Hochschule Aalen
Fakultät Elektronik und Informatik
Studienbereich Informatik

Advanced Programming with MOSTflexiPL

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

6. Task Sheet (Dezember 9, 2025)

Task 11: Forwarding of Operator Applications

Subtask 11.a)

Define an operator that exactly covers the intersection of the operators from task 7 and 8.b, causing expressions such as `1 = 2 /= 3` to be unambiguous even if both of these operators are visible.

Implement this new operator by forwarding its applications to one of these two other operators.

Just like in task 9.a, every operand shall be evaluated only if its value is needed to determine the result value of the entire comparison.

This operator shall also have the same binding properties as the predefined equality test •=•.

Note: When forwarding operator applications, a lambda parameter might only be forwarded to another lambda parameter, and an ordinary parameter might only be forwarded to another ordinary parameter, even though the compiler erroneously accepts other combinations, which will not work correctly at run time, however.

Subtask 11.b)

Eliminate, if possible, code replications in the implementations of your operators defined in previous tasks by means of complete or partial forwarding to suitable local auxiliary operators.

Because partial forwarding sometimes causes run time errors, try to use complete forwarding whenever possible. For example:

```
(while|until|do) [(X:type)] (\ x -> (X))
{ (while|until|do) [(Y:type)] (\ y -> (Y)) }
end -> (int =
    ......

    (?* (
        $$ Direct implementation:
        ......;                 $$ Processing of x.
        { ...... };             $$ Repeated processing of y
                                $$ in the same way as x.
        $$ Alternatively with local auxiliary operator:
        aux { (while|until|do) [(Z:type)] (\ z -> (Z)) } -> (int =
            { ...... }          $$ Repeated processing of z.
        );
        <>(aux);

        ......
    )) - 1
)
```

Task 12: Flexible Output of `int`, `char`, and `bool` Values

Define an operator `print` that consecutively prints any number of `int`, `char`, and `bool` values.

As with the predefined operator `print`, a terminating line separator is printed after the last value if `only` has not been given.

After `print` and optionally `only`, any number of operands with types `int`, `char`, and `bool` and any number of the following words and phrases can be given in any order:

- `bin`: Binary output of `int` values.
- `oct`: Octal output of `int` values.
- `dec`: Decimal output of `int` values (default).
- `hex`: Hexadecimal output of `int` values.
- `lower`: Use of lower case letters for the hexadecimal output of `int` values and for the output of `bool` values with letters (default).
- `upper`: Use of upper case letters for the hexadecimal output of `int` values and for the output of `bool` values with letters.
- `letter`: Output of the `bool` values `true` and `false` with the letters `t` and `f` or `T` and `F`, respectively, according to `lower` or `upper`.
- `digit`: Output of the `bool` values `true` and `false` with the digits `1` and `0` (default).
- `sign`: Output of the `bool` values `true` and `false` with the characters `+` and `-`.
- `chars t f` with two arbitrary `char` values `t` and `f`:
  Output of the `bool` values `true` and `false` with the characters `t` and `f`.
- `sep s`: The separator character `s` is printed between consecutive values (default is a space character).
- `tight`: No separator character is printed between consecutive values.
- `reset`: All settings are reset to the default values mentioned above.

Each of these specifications is applied to all subsequent operands of the current and all subsequent applications of `print` until it is overridden by another specification of the same category (`bin`/`oct`/`dec`/`hex`, `lower`/`upper`, `letter`/`digit`/`sign`/`chars`, `sep`/`tight`) or until `reset` is used to reset it to the default value of this category.

After each of the values, an optional `width` and an `int` value specifying the minimum output width for this value can be given. If the value's width (i. e., the number of characters for its output) is less than the minimum width, the corresponding number of space characters is printed before the value. (This means conversely: If the value's width is greater than or equal to the minimum width, or if the minimum width is an unnatural value, no additional spaces are printed.)

As with the predefined operator `print`, the output of unnatural `int` and `char` values as such shall be empty. Every `bool` value except `false` shall be printed in the same way as `true`.

For example:

```
$$ Application of print:                $$ Corresponding output:

print 10 20 30;                         $$ 10 20 30
print 10 hex 20 30;                     $$ 10 14 1e
print true false;                       $$ 1 0
print sep '|' letter upper true false;  $$ T|F
print 0 tight chars 'W' 'F' true;       $$ 0W
print dec sep '|' 0 1:0 width 2 -123 width 5 456 width 1:0;
                                        $$ 0|  | -123|456
print;                                  $$ (Just a blank line)
print only reset                        $$ (Reset all settings without any output)
                                        %
```