Operational Semantics

Initial configuration of executing statement S: $\langle S, \emptyset, \emptyset \rangle$. Repeatedly apply these rules until there are no commands left ($c = \emptyset$).

Arithmetic expressions (E):

$$\begin{aligned}
\langle n\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \boldsymbol{c}, n\boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle v\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \boldsymbol{c}, m(v), \boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle (E_1 \ \mathit{iop}\ E_2)\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle E_1\ E_2\ \mathit{iop}\ \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle \mathit{iop}\ \boldsymbol{c}, n_2 n_1 \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \boldsymbol{c}, n\boldsymbol{s}, \boldsymbol{m} \rangle \text{ where } n = n_1\ \underline{\mathit{iop}}\ n_2
\end{aligned}$$

Execution will be guarded against division by zero.

Boolean conditions (C):

$$\begin{aligned}
\langle b\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle b, n\boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle (E_1\ \mathit{bop}\ E_2)\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle E_1\ E_2\ \mathit{bop}\ \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle \mathit{bop}\ \boldsymbol{c}, n_2 n_1 \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \boldsymbol{c}, b\boldsymbol{s}, \boldsymbol{m} \rangle \text{ where } b = n_1\ \underline{\mathit{bop}}\ n_2
\end{aligned}$$

Statements (S):

$$\begin{aligned}
\langle ()\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle (S_1; S_2)\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle S_1 S_2 \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle v = E\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle E\ \mathit{save}\ \boldsymbol{c}, v\boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle \mathit{save}\ \boldsymbol{c}, nv\boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m}[v = n] \rangle \\
\langle \mathit{continue}\ \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \mathbf{c'}, \boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle \mathit{break}\ \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \mathbf{c''}, \boldsymbol{s}, \boldsymbol{m} \rangle \\
\langle \mathit{exit}\ \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m} \rangle &\to \langle \emptyset, \emptyset, \emptyset \rangle
\end{aligned}$$

where $\mathbf{c'}$:= the first *entered while* in $\boldsymbol{c}$ and the commands after it; and $\mathbf{c''}$:= the commands after the first *entered while* (excluding it).

Branching (if):

$$\begin{aligned}\langle (if\ C\ then\ S_t\ else\ S_f)\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m}\rangle &\to \langle C\ branch\ \boldsymbol{c}, S_t S_f \boldsymbol{s}, \boldsymbol{m}\rangle\\ \langle branch\ \boldsymbol{c}, true\ S_t S_f \boldsymbol{s}, \boldsymbol{m}\rangle &\to \langle S_t \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m}\rangle\\ \langle branch\ \boldsymbol{c}, false\ S_t S_f \boldsymbol{s}, \boldsymbol{m}\rangle &\to \langle S_f \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m}\rangle\end{aligned}$$

Looping ($while$):

$$\begin{aligned}\langle (while\ C\ do\ S)\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m}\rangle &\to \langle C\ loop\ \boldsymbol{c}, CS\boldsymbol{s}, \boldsymbol{m}\rangle\\ \langle loop\ \boldsymbol{c}, false\ CS\boldsymbol{s}, \boldsymbol{m}\rangle &\to \langle \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m}\rangle\\ \langle loop\ \boldsymbol{c}, true\ CS\boldsymbol{s}, \boldsymbol{m}\rangle &\to \langle S(while\ C\ do\ S)\boldsymbol{c}, \boldsymbol{s}, \boldsymbol{m}\rangle\end{aligned}$$

When applying the last rule, we also mark the *while* statement as *entered*. That is because encountering a loop with its condition evaluating to *true* causes us to enter it (at least one execution of its *body*). The *entered* flag is used for *break* and *continue* statements.

Execution will be guarded against infinite cycles.