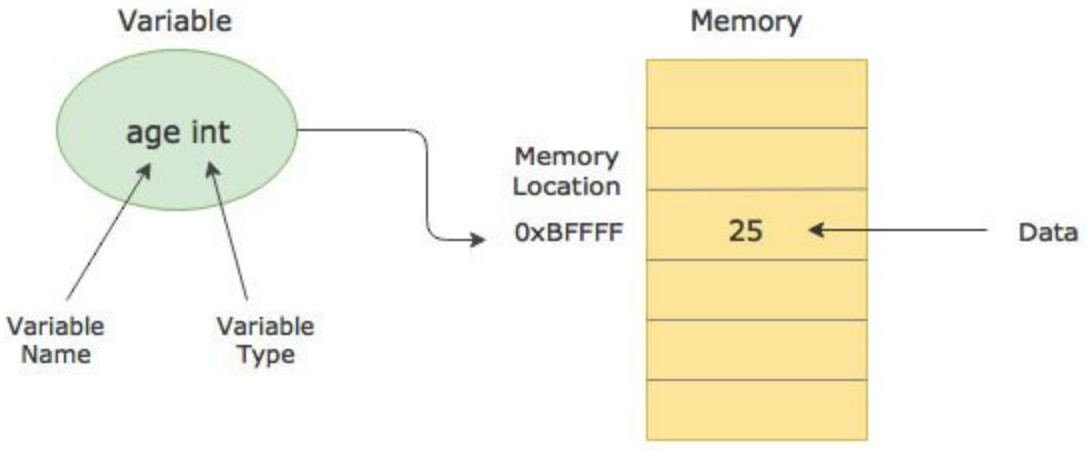
Backend Golang
Урок 4

Указатели. Структуры и их методы. Arrays, slices, maps

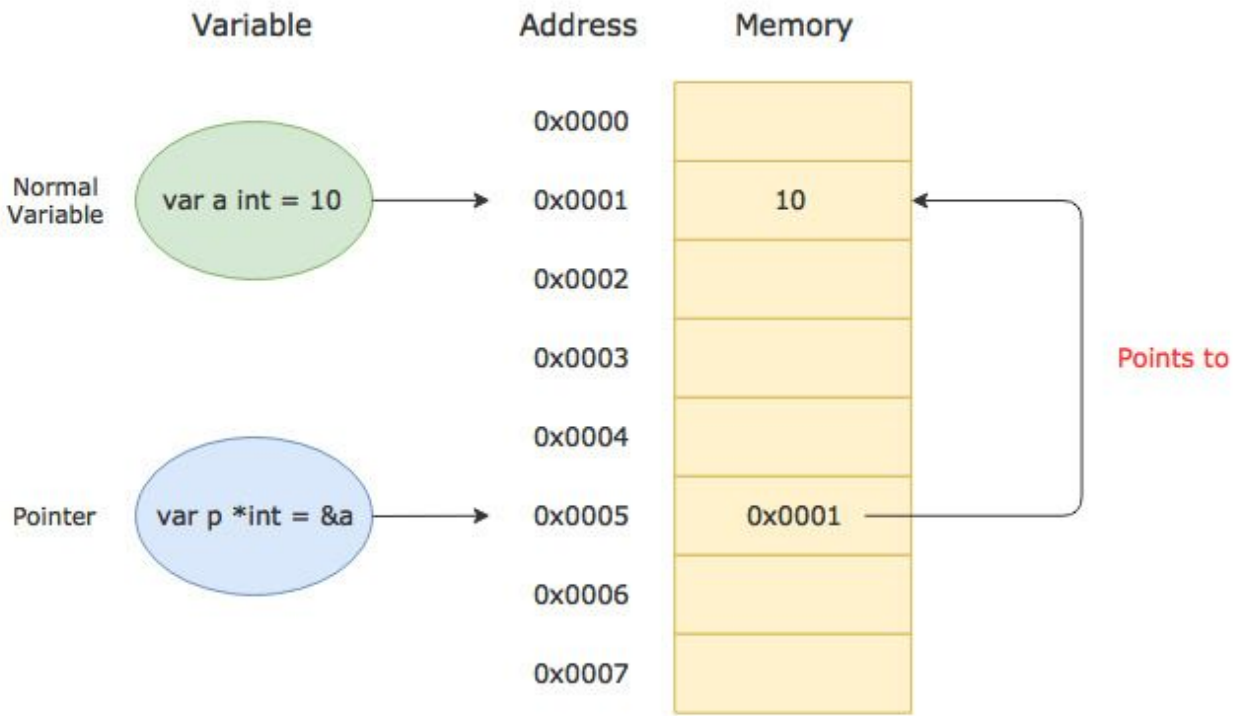
Цели занятия

- Научиться работать с **указателями**
- Познакомиться со **структурами и их методами**
- Понять, как устроены **массивы, слайсы, мапы** и научиться с ними работать

Переменные в памяти



Указатели в памяти



Указатели

Указатель - это адрес некоторого значения в памяти. Указатели строго типизированы. Zero Value для указателя - nil.

Для получения адреса используется оператор &.

```
package main

func main() {
  var a int = 10
  var p *int = &a
}
```

Разыменовывание указателей

Разыменование осуществляется с помощью оператора `*`.

```
package main

func main() {
  var a int = 10
  var p *int = &a
  println(p)  // 0xc00003c768 - какой-то адрес
  println(*p) // 10
}
```

Копирование указателей

При копировании указателя копируется только адрес данных.

```
package main

func main() {
  s := "some string"
  ptr := &s
  ptr2 := ptr
  println(ptr == ptr2)   // true
  println(&ptr == &ptr2) // false
}
```

Указатели в параметрах функций

Если необходимо в функции изменить оригинальное значение переменной, необходимо передать указатель.

```
package main

func main() {
  var counter int
  for counter < 10 {
     increment(&counter)
  }
  println(counter) // 10
}

func increment(n *int) {
  *n++
}
```

Структуры

Структуры - фиксированный набор переменных. Переменные размещаются рядом в памяти и обычно используются совместно.

```
type emptyStruct struct{} // пустая структура, не занимает памяти

type user struct { // структура с именованными полями
  name  string
  age   int
  phone string
}
```

Литералы структур

```
