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**

О Т Ч Е Т

по лабораторной работе № 5

Название: Основы асинхронного программирования на Golang

Дисциплина: Языки интернет-программирования

Студент	ИУ6-32Б		22.10.24	В.А. Баринова
	(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: в рамках данной лабораторной работы предлагается продолжить изучение Golang и познакомиться с продвинутыми конструкциями языка.

Ход работы

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.

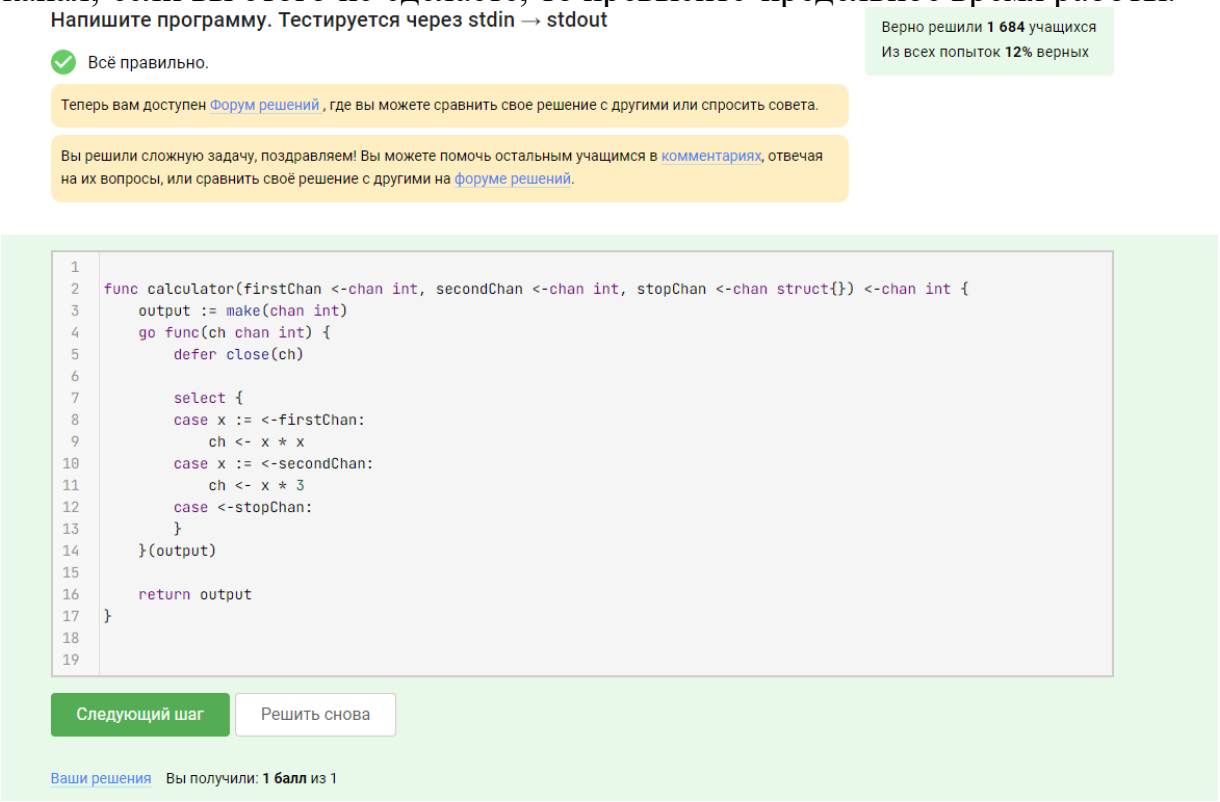

Рисунок 1—Прохождение тестов по заданию 1 на Stepik

Проведем тестирование программы, приведенной далее.

```
package main

import "fmt"

// calculator принимает два канала чисел и один канал для остановки,
// а затем возвращает канал с обработанными значениями.
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan
struct{}) <-chan int {
    outChan := make(chan int)

    go func() {
        defer close(outChan)

        for {
            select {
            case val := <-firstChan:
                outChan <- val * val // отправляем квадрат значения
            case val := <-secondChan:
                outChan <- val * 3 // отправляем значение, умноженное на 3
            case <-stopChan:
                return // выход из функции при получении сигнала остановки
            }
        }
    }()

    return outChan
}

func main() {
    firstChan := make(chan int)
    secondChan := make(chan int)
    stopChan := make(chan struct{})

    outChan := calculator(firstChan, secondChan, stopChan)

    // Пример использования
    go func() {
        firstChan <- 6
        secondChan <- 1
        stopChan <- struct{}{}
    }()

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

