**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**



**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

# ОТЧЕТ

**по лабораторной работе № 5**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент | ИУ6-31БВ |  | Бокова В.О. |
|  | (Группа) | (Подпись, дата) | (И.О. Фамилия) |
| Преподаватель |  |  | В.Д.Шульман |
|  |  | (Подпись, дата) | (И.О. Фамилия) |

**Название:** Основы асинхронного программирования на Golang **Дисциплина:** Языки интернет-программирования

Москва, 2024

**Цель работы** — изучение основ асинхронного программирования с использованием языка

Golang.

## Порядок выполнения

1. Ознакомьтесь с разделом "3. Map, файлы, интерфейсы, многопоточность и многое

другое" курса <https://stepik.org/course/54403/info>

1. Сделайте форк данного репозитория в GitHub, склонируйте получившуюся копию локально, создайте от мастера ветку дев и переключитесь на неё
2. Выполните задания. Ссылки на задания содержатся в README-файлах в директории

projects

1. Сделайте отчёт и поместите его в директорию docs
2. Зафиксируйте изменения, сделайте коммит и отправьте полученное состояние ветки дев в ваш удаленный репозиторий GitHub
3. Через интерфейс GitHub создайте Pull Request dev --> master
4. На защите лабораторной работы продемонстрируйте открытый Pull Request. PR должен быть направлен в master ветку вашего репозитория

**Ход работы**

**A) Задание «Calculator» (рис 1)**

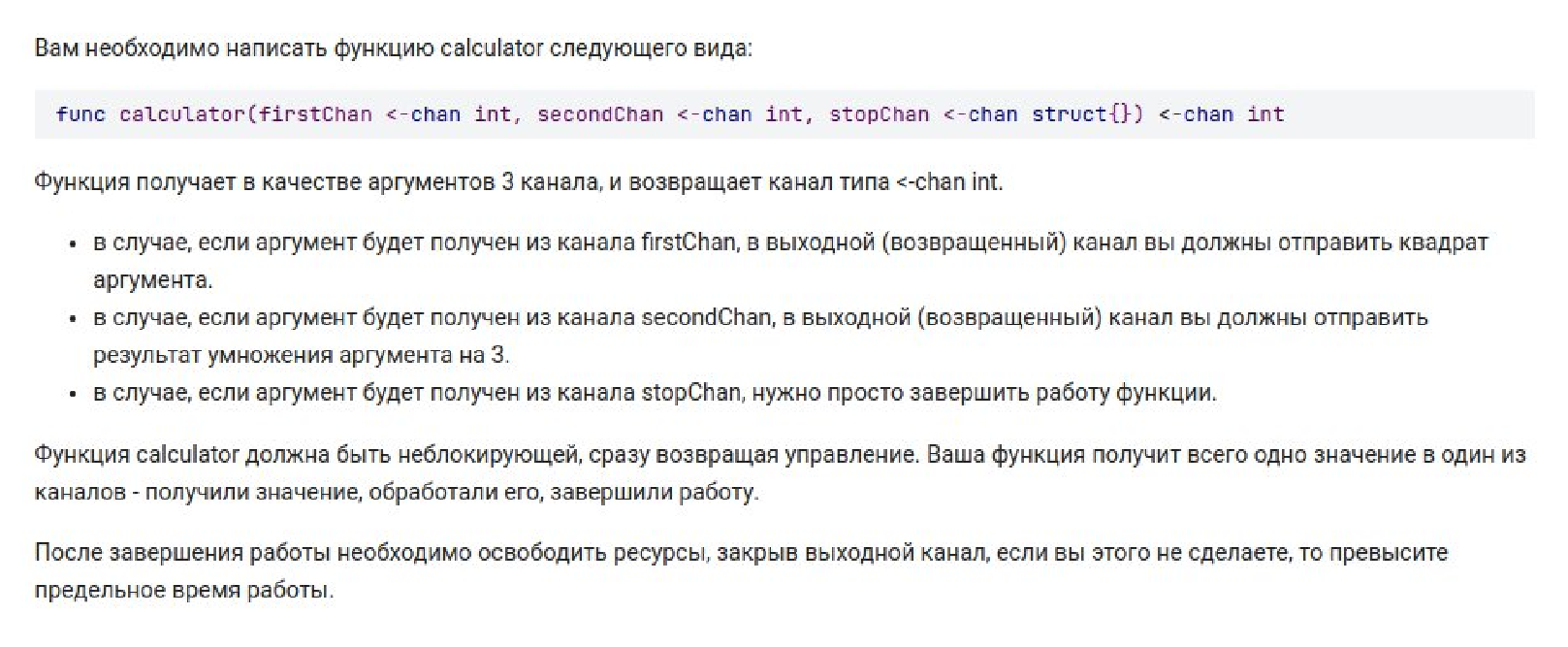


Рисунок 1

**Решение:** package main import "fmt"

// calculator - функция, которая принимает три канала:

// firstChan, secondChan, stopChan, и возвращает канал resChan типа <-chan int.

