PES UNIVERSITY
CELEBRATING 50 YEARS

Go Programming Language

Program Structure

Department of Computer Science and Engineering

Go Programming Language

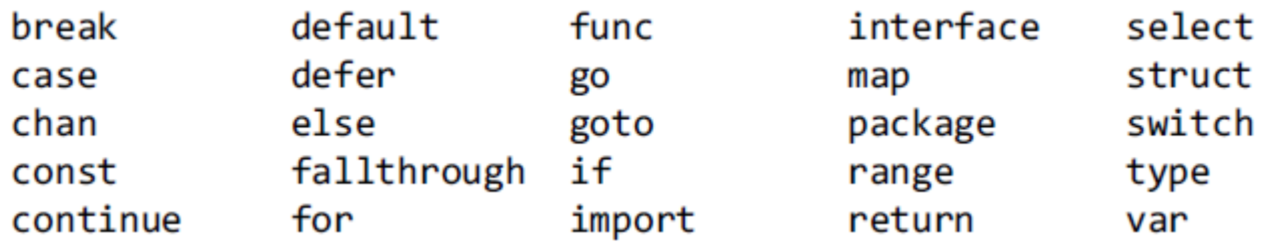
Program structure

- In Go, as in any other programming language, a program is built from a small set of basic constructs.
- **Variables** store values.
- Simple **expressions** are combined into larger ones with **operations** like addition and subtraction.
- Basic types are collected into aggregates like **arrays** and **structs**.
- Expressions are used in statements whose execution order is determined by control-flow statements like if and for.
- Statements are grouped into **functions** for isolation and reuse.
- Functions are gathered into source files and **packages**.

Go Programming Language

Names

- The names of Go - functions, variables, constants, types, statement labels, and packages follow a simple rule
- A name begins with a letter or an underscore and may have any number of additional letters, digits, and underscores.
- Names are case sensitive. heapSort and Heapsort are different names.
- Keywords:

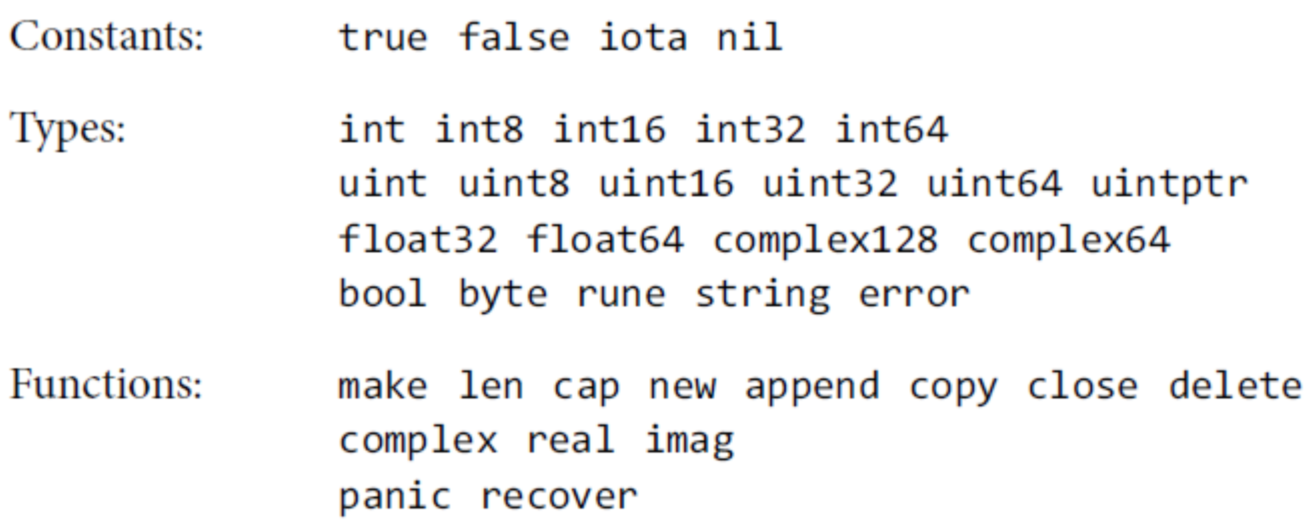


break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

- Keywords can't be used as names.

