WB Tech: level # 1 (Golang)

# Как делать задания

В заданиях никаких устных решений — только код. Одно решение — один файл с хорошо откомментированным кодом. Каждое решение или невозможность решения надо объяснить.

Разрешается и приветствуется использование любых справочных ресурсов, привлечение сторонних экспертов и т.д. и т.п.

Основной критерий оценки — четкое понимание «как это работает». Некоторые задачи можно решить несколькими способами, в этом случае требуется привести максимально возможное количество вариантов.

Можно задавать вопросы, как по условию задач, так и об их решении. Идеальный вариант — продемонстрировать свои решения и получить максимальный фидбэк от опытных разработчиков Wildberries.

# Задания

1. Дана структура Human (с произвольным набором полей и методов). Реализовать встраивание методов в структуре Action от родительской структуры Human (аналог наследования).

package L1  
  
import (  
 "fmt"  
 "math"  
 \_ "math"  
)  
  
type Human struct {  
 weight, height float64  
}  
  
func (h Human) BMI() float64 {  
 return h.weight / math.Pow(h.height, 2)  
}  
  
type Action struct {  
 human Human  
 activity float64  
 bmi float64  
}  
  
func (a Action) workout() {  
 a.human.weight -= 0.5 \* a.activity  
 a.bmi = a.human.BMI()  
}  
  
func main() {  
 newHuman := Human{height: 163, weight: 75}  
 newAction := Action{human: newHuman, activity: 3, bmi: newHuman.BMI()}  
 fmt.Println(newAction.bmi)  
 newAction.workout()  
 fmt.Println(newAction.bmi)  
}

1. Написать программу, которая конкурентно рассчитает значение квадратов чисел взятых из массива (2,4,6,8,10) и выведет их квадраты в stdout.

package main  
  
import (  
 "fmt"  
 "math"  
 "os"  
 "time"  
)  
  
func main() {  
 start := time.Now()  
 numbers := []float64{2, 4, 6, 8, 10}  
 go func() {  
 for \_, num := range numbers {  
 \_, err := fmt.Fprintln(os.Stdout, math.Pow(num, 2))  
 if err != nil {  
 return  
 }  
 }  
 }()  
  
 elapsedTime := time.Since(start)  
  
 fmt.Println("Total Time For Execution: " + elapsedTime.String())  
  
 time.Sleep(time.*Second*)  
}

1. Дана последовательность чисел: 2,4,6,8,10. Найти сумму их квадратов(22+32+42….) с использованием конкурентных вычислений.

package main  
  
import (  
 "fmt"  
 "math"  
 "os"  
 "time"  
)  
  
func main() {  
 start := time.Now()  
 numbers := []float64{2, 4, 6, 8, 10}  
 go func() {  
 var sum float64 = 0  
 for \_, num := range numbers {  
 sum += math.Pow(num, 2)  
 }  
 \_, err := fmt.Fprintln(os.Stdout, sum)  
 if err != nil {  
 return  
 }  
 }()  
  
 elapsedTime := time.Since(start)  
  
 fmt.Println("Total Time For Execution: " + elapsedTime.String())  
  
 time.Sleep(time.*Second*)  
}

1. Реализовать постоянную запись данных в канал (главный поток). Реализовать набор из N воркеров, которые читают произвольные данные из канала и выводят в stdout. Необходима возможность выбора количества воркеров при старте.  
     
   Программа должна завершаться по нажатию Ctrl+C. Выбрать и обосновать способ завершения работы всех воркеров.

package main  
  
import (  
 "fmt"  
 "os"  
)  
  
func work(ch chan []byte) {  
 val, err := <-ch  
 if err {  
 panic(err)  
 }  
 fmt.Fprintln(os.Stdout, val)  
}  
  
func main() {  
 //start := time.Now()  
 c := make(chan []byte)  
 var workers int  
 fmt.Println("Сколько будет workers?")  
 fmt.Scanf("%s\n", &workers)  
  
 go func() {  
 for i := 0; i != workers; i++ {  
 work(c)  
 }  
 }()  
  
 //elapsedTime := time.Since(start)  
 //  
 //fmt.Println("Total Time For Execution: " + elapsedTime.String())  
 //  
 //time.Sleep(time.Second)  
}

1. Разработать программу, которая будет последовательно отправлять значения в канал, а с другой стороны канала — читать. По истечению N секунд программа должна завершаться.

