

**INTRODUCTION**

1.Go language is a programming language initially developed at Google in the year **2007** by **Robert Griesemer, Rob Pike, and Ken Thompson.**

2. **Statically-typed** language

3. It provides **garbage collection,** **type safety, dynamic-typing capability, many advanced built-in types such as variable length arrays and key-value maps**. It also provides a rich standard library

4. **Launched in November 2009.**

5. Used in some of the **Google's production systems.**

6.Designed with **systems programming** in mind.

7. Programs are constructed using packages, for **efficient management of dependencies.**

8. Go programming implementations use a **traditional compile and link model to generate executable binaries**.

**FEATURES**

1. **Dynamic** languages.

2. **Compilation** time is **fast**.

3. Inbuilt concurrency support: **lightweight processes** (via go routines), channels, select statement.

4. Go programs are simple, concise, and safe.

5. Support for **Interfaces** and **Type embedding.**

6. Production of statically linked native binaries without external dependencies.

**NOT SUPPORTED FEATURES**

1. Support for type inheritance

2. Support for method or operator overloading

3. Support for circular dependencies among packages

4. Support for pointer arithmetic

5. Support for assertions

6. Support for generic programming

**GO PROGRAM STRUCTURE**

1.A Go program basically consists of the following parts

* Package Declaration
* Import Packages
* Functions
* Variables
* Statements and Expressions
* Comments

**EXAMPLE**

package main

import "fmt"

func main() {

/\* This is my first sample program. \*/

fmt.Println("Hello, World!")

}

**Package**

1. Define Package Name.

2. Package Statement **mendatory ,** Bcs Go Programs **Run in Packages**

3. The **main** package is the **starting point to run the program**.

4. Each package has a path and name associated with it.

**Import**

1.import "fmt" is a **preprocessor command** which tells the Go compiler to **include the files** lying in the **package fmt.**

**func main()**

1. main function where the **program** **execution begins.**

**Comments /\* \*/ AND //**

1. **/\*\*/** For Multiline Comments

2. **//** For Single Line Comment

**fmt.Println(val)**

1. fmt package has exported Println method which is used to display the message on the screen.

2.A name is **exported** if it starts with **capital letter**. Ex fmt.Println(val)

3. Exported means the **function or variable/constant is accessible to the importer of the respective package.**

**Tokens**

1.A token is either a keyword, an identifier, a constant, a string literal, or a symbol

**Line Separator**

1.the line separator key is a statement terminator.

2.he Go compiler internally places “;” as the statement terminator to indicate the end of one logical entity.

**Comments**

/\* my first program in Go \*/

1.You cannot have comments within comments and they do not occur within a string or character literals.

**Identifiers**

1. Identify a variable, function, or any other user-defined item.
2. An identifier starts with a letter A to Z or a to z or an underscore \_ followed by zero or more letters, underscores, and digits (0 to 9).
3. @, $, and % not allowed
4. Go is a **case-sensitive** programming language.

**Keywords**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| break | default | func | interface | select |
| case | defer | Go | map | Struct |
| chan | Else | Goto | package | Switch |
| const | fallthrough | if | range | Type |
| continue | For | import | return | Var |

**Whitespace in Go**

Whitespace is the term used in Go to describe blanks, tabs, newline characters, and comments.

**DATA TYPES**

The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

**1.Boolean types**

**-** two predefined constants: (a) true (b) false

**2.Numeric types**

a) integer types b) floating point

**3.String types**

1.Its value is a sequence of bytes

2.Strings are immutable types that is once created, it is not possible to change the contents of a string.

3.The predeclared string type is string.

**4.Derived types**

(a) Pointer types

(b) Array types

(c) Structure types

(d) Union types

(e) Function types

f) Slice types

g) Interface types

h) Map types

i) Channel Types

**Integer Types**

**unsigned**

**1. uint8**

Unsigned 8-bit integers (0 to 255)

**2**. **uint16**

Unsigned 16-bit integers (0 to 65535)

**3. uint32**

Unsigned 32-bit integers (0 to 4294967295)

**4.uint64**

Unsigned 64-bit integers (0 to 18446744073709551615)

**Signed**

**1.int8**

Signed 8-bit integers (-128 to 127)

**2.int16**

Signed 16-bit integers (-32768 to 32767)

**3.int32**

Signed 32-bit integers (-2147483648 to 2147483647)

**4.int64**

Signed 64-bit integers (-9223372036854775808 to 9223372036854775807)