```
[Running] go run "c:\github\laba-5\projects\calculator\main.go"
36
3
```

Рисунок 2—Результаты тестирования (задание 1)

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()`

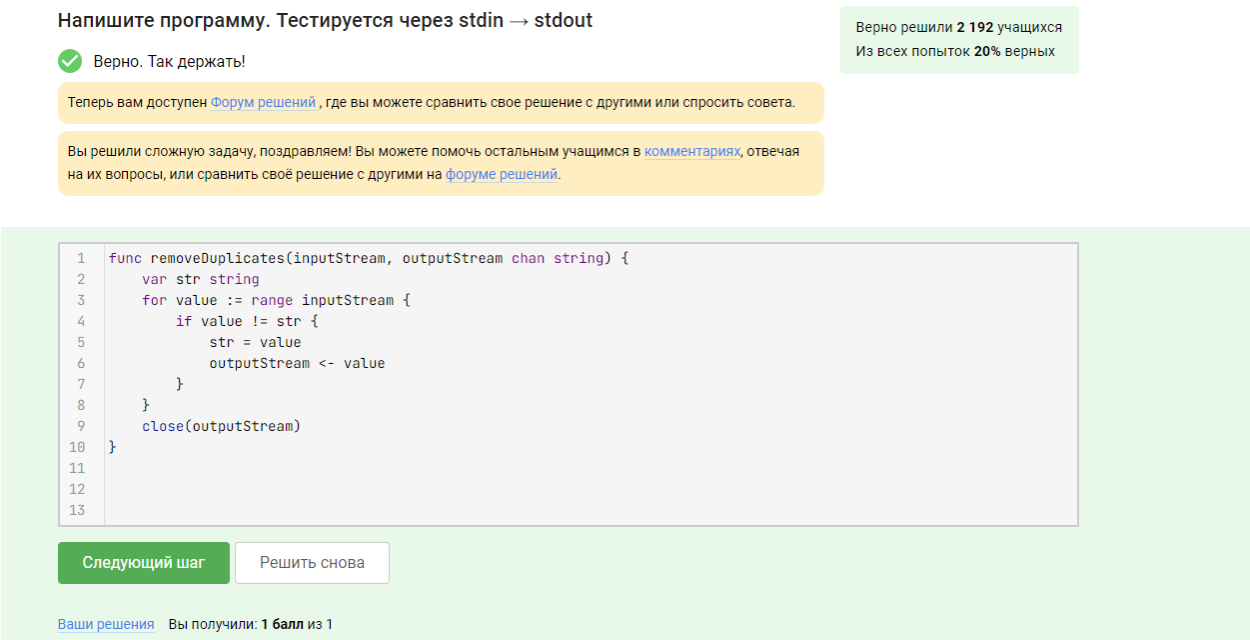

Рисунок 3—Прохождение тестов по заданию 2 на Stepik

Проведем тестирование программы, приведенной далее.

```
package main

import "fmt"

func removeDuplicates(inputStream, outputStream chan string) {
    var str string
    for value := range inputStream {
```

```
        if value != str {
            str = value
            outputStream <- value
        }
    }
    close(outputStream)
}

func main() {
    inputStream := make(chan string)
    outputStream := make(chan string)
    go removeDuplicates(inputStream, outputStream)

    go func() {
        defer close(inputStream)

        for _, r := range "112334456" {
            inputStream <- string(r)
        }
    }()