package main  
  
import (  
 "fmt"  
 "time"  
)  
  
func main() {  
 //start := time.Now()  
 result := make(chan string)  
 var n uint8  
 fmt.Println("how many seconds for program to work?")  
 fmt.Scanf("%s\n", &n)  
  
 go func() {  
 var data string  
 fmt.Scanf("%s\n", &data)  
 result <- data  
 }()  
  
 select {  
 case <-time.After(10 \* time.*Second*):  
 fmt.Println("timeout")  
 case r := <-result:  
 fmt.Println("data:", r)  
 }  
  
 //elapsedTime := time.Since(start)  
 //  
 //fmt.Println("Total Time For Execution: " + elapsedTime.String())  
 //  
 //time.Sleep(time.Second)  
}

1. Реализовать все возможные способы остановки выполнения горутины.

Первый способ

package main

import "fmt"

func main() {

quit := make(chan bool)

go func() {

fmt.Println("Текст внутри горутины.")

for {

select {

case <-quit:

return

default:

// Do other stuff

}

}

}()

fmt.Println("Текст перед выходом.")

// Выйти из горутины

quit <- true

// Вывод:

//Текст перед выходом.

//Текст внутри горутины.

}

Второй пример

package main  
  
import (  
 "context"  
 "fmt"  
 "time"  
)  
  
func main() {  
 forever := make(chan struct{})  
 ctx, cancel := context.WithCancel(context.Background())  
  
 go func(ctx context.Context) {  
 for {  
 select {  
 case <-ctx.Done(): // if cancel() execute  
 forever <- struct{}{}  
 return  
 default:  
 fmt.Println("for loop")  
 }  
  
 time.Sleep(500 \* time.*Millisecond*)  
 }  
 }(ctx)  
  
 go func() {  
 time.Sleep(3 \* time.*Second*)  
 cancel()  
 }()  
  
 <-forever  
 fmt.Println("finish")  
}

Третий способ

**func** main() {

ch := make(**chan** string, 6)

done := make(**chan** **struct**{})

**go** **func**() {

**for** {

**select** {

**case** ch <- "foo":

**case** <-done:

close(ch)

**return**

}

time.Sleep(100 \* time.Millisecond)

}

}()

**go** **func**() {

time.Sleep(3 \* time.Second)

done <- **struct**{}{}

}()

**for** i := **range** ch {

fmt.Println("Received: ", i)

}

fmt.Println("Finish")

}

1. Реализовать конкурентную запись данных в map.

var contexts = make(map[string]cronMetadata)  
var mutex = &sync.RWMutex{}  
  
// ...  
  
mutex.Lock()  
contexts[keyName] = \*request  
mutex.Unlock()  
  
// retrieve data  
mutex.RLock()  
var cronScaler = contexts[in.GetName()]  
mutex.RUnlock()

или

var contexts = sync.Map{}  
  
// ....  
  
contexts.Store(keyName, \*request)  
  
//retrieve data  
cronContext, \_ := contexts.Load(keyName)

1. Дана переменная int64. Разработать программу которая устанавливает i-й бит в 1 или 0.

package main  
  
import (  
 "fmt"  
)  
  
func main() {  
 var num int  
 \_, err := fmt.Scanf("%d", &num)  
 if err != nil {  
 panic(err)  
 }  
 var res = num | 2  
 fmt.Println(res)  
}

1. Разработать конвейер чисел. Даны два канала: в первый пишутся числа (x) из массива, во второй — результат операции x\*2, после чего данные из второго канала должны выводиться в stdout.
2. package main  
     
   import (  
    "fmt"  
   )  
     
   func gen(nums []int) <-chan int {  
    out := make(chan int)  
    go func() {  
    for \_, n := range nums {  
    out <- n  
    }  
    wg.Wait()  
    close(out)  
    }()  
    return out  
   }  
     
   func double(in <-chan int) <-chan int {  
    out := make(chan int)  
    go func() {  
    for n := range in {  
    out <- n \* 2  
    }  
    close(out)  
    }()  
    return out  
   }  
     
   func main() {  
    length := 0  
    fmt.Println("Enter the length of array")  
    fmt.Scanln(&length)  
    fmt.Println("Enter the inputs")  
    numbers := make([]int, length)  
    for i := 0; i < length; i++ {  
    fmt.Scanln(&numbers[i])  
    }  
    a := gen(numbers)  
    b := double(a)  
     
    for num := range b {  
    fmt.Println(num)  
    }  
   }

1. Дана последовательность температурных колебаний: -25.4, -27.0 13.0, 19.0, 15.5, 24.5, -21.0, 32.5. Объединить данные значения в группы с шагом в 10 градусов. Последовательность в подмножноствах не важна.