var u1 user   // Zero Value для типа user
u2 := user{}  // Zero Value для типа user
u3 := &user{} // указатель на Zero Value

u4 := user{ // значения по порядковым номерам полей
  "Jerry",
  4,
  "+77778889900",
}
u5 := user{ // значения по именам полей
  name:  "Jerry",
  age:   4,
  phone: "+77778889900",
}
```

Анонимные типы и структуры

Анонимные типы задаются литералом, у такого типа нет имени.

Типичный сценарий использования: когда структура нужно только внутри одной функции.

```
package main

func main() {
  point := struct {
     x int
     y int
  }{}
  point.x = 10
  point.y = 15
}
```

Можно узнать размер памяти, занимаемый структурой

```go
package main

import (
  "fmt"
  "unsafe"
)

type user struct {
  a byte
  b int
  c bool
  d int
}

func main() {
  fmt.Println(unsafe.Sizeof(user{})) // 32 байта
}
```

Порядок полей влияет на расположение в памяти

```go
package main

import (
  "fmt"
  "unsafe"
)

type user struct {
  b int
  d int
  a byte
  c bool
}

func main() {
  fmt.Println(unsafe.Sizeof(user{})) // 24 байта
}
```

Копирование структур

При копировании структуры копируются все поля

```
package main

type user struct {
  name  string
  age   int
  phone string
}

func main() {
  user1 := user{name: "Jerry"}
  user2 := user1
  println(user2.name) // Jerry

}
```

Определение методов

В Go можно определять методы у именованных типов (кроме интерфейсов)

```
package main

type user struct {
  name  string
  age   int
  phone string
}

func (u user) getName() string {
  return u.name
}

func main() {
  user1 := user{name: "Jerry"}
  println(user1.getName()) // Jerry

}
```

Zero Value структур

Zero Value для структур — это структура, в которой все поля равны соответствующим Zero Value.