// Она запускает горутину, которая обрабатывает значения из каналов.

func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {

resChan := make(chan int) // Создаем канал для результата go func() { // Запускаем анонимную горутину

defer close(resChan) // Закрываем канал при завершении // select позволяет ожидать значения из любого из каналов select {

// Получение значения из firstChan case val := <-firstChan: // Получаем значение из канала

resChan <- val \* val // Квадрат значения и отправка в resChan

// Получение значения из secondChan case val := <-secondChan: // Получаем значение из канала

resChan <- val \* 3 // Умножение значения на 3 и отправка в resChan

// Получение сигнала из stopChan case <-stopChan: // Получаем сигнал из канала

return // Завершаем горутину

}

}()

return resChan // Возвращаем канал результата

}

func main() {

ch1 := make(chan int) // Создаем канал ch1 ch2 := make(chan int) // Создаем канал ch2 ch3 := make(chan struct{}) // Создаем канал ch3 (для сигнала остановки) res := calculator(ch1, ch2, ch3) // Вызываем функцию calculator и получаем канал res ch2 <- 16 // Отправляем значение 16 в ch2 fmt.Println(<-res) // Получаем результат из канала res и выводим его

}

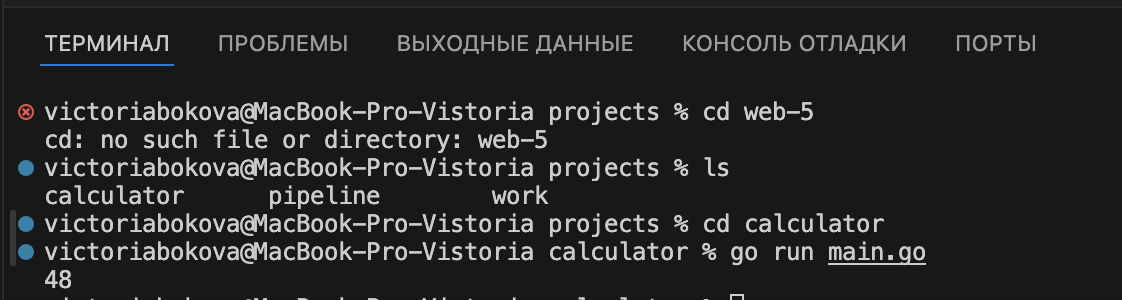


Рисунок 2 – вывод результата

**B) Задача «Pipeline» (рис 3)**

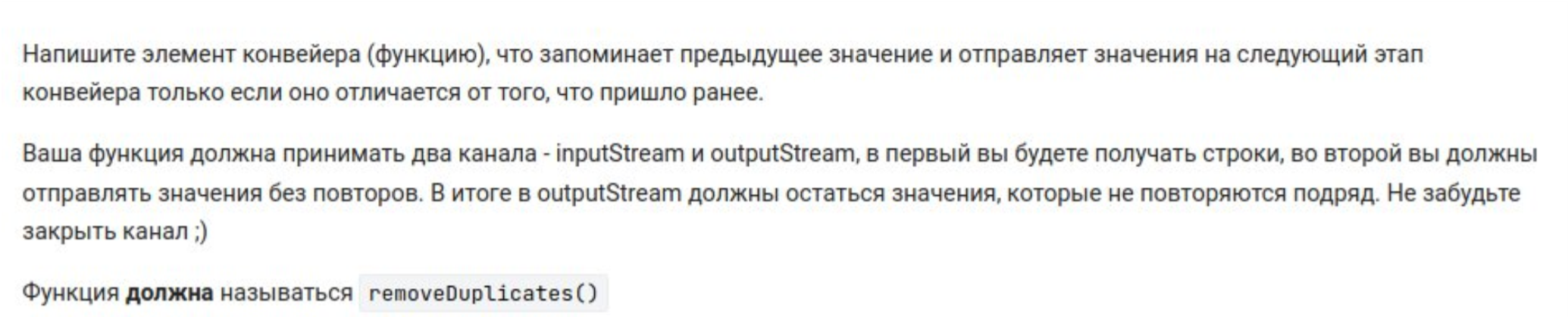


Рисунок 3

**Решение:** package main

import "fmt" func removeDuplicates(inputStream <-chan string, outputStream chan<- string) {

defer close(outputStream) // Закрываем выходной канал при завершении var prevValue string // Храним предыдущее значение for value := range inputStream {

if value != prevValue { // Проверка на дубликат

outputStream <- value // Отправка в выходной канал, если значение не дубликат prevValue = value // Обновляем предыдущее значение

}

}

}

func main() {

in := make(chan string) out := make(chan string) go func() {

defer close(in) // Закрываем входной канал при завершении in <- "one" in <- "two" in <- "two" in <- "three" in <- "three" in <- "three" in <- "four" in <- "five"

}()

go removeDuplicates(in, out) for value := range out { fmt.Println(value)

