MAVENIR™
Trust the Future

Golang Training Pointers

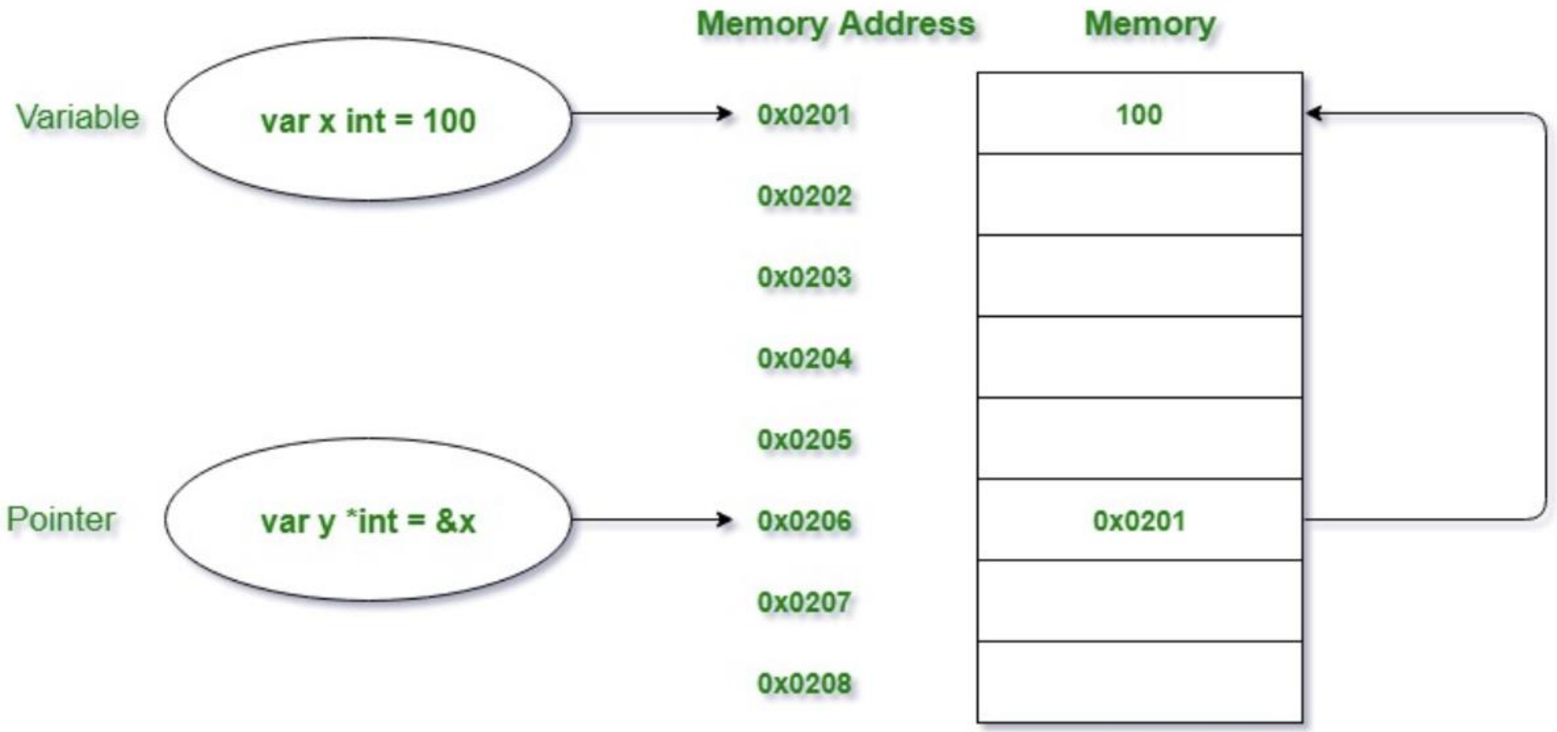
24/01/2023

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Pointer

Overview

> Pointer is a variable that stores the address of another variable



Golang: Pointer

Declaring pointer

> Pointer to a type is declared by prefixing * to type

E.g.

```
 a := 10

 var intPtr *int // Integer pointer

 intPtr = &a.  // & return reference(address) of a

b:= "Hello"

var stringPtr *string// string pointer

stringPtr = &b //& return reference of b
```

