ЛЕКЦИЯ 3

Композитные типы. Циклы. Указатели

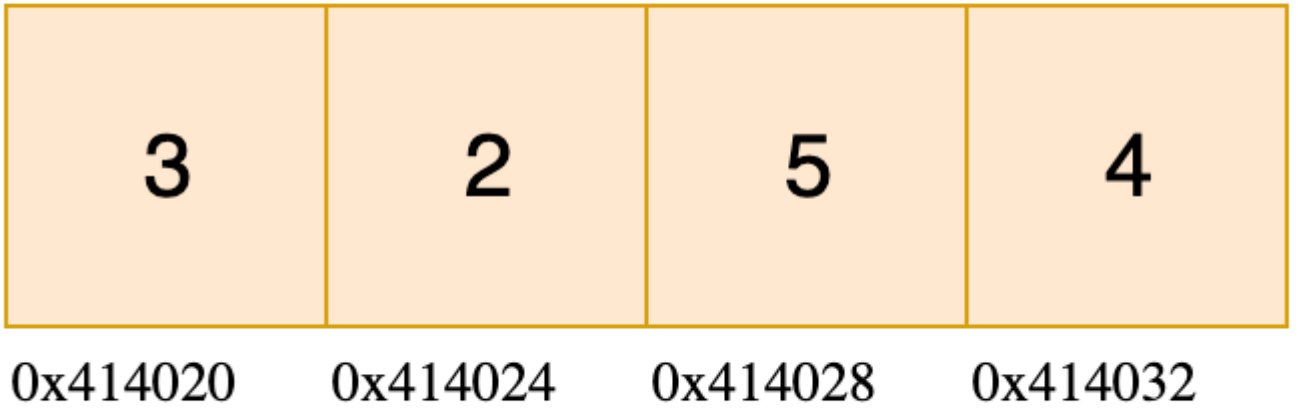
Массивы. Arrays

Инициализация массива:

```
testArr := [5]int{1,2,3,4,5}
```

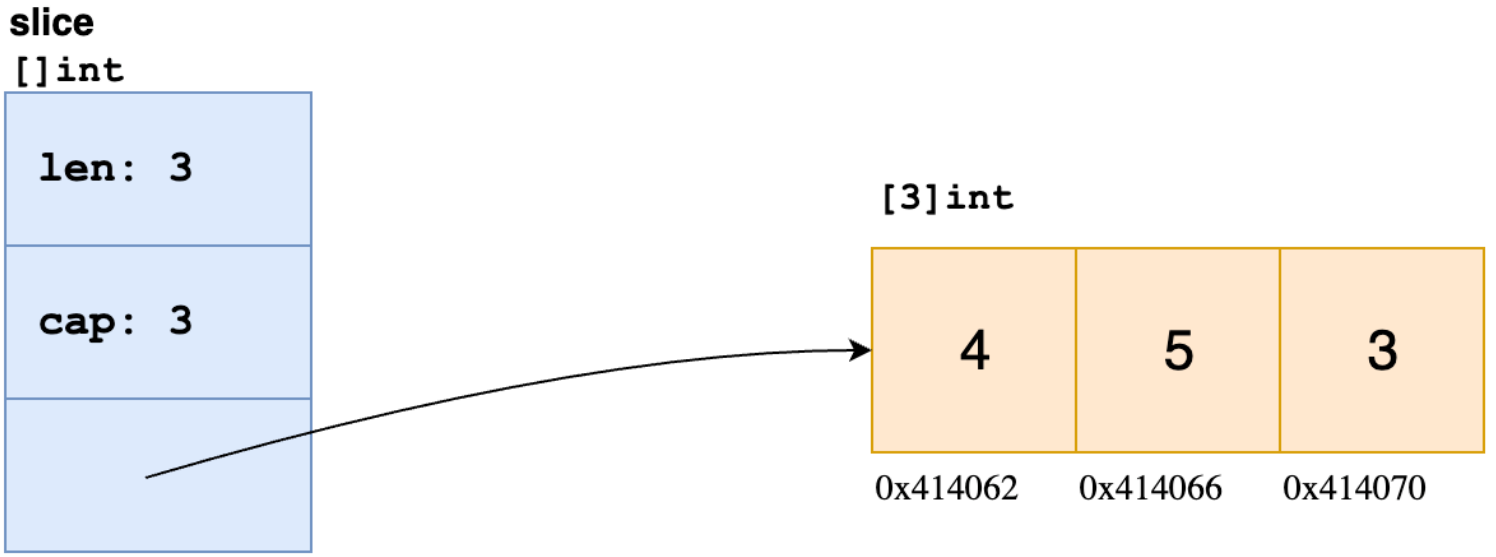
Массивы. Arrays

arr
`[4]int`



Срезы. Slices

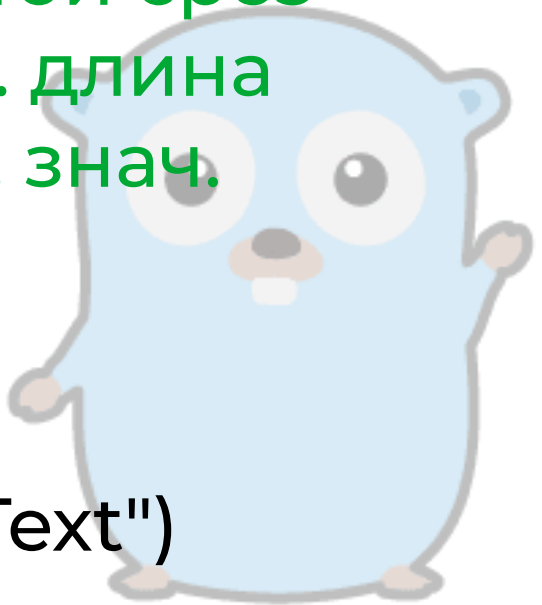
```
slice:= []int{4, 5, 3}
```



Срезы. Slices

Инициализация среза:

```
var testSlice []string                  // Пустой срез
testSlice := make([]string, 12)  // Опр. длина
testSlice := []string{"a", "b"}     // Опр. знач.
```



Добавление элементов:

