|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 Название направления «Информатика и вычислительная техника»**

**Отчет**

**по лабораторной работе № 5**

**Название: Основы асинхронного программирования на Golang**

**Дисциплина:** Языки интернет программирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-32Б |  |  | Е.Б. Гаппова |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | В.Д. Шульман |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

Цель работы — изучение основ асинхронного программирования с использованием языка Golang.

**Задание 1**

Вам необходимо написать функцию calculator следующего вида:

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

Функция получает в качестве аргументов 3 канала, и возвращает канал типа <-chan int.

в случае, если аргумент будет получен из канала firstChan, в выходной (возвращенный) канал вы должны отправить квадрат аргумента.

в случае, если аргумент будет получен из канала secondChan, в выходной (возвращенный) канал вы должны отправить результат умножения аргумента на 3.

в случае, если аргумент будет получен из канала stopChan, нужно просто завершить работу функции.

Функция calculator должна быть неблокирующей, сразу возвращая управление. Ваша функция получит всего одно значение в один из каналов - получили значение, обработали его, завершили работу.

После завершения работы необходимо освободить ресурсы, закрыв выходной канал, если вы этого не сделаете, то превысите предельное время работы.

Код программы

package main

import "fmt"

func main() {

    c1 := make(chan int)

    c2 := make(chan int)

    s := make(chan struct{})

    var r int = 1000000

    go func(r \*int) {

        \*r = <-calculator(c1, c2, s)

    }(&r)

    c1 <- 3

    //c2 <- 2

    //s <- struct{}{}

    fmt.Print(r)

}

func calculator(firstChan <-chan int, secondChan <-chan int,

    stopChan <-chan struct{}) <-chan int {

    outp := make(chan int)

    go func(outp chan int) {

        defer close(outp)

        select {

        case a := <-firstChan:

            outp <- a \* a

        case a := <-secondChan:

            outp <- a \* 3

        case <-stopChan:

            return

        }

    }(outp)

    return outp

}

Тестирование программы

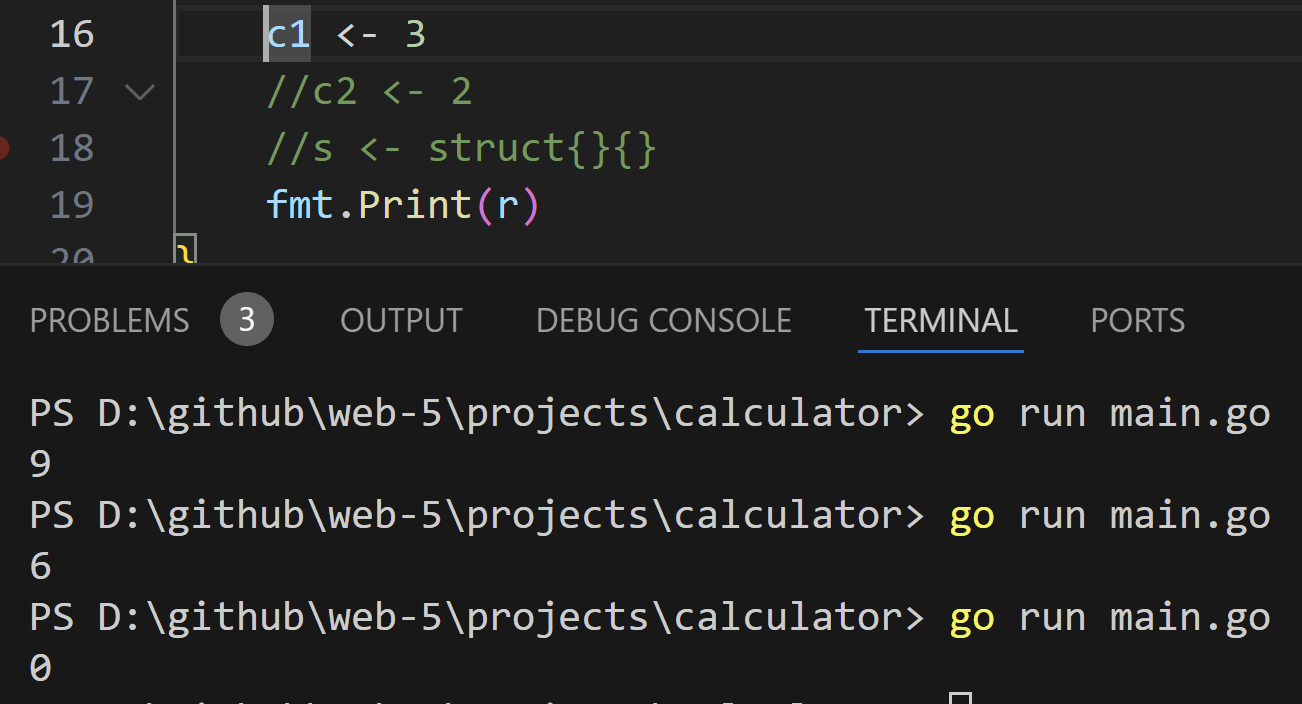


Рисунок 1 – тестирование программы

На рисунке 1 представлены тесты при:

1) ввод в первый канал – возведение в квадрат

2)ввод во второй канал - умножение на 3

3)ввод в третий канал - ничего происходит, выведено значение int по умолчанию

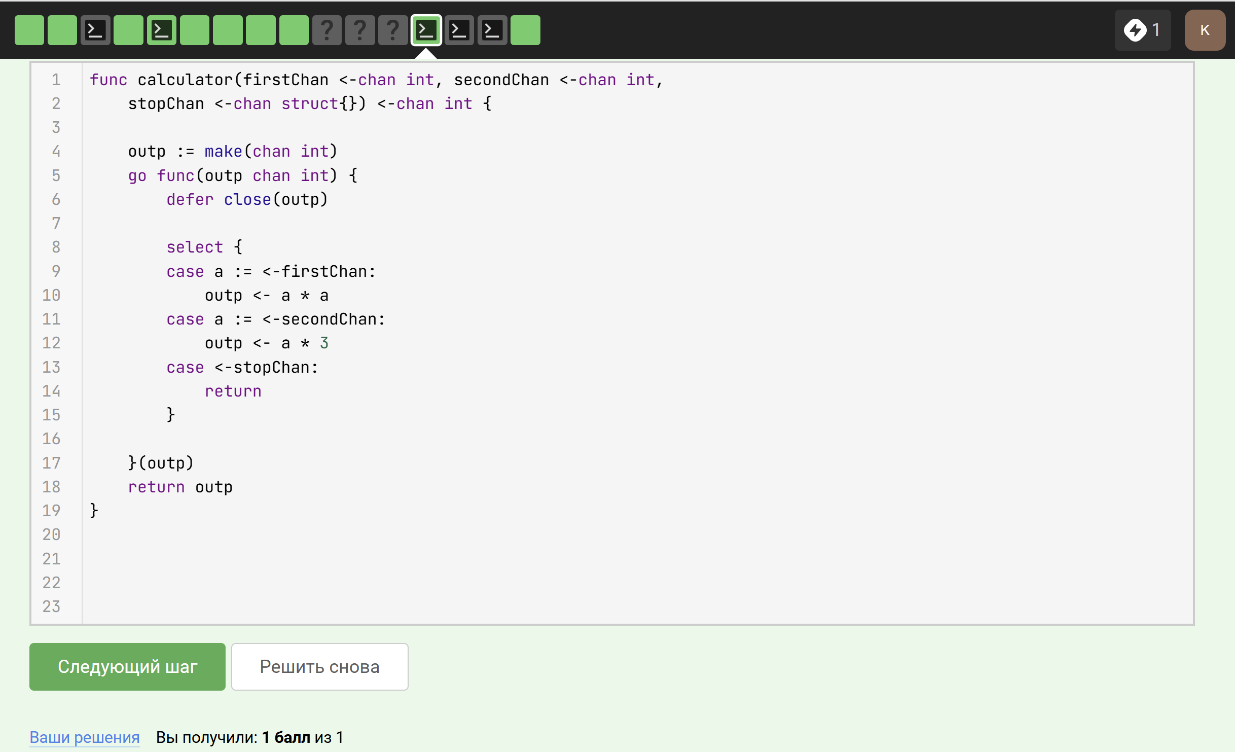


Рисунок 2 – тесты на stepik прошли

**Задание 2**

Напишите элемент конвейера (функцию), что запоминает предыдущее значение и отправляет значения на следующий этап конвейера только если оно отличается от того, что пришло ранее.

Ваша функция должна принимать два канала - inputStream и outputStream, в первый вы будете получать строки, во второй вы должны отправлять значения без повторов. В итоге в outputStream должны остаться значения, которые не повторяются подряд. Не забудьте закрыть канал ;)

Функция **должна** называться removeDuplicates()  
  
Выводить или вводить ничего не нужно!

package main

import "fmt"

func removeDuplicates(inputStream, outputStream chan string) {

    var tmp string = ""

    for v := range inputStream {

        if v != tmp {

            outputStream <- v

        }

        tmp = v

    }

    close(outputStream)

}

func main() {

    inputStream := make(chan string)

    outputStream := make(chan string)

    var s string = "4444444444eiei4"

    go removeDuplicates(inputStream, outputStream)

    go func() {

        for v := range outputStream {

            fmt.Print(v)