Пример: -20:{-25.0, -27.0, -21.0}, 10:{13.0, 19.0, 15.5}, 20: {24.5}, etc.

package main  
  
import "fmt"  
  
func main() {  
 arr := [8]float64{-25.4, -27.0, 13.0, 19.0, 15.5, 24.5, -21.0, 32.5}  
 var m = make(map[int][]float64)  
 max := 30.0  
  
 for \_, num := range arr {  
 for curr := -30.0; curr != max; curr += 10 {  
 if num >= curr && num <= curr+10 {  
 key := curr + 10  
 if num > 0 {  
 key = curr  
 }  
 a := m[int(key)]  
 if a != nil {  
 a = append(a, num)  
 m[int(key)] = a  
 }  
 if a == nil {  
 m[int(key)] = []float64{num}  
 }  
 break  
 }  
 }  
 }  
  
 fmt.Println(m)  
}

1. Реализовать пересечение двух неупорядоченных множеств.

­­­­­­package main  
  
func intersection(nums1 []int, nums2 []int) []int {  
 var count = map[int]bool{}  
 var result = []int{}  
 for \_, num := range nums1 {  
 count[num] = true  
 }  
 for \_, num := range nums2 {  
 if count[num] == true {  
 result = append(result, num)  
 count[num] = false  
 }  
 }  
 return result  
}

1. Имеется последовательность строк - (cat, cat, dog, cat, tree) создать для нее собственное множество.

package main  
  
import "fmt"  
  
func main() {  
 arr := [5]string{"cat", "cat", "dog", "cat", "tree"}  
 collection := make(map[string]int)  
 for \_, word := range arr {  
 collection[word]++  
 }  
 fmt.Println(collection)  
}

1. Поменять местами два числа без создания временной переменной.

package main  
  
import "fmt"  
  
func main() {  
 a, b := 1, 2  
 fmt.Println("before swap")  
 fmt.Println(a, b)  
 b, a = a, b  
 fmt.Println("after swap")  
 fmt.Println(a, b)  
}

1. Разработать программу, которая в рантайме способна определить тип переменной: int, string, bool, channel из переменной типа interface{}.

package main  
  
import "fmt"  
  
func main() {  
 switch v := myInterface.(type) {  
 case int:  
 fmt.Printf("Integer: %v", v)  
 case bool:  
 fmt.Printf("Bool: %v", v)  
 case string:  
 fmt.Printf("String: %v", v)  
 case chan interface{}:  
 fmt.Printf("Chan: %v", v)  
 default:  
 // And here I'm feeling dumb. ;)  
 fmt.Printf("I don't know lol")  
 }  
  
}

1. К каким негативным последствиям может привести данный фрагмент кода, и как это исправить? Приведите корректный пример реализации.

**var** justString string

**func** someFunc() {

v := createHugeString(1 << 10)

justString = v[:100]

}

**func** main() {

someFunc()

}

1. Реализовать быструю сортировку массива (quicksort) встроенными методами языка.

package main  
  
import "fmt"  
  
func partition(arr []int, low, high int) ([]int, int) {  
 pivot := arr[high]  
 i := low  
 for j := low; j < high; j++ {  
 if arr[j] < pivot {  
 arr[i], arr[j] = arr[j], arr[i]  
 i++  
 }  
 }  
 arr[i], arr[high] = arr[high], arr[i]  
 return arr, i  
}  
  
func quickSort(arr []int, low, high int) []int {  
 if low < high {  
 var p int  
 arr, p = partition(arr, low, high)  
 arr = quickSort(arr, low, p-1)  
 arr = quickSort(arr, p+1, high)  
 }  
 return arr  
}  
  
func quickSortStart(arr []int) []int {  
 return quickSort(arr, 0, len(arr)-1)  
}  
  
func main() {  
 fmt.Println(quickSortStart([]int{12, 3, 5, 54, 32, 99}))  
}

1. Реализовать бинарный поиск встроенными методами языка.

