# Project Language Specifications (Denotational Semantics)

## 1 Dynamic Semantics

This section gives the dynamic semantics of the language using denotational semantics. Consider the *demsem* function the denotational semantics for this language. We will use a mapping from variable name to value to represent the symbol table of the program during execution, and in code can be represented as a HashMap or similar datatype in your language of choice. We will use a sequence of characters to represent the output of a program, with $\epsilon$ representing the empty sequence. I will also assume that all strings will be represented as sequences of characters. Assume there is a function *append* that, when given two sequences, appends the second sequence to the first. Also assume, there is a function *seq* that takes an integer and gives a sequence of characters representing that integer as text. Assume there are the functions *head*, which maps a sequence to its first element, *tail*, which maps a sequence to a new one created by removing the first element, *clean*, which maps a sequence of input characters to a new sequence by removing any non-digits from the front of the sequence, and *int* that maps a sequence of digits to the corresponding integer. If the sequence is empty, *int* will give zero. A state, as well as the meaning of a program, will be a 3-tuple consisting of a variable name mapping function, a sequence of input characters and an output sequence. The initial state for any program is $(\{\}, i, \epsilon)$, where $i$ is some sequence of characters the user will input. If a token (represented by all caps and bold font) appears as a value on the right hand side of a function definition, then replace it with its lexeme. So if a **ID** was generated by the lexer from an $x$, then replace **ID** with $x$.

$$
\begin{aligned}
\mathit{densem}(\epsilon, (\theta, i, p)) &= (\theta, i, p)\\
\mathit{densem}(\texttt{<stmt>}\ \text{“;”}\ \texttt{<stmt\_list>}, (\theta, i, p)) &= \mathit{densem}(\texttt{<stmt\_list>}, \mathit{densem}(\texttt{<stmt>}, (\theta, i, p)))\\
\mathit{densem}(\text{“print”}\ \mathbf{STRING}, (\theta, i, p)) &= (\theta, i, \mathit{append}(p, \mathbf{STRING}))\\
\mathit{densem}(\text{“print”}\ \texttt{<expr>}, (\theta, i, p) &= (\theta, i, \mathit{append}(p, \mathit{seq}(\mathit{out})))\\
&\quad \text{where } \mathit{out} = \mathit{exprsem}(\texttt{<expr>})\\
\mathit{densem}(\text{“get”}\ \mathbf{ID}, (\theta, i, p)) &= (\theta', i', p)\\
&\quad \text{where}\\
&(x, i') = \mathit{getInt}(\mathit{clean}(i))\\
&\theta'(n) = \text{if } n = \mathbf{ID} \text{ then } x \text{ else } \theta(n)\\
\mathit{densem}(\mathbf{ID}\ \text{“=”}\ \texttt{<expr>}, (\theta, i, p)) &= (\theta', i, p)\\
&\quad \text{where}\\
&\theta'(n) = \text{if } n = \mathbf{ID} \text{ then } \mathit{exprsem}(\texttt{<expr>}, \theta) \text{ else } \theta(n)\\
\mathit{densem}(\texttt{<if>}, (\theta, i, p)) &= \text{if } \mathit{exprsem}(\texttt{<if>}.\texttt{<expr>}, \theta) \neq 0\\
&\quad \text{then } \mathit{densem}(\texttt{<if>}.\texttt{<stmt\_list>}[0], (\theta, i, p))\\
&\quad \text{else } \mathit{densem}(\texttt{<if>}.\texttt{<stmt\_list>}[1], (\theta, i, p))
\end{aligned}
$$

