Section 3 (Pointers)

1- What is pointer?

A pointer is a variable that stores the address it points to. A pointer of a specific type can only point to that type.

2- GoLang pointer syntax

```
var ptr *type
var ptrint *int     // pointer to an int
```

3- Pointer initialization

```
package main

import (
    "fmt"
)

func main() {
    var q int = 42
    var p *int     // declare the pointer
    p = &q         // initialize the pointer
    fmt.Println(p)  // 0x40e020
}
```

4- Pointer dereferencing

Dereferencing a pointer means getting the value inside the address the pointer holds. If we have a memory address, we can dereference the pointer to that memory address to get the value inside it. Here is the same example showing the dereference operation using the star(*) operator.

```
func main() {
    var q int = 42
    var p *int
    p = &q
    fmt.Println(p)  // 0x40e020
    fmt.Println(*p) // 42
}
```

5- Pointer to pointer

A pointer variable can store even a pointers address since a pointer is also a variable just like others

```
func main() {
    i := 64
    j := &i  // j is pointer to an int
    k := &j  // k is pointer to a pointer to an int (pointer to anc

    fmt.Println(i)  // 64

    fmt.Println(j)  // 0x40e020

    fmt.Println(*j) // 64 (value inside that address)

    fmt.Println(k)  // 0x40c138

    fmt.Println(*k) // 0x40e020 (address of j)
}
```

6- Pointers as function argument

Pointers can be used in function arguments just like value. It has some advantages over using values directly. It is a very efficient way to pass large objects to function.

```
// declare pointer as argument
func f(a *int) {
    fmt.Println(*a)
}

func main() {
    var a int = 42

    // pass the address
    f(&a) // 42
}
```

Examples

- Access the memory address

```
// Program to illustrate how memory address works

package main
import "fmt"

func main() {

  var num int = 5
  // prints the value stored in variable
  fmt.Println("Variable Value:", num)

  // prints the address of the variable
  fmt.Println("Memory Address:", &num)

}
```

Output :

```
Variable Value: 5
Memory Address: 0xc000018030
```

- Pointer variables to store the memory address

```
var num int = 5

// create the pointer variable
var ptr *int = &num
```

we have created the pointer variable named ptr that stores the memory address of the num variable.

*int represents that the pointer variable is of int type (stores the memory address of int variable).

We can also create pointer variables of other types.

```
// pointer variable of string type
var ptr1 *string

// pointer variable of double type
var ptr2 * float32
```

- assign the memory address of a variable to a pointer variable.

```
// Program to assign memory address to pointer

package main
import "fmt"

func main() {

  var name = "John"
  var ptr *string

  // assign the memory address of name to the pointer
  ptr = &name

  fmt.Println("Value of pointer is", ptr)
  fmt.Println("Address of the variable", &name)

}
```

Output:

```
Value of pointer is 0xc00007c1c0
Address of the variable 0xc00007c1c0
```

In the above example, we have created a pointer variable named ptr of type string. Here, both the pointer variable and the address of the name variables are the same.

This is because the pointer ptr stores the memory address of the name variable.

ptr = &name

- Get value pointed by pointer

```
// Program to get the value pointed by a pointer

package main
import "fmt"

func main() {

  var name = "John"
  var ptr *string

  ptr = &name

  // * to get the value pointed by ptr
  fmt.Println(*ptr) // John

}
```

We use the * operator to access the value present in the memory address pointed by the pointer

Here, we have used the *ptr to access the value stored in the memory address pointed by the pointer.

Since the pointer stores the memory address of the name variable, we get the value "John" as output.

In the above example, ptr is a pointer, not *ptr.
You cannot and should not do something like *ptr = &name

The * is called the dereference operator (when working with pointers). It operates on a pointer and gives the value stored in that pointer.