package main  
  
import "fmt"  
  
func binarySearch(target int, nums []int) bool {  
  
 low := 0  
 high := len(nums) - 1  
  
 for low <= high {  
 median := (low + high) / 2  
  
 if nums[median] < target {  
 low = median + 1  
 } else {  
 high = median - 1  
 }  
 }  
  
 if low == len(nums) || nums[low] != target {  
 return false  
 }  
  
 return true  
}  
  
func main() {  
 items := []int{1, 7, 15, 26, 33, 47, 65, 88, 101}  
 fmt.Println(binarySearch(47, items))  
}

1. Реализовать структуру-счетчик, которая будет инкрементироваться в конкурентной среде. По завершению программа должна выводить итоговое значение счетчика.
2. package main  
     
   import (  
    "fmt"  
    "sync"  
   )  
     
   func main() {  
    c := 0  
    n := 200  
    m := sync.Mutex{}  
    wg := sync.WaitGroup{}  
    wg.Add(n)  
    for i := 0; i < n; i++ {  
    go func(i int) {  
    m.Lock()  
    c++  
    m.Unlock()  
    wg.Done()  
    }(i)  
    }  
    wg.Wait()  
     
    fmt.Println(c)  
   }

1. Разработать программу, которая переворачивает подаваемую на ход строку (например: «главрыба — абырвалг»). Символы могут быть unicode.

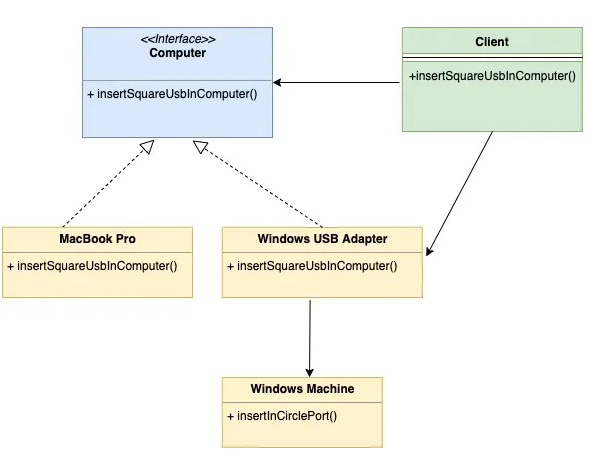
package main  
  
import "fmt"  
  
func flipAstring(s string) string {  
 r := []rune(s)  
 begin := 0  
 end := len(s) - 1  
 for begin <= end {  
 r[begin], r[end] = r[end], r[begin]  
 begin++  
 end--  
 }  
 return string(r)  
}  
  
func main() {  
 line := "sample"  
 fmt.Println("before swap")  
 fmt.Println(line)  
 line = flipAstring(line)  
 fmt.Println("after swap")  
 fmt.Println(line)  
}

1. Разработать программу, которая переворачивает слова в строке.   
   Пример: «snow dog sun — sun dog snow».

package main  
  
import (  
 "fmt"  
 "strings"  
)  
  
func flipAline(s string) string {  
 var arr []string  
 last := 0  
 for i, c := range s {  
 if c == ' ' {  
 arr = append(arr, s[last:i])  
 last = i + 1  
 }  
 if i == len(s)-1 {  
 arr = append(arr, s[last:i+1])  
 }  
 }  
 var rev\_arr []string  
 for i := range arr {  
 rev\_arr = append(rev\_arr, arr[len(arr)-1-i])  
 }  
  
 return strings.Join(rev\_arr, " ")  
}  
  
func main() {  
 line := "snow dog sun"  
 fmt.Println("before swap")  
 fmt.Println(line)  
 line = flipAline(line)  
 fmt.Println("after swap")  
 fmt.Println(line)  
}

1. Реализовать паттерн «адаптер» на любом примере

Адаптер на примере разных соединений ОС

.

