the maximum of any number of `int` values, if it exists, i. e., one of the values that is less than or equal (minimum) or greater than or equal (maximum) to all other values, respectively. If no such value exists – which might happen because of unnatural values, because they are neither less than nor greater than nor equal to other values –, nil shall be returned in these cases.

For example:

```
s : int;
min 1 end;                      $$ 1
min 3 1 2 end;                  $$ 1
max 1 1 end;                    $$ 1
max 1 nil 2 end;                $$ nil
min s 3 4 5 end;                $$ nil
min s s end                     $$ s
```

A minimum or maximum exists if and only if every two consecutive values `x` and `y` are comparable, i. e., if `x <= y` or `y <= x` is true for them. (This is, for example, also the case if all values are the same unnatural value, which is both the minimum and the maximum in this case.)

# Aufgabe 7: Chained Integer Comparisons

Generalize the comparison operators for integer numbers from task 2 into a single operator that allows chained comparisons of all kinds with two or more operands, for example:

```
a < b < c
a <= b = c > d /= e
```

For that purpose, the values of every two consecutive operands are compared.
The result value is `true` if and only if each of these comparisons returns `true`.

Use indirect exclusions to specify that this new operator has the same binding properties as the predefined equality test and vice versa (that means in particular, that the predefined equality test is forbidden in all operands of the operator, and conversely, that the new operator is also forbidden in the operands of the predefined equality test).

## Notice

The definition of this new comparison operator makes comparisons `a = b` of two `int` values ambiguous, because they fit on the new operator as well as on the predefined equality test. This ambiguity can be removed as follows (precise explanation in §3.9.2 of the lecture slides):

```
$$ Auxiliary operator.
equal (x:int) (y:int) -> (bool =
    $$ Invocation of the predefined equality test.
    x = y
);

$$ Specialization of the predefined equality test,
$$ which can compare values of any type, for int values.
(x:int) "=" (y:int) -> (bool =
    $$ Indirect invocation of the predefined equality test
    $$ via the auxiliary operator equal.
    equal x y
);

$$ This specialization shall have the same binding properties
$$ as the predefined equality test and vice versa.
excl (true = false) <-> (1 = 2); true <-> 1; false <-> 2 end
```

Because comparisons of two `int` values are already needed in the implementation of the chained comparison, these definitions must appear before the definition of the chained comparison.