The Go Programming Language Specification

Language version go1.25 (Feb 25, 2025)

Table of Contents

Introduction

This is the reference manual for the Go programming language. For more information and other documents, see go.dev.

Go is a general-purpose language designed with systems programming in mind. It is strongly typed and garbage-collected and has explicit support for concurrent programming. Programs are constructed from *packages*, whose properties allow efficient management of dependencies.

The syntax is compact and simple to parse, allowing for easy analysis by automatic tools such as integrated development environments.

Notation

The syntax is specified using a variant of Extended Backus-Naur Form (EBNF):

```
Syntax      = { Production } .
Production  = production_name "=" [ Expression ] "." .
Expression  = Term { "|" Term } .
```

```
Term        = Factor { Factor } .
Factor      = production_name | token [ "…" token ] | Group | Option | Repetition
Group       = "(" Expression ")" .
Option      = "[" Expression "]" .
Repetition  = "{" Expression "}" .
```

Productions are expressions constructed from terms and the following operators, in increasing precedence:

```
|   alternation
()  grouping
[]  option (0 or 1 times)
{}  repetition (0 to n times)
```

Lowercase production names are used to identify lexical (terminal) tokens. Non-terminals are in CamelCase. Lexical tokens are enclosed in double quotes "" or back quotes ``.

The form a … b represents the set of characters from a through b as alternatives. The horizontal ellipsis … is also used elsewhere in the spec to informally denote various enumerations or code snippets that are not further specified. The character … (as opposed to the three characters `...`) is not a token of the Go language.

A link of the form [Go 1.xx] indicates that a described language feature (or some aspect of it) was changed or added with language version 1.xx and thus requires at minimum that language version to build. For details, see the linked section in the appendix.

Source code representation

Source code is Unicode text encoded in UTF-8. The text is not canonicalized, so a single accented code point is distinct from the same character constructed from combining an accent and a letter; those are treated as two code points. For simplicity, this document will use the unqualified term *character* to refer to a Unicode code point in the source text.

Each code point is distinct; for instance, uppercase and lowercase letters are different characters.

Implementation restriction: For compatibility with other tools, a compiler may disallow the NUL character (U+0000) in the source text.

Implementation restriction: For compatibility with other tools, a compiler may ignore a UTF-8-encoded byte order mark (U+FEFF) if it is the first Unicode code point in the source text. A byte order mark may be disallowed anywhere else in the source.

Characters

The following terms are used to denote specific Unicode character categories:

```
newline        = /* the Unicode code point U+000A */ .
unicode_char   = /* an arbitrary Unicode code point except newline */ .
unicode_letter = /* a Unicode code point categorized as "Letter" */ .
unicode_digit  = /* a Unicode code point categorized as "Number, decimal digit" *
```

In The Unicode Standard 8.0, Section 4.5 "General Category" defines a set of character categories. Go treats all characters in any of the Letter categories Lu, Ll, Lt, Lm, or Lo as Unicode letters, and those in the Number category Nd as Unicode digits.

Letters and digits

The underscore character _ (U+005F) is considered a lowercase letter.

```
letter        = unicode_letter | "_" .
decimal_digit = "0" … "9" .
binary_digit  = "0" | "1" .
octal_digit   = "0" … "7" .
hex_digit     = "0" … "9" | "A" … "F" | "a" … "f" .
```

Lexical elements

Comments

Comments serve as program documentation. There are two forms:

1. *Line comments* start with the character sequence `//` and stop at the end of the line.
2. *General comments* start with the character sequence `/*` and stop with the first subsequent character sequence `*/`.

A comment cannot start inside a rune or string literal, or inside a comment. A general comment containing no newlines acts like a space. Any other comment acts like a newline.

Tokens

Tokens form the vocabulary of the Go language. There are four classes: *identifiers*, *keywords*, *operators and punctuation*, and *literals*. *White space*, formed from spaces (U+0020), horizontal tabs (U+0009), carriage returns (U+000D), and newlines (U+000A), is ignored except as it separates tokens that would otherwise combine into a single token. Also, a newline or end of file may trigger the insertion of a semicolon. While breaking the input into tokens, the next token is the longest sequence of characters that form a valid token.

Semicolons