**computer.go**

package main  
  
type computer interface {  
 insertInSquarePort()  
}

**mac.go**

package main  
  
import "fmt"  
  
type mac struct {  
}  
  
func (m \*mac) insertInSquarePort() {  
 fmt.Println("Insert square port into mac machine")  
}

**windowsAdapter.go**

package main  
  
type windowsAdapter struct {  
 windowMachine \*windows  
}  
  
func (w \*windowsAdapter) insertInSquarePort() {  
 w.windowMachine.insertInCirclePort()  
}

**windows.go**

package main  
  
import "fmt"  
  
type windows struct{}  
  
func (w \*windows) insertInCirclePort() {  
 fmt.Println("Insert circle port into windows machine")  
}

**client.go**

package main  
  
type client struct {  
}  
  
func (c \*client) insertSquareUsbInComputer(com computer) {  
 com.insertInSquarePort()  
}

**main.go**

package main  
  
func main() {  
 client := &client{}  
 mac := &mac{}  
 client.insertSquareUsbInComputer(mac)  
 windowsMachine := &windows{}  
 windowsMachineAdapter := &windowsAdapter{  
 windowMachine: windowsMachine,  
 }  
 client.insertSquareUsbInComputer(windowsMachineAdapter)  
}

1. Разработать программу, которая перемножает, делит, складывает, вычитает две числовых переменных a,b, значение которых > 2^20.

package main  
  
import (  
 "fmt"  
 "math/big"  
)  
  
func main() {  
 var a, b int64  
  
 fmt.Scanln(&a)  
 fmt.Scanln(&b)  
  
 first := big.NewInt(a)  
 second := big.NewInt(b)  
  
 result := big.NewInt(0)  
 //Add  
 fmt.Println(result.Add(first, second))  
 //Subtract  
 fmt.Println(result.Sub(first, second))  
 //Multiply  
 fmt.Println(result.Mul(first, second))  
 //Divide  
 fmt.Println(result.Div(first, second))  
}

1. Удалить i-ый элемент из слайса.

Быстрый способ(меняет порядок)

a := []string{"A", "B", "C", "D", "E"}  
i := 2  
  
a[i] = a[len(a)-1]   
a[len(a)-1] = ""   
a = a[:len(a)-1]   
  
fmt.Println(a) // [A B E D]

Медленный способ(не меняет порядок)

a := []string{"A", "B", "C", "D", "E"}  
i := 2  
  
copy(a[i:], a[i+1:])   
a[len(a)-1] = ""   
a = a[:len(a)-1]   
  
fmt.Println(a) // [A B D E]

1. Разработать программу нахождения расстояния между двумя точками, которые представлены в виде структуры Point с инкапсулированными параметрами x,y и конструктором.

package main  
  
import (  
 "fmt"  
 "math"  
)  
  
type Point struct {  
 x float64  
 y float64  
}  
  
func calculateDistance(a Point, b Point) float64 {  
 return math.Sqrt(math.Pow(math.Abs(a.x-b.x), 2) +  
 math.Pow(math.Abs(a.y-b.y), 2))  
}  
  
func main() {  
 a := Point{0, 0}  
 b := Point{4, 3}  
 fmt.Println(calculateDistance(a, b))  
}

1. Реализовать собственную функцию sleep.
2. Разработать программу, которая проверяет, что все символы в строке уникальные (true — если уникальные, false etc). Функция проверки должна быть регистронезависимой.

Например:   
abcd — true

abCdefAaf — false  
 aabcd — false

package main  
  
import (  
 "fmt"  
 "strings"  
)  
  
func validate(s string) bool {  
 m := make(map[string]int)  
  
 for i := 0; i != len(s); i++ {  
 c := strings.ToLower(string(s[i]))  
 if m[c] == 0 {  
 m[c]++  
 continue  
 }  
 if m[c] > 0 {  
 return false  
 }  
 }  
 return true  
}  
  
func main() {  
 var s string  
 fmt.Scanln(&s)  
 fmt.Println(validate(s))  
}

# Устные вопросы

1. Какой самый эффективный способ конкатенации строк?
2. Что такое интерфейсы, как они применяются в Go?
3. Чем отличаются RWMutex от Mutex?
4. Чем отличаются буферизированные и не буферизированные каналы?
5. Какой размер у структуры struct{}{}?
6. Есть ли в Go перегрузка методов или операторов?
7. В какой последовательности будут выведены элементы map[int]int?  
     
   *Пример:*

m[0]=1

m[1]=124

m[2]=281

1. В чем разница make и new?
2. Сколько существует способов задать переменную типа slice или map?
3. 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. 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"**)

}

1. Что выведет данная программа и почему?

**func** main() {

n := 0

**if *true*** {

n := 1

n++

}

fmt.Println(n)

}

1. 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)

}

1. 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)

}