MAVENIR

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Pointer

Dereferencing Pointer

```
a := 10

var intPtr  *int // Integer pointer

intPtr = &a.  // & return reference(address) of a

c := *intPtr // dereferencing pointer

b:= "Hello"

var stringPtr *string// string pointer

stringPtr = &b //& return reference of b

d := *stringPtr // dereferencing pointer
```

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Pointer

Important Note

> Pointer arithmetic is not possible (C and C++ Supports Pointer arithmetic)

> Pointers are special variables in Golang

> '&' // referencing operator

> '*' // dereferencing operator

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Array

- An array is collection of objects of same type
- Array has a fixed size/length
- Array can neither expand not shrink
- Array is specified by prefixing size in square brackets before the type.

var array [size]type

Eg. var arrayInt [10]int

var arrayString [10] string

Golang: Array

- Initializing array with array literal
- Initializing specific elements of array
- Copying Array
  - By value
  - By reference
- len function and range operator

Copyright Mavenir 2022. Proprietary and Confidential.

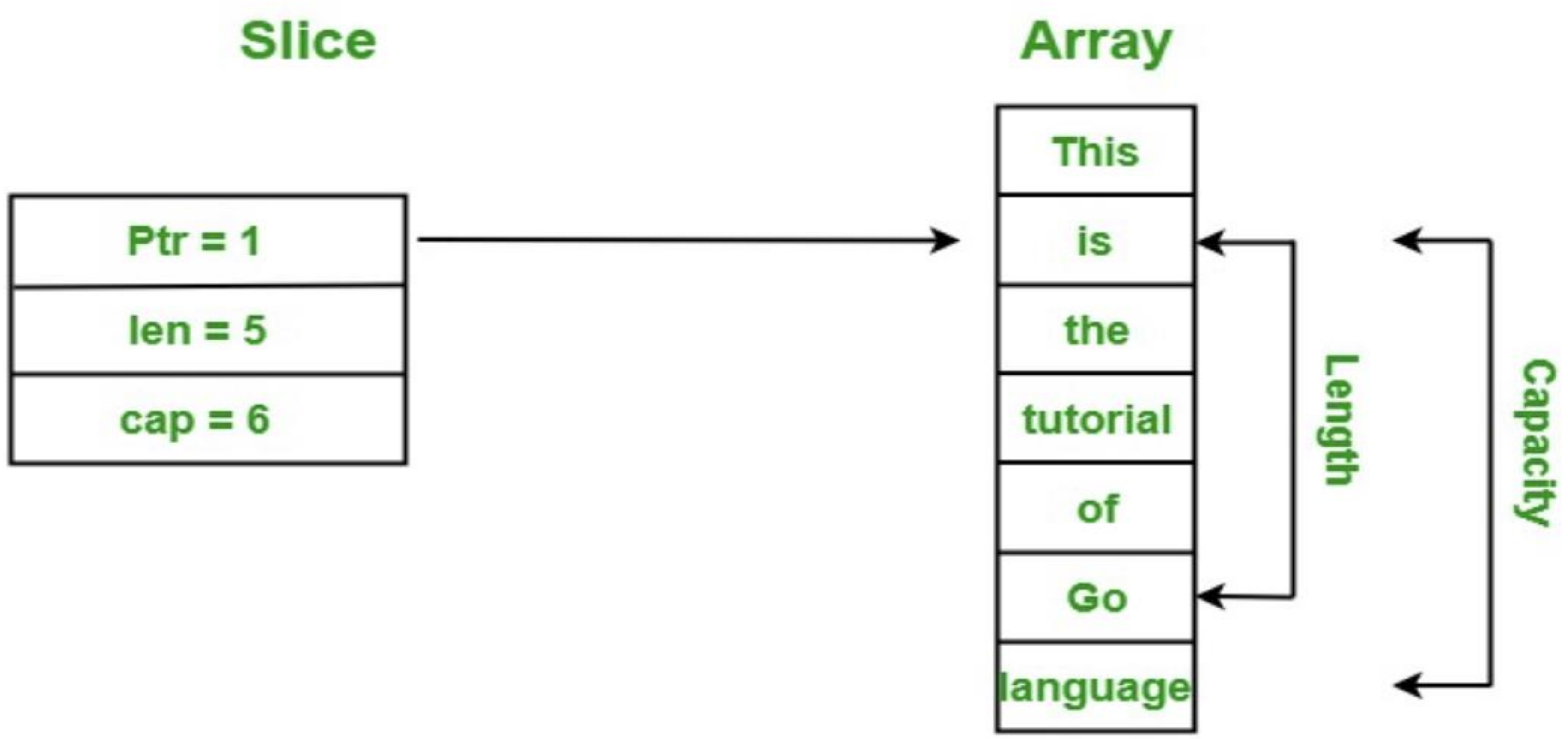
Golang: Slice

Overview

- Slice is a variable-length sequence which stores elements of a similar type
- Slice is a lightweight data structure
- Slice contains 3 elements
  - **Pointer:** The pointer is used to points to the first element of the array that is accessible through the slice
  - **Length:** The length is the total number of elements present in the array.
  - **Capacity:** The capacity represents the maximum size upto which it can expand.

Copyright Mavenir 2022. Proprietary and Confidential.

Golang:Slice



Golang:Slice

Declaring slice

Slice can be declared like an array, however size is not specified

```
var my_slice_1 = []string{"Mavenir", "Golang", "Training"}
```

Using make function to create slice

```
func make([]T, len, cap) []T
```

- Slice can be created using existing array

```
array_name[lowIdx:highIdx]
```

- Slice can be created using existing slice

```
