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 ИУ6-33Б _____ Н.Р. Гусниев
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ _____
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

Цель работы

изучение основ асинхронного программирования с использованием языка Golang. В рамках данной лабораторной работы предлагается продолжить изучение Golang и познакомиться с продвинутыми конструкциями языка.

Задание

Задание1 - https://stepik.org/lesson/345547/step/13?unit=329291
Задание2 - https://stepik.org/lesson/360357/step/10?unit=344766
Задание3 - https://stepik.org/lesson/345547/step/5?&unit=329291

Ход работы

Задание 1

```go
package main

import (
    "fmt"
    "time"
)

func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-
chan struct{}) <-chan int {
    out := make(chan int)

    go func() {
       defer close(out)
       select {
       case x := <-firstChan:
          out <- x * x
       case x := <-secondChan:
          out <- x * 3
       case <-stopChan:
          return
       }
    }()

    return out
}

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

    resultChan := calculator(firstChan, secondChan, stopChan)

    // Пример 1: Отправка значения в первый канал (квадрат числа)
    go func() {
       firstChan <- 4
    }()

    // Пример 2: Отправка значения во второй канал (умножение на 3)
    //go func() {
    // secondChan <- 5
    //}()

    // Пример 3: Завершение работы через канал stopChan
    // go func() {
    //     time.Sleep(1 * time.Second)
    //     close(stopChan)
    // }()

    select {
    case result := <-resultChan:
       fmt.Println("Результат:", result)
    case <-time.After(2 * time.Second):
       fmt.Println("Таймаут!")
    }
}
```

Main.go

Напишите программу. Тестируется через stdin → stdout

Верно решили **1 684** учащихся
Из всех попыток **12%** верных

Верно. Так держать!

Теперь вам доступен Форум решений , где вы можете сравнить свое решение с другими или спросить совета.

Вы решили сложную задачу, поздравляем! Вы можете помочь остальным учащимся в комментариях, отвечая на их вопросы, или сравнить своё решение с другими на форуме решений.

```go
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
    out := make(chan int)

    go func() {
        defer close(out)
        select {
        case x := <-firstChan:
            out <- x * x
        case x := <-secondChan:
            out <- x * 3
        case <-stopChan:
            return
        }
    }()

    return out
}
```

