WIKIPEDIA

Statement (computer science)

In computer programming, a **statement** is the smallest standalone element of an imperative programming language that expresses some action to be carried out. It is an instruction written in a high-level language that commands the computer to perform a specified action.[1] A program written in such a language is formed by a sequence of one or more statements. A statement may have internal components (e.g., expressions).

Many languages (e.g. C) make a distinction between statements and definitions, with a statement only containing executable code and a definition instantiating an identifier, while an expression evaluates to a value only. A distinction can also be made between simple and compound statements; the latter may contain statements as components.

Contents

Kinds of statements

The following are some of the major generic kinds of statements, pseudocoded. These pseudocoded statements have an immediate corresponding syntax in any typical imperative language like Pascal, C, Fortran etc.:

Simple statements

- **assertion**
  - in C: `assert(ptr != NULL);`
- **assignment**
  - in Pascal: `A := A + 5`
  - in C and Fortran: `A = A + 5`
- **goto**
  - in Fortran: `goto label;`
- **return**
  - in C, Pascal and Fortran: `return value;`

- **call**
  - in C and Pascal: **`CLEARSCREEN()`**
  - in Fortran: **`call CLEARSCREEN()`**

Compound statements

- **block**:
  - in Pascal: **`begin ... end`**
  - in C: **`{ ... }`**
- **do-loop**: `do { computation(&i); } while (i < 10);`
- **for-loop**: `for A := 1 to 10 do WRITELN(A) end`
- **if-statement**: `if A > 3 then WRITELN(A) else WRITELN("NOT YET"); end`
- **switch-statement**: `switch (c) { case 'a': alert(); break; case 'q': quit(); break; }`
- **while-loop**: `while NOT EOF do begin READLN end`
- **with-statement**: `with open(filename) as f: use(f)`

Syntax

The appearance of statements shapes the look of programs. Programming languages are characterized by the type of statements they use (e.g. the curly brace language family). Many statements are introduced by identifiers like *if*, *while* or *repeat*. Often statement keywords are reserved such that they cannot be used as names of variables or functions. Imperative languages typically use special syntax for each statement, which looks quite different from function calls. Common methods to describe the syntax of statements are Backus–Naur form and syntax diagrams.

Semantics

Semantically many statements differ from subroutine calls by their handling of parameters. Usually an actual subroutine parameter is evaluated once before the subroutine is called. This contrasts to many statement parameters that can be evaluated several times (e.g. the condition of a while loop) or not at all (e.g. the loop body of a while loop). Technically such statement parameters are call-by-name parameters. Call-by-name parameters are evaluated when needed (see also lazy evaluation). When call-by-name parameters are available a statement like behaviour can be implemented with subroutines (see Lisp). For languages without call-by-name parameters the semantic description of a loop or conditional is usually beyond the capabilities of the language. Therefore, standard documents often refer to semantic descriptions in natural language.

Expressions

In most languages, statements contrast with expressions in that statements do not return results and are executed solely for their side effects, while expressions always return a result and often do not have side effects at all. Among imperative programming languages, Algol 68 is one of the few in which a statement can return a result. In languages that mix imperative and functional styles, such as the Lisp family, the distinction between expressions and statements is not made: even expressions executed in sequential contexts solely for their side effects and whose return values are not used are considered 'expressions'. In purely functional programming, there are no statements; everything is an expression.

This distinction is frequently observed in wording: a statement is *executed*, while an expression is *evaluated*. This is found in the `exec` and `eval` functions found in some languages: in Python both are found, with `exec` applied to statements and `eval` applied to expressions.

Extensibility

Most languages have a fixed set of statements defined by the language, but there have been experiments with extensible languages that allow the programmer to define new statements.

See also

- Comparison of Programming Languages - Statements
- Control flow
- Expression (contrast)
- Extensible languages

References

1. "statement" (http://www.webopedia.com/TERM/S/statement.html). webopedia. Retrieved 2015-03-03.

External links

- PC ENCYCLOPEDIA: Definition of: program statement (https://www.pcmag.com/encyclopedia_term/0,2542,t=program+statement&i=49804,00.asp)