    for x := range outputStream {
        fmt.Print(x)
    }
    // 123456
}
```

```
[Running] go run "c:\github\laba-5\projects\pipeline\main.go"
123456
[Done] exited with code=0 in 1.485 seconds
```

Рисунок 4—Результаты тестирования (задание 2)

Задание 3.

Внутри функции main (функцию объявлять не нужно), вам 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.

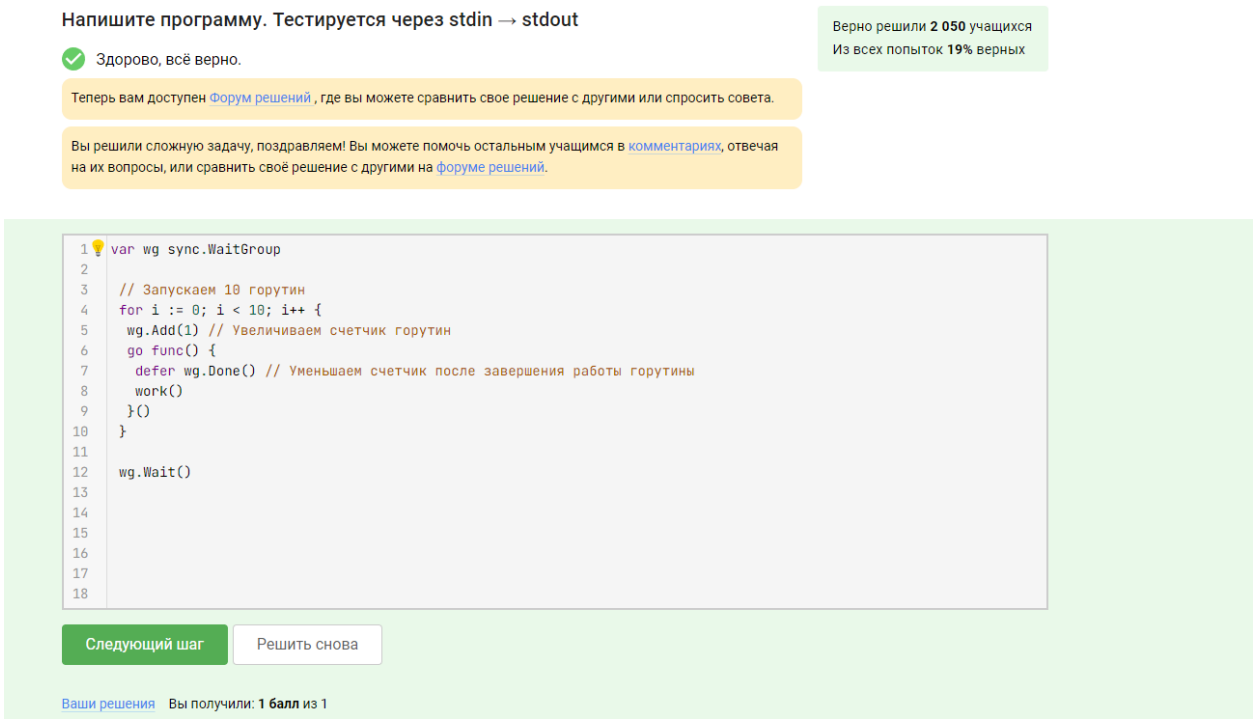

Рисунок 5—Прохождение тестов по заданию 3 на Stepik

Проведем тестирование программы, приведенной далее.

```go
package main

import (
    "fmt"
    "sync"
)

func work() {
    fmt.Println("Работа выполнена")
}

func main() {
    var wg sync.WaitGroup

    // Запускаем 10 горутин
    for i := 0; i < 10; i++ {
        wg.Add(1) // Увеличиваем счетчик горутин
        go func() {
            defer wg.Done() // Уменьшаем счетчик после завершения работы горутины
            work()          // Вызываем функцию work
        }()
    }

    // Ожидаем завершения всех горутин
    wg.Wait()
    fmt.Println("Все работы завершены")
```

```
}
```

```
[Running] go run "c:\github\laba-5\projects\work\main.go"
Работа выполнена
Работа выполнена
Работа выполнена
Работа выполнена
Работа выполнена
Работа выполнена
Работа выполнена
Работа выполнена
Работа выполнена
Работа выполнена
Все работы завершены

[Done] exited with code=0 in 1.384 seconds
```

Рисунок 6—Результаты тестирования (задание 3)

Заключение: в рамках данной лабораторной работы было продолжено изучение Golang и ознакомление с продвинутыми конструкциями языка.

Список использованных источников

1. https://stepik.org/course/54403/info
2. https://go.dev/doc