Методы типа и указателя на тип

- Геттеры: приемник — тип
- Сеттеры: приемник — указатель на тип

```go
func (u user) HappyBirthday() {
  u.age++ // приемник метода - u - копия оригинального объекта
}
```

```go
func (u *user) HappyBirthday() {
  u.age++ // приемник метода - u - указатель на оригинальный обхект
}
```

```go
package main

import (
  "encoding/json"
)

type User struct {
  Name  string `json:"name"`
  Age   int    `json:"age"`
  Phone string `json:"phone"`
}

func main() {
  user := User{
     Name:  "Tom",
     Age:   5,
     Phone: "466422",
  }
  bytes,   := json.MarshalIndent(user, "", "  ")
  println(string(bytes))
}
```

Тэги элементов структур

Задание 1:

Реализовать структуру Point, конструктор и метод DistanceTo

https://play.golang.org/p/h0XPCIS5UcR

```
// Point - TODO: implement
type Point struct {
}

// NewPoint - TODO: implement
func NewPoint(x, y float64) *Point {
  return nil
}

// DistanceTo - TODO: implement
func (p1 Point) DistanceTo(p2 *Point) float64 {
  return 0
}
```

Массивы

Массив - нумерованная последовательность элементов фиксированной длины. Массив располагается последовательно в памяти и не меняет своей длины.

```
var arr [256]int        // фиксированная длина
var arr [10][10]string // может быть многомерным
var arr [...]{1, 2, 3}
arr := [10]int{1, 2, 3, 4, 5}
```

Длина массива - часть типа, т.е. массивы разной длины это разные типы данных.

Операции над массивами

```
arr[3] = 1 // индексация
len(arr)   // длина массива
arr[3:5]   // получение среза
```

Слайсы

Слайсы - это те же "массивы", но переменной длины.

```
var s []int          // неинициализированный слайс, nil
s := []int{}         // с помощью литерала слайса
s := make([]int, 3) // с помощью функции make, s == {0,0,0}
s := make([]int, 3, 10)
```

Добавление элементов в слайс

Добавить новые элементы в слайс можно с помощью функции append

```
var s []int              // s == nil
s = append(s, 1)         // s == {1} append умеет работать с nilслайсами
s[i] = 1                 // работает если i < len(s)
s[len(s)+10] = 1         // случится panic
s = append(s, 1)         // добавляет 1 в конец слайса
s = append(s, 1, 2, 3)   // добавляет 1, 2, 3 в конец слайса
s = append(s, s2...)     // добавляет содержимое слайса s2 в конец s
```

Получение под-слайсов (нарезка)

s[i:j] - возвращает под-слайс, с i-ого элемента включительно, по j-ый не включительно. Длина нового слайса будет равна j-i.

```
s := []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
s2 := s[:]   // копия s (shallow)
s3 := s[3:5] // []int{3,4}
s4 := s[3:]  // []int{3, 4, 5, 6, 7, 8, 9}
s5 := s[:5]  // []int{0, 1, 2, 3, 4}
```

Задание 2:

Сгенерировать слайс из N случайных целых чисел и распечатать его.

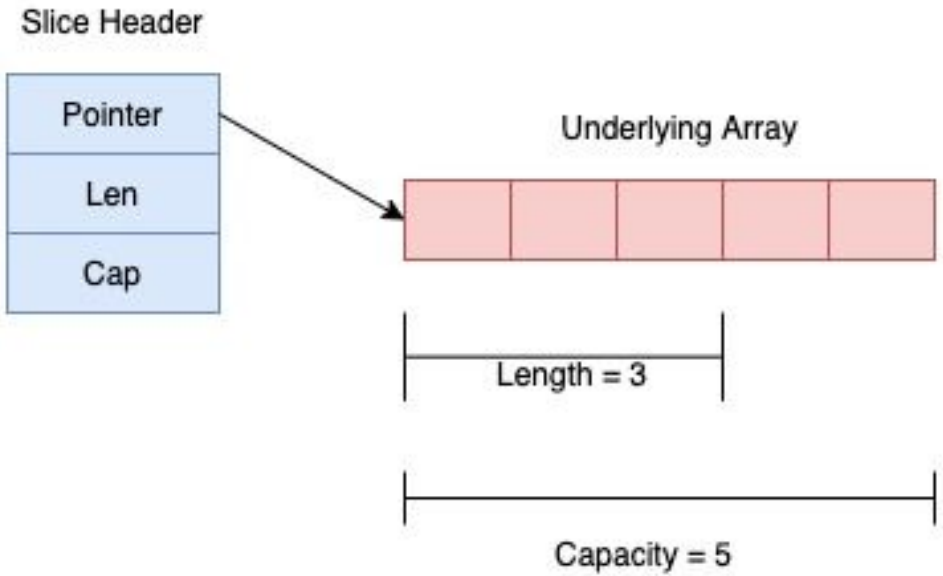
Получить все числа справа от минимального и распечатать.