slice_name[low:high]
```

- Declare Slice using new Keyword

```
var intSlice = new([50]type)[0:10]
```

- Slice can be iterated like a array

Copyright Mavenir 2022. Proprietary and Confidential.

Golang:Slice

Important points

- **Zero value slice:** In Go language, you are allowed to create a nil slice that does not contain any element in it

  var myslice []**string**

- **Modifying Slice:**
  - slice is a reference type
  - changing some elements in the slice, will change content of underlying array
- **Comparison of Slice:** In Slice, you can only use **==** operator to check the given slice is nill or not
- **Multi-Dimensional Slice:** Multi-dimensional slice are just like the multidimensional array,
- **Sorting of Slice:** In Go language, you are allowed to sort the elements present in the slice using sort package

MAVENIR™

Copyright Mavenir 2022. Proprietary and Confidential.

Golang:Slice

Add/Remove/Copy

- To add an item to the end of the slice, use the append() method.
- Removing element at index

  append(s[:index], s[index+1:]...)
- Slice can be copied using copy()

Golang: Slice

Sort/Search/Reverse

- Sort package provides functions to sort/search/reverse
  - Ints function [ascending order]
  - func Strings(strSlice []string)
  - Float64s function [ascending order]
  - IntsAreSorted function
  - StringsAreSorted function
  - Float64sAreSorted function
  - SearchInts(intSlice []int, x int) int
  - SearchStrings(strSlice []string, x string) int
  - SearchFloat64s(fltSlice []float64, x float64) int
  - Reverse(data Interface) Interface
- Revisit sort search reverse after structures

MAVENIR™

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Slice

Important functions

- make() used to create slice
- append() used to append elements
- copy() used to copy
- len() returns length of slice
- range used as similar to arrays

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: structure

Overview

- A structure or struct in Golang is a user-defined type
- Groups items of possibly different types into a single type
- Any real-world entity which has some set of properties/fields can be represented as a struct.

