Assert Statements
Addendum to Appendix 3: Pseudocode

We use an assert statement to require a condition to be true at that point in the program (compare to Java’s assert statement). An assert statement of the form:

\[ \text{assert } p \]

results in an error if \( p \) is false when the statement is executed. \( p \) is a condition that can freely use the variables in the program. If \( p \) is a predicate logic statement, the program’s variables must not be quantified.

Consider this procedure to return the maximum value of two numbers.

\[
\text{procedure } \text{max2}(x, y) \\
\quad \text{assert } x \in \mathbb{R} \text{ and } y \in \mathbb{R} \\
\quad \text{if } x < y \\
\quad \quad \text{then } \text{max} := y \\
\quad \quad \text{else } \text{max} := x \\
\quad \quad \text{assert } \text{max} = x \lor \text{max} = y \\
\quad \quad \text{assert } \text{max} \geq x \land \text{max} \geq y \\
\quad \text{return } \text{max}
\]

Here is an explanation of the assert statements.

- The first assert statement of \( \text{max2} \) requires that both inputs be real numbers.

  Assert statements at the beginning of a procedure should be used to describe the preconditions (also called the initial assertion) of the procedure. The intent is that the procedure will not run correctly unless the preconditions are true.

- The last two assert statements describes what the procedure accomplishes: \( \text{max} \) is equal to one of the inputs, and \( \text{max} \) is greater than or equal to both inputs.

  Assert statements at the end of a procedure should be used to describe the postconditions (also called the final assertion) of the procedure. Ideally, whenever the preconditions of the procedure are true before the procedure is executed, the postconditions are true after the procedure is executed.

Assert statements can also be used within a procedure to describe what progress has been made up to that point. Assertions within loops are especially informative. For example, consider this procedure to return the maximum value of a sequence.
procedure maximum \((a_1, a_2, \ldots, a_n)\)

assert \(n \in \mathbb{Z}^+ \land \{a_1, a_2, \ldots, a_n\} \subseteq \mathbb{R}\)

\(\max := a_1\)

\(i := 2\)

assert \(\max = a_1 \land i = 2\)

while \(i \leq n\)

assert \(\max\) is the maximum of \((a_1, a_2, \ldots, a_{i-1})\)

if \(\max < a_i\) then \(\max := a_i\)

assert \(\max\) is the maximum of \((a_1, a_2, \ldots, a_i)\)

\(i := i + 1\)

assert \(\max\) is the maximum of \((a_1, a_2, \ldots, a_n)\)

return \(\max\)

Here is an explanation of the assert statements.

- The first assert statement of \textit{maximum} (its preconditions) requires that the sequence contains at least one element (hard to find the maximum of an empty sequence) and that the sequence contains real numbers. This assert statement could be restated in predicate logic as:

  \[\text{assert } n \in \mathbb{Z}^+ \land \forall x (x \in \{1, 2, \ldots, n\} \rightarrow a_x \in \mathbb{R})\]

- The second assert statement simply describes the result of the initial assignments to \(\max\) and \(i\).

- The assert statements inside the loop describe the progress that is made by each iteration and the if statement in the loop. Before the if statement, \(\max\) is the maximum of the first \(i - 1\) elements of the sequence; this is the result of previous iterations of the loop. After the if statement, \(\max\) is the maximum of the first \(i\) elements of the sequence; the if statement processes one more element of the sequence. Note that the assertions are with respect to the value of \(i\) at that point in the program.

- The first assert statement in the loop is a special kind of assertion called the \textit{loop invariant} (an odd name because the program’s variables change). A loop invariant is true before the loop begins, is true at the beginning of each repetition, and is true after the loop ends. It should describe how much progress has been made at that point relative to the loop variable.

- The last assert statement (the postcondition) describes what the procedure accomplishes, that \(\max\) is the maximum of the sequence. This could be restated in predicate logic as:\footnote{We use “//” to start comments. The book uses curly braces \{ \} for comments, but this can be confused with set notation. To further avoid confusion, each // comment should be on a separate line by itself.}

\[\text{assert } \max\] is the maximum of \((a_1, a_2, \ldots, a_n)\)
// max is greater or equal to $a_1, \ldots, a_n$
assert $\forall x \in \{1, 2, \ldots, n\} \rightarrow max \geq a_x$

// max is equal to one of $a_1, \ldots, a_n$
assert $\exists x \in \{1, 2, \ldots, n\} \land max = a_x$

There should be more assertions in the loop and after the loop stating how $i$ relates to $n$. Who knew finding the maximum was so complicated?

If the program has been completely specified, then we should be able to determine the correctness of the program. Suppose, there is a code segment surrounded by assert statements.

assert $p$
code $C$
assert $q$

Ideally, in every possible case where $p$ is true before code segment $C$, it should be also be the case that $q$ is true after $C$ is executed. Proving this is called program verification, which is introduced in Section 5.5 of the book.