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**

О Т Ч Е Т

по лабораторной работе № 5

Название: **Основы асинхронного программирования на Golang**

Дисциплина: **Языки интернет-программирования**

Студент ИУ6-31Б _____ Н.Е. Мамаев
(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. 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.

Фрагмент кода программы изображен на рисунке 1.

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

wg.Wait()
```

Рисунок 1: Фрагмент кода программы 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() Выводить или вводить ничего не нужно!

Фрагмент кода программы приведен на рисунке 2.

```go
func removeDuplicates(inputStream, outputStream chan string){
    tmp1 := <- inputStream
    outputStream <- tmp1
    for tmp := range inputStream {
        if tmp != tmp1 {
            outputStream <- tmp
            tmp1 = tmp
        }
    }
    close(outputStream)
}
```

Рисунок 2: Фрагмент кода программы 2.

Задание 3. 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.

Фрагмент кода программы приведен на рисунке 3.

```go
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int{
    result := make(chan int)
    go func(){
        defer close(result)
        select{
        case a := <- firstChan:
            result <- a*a
        case b := <- secondChan:
            result <- 3*b
        case <- stopChan:
            return
        }
    }()
    return result
}
```

Рисунок 3: Фрагмент кода программы 3.

Вывод: в ходе лабораторной изучены основы работы асинхронного программирования с использованием golang.