Внутреннее устройство слайсов

```
// runtime/slice.go
type slice struct {
  array unsafe.Pointer
  len   int
  cap   int
}
```

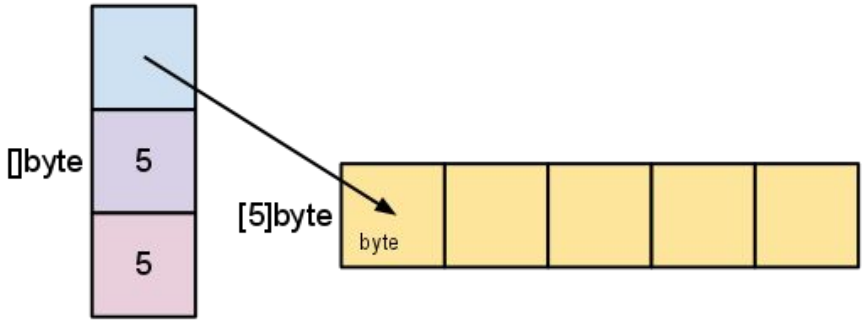
```
l := len(s) // получение длины
c := cap(s) // получение ёмкости
```

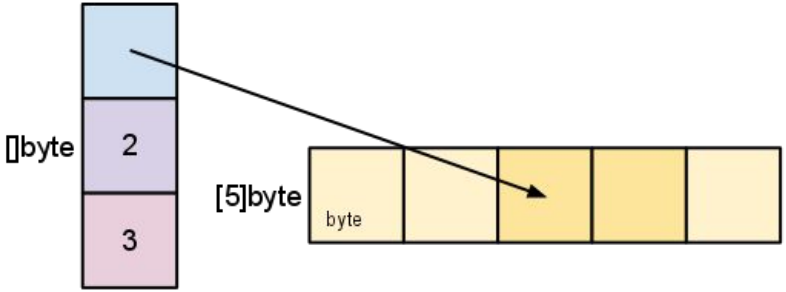
Внутреннее устройство слайсов



Получение под-слайса

```
s := []byte{1, 2, 3, 4, 5}
s2 := s[2:4]
```





Авто-увеличение слайса

Если `len < cap` - увеличивается `len`

Если `len = cap` - увеличивается `cap` , выделяется новый кусок памяти, данные копируются.

```
package main

import "fmt"

func main() {
  var s []int
  fmt.Printf("before loop. len(s): %d,\tcap(s): %d\n", len(s), cap(s))

  for i := 0; i < 100; i++ {
     s = append(s, i)
     fmt.Printf("in loop.     len(s): %d,\tcap(s): %d\n", len(s),
cap(s))
  }
}
```

“Неправильное” копирование слайса

```go
package main

import "fmt"

func main() {
  s1 := []int{1, 2, 3, 4, 5}
  s2 := s1
  s2[0] = 9999999
  fmt.Println("s1", s1)
  fmt.Println("s2", s2)
}
```

```
s1 [9999999 2 3 4 5]
s2 [9999999 2 3 4 5]
```

В итоге, скопировали только заголовок, но не подкапотный массив.

“Правильное” копирование слайса

```go
package main

import "fmt"

func main() {
  s1 := []int{1, 2, 3, 4, 5}
  s2 := make([]int, len(s1))
  copy(s1, s1)
  s2[0] = 9999999
  fmt.Println("s1", s1)
  fmt.Println("s2", s2)
}
```

```
s1 [1 2 3 4 5]
s2 [9999999 0 0 0 0]
```

Задание 3:

Реализовать функцию Flatten, которая получает несколько слайсов на вход и склеивает элементы в один слайс.

https://play.golang.org/p/xRH125W9Ync

```
// Flatten - TODO: implement
func Flatten(slices [][]int) []int {
  return nil
}
```

map (словари)

Словари в Go - это отображение ключ => значение. Словари реализованы как хэш-таблицы. Аналогичные типы в других языках: в Python - dict , в Java - HashMap , в JavaScript - Object .

