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)**

О Т Ч Е Т

по лабораторной работе № 5

Название: **Основы асинхронного программирования на Golang**

Дисциплина: **Языки интернет программирования**

Студент ИУ6-32Б _____ Кирикович М.А
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2024

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

Часть 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 - Задание 1

Ниже приведен код решающий данную задачу.

```go
package main

import (
	"fmt"
	"time"
)

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

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

}
func main() {

	ch1 := make(chan int, 1)
	ch2 := make(chan int, 1)
	ch3 := make(chan struct{}, 1)
	defer close(ch1)
	defer close(ch2)
	defer close(ch3)
	// ch3 <- a
	var a int
	var num int
	fmt.Println("Введите число")
	fmt.Scan(&a)
	fmt.Println("Введите номер канала")
	fmt.Scan(&num)

	switch {
	case num == 1:
		ch1 <- a
	case num == 2:
		ch2 <- a
	case num == 3:
		ch3 <- struct{}{}
	}
	fmt.Println(<-calc(ch1, ch2, ch3))
	time.Sleep(time.Second)
}

```

```
maksim@Maksim:~/web/web-5/projects/calculator$ go run main.go
Введите число
2
Введите номер канала
2
6
maksim@Maksim:~/web/web-5/projects/calculator$ go run main.go
Введите число
2
Введите номер канала
1
4
```

Рисунок 2 - Тестирование программы

Часть 2

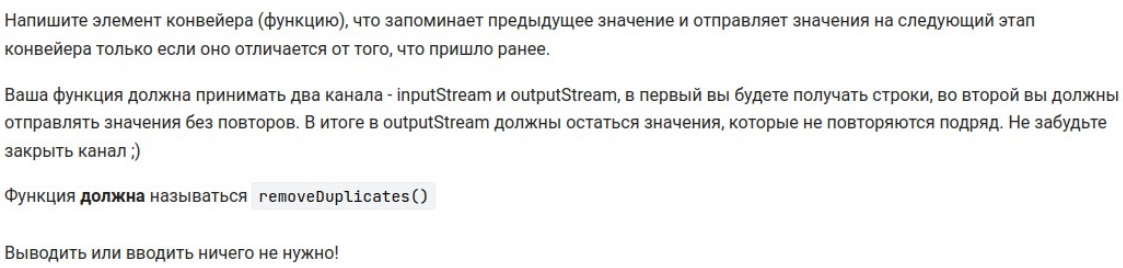

Рисунок 3 - Задание

Код приведен ниже.

```
1  package main
2
3  import (
4          "fmt"
5          "time"
6  )
7
8  func removeDuplicates(in, out chan string) {
9          defer close(out)
10         var temp string
11         for val := range in {
12                 if (temp) != val {
13                         out <- val
14
15                 }
16                 temp = val
17                 time.Sleep(time.Millisecond * 20)
18         }
19
20 }
21 func main() {
22         input := make(chan string)
23         output := make(chan string)
24         var a string
25         fmt.Scan(&a)
26         go func() {
27                 for _, r := range a {
28                         input <- string(r)
29                 }
30                 defer close(input)
31         }()
32         go removeDuplicates(input, output)
33         go func() {
34                 for val := range output {
35                         fmt.Print(val)
36                 }
37         }()
38         time.Sleep(time.Second)
39         fmt.Println()
40 }
41
```

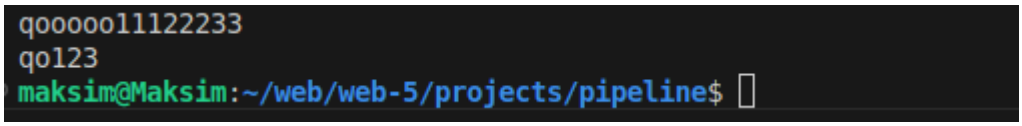

Рисунок 4 - Результат тестирования программы

Часть 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 - Задание 3

Код приведен ниже.

```go
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(i int) {
			work()
			defer wg.Done()
		}(i)
	}
	wg.Wait()
}
```

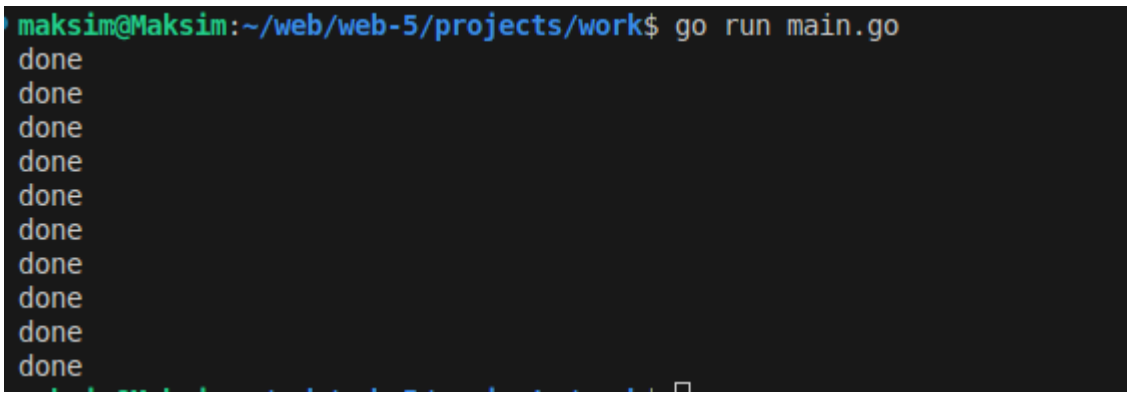

Рисунок 6 - Тестирование программы

Вывод: были изучены основы асинхронного программирования