Intelligent programming

Lecture two

Data Types in Go

In Go language, the type is divided into four categories which are as follows:

Basic type: Numbers, strings, and booleans

Derived types

1.Aggregate type: Array and structs

2. Reference type: Pointers, slices, maps, and functions

Data Types

- 1. Boolean types

Boolean type consists of the two predefined constants: (a) true (b) false

- 2. Numeric types

They are arithmetic types and they represents a) integer types or b) floating point values.

- 3. String types

A string type represents the set of string values. Its value is a sequence of bytes.

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

- 4.Derived types

They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types f) Slice types g) Interface types h) Map types i) Channel Types

Integers and float

int8	8-bit signed integer
int16	16-bit signed integer
int32	32-bit signed integer
int64	64-bit signed integer
float32	32-bit floating-point number
float64	64-bit floating-point number

Strings

- It is a sequence of characters where each and every character is represented by one or more bytes

```
import "fmt"

func main() {

      str1 := “name"

   var str2 string
   str2 = “age"

      fmt.Println( str1)
     fmt.Println( str2)
}
```

```
Output
name
age
```

Variable Definition in Go

A variable definition tells the compiler where and how much storage to create for the variable.

- A variable definition specifies a data type and contains a list of one or more variables of that type as follows −

- **var variable_list optional_data_type**
- var i, j, k int;
- var c, ch byte;
- var f, salary float32;

Variable initialization

- Variables can be initialized (assigned an initial value) in their declaration.

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

- The initializer consists of an equal sign followed by a constant expression

- var d = 3 // declaration of d Here d is int

Dynamic Type Declaration
Type Inference in Go

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

- y:=20

Printing type of 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)
}
```

Println( )

- Println ( ) function formats the string using the default formats for its operands. It add spaces and line breaker automatically

```
package main

import "fmt"

func main() {

    s := "Sam"
   age := 25
   fmt.Println("His name is", s)

   fmt.Println("His age is", age, "years")
}
```

```
His name is Sam
His age is 25 yearsh
```

Printf ( )

- Means “Print Formatter”, it prints formatted strings.
-
- It does not insert a new line at the end like Println ().

- For that, you’ll have to add “\n” in the end.

- It formats the string according to a format specifier.
- %s : string
- %d : decimal
- %f :floating point
- %T : type

Pointers in Go language

- Variables are the names given to a memory location where the actual data is stored.
- To access the stored data we need the address of that particular memory location.
- Variables can be accessed just by using their name.

Pointers in Go language

- Go programming tasks are performed more easily with pointers, and other tasks, such as **call by reference**, cannot be performed without using pointers.

- Every variable is a memory location and every memory location has its address defined which can be accessed using ampersand **(&)** operator, which denotes an address in memory.

Dealing with hexadecimal numbers

```
package main
import "fmt"
func main() {
    x := 0xFF
    y := 0x9C

    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)
}
```

X is a variable of type integer and its value in hexadecimal is FF

```
Type of variable x is int
Value of x in hexadecimal is FF
Value of x in decimal is 255

Type of variable y is int
Value of y in hexadecimal is 9C
Value of y in decimal is 156
```

Pointers in Go language

```
package main
import "fmt"
func main() {
   var a int = 10
   fmt.Printf("Address of a variable: %x\n", &a  )
}
```