```
testSlice = append(testSlice, "someText")
```

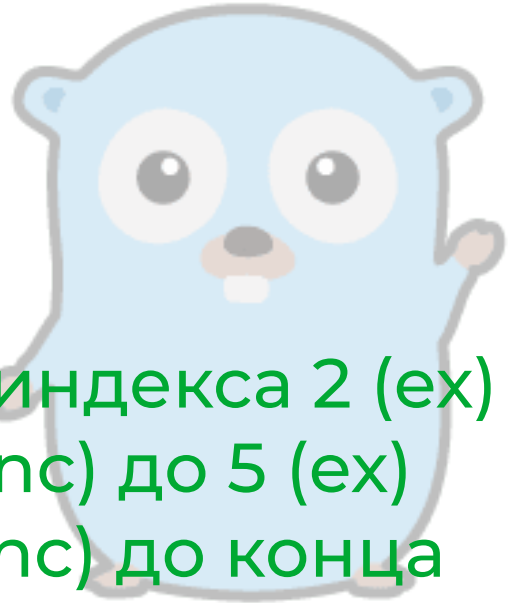
Срезы. Slices

Работа с элементами массива / среза:

```
someVar := testArr[0]
testArr[0] = "some value"
```

Работа со срезами:

```
someVar := testArr[:2]   // От начала до индекса 2 (ex)
someVar := testArr[3:5] // С индекса 3 (inc) до 5 (ex)
someVar := testArr[5:]   // С индекса 5 (inc) до конца
```



Срезы. Slices

Определение длины и емкости массива / среза:

```
len(testArr)    // Вернет кол-во элементов
cap(testArr)    // Вернет емкость
```