        }

    }()

    for \_, a := range s {

        inputStream <- string(a)

    }

Тестирование программы

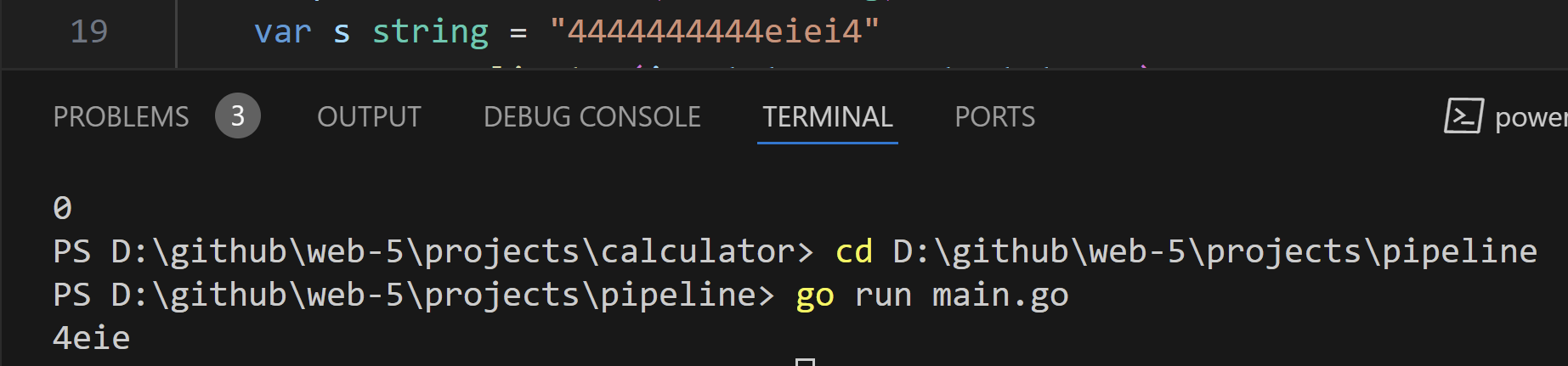


Рисунок 3 – тест 1

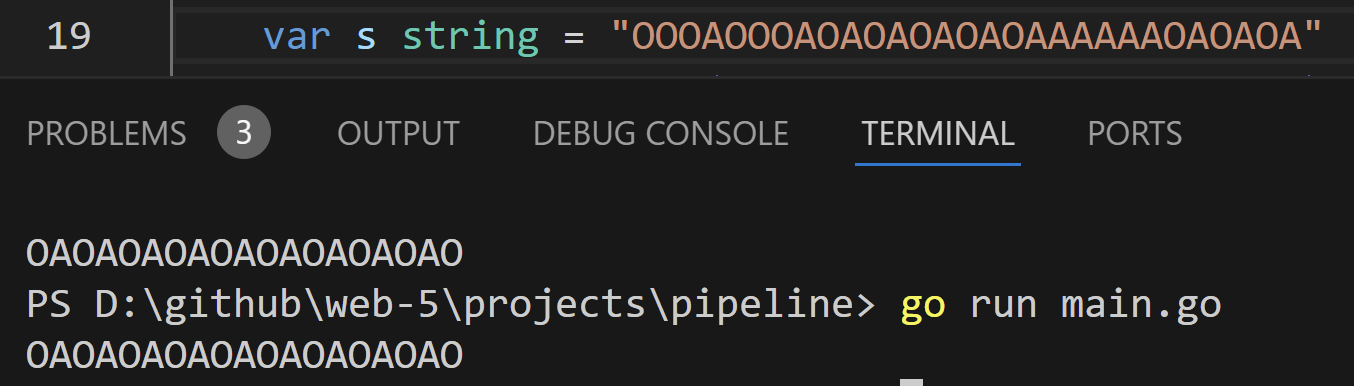


Рисунок 4 – тест 2



Рисунок 5 - тесты на stepik прошли

**Задание 3**

Внутри функции main необходимо в отдельных горутинах вызвать функцию work() 10 раз и дождаться результатов выполнения вызванных функций.

Функция work() ничего не принимает и не возвращает. Пакет "sync" уже импортирован.