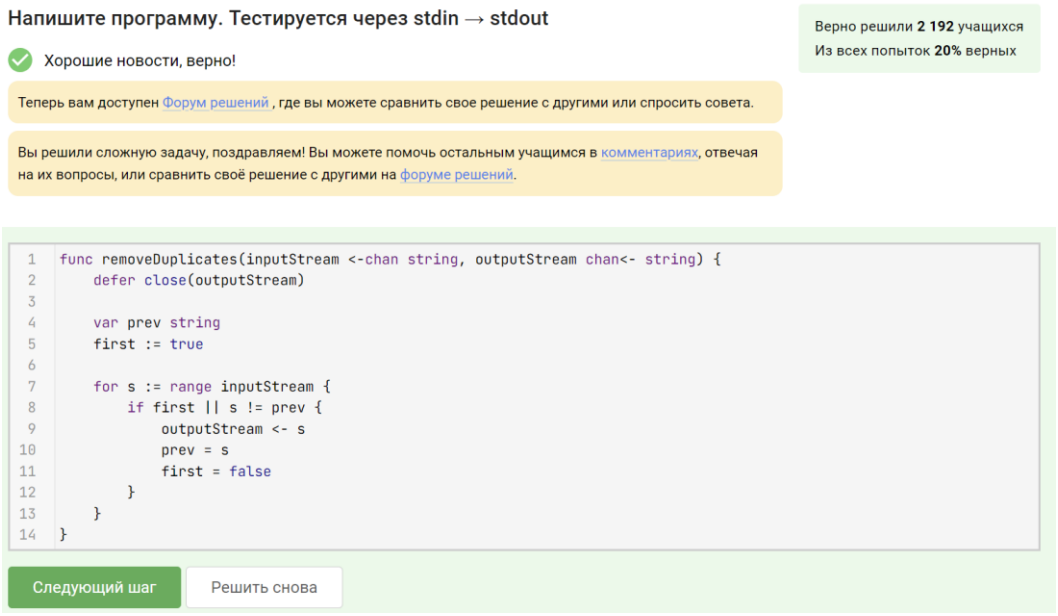
Следующий шаг | Решить снова

Результат

Задание 2

```go
package main

import "fmt"

func removeDuplicates(inputStream <-chan string, outputStream
chan<- string) {
    defer close(outputStream)

    var prev string
    first := true

    for s := range inputStream {
       if first || s != prev {
          outputStream <- s
          prev = s
          first = false
       }
    }
}

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

    // Запускаем функцию removeDuplicates в отдельной горутине
    go removeDuplicates(inputStream, outputStream)

    // Отправляем данные в inputStream
    go func() {
       values := []string{"apple", "apple", "banana",
"banana", "apple", "pear", "pear", "banana"}
       for _, v := range values {
          inputStream <- v
       }
       close(inputStream) // Закрываем канал, чтобы завершить
цикл в removeDuplicates
    }()

    // Читаем данные из outputStream и выводим их
    for v := range outputStream {
       fmt.Println(v)
    }
}
```

Main.go

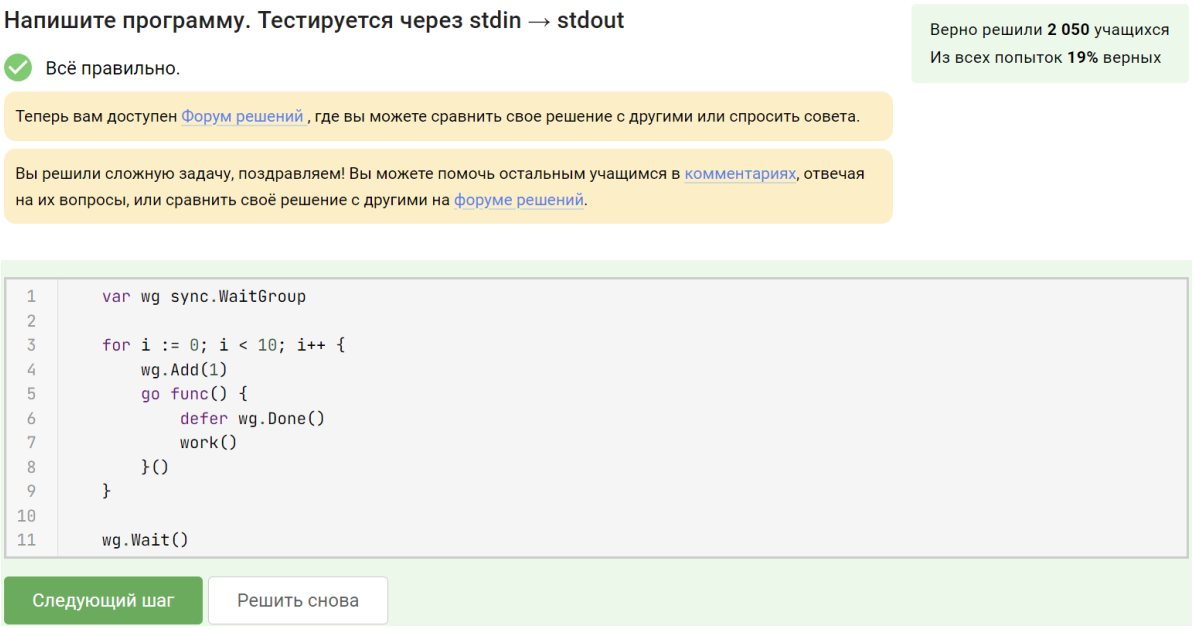


Результат

Задание 3

```go
package main

import (
    "fmt"
    "sync"
    "time"
    // "sync"
)

func work() {
    time.Sleep(time.Millisecond * 50)
    fmt.Println("done")
}

func main() {
    var wg sync.WaitGroup

    for i := 0; i < 10; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            work()
        }()
    }

    wg.Wait()
}
```

Main.go



Результат

Заключение

В ходе лабораторной работы изучены основы асинхронного программирования на языке Go, такие как использование горутин, каналов и механизмов синхронизации. Были выполнены все задания, что позволило закрепить навыки параллельного выполнения задач и обмена данными между потоками. Цели работы достигнуты.