Создание словарей

```
var cache map[string]string  // неинициализированный словарь, nil
cache := map[string]string{} // с помощью литерала, len(cache) == 0
cache := map[string]string{ // литерал с первоначальным значением
  "one":   "один",
  "two":   "два",
  "three": "три",
}
cache := make(map[string]string)      // тоже что и map[int]string{}
cache := make(map[string]string, 100) // заранее выделить память на 100
ключей
```

Работа со словарями

```
value := cache[key] // получение значения, если ключ не найден Zero Value
```

```
value, ok := cache[key] // получить значение, и флаг того что ключ найден
```

```
_, ok := cache[key] // проверить наличие ключа в словаре
```

```
cache[key] = value // записать значение в инициализированный(!) словарь
```

```
delete(cache, key) // удалить ключ из словаря, работает всегда
```

Итерация по словарю

```
for key, val := range cache {
  // do something
}
```

```
for key := range cache { // если значение не нужно
  // do something
}
```

```
for  , val := range cache { // если ключ не нужен
  // do sometning
}
```

Порядок ключей при итерации не гарантирован. Более того в современных версиях Go этот порядок рандомизирован, т.е. Go будет возвращать ключи в разном порядке каждый раз.

Список ключей и список значений

В Go нет функций, возвращающих списки ключей и значений словаря

```
// Список ключей
var keys []string
for key := range cache {
  keys = append(keys, key)
}
```

```
// Список значений
var values []string
for  , val := range cache {
  values = append(values, val)
}
```

Требования к ключам

Ключём может быть любой тип данных, для которого определена операция `==`:

- числа, строки, bool
- каналы
- интерфейсы
- указатели
- структуры и массивы, содержащие сравнимые типы

```
type User struct {
  Name string
  Host string
}
var cache map[User][]Permission
```

Задание 4

Реализовать функцию подсчета символов в строке `countRunes(s string) map[rune]int`

https://play.golang.org/p/Ua8gNAHrcVe

```
// countRunes - TODO: implement
func countRunes(s string) map[rune]int {
  return nil
}
```

Почитать:

1. Почитать про “стандартную” структуру проекта: https://github.com/golang-standards/project-layout
2. Почитать про представление данных в памяти: https://research.swtch.com/godata
3. Почитать про указатели: https://www.callicoder.com/golang-pointers/
4. Почитать про мапы: https://medium.com/a-journey-with-go/go-map-design-by-example-part-i-3f78a064a352, https://medium.com/a-journey-with-go/go-map-design-by-code-part-ii-50d111557c08, https://medium.com/a-journey-with-go/go-concurrency-access-with-maps-part-iii-8c0a0e4eb27e

Почитать:

5. Почитать про массивы и слайсы: https://golang.org/doc/effective_go#arrays, https://golang.org/doc/effective_go#slices, https://blog.golang.org/slices-intro
6. Почитать про разницу между new и make: https://programmer.help/blogs/the-difference-between-golang-new-and-make.html
7. Почитать про nil: https://golangify.com/nil
8. Разобраться с копированием слайса

ДЗ 1

Реализовать функцию, которая возвращает n самых часто встречающихся слов в строке. Написать тесты для этой функции (https://youtu.be/fMUNBJPhP6Y)

```
// topWords - TODO: implement
func topWords(s string, n int) []int {
  return nil
}
```

ДЗ — Тетрис (оптимизация структуры)

Порядок полей в структуре влияет на ее конечный размер в памяти (https://medium.com/german-gorelkin/go-alignment-a359ff54f272)

Написать функцию, которая на вход принимает путь к файлу со структурой:

```
package any

type SomeName struct {
  FirstField  int
  SecondField *bool
  ThirdField  string
  FourthField uint64
  FifthField  string
  SixthField  *byte
}
```

Количество полей в структуре — 6, типы могут отличаться. Задача — расположить поля в таком порядке, чтобы структура занимала наименьший возможный размер в памяти.

*** Вывести топ 3 самых эффективных (в плане памяти) расположения.