Intelligent programming

Lecture Five

type **structure Name** struct {

member definition

member definition

…

member definition

}

Structure in Go

First define the type of structure

```
type  Address  struct {
      name string
      street string
      city string
      country string
      Pincode int
}
```

```
type Books struct {
   title string
   author string
   subject string
   book_id int
}
func main() {
   var Book1 Books
   var Book2 Books
    Book1.title = “Intelligent Programming"
   Book1.author = " Kumar"
   Book1.subject = "Go Programming "
   Book1.book_id = 6452307
   Book2.title = "Telecom Billing"
   Book2.author = "Zara Ali"
   Book2.subject = "Telecom Billing Tutorial"
   Book2.book_id = 6495700
   printBook(Book1)
    printBook(Book2)
}
```

```
func  printBook ( book Books ) {

   fmt.Printf( "Book title : %s\n", book.title);

   fmt.Printf( "Book author : %s\n", book.author);

   fmt.Printf( "Book subject : %s\n", book.subject);

   fmt.Printf( "Book book_id : %d\n", book.book_id);

}
```

Pointers to Structures

You can define pointers to structures in the same way as you define pointer to any other variable

var **struct_pointer** *Books

Now, you can store the address of a structure variable in the above defined pointer variable.
To find the address of a structure variable, place the & operator before the structure's name as follows

struct_pointer = &Book1;

To access an member (titel)
struct_pointer.title;

```
type Books struct {
   title string
   author string
   subject string
   book_id int
}
func main() {
   var Book1 Books
    var Book2 Books
 Book1.title = “Intelligent Programming"
   Book1.author = "Kumar"
   Book1.subject = "Go Programming "
   Book1.book_id = 6495407
  Book2.title = "Telecom Billing"
   Book2.author = "Zara Ali"
   Book2.subject = "Telecom Billing Tutorial"
   Book2.book_id = 6495700
     printBook  (&Book1)
      printBook (&Book2)
}
```

```
func printBook( book *Books ) {

   fmt.Printf( "Book title : %s\n", book.title);

   fmt.Printf( "Book author : %s\n", book.author);

   fmt.Printf( "Book subject : %s\n", book.subject);

   fmt.Printf( "Book book_id : %d\n", book.book_id);

}
```

Define an employee structure containing first name and last name as strings and age, salary as int the define a pointer to structure.
Define a pointer of type pointer to employee and assign values to it in one statement.
Using this pointer print the employee first name and age

```
mport "fmt"

type Employee struct {
   firstName, lastName string
   age, salary int
}

func main() {

emp* := &Employee{“Ahmed", “Sayed", 45, 6000}

    fmt.Println("First Name:", (*emp8).firstName)
   fmt.Println("Age:", (*emp8).age)
}
```

Output:

First Name: Ahmed
Age: 45

Arrays in Go language

The type [n]T is an array of n values of type **T**.

The expression

var a [10]int

declares a variable a as an array of ten integers.

```
func main() {
        var a [2]string
        a[0] = "Hello"
        a[1] = "World"
        fmt.Println(a[0], a[1])
        fmt.Println(a)

        primes := [6]int{2, 3, 5, 7, 11, 13}
        fmt.Println(primes)
}
```

Output

```
Hello World

[Hello World]

[2 3 5 7 11 13]
```

Slices

An array has a fixed size.

- A slice, on the other hand, is a dynamically-sized, flexible view into the elements of an array. In practice, slices are much more common than arrays.

- The type []T is a slice with elements of type T.

- A slice is formed by specifying two indices, a low and high bound, separated by a colon:

- **a[low : high]**

- This includes the first element, but excludes the last one.

- **a[1:4]** creates a slice which includes elements 1 through 3 of a:

```
package main

import "fmt"

func main() {
        primes := [6]int{2, 3, 5, 7, 11, 13}
        var s [ ]int = primes[1:4]
        fmt.Println(s)
}
```

[357]

Slices are like references to arrays (same address)

A slice does not store any data, it just describes a section of an underlying array.

Changing the elements of a slice modifies the corresponding elements of its underlying array.

Other slices that share the same underlying array will see those changes.

```
func main() {
        names := [4]string{
                "Ahmed",
                "Alil",
                "Mai",
                "Ramy",
        }
        fmt.Println(names)
        a := names[0:2]
        b := names[1:3]
        fmt.Println(a, b)
        b[0] = "XXX"
        fmt.Println(a, b)
        fmt.Println(names)
}
```

Output

[Ahmed Ali Mai Ramy]

[Ahmed Ali] [Ali Mai]

[Ahmed XXX] [XXX Mai]

[Ahmed XXX Mai Ramy]

Array of structure

```
import "fmt"

func main() {
		q := []int{2, 3, 5, 7, 11, 13}
		fmt.Println(q)

		r := []bool{true, false, true, true, false, true}
		fmt.Println(r)

		s := []struct {
			i int
			b bool
		}{
			{2, true},
			{3, false},
			{5, true},
			{7, true},
			{11, false},
			{13, true},
		}
		fmt.Println(s)
}
```

```
[2 3 5 7 11 13]

[true false true true false true]

[{2 true} {3 false} {5 true} {7 true} {11 false} {13 true}]
```

Slice defaults

You may omit the high or low bounds to use their defaults instead.

The default is zero for the low bound and the length of the slice for the high bound.

For the array

var a [10]int

these slice expressions are equivalent:

a[0:10]
a[:10]
a[0:]
a[:]

```
package main

import "fmt"

func main() {
        s := []int{2, 3, 5, 7, 11, 13}
        sa := s[1:4]
        fmt.Println(sa)
        sa = sa[:2]
        fmt.Println(sa)
        sa = sa[1:]
        fmt.Println(sa)
}
```

```
[3 5 7]
[3 5]
[5]
```

Slice length and capacity

- The **length** of a slice is the **number of elements** it contains.

- The **capacity** of a slice is the number of elements in the underlying array, counting from the first element in the slice.

- The length and capacity of a slice s can be obtained using the expressions len(s) and cap(s).

```
import "fmt"

func main() {
		s := []int{2, 3, 5, 7, 11, 13}
		printSlice(s)

		// Slice the slice to give it zero length.
		s = s[:0]
		printSlice(s)

		// Extend its length.
		s = s[:4]
		printSlice(s)

		// Drop its first two values.
		s = s[2:]
		printSlice(s)
}

func printSlice(s []int) {
		fmt.Printf("len=%d cap=%d %v\n", len(s), cap(s), s)
}
```

Output

```
len=6 cap=6 [2 3 5 7 11 13]

len=0 cap=6 []

len=4 cap=6 [2 3 5 7]

len=2 cap=4 [5 7]
```