L1 устные вопросы

▼ 1. Какой самый эффективный способ конкатенации строк?

strings.Builder и метод **WriteString**

При каждом вызове `res+=str` в памяти создаётся новая строка. Это происходит ввиду неизменяемости строк в Go (при любом изменении создаётся новая строка). Чтобы избавиться от лишних аллокаций, мы можем воспользоваться типом **strings.Builder** и методом **WriteString:**

```
func join(strs...string)string {
var sb strings.Builder
for _, str:=range strs {
            sb.WriteString(str)
    }
return sb.String()
 }
```

Другие методы stings.Builder:

- **WriteRune** и **WriteByte**
- **Reset**

▼ 2. Что такое интерфейсы, как они применяются в Go?

Интерфейс – это тип, описывающий какими методами должен обладать другой тип, чтобы удовлетворять интерфейсу. Он содержит множество функций, включая их названия, параметры, и возвращаемый тип. Интерфейсы используются для выражения концептуальной схожести разных типов. С интерфейсами можно обращаться как с другими типами, создавать переменные, указывая интерфейсный тип, передавать и возвращать.

▼ 3. Чем отличаются RWMutex от Mutex?

У RWMutex есть дополнительные методы RLock() и RUnlock(), которые позволяют читать параллельно нескольким горутинам. В то время как Lock()

блокирует запись и чтение. RLock() не блокирует другие RLock(), но не позволяет выполниться Lock().

▼ 4. Чем отличаются буферизированные и не буферизированные каналы?

После отправки значения в не буферизированный канал блокируется выполнение горутины, пока значение не будет прочитано.
Буферизированный канал можно заполнять в соответствии с его размером, выполнение горутины не будет прерываться, пока есть свободное место.
Размер буферезированого канала n задается при его создании: ch:=make(chan int, 5).

```
func join(strs...string)string {
var sb strings.Builder
for _, str:=range strs {
            sb.WriteString(str)
    }
return sb.String()
 }
```

▼ 5. Какой размер у структуры struct{}{}?

0 байт

▼ 6. Есть ли в Go перегрузка методов или операторов?

Нет

▼ 7. В какой последовательности будут выведены элементы map[int]int?

В случайной

▼ 8. В чем разница make и new?

make	new
возвращает экземпляр типа данных	возвращает указатель
выделяет память и инициализирует нулевым значением	выделяет память и инициализирует нулевым значением (кроме ссылочных типов)
slice, map, chan	все типы
можно указать размер	указывается только тип

▼ 9. Сколько существует способов задать переменную типа slice или map?

slice:

```
sl:=make([]int, 10)

sl:=make([]int, 10, 20)

var sl []int // sl=append(sl, 2)

var sl []int = []int{1,2,3}
sl:=[]int{1,2,3}
var sl=[]int{1,2,3}

sl:=new([10]int)[0:4]

p := new([]int) //*p = append(*p, 1)
```

map:

```
m:=make(map[int]int)

var m map[int]int=map [int] int{1:3, 2:4}
m:=map [int] int{1:3, 2:4}

  var a map[string]int
  a = map[string]int{}
  a["z"] = 10

  p := new(map[string]int)
  *p = map[string]int{}
  (*p)["z"] = 1
```

map должна быть объявлена и инициализирована для возможности записи в нее. Чтение из nil map производится как из пустой, а запись сопровождается паникой.

▼ 10. Что выведет данная программа и почему?

```
func update(p *int) {
  b := 2
  p = &b
}

func main() {
  var (
    a = 1
    p = &a
  )
  fmt.Println(*p)
  update(p)
  fmt.Println(*p)
}
```

```
1
1
```

Так как копии переменной p, содержащей адрес, по которому содержится 1, присваивается значение с новым адресом (переменной b). А ячейка памяти, на которую указывает переменная p из main остается не тронутой.

▼ 11. Что выведет данная программа и почему?

```
func main() {
  wg := sync.WaitGroup{}
  for i := 0; i < 5; i++ {
     wg.Add(1)
     go func(wg sync.WaitGroup, i int) {
        fmt.Println(i)
        wg.Done()
     }(wg, i)
  }
  wg.Wait()
  fmt.Println("exit")
}
```

```
4
2
1
0
3
fatal error: all goroutines are asleep - deadlock!
```

В функцию передается копия wg, из-за этого уменьшение счетчика влияет только на копию, а не на wg в main. И таким образом значение wg навсегда остается равным 5.
Нужно передавать указатель или вообще не передавать в данном случае, так как переменная в области видимости.

▼ 12. Что выведет данная программа и почему?

```
func main() {
  n := 0
  if true {
     n := 1
     n++
  }
  fmt.Println(n)
}
```

```
0
```

Переменная n переобъявлена в блоке, и вне блока именно ее не существует. А существует n, объявленная и инициализированная изначально.

▼ 13. Что выведет данная программа и почему?

```
func someAction(v []int8, b int8) {
  v[0] = 100
  v = append(v, b)
}

func main() {
  var a = []int8{1, 2, 3, 4, 5}
  someAction(a, 6)
```

```
  fmt.Println(a)
}
```

`[100 2 3 4 5]`

append влияет только на копию слайса внутри функции someAction. v[0] меняет элемент в опорном массиве, который опорный и для слайса из main, поэтому это отражается на слайсе a.

▼ 14. Что выведет данная программа и почему?

```
func main() {
  slice := []string{"a", "a"}

  func(slice []string) {
     slice = append(slice, "a")
     slice[0] = "b"
     slice[1] = "b"
     fmt.Print(slice)
  }(slice)
  fmt.Print(slice)
}
```

`[b b a][a a]`

Был создан новый опорный массив из-за функции append (так как необходимо было расширение). Поэтому изменение нулевого и первого элемента не затрагивает изначальный слайс из main. append меняет копию слайса, а не сам слайс, который был передан функции.