```
Address of a variable: 10328000
```

Pointers for integers

var x int

If px is a pointer to x

Its value will be 1000

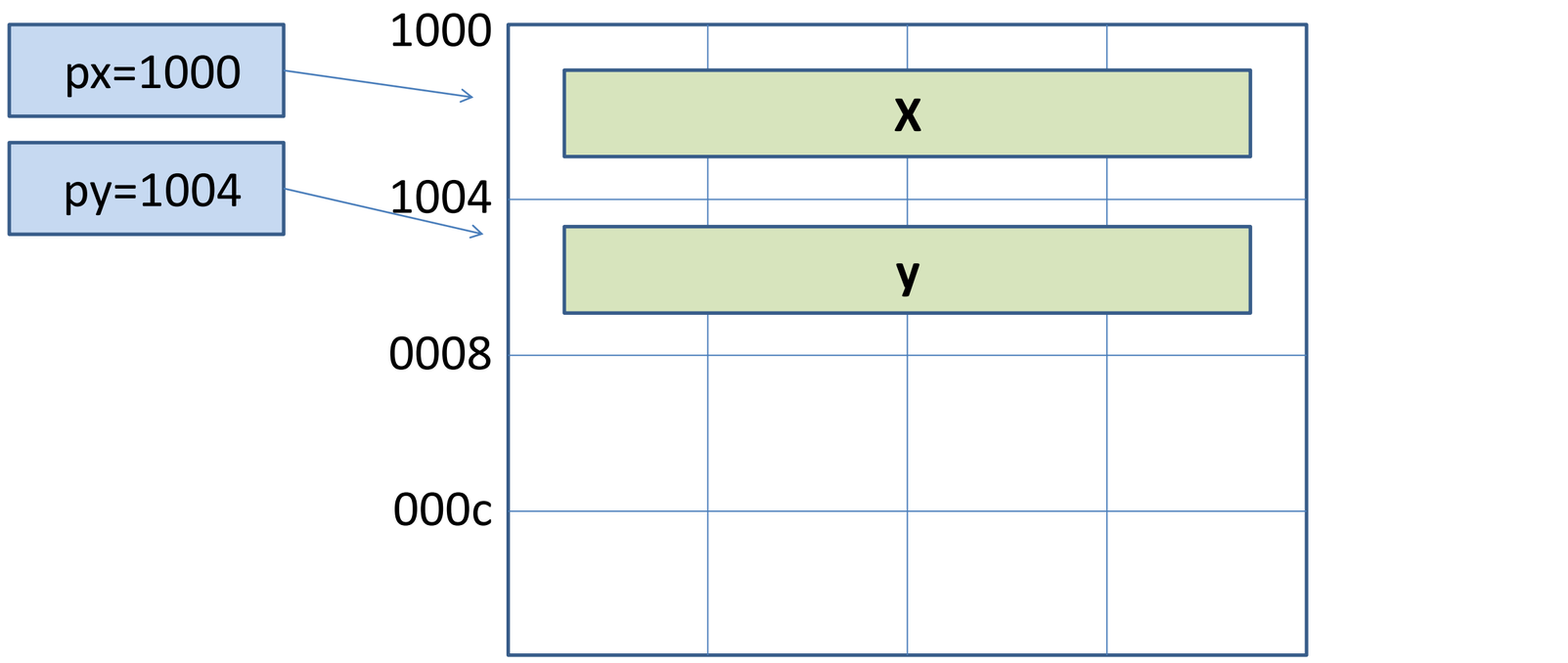
Px=1000

var y int

If py is a pointer to x

Its value will be 1004

Py=1004



Pointers in GO

- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.

- Like any variable, you must declare a pointer before you can use it to store any variable address.

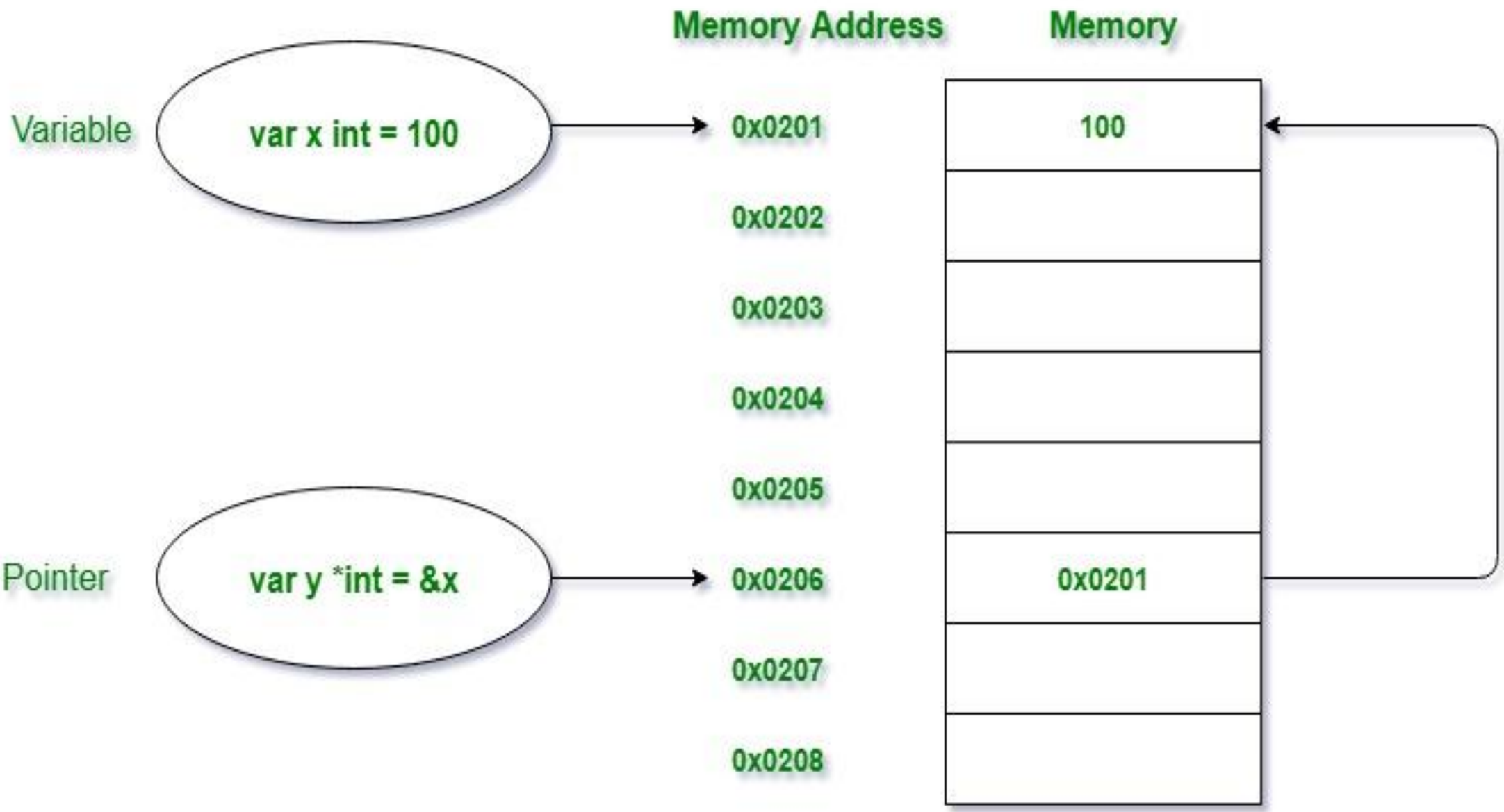
- The general form of a pointer variable declaration is –
- **var var_name *var-type**

Declaration of pointer