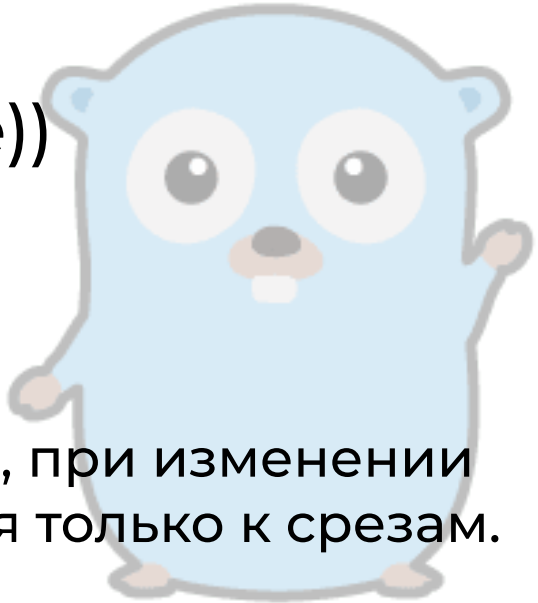
Срезы. Slices

Копирование среза:

```
firstSlice := []int{1,2,3,4,5,6,7}

secondSlice := make([]int, len(firstSlice))

copy(secondSlice, firstSlice)
```



Если просто приравнять secondSlice := firstSlice, при изменении secondSlice изменится и firstSlice. Это относится только к срезам.

Срезы. Slices

Сортировка среза (стандартным пакетом sort):

```
dataArr := []int{9,5,4,3,8,11,71}

sort.Slice(data,  func(i, j int) bool {
    return data[i] < data[j]  // По возрастанию ASC
})
sort.Slice(data,  func(i, j int) bool {
    return data[i] > data[j]  // По убыванию DESC
})
//Проверить отсортирован ли срез - SliceIsSorted
```

Карты. Maps

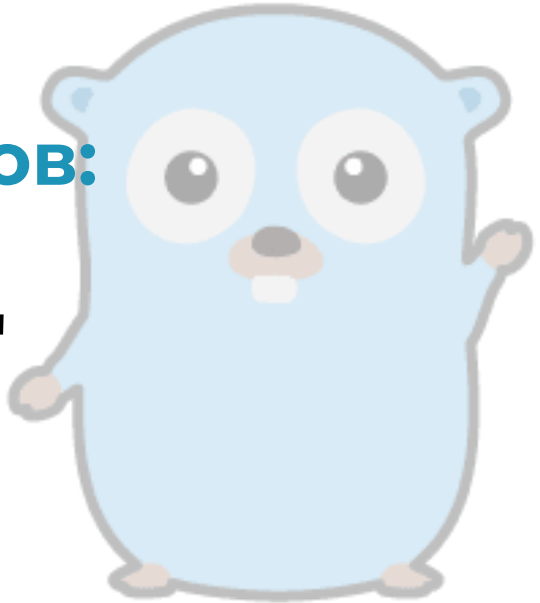
Инициализация карты:

```
testMap := make(map[string]string)
```

Добавление / удаление элементов:

```
testMap["someKey"] = "Some value"

delete(testMap, "someKey")
```



Карты. Maps

Количество элементов в карте:

```
len(testMap)
```

Чтение элемента с карты по ключу:

```
value, isExist := testMap["someKey"]

if isExist { // Do something }
```

Циклы. For

Цикл (init, condition, post):

```
for i := 0; i < 100; i++ {
    // do something
}
```

Циклы. For

Сокращенные циклы:

```
for i := 0; i < 100; {  // Init and condition
    i++
}

for i < 100 {    // Only condition (like while)
    i++
}
```

Циклы. For

Обход массива / среза (Вариант 1):

```
testArr := []int{1,2,3,4,5,6,7,8,9,10}

for i := 0; i < len(testArr); i++ {
    fmt.Println(testArr[i])
}
```