**Floating Types**

**1.float32**

IEEE-754 32-bit floating-point numbers

**2.float64**

IEEE-754 64-bit floating-point numbers

**3.complex64**

Complex numbers with float32 real and imaginary parts

**4.complex128**

Complex numbers with float64 real and imaginary parts

**Other Numeric Types**

**1.byte**

same as uint8

**2.rune**

same as int32

**3.uint**

32 or 64 bits

**4.int**

same size as uint

**5.uintptr**

an unsigned integer to store the uninterpreted bits of a pointer value

**VARIABLES**

1.Each variable in Go has a specific type, which determines the size and layout of the variable's memory, the range of values that can be stored within that memory, and the set of operations that can be applied to the variable.

var variable\_list optional\_data\_type;

2**.optional\_data\_type** is a valid Go data type including byte, int, float32, complex64, boolean or any user-defined object, etc., and **variable\_list** may consist of one or more identifier names separated by commas.

var i, j, k int;

var c, ch byte;

var f, salary float32;

d = 42;

3.The type of variable is automatically judged by the compiler based on the value passed to it

variable\_name = value;

4.variables with static storage duration are implicitly initialized with nil (all bytes have the value 0);

5.the initial value of all other variables is **zero** value of their data type.

**Static Type Declaration in Go**

1.A static type variable declaration **provides assurance to the compiler that there is one variable available with the given type and name so that the compiler can proceed for further compilation** without requiring the complete detail of the variable.

package main

import "fmt"

func main() {

var x float64

x = 20.0

fmt.Println(x)

fmt.Printf("x is of type %T\n", x)

}

2.When the above code is compiled and executed, it produces the following result o/p : 20

x is of type float64

**Dynamic Type Declaration / Type Inference**

1.A dynamic type variable declaration requires the compiler to interpret the type of the variable based on the value passed to it.

2.The compiler does not require a variable to have type statically as a necessary requirement.

3.y := 42

in case of type inference, we initialized the variable **y** with **:=** operator,

package main

import "fmt"

func main() {

    var c = "I LOVE DANCE"

    love := fmt.Sprintf("%s", c)

    fmt.Println(love)

}

**The lvalues and the rvalues in Go**

**A) lvalue** −

1. Expressions that refer to a memory location is called "lvalue" expression.

2.An lvalue may appear as either the left-hand or right-hand side of an assignment.

3. Variables are lvalues and so may appear on the left-hand side of an assignment.

**B) rvalue** −

1. The term rvalue refers to a data value that is stored at some address in memory.

2.An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right- but not left-hand side of an assignment.

3. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side.

**VALID STATEMENT :**

x = 20.0

**INVALID STATEMENT :**

10 = 20.0

**CONSTANTS**

1.Constants refer to fixed values that the program may not alter during its execution.

2. Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are also enumeration constants as well.

3. Constants are treated just like regular variables except that their values cannot be modified after their definition

**Integer Literals**

1. An integer literal can be a decimal, octal, or hexadecimal constant

2. **0x or 0X** for **hexadecimal,eg.** 0xFeeL

3. **0 for octal**, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively.

eg.215u /\* Legal \*/

Following are other examples of various type of Integer literals −

85 /\* decimal \*/

0213 /\* octal \*/

0x4b /\* hexadecimal \*/

30 /\* int \*/

30u /\* unsigned int \*/

30l /\* long \*/

30ul /\* unsigned long \*/

**Floating-point Literals**

1.A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part.

2. You can represent floating point literals either in decimal form or exponential form.

3.The signed exponent is introduced by e or E.

3.14159 /\* Legal \*/

314159E-5L /\* Legal \*/

510E /\* Illegal: incomplete exponent \*/

210f /\* Illegal: no decimal or exponent \*/

.e55 /\* Illegal: missing integer or fraction \*/

**Escape Sequence**

1.When certain characters are preceded by a backslash, they will have a special meaning in Go.

2. These are known as Escape Sequence codes which are used to represent newline (\n), tab (\t), backspace, etc.

|  |  |
| --- | --- |
| \a | Alert or bell |
| \b | Backspace |
| \f | Form feed |
| \n | Newline |
| \r | Carriage return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \ooo | Octal number of one to three digits |
| \xhh . . . | Hexadecimal number of one or more digits |

**String Literals**

1.String literals or constants are enclosed in double quotes "".

2. You can break a long line into multiple lines using string literals and separating them using whitespaces.

"hello, \

dear"

**The *const* Keyword**

const variable type = value;