Go Programming Language

Predeclared Names



Constants:	`true false iota nil`

Types:	`int int8 int16 int32 int64`
	`uint uint8 uint16 uint32 uint64 uintptr`
	`float32 float64 complex128 complex64`
	`bool byte rune string error`

Functions:	`make len cap new append copy close delete`
	`complex real imag`
	`panic recover`

- If an entity is declared within a function, it is local to that function.
- If an entity is declared outside of a function, it is visible in all files of the package to which it belongs.
- The case of the first letter of a name determines its visibility across package boundaries.
- If the name begins with an upper-case letter, it is exported, which means that it is visible and accessible outside of its own package and may be referred to by other parts of the program
- Package names themselves are always in lower case

Go Programming Language
Variables

- A **var** declaration creates a variable of a particular type, attaches a name to it, and sets its initial value.
- Each declaration has the general form **var name type = expression**
- Either the **type** or the **= expression** part may be omitted, but not both
- If the type is omitted, it is determined by the initializer expression.
- If the expression is omitted, the initial value is the zero value for the type, which is 0 for numbers, false for booleans, "" for strings, and nil for interfaces and reference types (slice, pointer, map, channel, function).
- The **zero value** of an aggregate type like an **array** or a **struct** has the zero value of all of its elements or fields.
- The zero-value mechanism ensures that a variable always holds a well-defined value of its type
- In Go there is no such thing as an uninitialized variable
- Within a function, a variable may be declared using the form **name := expression**
- Note that **:=** is a declaration, whereas **=** is an assignment

Go Programming Language

Pointers

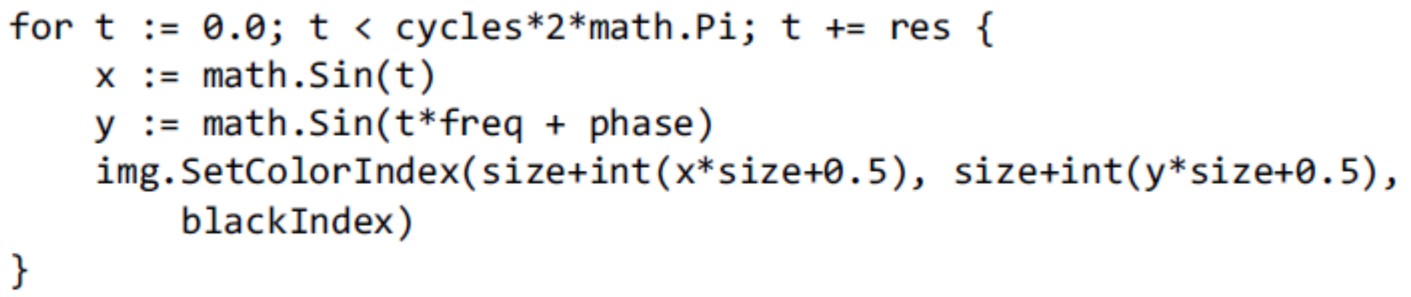
- A pointer value is the address of a variable.
- A pointer is thus the location at which a value of the variable is stored.
- Not every value has an address, but every variable does.
- With a pointer, we can read or update the value of a variable indirectly, without using or even knowing the name of the variable.
- If a variable is declared **var x int**, the expression &x (‘‘address of x’’) yields a pointer to an
- integer variable x
- The zero value for a pointer of any type is nil. The test p != nil is true if p points to a variable.
- Pointers are comparable. Two pointers are equal if and only if they point to the same variable or both are nil.
- Because a pointer contains the address of a variable, passing a pointer argument to a function makes it possible for the function to update the variable that was indirectly passed.

new Function

- Another way to create a variable is to use the built-in function new.
- The expression **new(T)** creates an unnamed variable of type T, initializes it to the zero value of T, and returns its address, which is a value of type *T.
- A variable created with new is no different from an ordinary local variable whose address is taken, except that there’s no need to invent (and declare) a dummy name, and we can use new(T) in an expression.
- Thus new is only a syntactic convenience, not a fundamental notion

Go Programming Language
Lifetime of variables

- The lifetime of a variable is the interval of time during which it exists as the program executes.
- The lifetime of a package-level variable is the entire execution of the program.
- Local variables have dynamic lifetimes: a new instance is created each time the declaration statement is executed, and the variable lives on until it becomes unreachable, at which point its storage may be recycled.
- Function parameters and results are local variables too; they are created each time their enclosing function is called.



```
for t := 0.0; t < cycles*2*math.Pi; t += res {
    x := math.Sin(t)
    y := math.Sin(t*freq + phase)
    img.SetColorIndex(size+int(x*size+0.5), size+int(y*size+0.5),
        blackIndex)
}
```

- The va x and y are created on each iteration of the loop.

