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 read(a chan int, b chan int, nums []int) {  
    for \_, x := range nums {  
    a <- x  
    b <- x \* 2  
    }  
   }  
     
   func main() {  
    a := make(chan int)  
    b := make(chan int)  
    length := 0  
    fmt.Println("Enter the number of inputs")  
    fmt.Scanln(&length)  
    fmt.Println("Enter the inputs")  
    numbers := make([]int, length)  
    for i := 0; i < length; i++ {  
    fmt.Scanln(&numbers[i])  
    }  
     
    go read(a, b, numbers)  
     
    res := <-b  
    fmt.Printf(string(rune(res)))  
   }

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.

1. Реализовать пересечение двух неупорядоченных множеств.
2. Имеется последовательность строк - (cat, cat, dog, cat, tree) создать для нее собственное множество.
3. Поменять местами два числа без создания временной переменной.
4. Разработать программу, которая в рантайме способна определить тип переменной: int, string, bool, channel из переменной типа interface{}.
5. 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.
2. Реализовать бинарный поиск встроенными методами языка.
3. Реализовать структуру-счетчик, которая будет инкрементироваться в конкурентной среде. По завершению программа должна выводить итоговое значение счетчика.
4. Разработать программу, которая переворачивает подаваемую на ход строку (например: «главрыба — абырвалг»). Символы могут быть unicode.
5. Разработать программу, которая переворачивает слова в строке.   
   Пример: «snow dog sun — sun dog snow».
6. Реализовать паттерн «адаптер» на любом примере.
7. Разработать программу, которая перемножает, делит, складывает, вычитает две числовых переменных a,b, значение которых > 2^20.
8. Удалить i-ый элемент из слайса.
9. Разработать программу нахождения расстояния между двумя точками, которые представлены в виде структуры Point с инкапсулированными параметрами x,y и конструктором.
10. Реализовать собственную функцию sleep.
11. 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

# 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)

}