We present the syntax and semantics of an imperative programming language called C0. In a nutshell it is Pascal [HW73] with C syntax. Although C0 is powerful enough to do some serious programming work, its context-free grammar $G$ shown in Table 12 fits on a single page.

We proceed in the obvious order. In Sect. 11.1 we present the grammar and give an informal introduction to the language. As an early programming example we indicate in Sect. 11.1.8 how to represent derivation trees as C0 data structures. This is meant as an aid for anybody who wishes to implement algorithms which process derivation trees like the compiler from Chap. 12. In Sect. 11.1.9 we develop some basic machinery for navigating in all kinds of sequences (statement sequences, declaration sequences, parameter sequences, etc.) occurring in derivation trees of grammar $G$.

Then we formally define the declared types in Sect. 11.2, configurations in Sect. 11.3, expression evaluation in Sect. 11.4, and statement execution in Sect. 11.5.

The start symbol of the grammar is

$$G.S = \langle \text{prog} \rangle.$$

The language $L(\text{prog})$ generated by the grammar is a superset of all C0 programs. Numerous restrictions on the syntax are defined when we define the semantics, i.e., the meaning of C0 programs. In the context of program semantics the restrictions will appear most natural: they are just the conditions which happen to make the definition of the semantics work. In the course of formulating these restrictions and studying their consequences a simple theory of type correctness evolves.

In Sect. 11.6 we illustrate the use of the C0 semantics with the correctness proofs of a few example programs. For the remainder of this chapter and the next chapter this grammar stays fixed. Thus we drop all subscripts $G$.

In lectures we take our time at two places: i) explaining that the range of types cannot be defined by the obvious straightforward induction, because struct types $t$ can contain components pointing to variables of type $t$; the standard example is linked lists; ii) introducing the components of C0 configurations. Once this is done,
expression evaluation and statement execution can be treated with amazing speed in the classroom. Most lemmas belonging to the theory of type correctness are just stated in the classroom, because their proofs boil down to bookkeeping. The only exception is in Sect. 11.4.7, where the absence of dangling pointers is derived from the software condition that ‘address of’ is never taken for variables (or subvariables) on the stack.

11.1 Grammar of C0

In order to give the reader some overview of the language we proceed first with a brief, preliminary, and quite informal discussion of the grammar. The language uses well-known keywords like \texttt{int}, \texttt{typedef}, etc. We treat these as single symbols of the terminal alphabet. Similarly, certain mathematical operators such as \texttt{! =} or \texttt{&&} are treated like single symbols of the terminal alphabet.

11.1.1 Names and Constants

Symbol \( \langle Di \rangle \) generates decimal digits

\[
L(Di) = \{0, \ldots, 9\}
\]

and symbol \( \langle DiS \rangle \) generates sequences of digits

\[
L(DiS) = \{0, \ldots, 9\}^+.
\]

Symbol \( \langle Le \rangle \) generates small and capital letters as well as the symbol ‘.’

\[
L(Le) = \{a, \ldots, z, A, \ldots, Z, .\}.
\]

Symbol \( \langle DiLe \rangle \) generates digits, small, and capital letters including the symbol ‘.’

\[
L(DiLe) = \{0, \ldots, 9, a, \ldots, z, A, \ldots, Z, .\}.
\]

Symbol \( \langle DiLeS \rangle \) generates sequences of such symbols

\[
L(DiLeS) = \{0, \ldots, 9, a, \ldots, z, A, \ldots, Z, .\}^+.
\]

Names are generated by symbol \( \langle Na \rangle \). They are sequences of digits and letters starting with a letter

\[
L(Na) = L(Le) \circ \{0, \ldots, 9, a, \ldots, z, A, \ldots, Z, .\}^*.
\]

Constants are generated by symbol \( \langle C \rangle \). They are decimal representations of numbers, possibly followed by the letter \( u \), or the symbolic value \texttt{null} representing the content of uninitialized pointer variables (null pointer constant)

\[
L(C) = L(Di)^+ \cup (L(Di)^+ \circ \{u\}) \cup \{\texttt{null}\}.
\]