General ex 1:

```go
package main
import "fmt"

func main() {
  var num  int
  var ptr *int

  num = 22
  fmt.Println("Address of num:",&num)
  fmt.Println("Value of num:",num)

  ptr = &num
  fmt.Println("\nAddress of pointer ptr:",ptr)
  fmt.Println("Content of pointer ptr:",*ptr)

  num = 11
  fmt.Println("\nAddress of pointer ptr:",ptr)
  fmt.Println("Content of pointer ptr:",*ptr)

  *ptr = 2
  fmt.Println("\nAddress of num:",&num)
  fmt.Println("Value of num:",num)
}
```

Output:

```
Address of num: 0xc000090020
Value of num: 22

Address of pointer ptr: 0xc000090020
Content of pointer ptr: 22

Address of pointer ptr: 0xc000090020
Content of pointer ptr: 11

Address of num: 0xc000090020
Value of num: 2
```

General ex 2:

```
// Program to pass pointer as a function argument

package main
import "fmt"

// function definition with a pointer argument
func update(num *int) {

  // dereference the pointer
  *num = 30

}

func main() {

  var number = 55

  // function call
  update(&number)

  fmt.Println("The number is", number)

}
```

Output:

```
The number is 30
```

Return pointer from function

```
// Program to return a pointer from a function

package main
import "fmt"

func main() {

  // function call
  result := display()
  fmt.Println("Welcome to", *result)

}

func display() *string {

  message := "Programiz"

  // returns the address of message
  return &message

}
```

Output:

```
Welcome to Programiz
```

Here, *string indicates that the function returns a pointer of string type.

Notice the return &message statement in the display() function.

This indicates that the function returns the address of the message variable to the main() function.

The returned address is assigned to the result pointer. To get the value stored in the memory address, we have used the code *result

Call by reference

- While passing pointers to a function, we are actually passing a reference (address) of the variable. Instead of working with the actual value, we are working with references like

- accessing value using reference
- changing value using reference

That's why this process of calling a function with pointers is called call by reference in Go.

```
// call by value
func callByValue(num int) {

  num = 30
  fmt.Println( num) // 30

}

// call by reference
func callByReference(num *int) {

  *num = 10
  fmt.Println(*num) // 10

}

func main() {

  var number int

  // passing value
  callByValue(number)

  // passing a reference (address)
  callByReference(&number)

}
```

Here, we have created two functions: callByValue() and callByReference().

In the callByValue() function, we are directly passing the number variable. Whereas in callByReference(), we are passing the memory address of number

Trace

```
package main

import (
    "fmt"
)
func main() {
    // storing the hexadecimal
    // values in variables
    x := 0xFF
    y := 0x9C
    // Displaying the values
    fmt.Printf("Type of variable x is %T\n", x)
    fmt.Printf("Value of x in hexadecimal is %X\n", x)
    fmt.Printf("Value of x in decimal is %v\n", x)

    fmt.Printf("Type of variable y is %T\n", y)
    fmt.Printf("Value of y in hexadecimal is %X\n", y)
    fmt.Printf("Value of y in decimal is %v\n", y)
}
```

```
package main
import (
    "fmt"
)
func main() {
    // taking a normal variable
    var x int = 5748
    // declaration of pointer
    var p *int
    // initialization of pointer
    p = &x
    // displaying the result
    fmt.Println("Value stored in x = ", x)
    fmt.Println("Address of x = ", &x)
    fmt.Println("Value stored in variable p = ", p)
}
```

```go
package main
import (
    "fmt"
)
func main() {

        // taking a pointer
        var s *int

        // displaying the result
        fmt.Println("s = ", s)
}
```

```go
package main
import (
    "fmt"
)
func main() {
// using var keyword
    // we are not defining
    // any type with variable
    var y = 458
    // taking a pointer variable using
    // var keyword without specifying
    // the type
    var p = &y
    fmt.Println("Value stored in y = ", y)
    fmt.Println("Address of y = ", &y)
    fmt.Println("Value stored in pointer variable p = ", p)
    // this is dereferencing a pointer
    // using * operator before a pointer
    // variable to access the value stored
    // at the variable at which it is pointing
    fmt.Println("Value stored in y(*p) = ", *p)

}
```

```go
package main
import (
    "fmt"
)
func main() {
// using var keyword
    // we are not defining
    // any type with variable
    var y = 458

    // taking a pointer variable using
    // var keyword without specifying
    // the type
    var p = &y

    fmt.Println("Value stored in y before changing = ", y)
    fmt.Println("Address of y = ", &y)
    fmt.Println("Value stored in pointer variable p = ", p)

    // this is dereferencing a pointer
    // using * operator before a pointer
    // variable to access the value stored
    // at the variable at which it is pointing
    fmt.Println("Value stored in y(*p) Before Changing = ", *p)

    // changing the value of y by assigning
    // the new value to the pointer
    *p = 500

     fmt.Println("Value stored in y(*p) after Changing = ",y)
}
```