```
var x  int =100
var y *int = &x
```

We have two variables
The first
Name :x
Type : int
Value :100
The second
Name :y
Type: pointer to integer
Value : the memory address of variable x

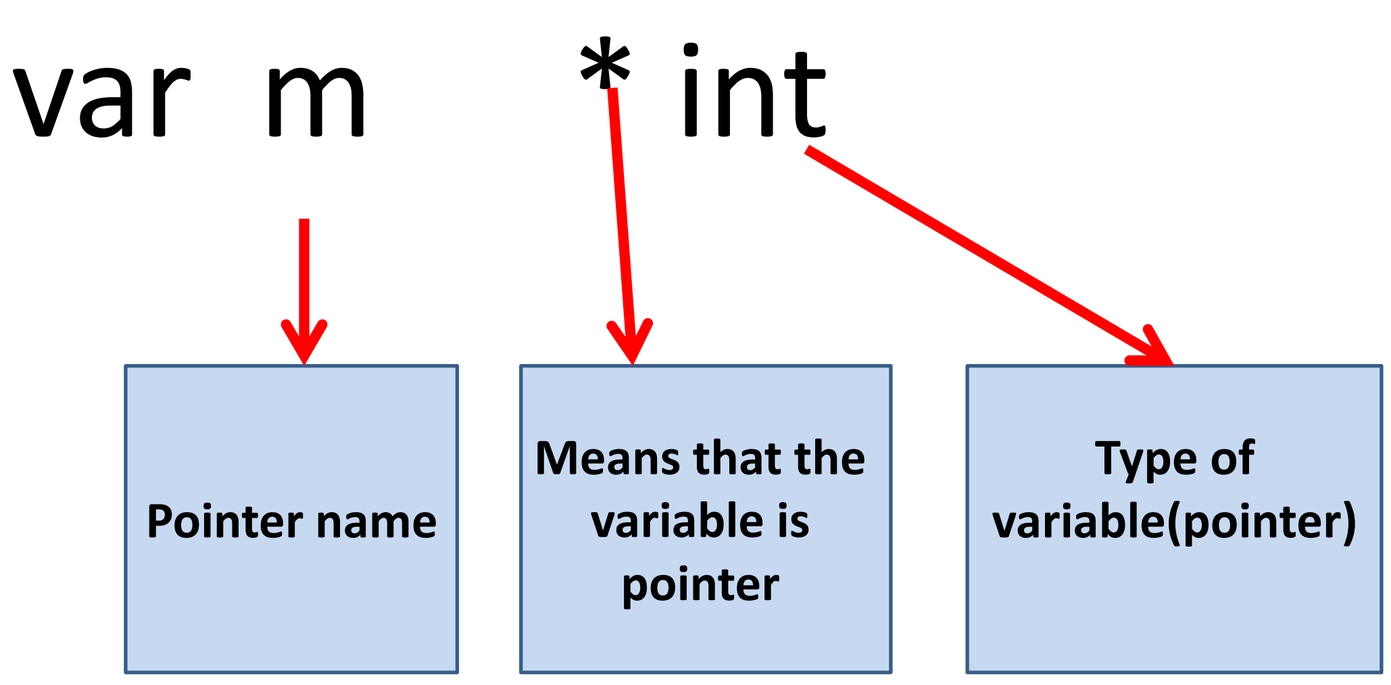
Pointers in Go language



Pointers declaration

Declaration

var pointer name *type of variable;



Pointers Declaration

Var p1 *int (p1 is a pointer to an integer)

Var p2 *float32 (p2 is a pointer to a float)

Here, **type** is the pointer's base type; it must be a valid Go data type

var-name is the name of the pointer variable.

The **asterisk *** used to declare a pointer.

Operators of pointers

- * **Operator** :dereferencing operator used to
- 1.Declare pointer variable
- 2.Access the value stored in the address.

- **& operator** : address operator used to
- 1. Returns the address of a variable or to access the address of a variable to a pointer.

```
package main
import "fmt"
func main() {
 var x int = 5748
 var p *int
    p = &x


     fmt.Println      (x)
    fmt.Println  ( &x)
    fmt.Println(p)
}
```

X is saved in 0x414024

Output:

5748
0x414024
0x414024

The default value or zero-value of a pointer is nil.

```
package main

import "fmt"

func main() {

    // taking a pointer
   var s *int

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

Output:

```
s =  <nil>
```

* operator returns the value at the address of.

```
package main

import "fmt"

func main() {

        var y = 458

        var p = &y

    fmt.Println( y)
    fmt.Println( &y)
    fmt.Println( p)
    fmt.Println(*p)

}
```

Y is saved in 0x414020

```
Output
458
0x414020
0x414020
458
```

```
func main() {

        var y = 458
        var p = &y

    fmt.Println (y)
    fmt.Println (&y)
    fmt.Println ( p)
    fmt.Println (*p)

     *p = 500
     fmt.Println (y)
    fmt.Println (&y)
    fmt.Println ( p)
    fmt.Println (*p)
}
```

Y is saved in 0x414020

```
Output
458
0x414020
0x414020
458


500
0x414020
0x414020
500
```