Преимущества: экономия памяти

Циклы. For

Обход массива / среза (Вариант 2):

```
testArr := []int{1,2,3,4,5,6,7,8,9,10}

for index, value := range testArr {
    fmt.Println(index, value)
}
```

Циклы. For

Обход карты:

```
testMap := make(map[string]int)

testMap["someKey"] = "Some value"

for key, value := range testMap {
    fmt.Println(key, value)
}
```

Переключатель. Switch

```
switch i {
case 0:
    fmt.Println("Zero")
case 1:
    fmt.Println("One")
...
default:
    fmt.Println("Unknown Number")
}
```

Указатели. Pointers

Указатель – переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало.

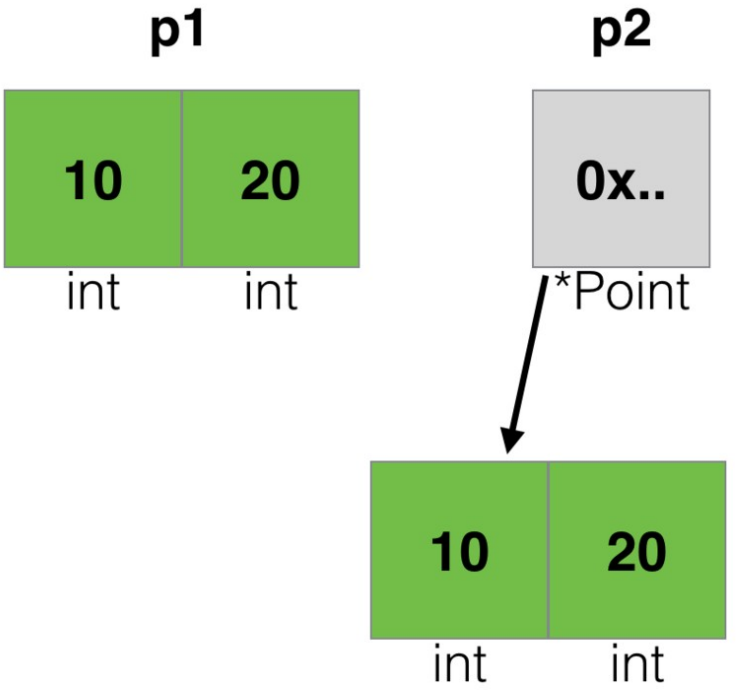
Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению.

Указатели. Pointers

Code:

```
type Point struct {
    X, Y int
}

p1 := Point{10, 20}
p2 := &Point{10, 20}
```