Go Programming Language

Assignment

- The value held by a variable is updated by an assignment statement, which in its simplest form
- has a variable on the left of the **=** sign and an expression on the right.
- An assignment, explicit or implicit, is always legal if the left-hand side (the variable) and the right-hand side (the value) have the same type.
- The assignment is legal only if the value is assignable to the type of the variable.
- Tuple assignment, allows several variables to be assigned at once.

Go Programming Language

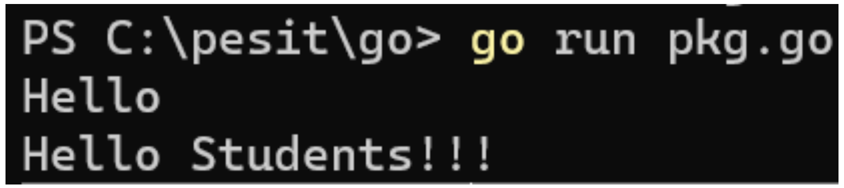
Type declaration

- The type of a variable or expression defines the characteristics of the values it may take on, such as their size, how they are represented internally, the intrinsic operations that can be performed on them, and the methods associated with them.
- A type declaration defines a new named type that has the same underlying type as an existing type.
- **type name underlying_type**
- Type declarations most often appear at package level, where the named type is visible throughout the package

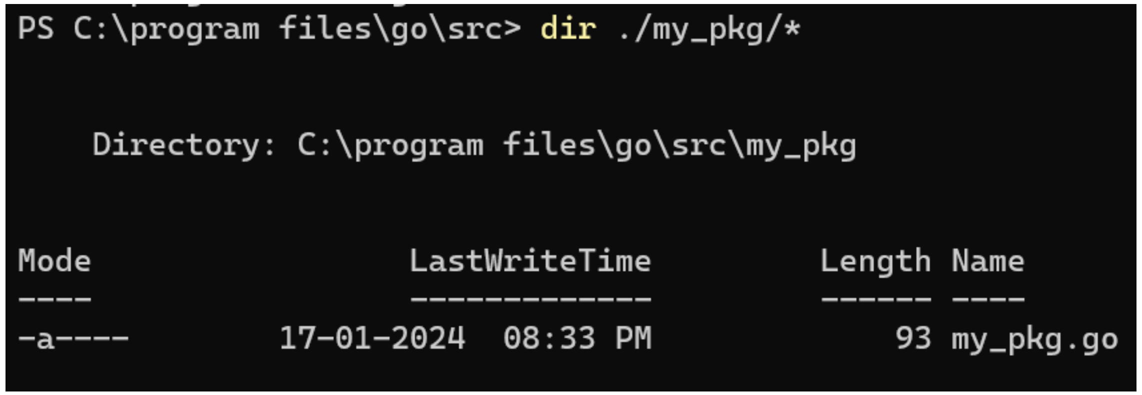
Go Programming Language

Packages and Files

- Packages in Go serve the same purposes as libraries or modules in other languages, supporting modularity, encapsulation, separate compilation, and reuse.
- The source code for a package resides in one or more .go files, usually in a directory whose name ends with the import path
- Each package serves as a separate name space for its declarations
- To refer to a function from outside its package, we must qualify the identifier to make explicit reference.



```
PS C:\pesit\go> go run pkg.go
Hello
Hello Students!!!
```



```
PS C:\program files\go\src> dir ./my_pkg/*


    Directory: C:\program files\go\src\my_pkg


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        17-01-2024   08:33 PM             93 my_pkg.go
```

Go Programming Language

Scope

- The scope of a declaration is a region of the program text
- It is a compile-time property
- A **syntactic block** is a sequence of statements enclosed in braces like those that surround the body of a function or loop.
- A name declared inside a syntactic block is not visible outside that block.
- Other groupings of declarations that are not explicitly surrounded by braces in the source code are called **lexical blocks**
- Lexical block for the entire source code is called the **universe block**
- The declarations of built-in types, functions, and constants like int, len, and true are in the universe
- block and can be referred to throughout the entire program.

Scope

- A program may contain multiple declarations of the same name so long as each declaration is in a different lexical block.
- When the compiler encounters a reference to a name, it looks for a declaration, starting with the inner most enclosing lexical block and working up to the universe block.
- If the compiler finds no declaration, it reports an ‘‘undeclared name’’ error.
- If a name is declared in both an outer block and an inner block, the inner declaration will be found first.

PES UNIVERSITY
CELEBRATING 50 YEARS

THANK YOU

Suresh Jamadagni

Department of Computer Science and Engineering