Syntax

```
type <struct-name> struct {
    Property1 type
    Property2 type
}
```

Golang: Structure

Declaration

> Structure type variables can be declared as below

Eg.

```
 type Address struct {
    name, street, city, state string
    Pincode int
 }
var a Address = Address {name: "Sanjana S", street: "HSR", city: "Bangalore", state: "Karnatak", Pincode:560102 }
```

Struct Instantiation Using new Keyword

```
 addr := new(Address) // here addr is pointer to struct.
```

Copyright Mavenir 2022. Proprietary and Confidential.

Golang:Structe

Discussion points

> Declaration of a Struct Type

> Creating a Struct Instance Using a Struct Literal

> Struct Instantiation Using new Keyword

> Struct Instantiation Using Pointer Address Operator

> Nested Struct Type

> Comparing Structs with the Different Values Assigned to Data Fields

> Copy Struct Type Using Value and Pointer Reference

> Anonymous Structure and Fields

> Promoted Fields in Structure

>

Copyright Mavenir 2022. Proprietary and Confidential.

Golang:Functions

Overview

> Functions are generally the block of codes or statements

> In a program that gives the user the ability to reuse the same code

> saves the excessive use of memory

> provides better readability of the code

Syntax:

```
func function_name(Parameter-list)(Return_type){
   // function body.....
}
```

> **func:** It is a keyword in Go language, which is used to create a function.

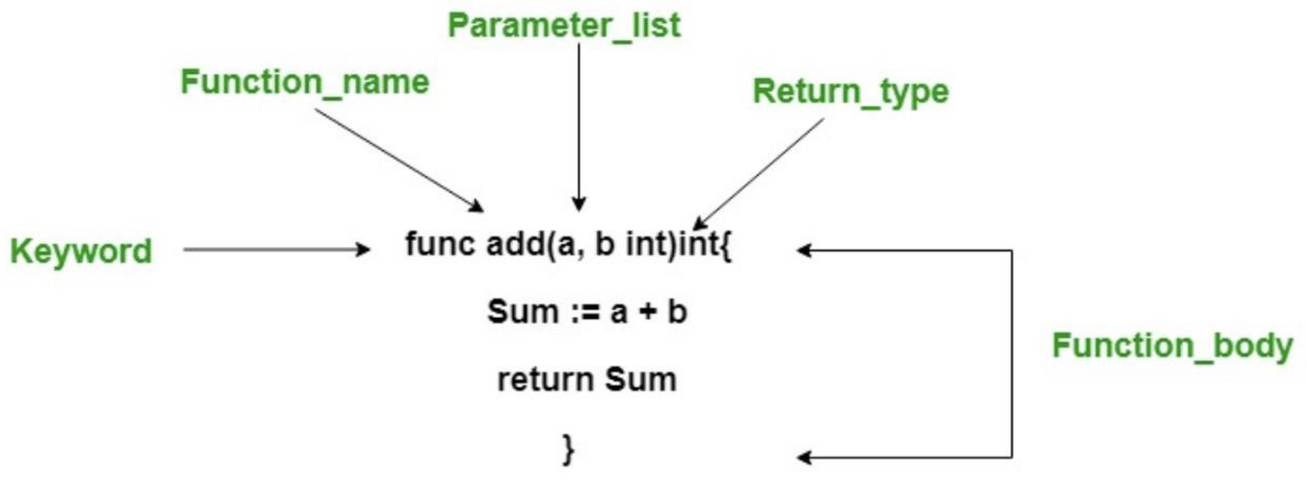
> **function_name:** It is the name of the function.

> **Parameter-list:** It contains the name and the type of the function parameters.

> **Return_type:** It is optional and it contain the types of the values that function returns. If you are using return_type in your function, then it is necessary to use a return statement in your function.