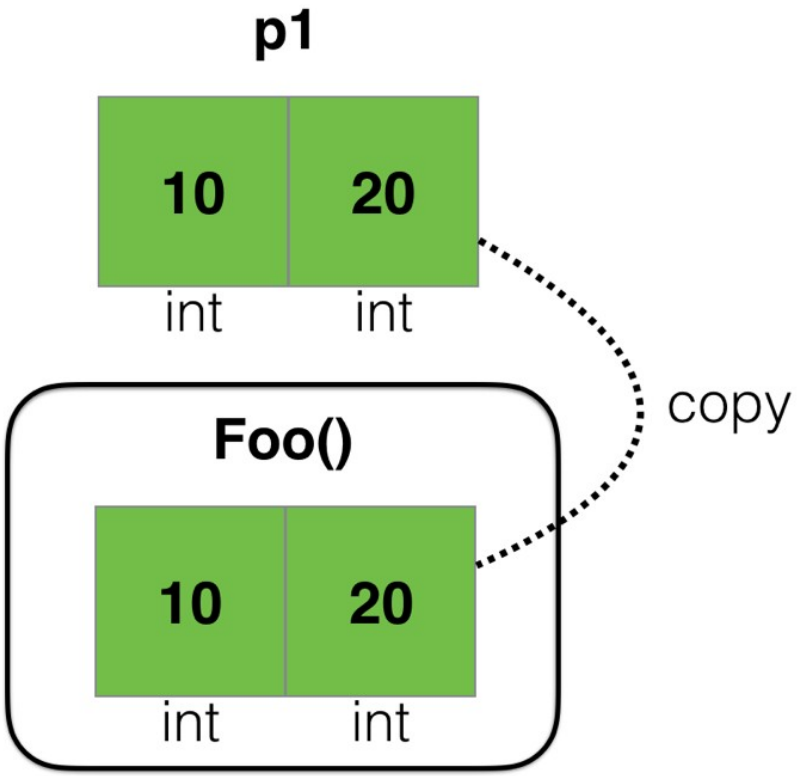


Указатели. Pointers

Code:

```
func Foo(p Point) {
    // ...
}

p1 := Point{10, 20}
Foo(p1)
```

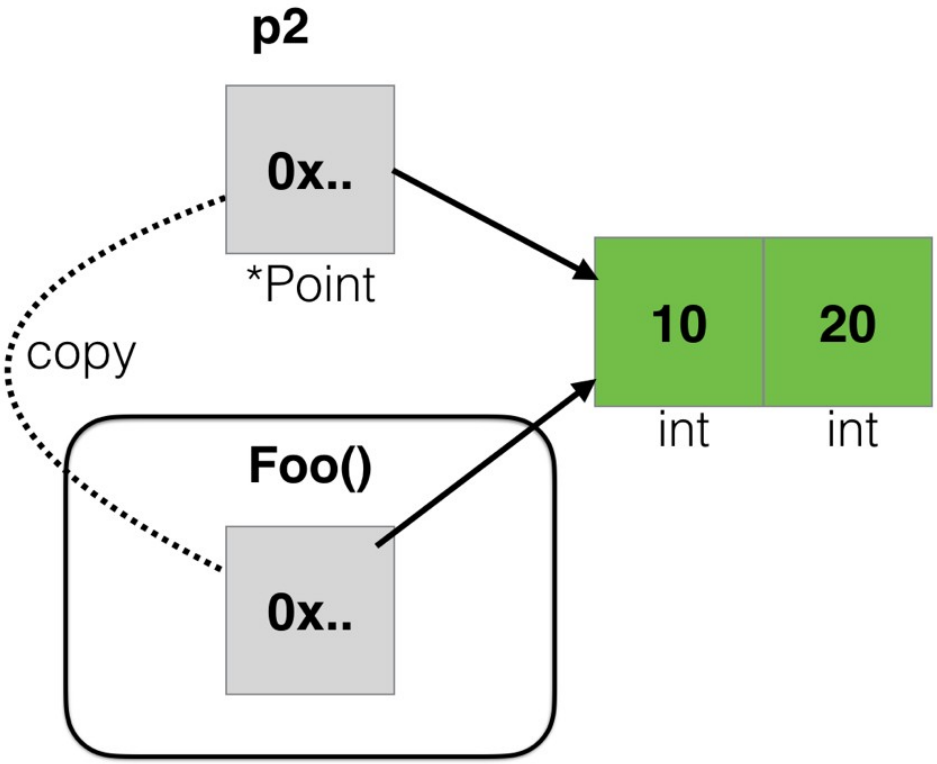


Указатели. Pointers

Code:

```
func Foo(p *Point) {
    // ...
}

p2 := &Point{10, 20}
Foo(p2)
```



Указатели. Pointers

Когда звездочка, а когда амперсанд?:

Звездочка в сигнатуре функции принимающей указатель или при копировании переменной по указателю:

```
funcUsedPointer(testVar *[]int) int {
}

sourceVar := &[]int{1,2,3}
clonedVar := *sourceVar
```

Указатели. Pointers

Когда звездочка, а когда амперсанд?:

Амперсанд при инициализации и при передаче в функцию (если переменная не является указателем):

```
testVar := &[]int{}
result := funcUsedPointer(testVar)

testVar := []int{}
result := funcUsedPointer(&testVar)
```