}

}

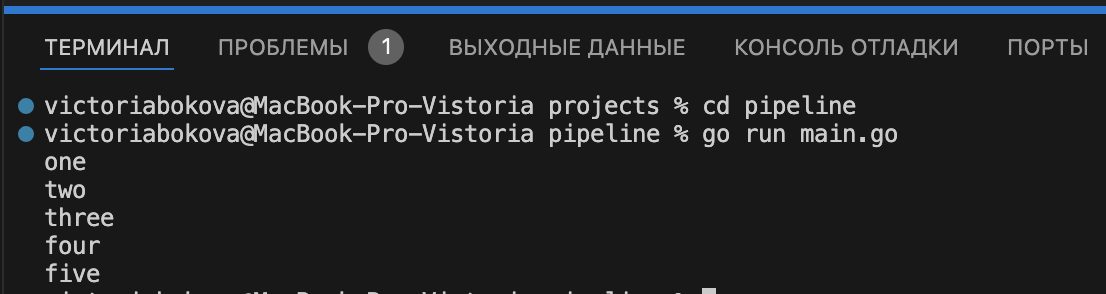


Рисунок 4 – результат программы

**C) Задача «Work» (рис 5):**

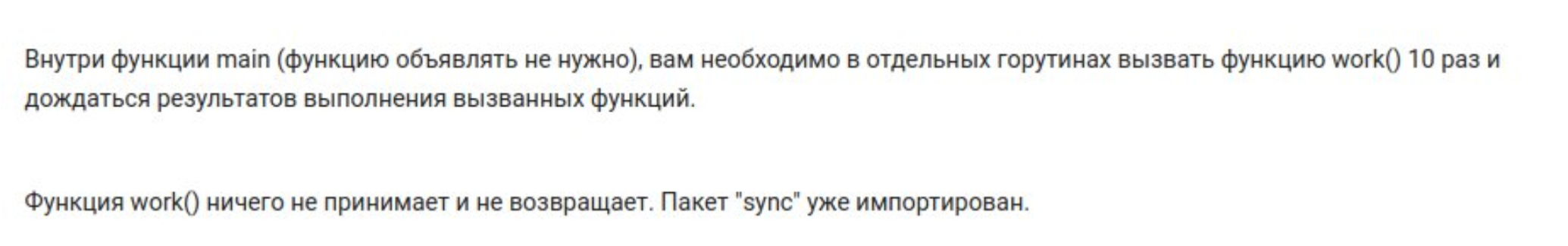


Рисунок 5

**Решение:**

package main

import (

"fmt"

"sync"

)

func work() {

// Здесь ваш код, который выполняется в каждой горутине fmt.Println("WORK!")

}

func main() {

var wg sync.WaitGroup // Создаем WaitGroup для синхронизации

wg.Add(10) // Увеличиваем счетчик на 10, так как запускаем 10 горутин

for i := 0; i < 10; i++ {

go func() {

defer wg.Done() // Уменьшаем счетчик после завершения горутины work() // Вызываем функцию work

}() }

wg.Wait() // Ждем завершения всех горутин fmt.Println("Все горутины завершились!")

}

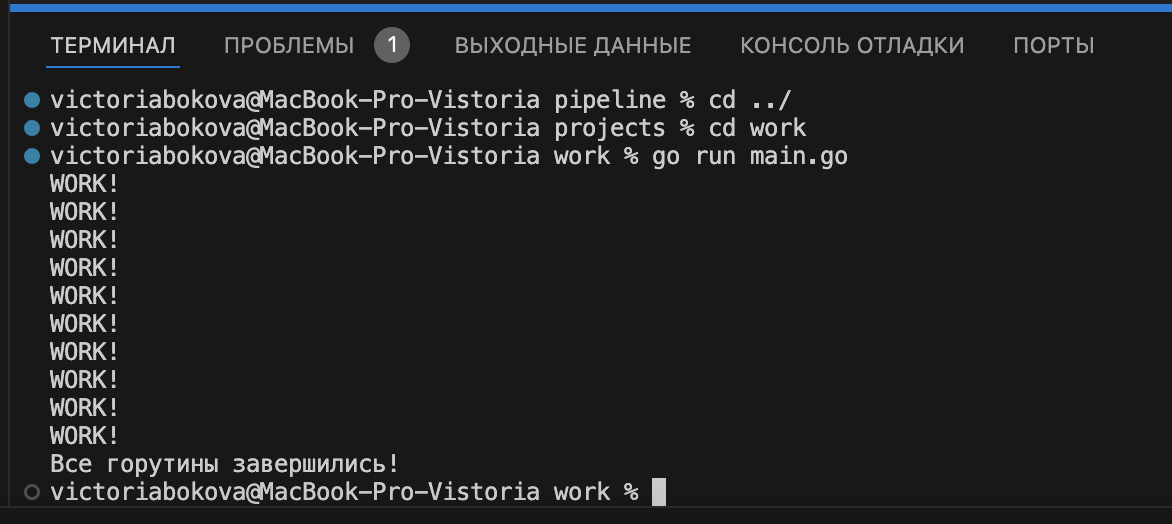


Рисунок 6 - тестирование

**Заключение:** В процессе выполнения лабораторной работы были изучены основы асинхронного программирования на Golang, а также получены практические навыки написания программ с использованием данной концепции программирования.

**Использованные источники**

* https://github.com/ValeryBMSTU/web-5
* https://stepik.org/course/54403/info