1. Какой самый эффективный способ конкатенации строк?

Пакет strings имеет тип builder, который является очень эффективным способом построения строк. Он использует гораздо меньше памяти при объединении строк и является лучшим способом объединения. Функция WriteString позволяет нам объединять строки более быстрым способом. Ниже приведен пример объединения строк с использованием метода strings builder.

```
package main



import (
    "fmt"
    "strings"
)

func main() {
    var sb strings.Builder
    sb.WriteString("First")
    sb.WriteString("Second")
    fmt.Println(sb.String()) // FirstSecond
}
```

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

В языке Go интерфейс — это набор методов, которые могут быть реализованы типом. Иными словами, интерфейс — описание того, что может сделать тип.

Если тип имеет методы, описанные в интерфейсе, то этот тип удовлетворяет интерфейсу.

Интерфейсы позволяют реализовывать код, работающий с разными типами данных. Важно только чтобы эти данные удовлетворяли интерфейсу.

3. Чем отличаются RWMutex от Mutex?

Mutex методом Lock блокирует все операции чтения и записи какого-либо объекта, а метод RLock в RWMutex блокирует только операции на запись, чтение разрешается.

4. Чем отличаются буферизированные и не буферизированные каналы?
   Буферизированные каналы позволяет записать определённое количество данных, прежде чем он остановит выполнение горутины.

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?

   Пример:

m[0]=1

m[1]=124

m[2]=281

Если выводить через fmt.Println(m), то вывод будет отсортирован по ключам автоматически, если же через цикл:

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

То здесь уже строгого порядка не будет.

8. В чем разница make и new?

map	**new**
Используется только для slice, map, chan	Любая структура, включая срез, карту, чан
Входящие параметры включают тип и размер.	Допускается только один параметр типа, без размера
Возвращается сам тип	Возврат — это указатель типа

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

Slice (4 способа):

```
var mySlice []int
mySlice := make([]int,5,10) // слайс [0 0 0 0 0], базовый массив [0 0 0 0 0 0 0 0 0 0]
s = []int{1, 2, 3} // [1 2 3]

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

Map:

```
m := make(map[string]string)    // создаём map
MyStringMap := map[string]string{"first": "первый", "second": "второй"}
MyMap := map[string]string{}
var MyMap = map[string]string{}
var MyMap map[string]string
```

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)
}
```

Будет выведено 1 в обоих случаях, так как в функцию передаётся копия указателя и меняется адрес этой копии.

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")
}
```

fatal error: all goroutines are asleep - deadlock!

Потому что в горутину передаётся копия переменной wg, но внешняя переменная не меняется, а потому ошибка возникнет в строке “wg.Wait()”, так как он никогда не завершится, так как его внутренний счётчик никогда не станет 0.

12. Что выведет данная программа и почему?

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

В условии создаётся новая переменная n, а за пределами условия у нас глобальная n, которая всё еще равна 0.

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)
        }
```

В функцию передаётся срез, а срез — это ссылочный тип, поэтому первая строка в функции someAction изменяет первоначальный слайс a, однако затем происходит операция добавления нового числа в слайс, которая не изменяет слайс, а создаёт новый, поэтому старый слайс остаётся неизменным, а потому будет выведено:

[100 2 3 4 5]

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)
}
```

Внутри функции после append создаётся новый слайс, а потому в функции изменяется новый слайс. За пределами функции будет выведен старый слайс, а потому будет выведено:

[b b a][a a]