Copyright Mavenir 2022. Proprietary and Confidential.

Golang:Functions



```
func add(a, b int)int{
Sum := a + b
return Sum
}
```

Golang: Functions

Call by Value/Call by reference

> **Call by value**

- values of actual parameters are copied to function's formal parameters
- Both parameters are stored in different memory locations.
- Any changes made inside functions are not reflected in actual parameters of the caller.

> **Call by reference**

- Both the actual and formal parameters refer to the same locations
- Any changes made inside the function are actually reflected in actual parameters of the caller.

MAVENIR™

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Functions

Anonymous function

> Go language provides a special feature known as an anonymous function
>
> Anonymous function doesn't contain any name
>
> It can be compared with an inline function
>
> an anonymous function can form a closure
>
> An anonymous function is also known as ***function literal***

Syntax:

```
    func(parameter_list)(return_type){
      // code..


     // Use return statement if return_type are given
     // if return_type is not given, then do not
     // use return statement
     return
}()
```

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Functions

Anonymous Functions

- Anonymous function can be assigned to a variable.
  - type of the variable is of function type
  - Anonymous function can be called using assigned variable
- Can pass arguments in the anonymous function.
- Can pass an anonymous function as an argument into other function.
- Can return an anonymous function from another function.

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: functions

Function Returning Multiple Values

> It is allowed to return multiple values from a function

> Single return statement can return multiple values

Syntax

```
    func function_name(parameter_list)(return_type_list){
        // code...
    }
```

> **function_name:** It is the name of the function.

> **parameter-list:** It contains the name and the type of the function parameters.

> **return_type_list:** It is optional and it contains the types of the values that function returns. If you are using return_type in your function, then it is necessary to use a return statement in your function.

Golang: Functions

Name to the Return Values

> It is allowed provide names to the return values.

> Names of the return values can be used as variable names in your code

> And this type of return is known as the bare return

Syntax

```
 func function_name(para1, para2 int)(name1 int, name2 int){
     // code...
 }
func function_name(para1, para2 int)(name1, name2 int){
   // code...
}
```

Golang: Functions

Variadic arguments

> function that is called with the varying number of arguments

Eg.

fmt.Printf, fmt.Println

> The type of the last parameter is preceded by an ellipsis

Syntax

```
function function_name(para1, para2...type)type {
// code...
}
```

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Functions

When to use variadic functions

> Variadic functions can be used instead of passing a slice to a function.
>
> Variadic function is used when we don’t know the number of parameters.
>
> It increases the readability of your program.

Golang: Functions

Topics to cover

> Array as argument to function

> Returning Array

> Slice as argument to function

> Returning slice

> Structure as argument

> Returning structures

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Functions

Methods

- Go methods are similar to Go function with one addition of a receiver argument in it
- Method can access the properties of the receiver
- Receiver can be of struct type or non-struct type

Syntax:

```
func(reciver_name Type) method_name(parameter_list)(return_type){
  // Code
}
```

- Method with Non-Struct Type Receiver
- Method with struct type receiver
- Methods with Pointer Receiver

MAVENIR™

Copyright Mavenir 2022. Proprietary and Confidential.

Golang: Functions

Methods with same name

> it is allowed to create two or more methods with the same name in the same package

> receiver of these methods must be of different types

Syntax:

```
func(reciver_name_1 Type) method_name(parameter_list)(return_type){
// Code
}

func(reciver_name_2 Type) method_name(parameter_list)(return_type){
// Code
}
```

Golang: Functions

defer

- defer statements delay the execution of the
  - function
  - method
  - anonymous method
- defer function or method call arguments evaluate instantly
- Multiple defer statements are allowed in the same program and they are executed in LIFO(Last-In, First-Out)
- In the defer statements, the arguments are evaluated when the defer statement is executed, not when it is called.
- Defer statements are generally used to ensure the cleanup

Copyright Mavenir 2022. Proprietary and Confidential.