Код программы:

package main

import (

    "fmt"

    "sync"

    "time"

)

func work() {

    time.Sleep(time.Millisecond \* 50)

    fmt.Println("done")

}

func main() {

    wg := new(sync.WaitGroup)

    for i := 0; i < 10; i++ {

        wg.Add(1)

        go func(wg \*sync.WaitGroup) {

            defer wg.Done()

            work()

        }(wg)

    }

    wg.Wait()

}

Тестирование программы

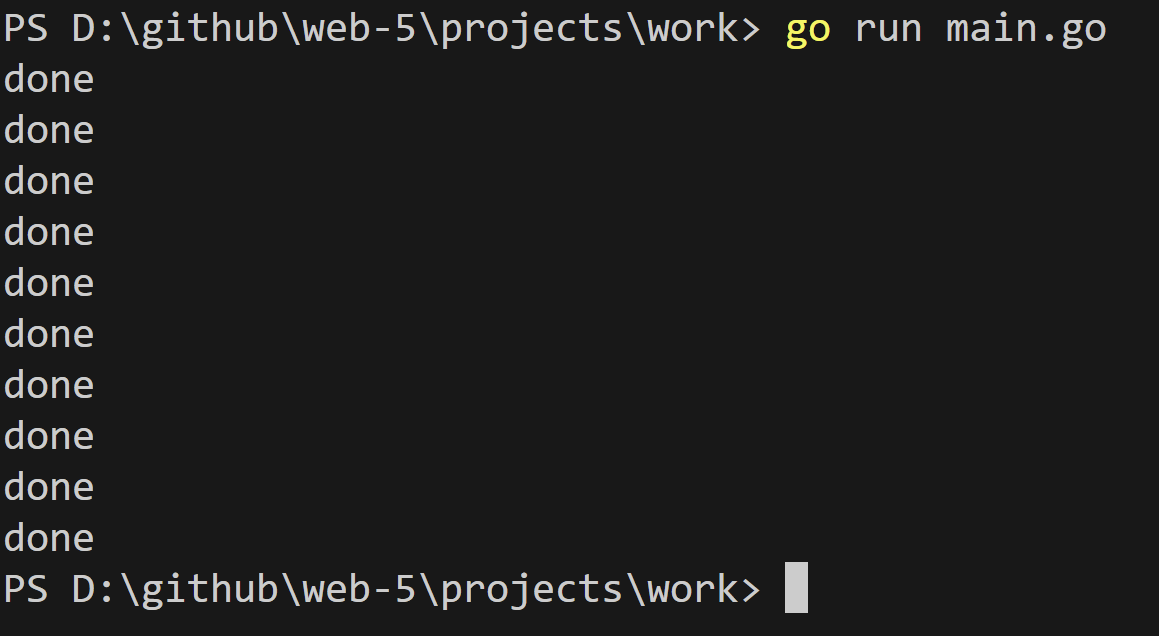


Рисунок – тестирование программы

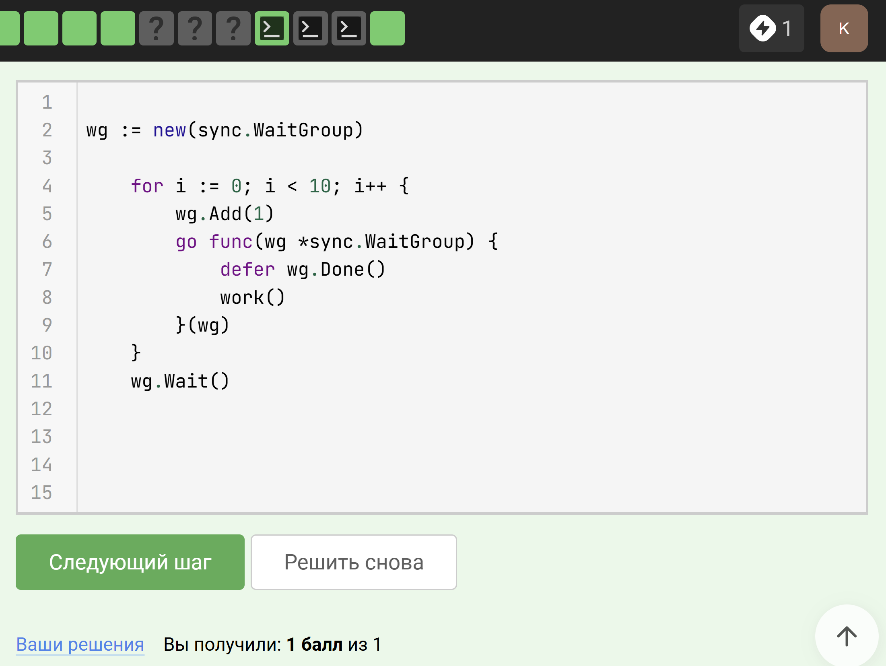


Рисунок – тесты на stepik прошли

Вывод: таким образом были освоены основы асинхронного программирования на Golang.

Источники:

1. <https://stepik.org/lesson/345547/step/13?unit=329291>
2. <https://stepik.org/lesson/360357/step/10?unit=344766>
3. <https://stepik.org/lesson/345547/step/5?&unit=329291>