$$
\begin{aligned}
densem(\texttt{<while>}, (\theta, i, p)) &= \text{if } exprsem(\texttt{<while>}\,.\,\texttt{<expr>}, \theta) = 0 \\
&\quad \text{then } (\theta, i, p) \\
&\quad \text{else } densem(\texttt{<while>}, \\
&\qquad\qquad\qquad densem(\texttt{<while>}\,.\,\texttt{<stmt\_list>}, (\theta, i, p))) \\
exprsem(\texttt{<expr>}, \theta) &= \text{if } \texttt{<expr>}\,.\,\texttt{<b\_expr>} = \epsilon \\
&\quad \text{then } exprsem(\texttt{<expr>}\,.\,\texttt{<n\_expr>}, \theta) \\
&\quad \text{else } bexprsem(\texttt{<expr>}\,.\,\texttt{<b\_expr>}, \\
&\qquad\qquad\qquad exprsem(\texttt{<expr>}\,.\,\texttt{<n\_expr>}\,), \theta) \\
exprsem(\texttt{<n\_expr>}, \theta) &= \text{if } \texttt{<n\_expr>}\,.\,\texttt{<t\_expr>} = \epsilon \\
&\quad \text{then } exprsem(\texttt{<n\_expr>}\,.\,\texttt{<term>}, \theta) \\
&\quad \text{else } texprsem(\texttt{<n\_expr>}\,.\,\texttt{<t\_expr>}, \\
&\qquad\qquad\qquad exprsem(\texttt{<n\_expr>}\,.\,\texttt{<term>}\,), \theta) \\
exprsem(\texttt{<term>}, \theta) &= \text{if } \texttt{<term>}\,.\,\texttt{<f\_expr>} = \epsilon \\
&\quad \text{then } exprsem(\texttt{<term>}\,.\,\texttt{<factor>}, \theta) \\
&\quad \text{else } fexprsem(\texttt{<term>}\,.\,\texttt{<f\_expr>}, \\
&\qquad\qquad\qquad exprsem(\texttt{<term>}\,.\,\texttt{<factor>}\,), \theta) \\
exprsem(\texttt{<factor>}, \theta) &= \text{if } \texttt{<factor>}\,.\,\texttt{<v\_expr>} = \epsilon \\
&\quad \text{then } exprsem(\texttt{<factor>}\,.\,\texttt{<value>}, \theta) \\
&\quad \text{else } vexprsem(\texttt{<factor>}\,.\,\texttt{<v\_expr>}, \\
&\qquad\qquad\qquad exprsem(\texttt{<factor>}\,.\,\texttt{<value>}\,), \theta) \\
exprsem(\text{“(”}\ \texttt{<expr>}\ \text{“)”}, \theta) &= exprsem(\texttt{<expr>}, \theta) \\
exprsem(\text{“not”}\ \texttt{<value>}, \theta) &= \text{if } exprsem(\texttt{<value>}, \theta) = 0 \text{ then } 1 \text{ else } 0 \\
exprsem(\text{“-”}\ \texttt{<value>}, \theta) &= -exprsem(\texttt{<value>}, \theta) \\
exprsem(\mathbf{ID}, \theta) &= \theta(\mathbf{ID}) \\
exprsem(\mathbf{INT}, \theta) &= \mathbf{INT} \\
bexprsem(\text{“and”}\ \texttt{<n\_expr>}, v, \theta) &= \text{if } v \neq 0 \text{ and } exprsem(\texttt{<n\_expr>}, \theta) \neq 0 \text{ then } 1 \text{ else } 0 \\
bexprsem(\text{“or”}\ \texttt{<n\_expr>}, v, \theta) &= \text{if } v \neq 0 \text{ or } exprsem(\texttt{<n\_expr>}, \theta) \neq 0 \text{ then } 1 \text{ else } 0 \\
texprsem(\text{“+”}\ \texttt{<n\_expr>}, v, \theta) &= v + exprsem(\texttt{<n\_expr>}, \theta) \\
texprsem(\text{“-”}\ \texttt{<n\_expr>}, v, \theta) &= v - exprsem(\texttt{<n\_expr>}, \theta) \\
fexprsem(\text{“*”}\ \texttt{<term>}, v, \theta) &= v \times exprsem(\texttt{<term>}, \theta) \\
fexprsem(\text{“/”}\ \texttt{<term>}, v, \theta) &= \frac{v}{exprsem(\texttt{<term>}, \theta)} \\
fexprsem(\text{“\%”}\ \texttt{<term>}, v, \theta) &= v \mod exprsem(\texttt{<term>}, \theta) \\
vexprsem(\text{“>”}\ \texttt{<value>}, v, \theta) &= \text{if } v > exprsem(\texttt{<value>}, \theta) \text{ then } 1 \text{ else } 0 \\
vexprsem(\text{“>=”}\ \texttt{<value>}, v, \theta) &= \text{if } v \geqslant exprsem(\texttt{<value>}, \theta) \text{ then } 1 \text{ else } 0 \\
vexprsem(\text{“<”}\ \texttt{<value>}, v, \theta) &= \text{if } v < exprsem(\texttt{<value>}, \theta) \text{ then } 1 \text{ else } 0 \\
vexprsem(\text{“<=”}\ \texttt{<value>}, v, \theta) &= \text{if } v \leqslant exprsem(\texttt{<value>}, \theta) \text{ then } 1 \text{ else } 0 \\
vexprsem(\text{“==”}\ \texttt{<value>}, v, \theta) &= \text{if } v = exprsem(\texttt{<value>}, \theta) \text{ then } 1 \text{ else } 0 \\
vexprsem(\text{“!=”}\ \texttt{<value>}, v, \theta) &= \text{if } v \neq exprsem(\texttt{<value>}, \theta) \text{ then } 1 \text{ else } 0
\end{aligned}
$$

$$
\begin{aligned}
getInt(i) &= (int(x), i') \\
&\text{where } (x, i') = getIntSeq(\epsilon, i) \\
getIntSeq(i_1, i_2) &= \text{if } digit(head(i_2)) \\
&\text{then } getIntSeq(append(i_1, head(i_2)), tail(i_2)) \\
&\text{else } (i_1, i_2)
\end{aligned}
$$