



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

(Группа)

 22.09.2024  
(Подпись, дата)

А.Д. Мартыненко  
(И.О. Фамилия)

Преподаватель

 28.09.2024  
(Подпись, дата)

В.Д. Шульман  
(И.О. Фамилия)

Москва, 2024

## 1) Цель лабораторной работы

Изучение основ асинхронного программирования с использованием языка Golang.

## 2) Задание

Продолжить изучение Golang и познакомиться с продвинутыми конструкциями языка.

## 3) Ход работы

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

*Код на языке 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, wg *sync.WaitGroup) {
            defer wg.Done()
            work()
        }(i, wg)
    }
    wg.Wait()
}
```

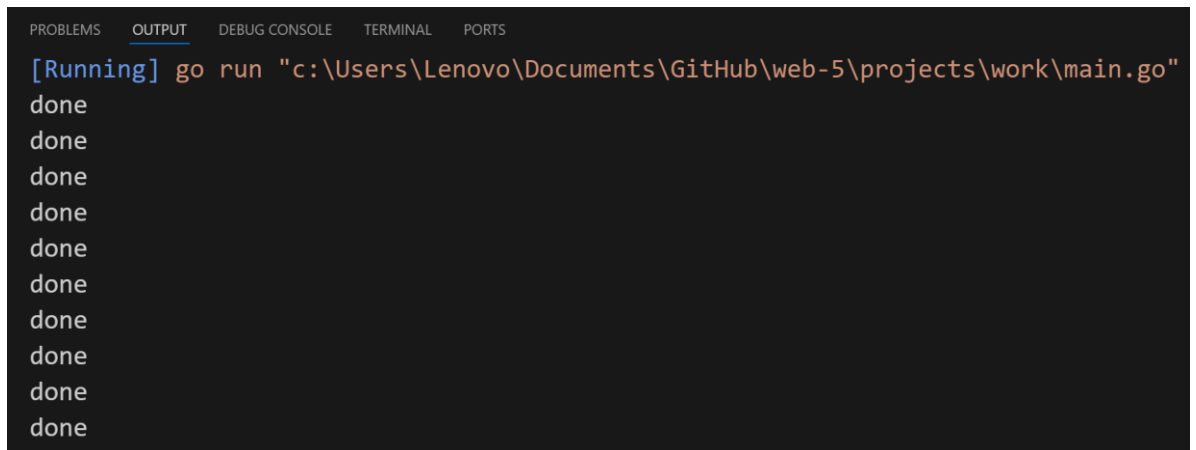
```

    }(i, wg)
}

wg.Wait()
}

```

*Результат:*



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[Running] go run "c:\Users\Lenovo\Documents\GitHub\web-5\projects\work\main.go"
done
done
done
done
done
done
done
done
done
done
done
done

```

*Рисунок 1 – Результат работы программы*

2. *Pipeline*. 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.

*Код на языке Go:*

```

package main

import (
    "fmt"
)

func removeDuplicates(in, out chan string) {
    var buf []string
    for i := range in {

```

```

        flag := false
        for _, j := range buf {
            if i == j {
                flag = true
            }
        }
        if flag == false {
            out <- i
        }
        buf = append(buf, i)
    }
    close(out)
}

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)
    }
}

```

*Результат:*



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[Running] go run "c:\Users\Lenovo\Documents\GitHub\web-5\projects\pipeline\main.go"
123456
[Done] exited with code=0 in 1.089 seconds

```

*Рисунок 2 – Результат работы программы*

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.

*Код на языке Go:*

```
package main

import (
    "fmt"
)

func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
    resChan := make(chan int)
    go func() {
        defer close(resChan)
        select {
        case val := <-firstChan:
            resChan <- val * val
        case val := <-secondChan:
```

```

        resChan <- val * 3
    case <-stopChan:
        return
    }
}()
return resChan
}

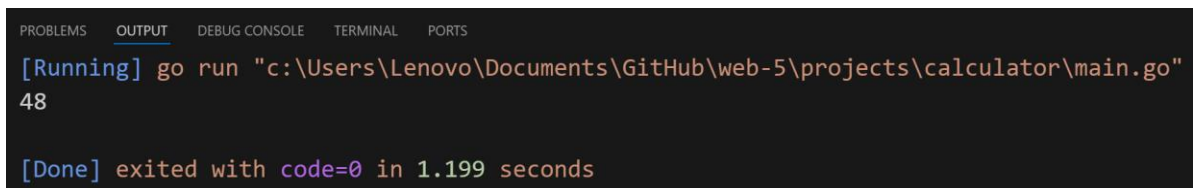
func main() {
    ch1 := make(chan int)
    ch2 := make(chan int)
    ch3 := make(chan struct{})
    res := calculator(ch1, ch2, ch3)

    ch2 <- 16

    fmt.Println(<-res)
}

```

*Результат:*



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[Running] go run "c:\Users\Lenovo\Documents\GitHub\web-5\projects\calculator\main.go"
48

[Done] exited with code=0 in 1.199 seconds

```

*Рисунок 3 – Результат работы программы*

#### 4) Заключение

В процессе выполнения лабораторной работы были изучены основы асинхронного программирования на Golang, а также получены практические навыки написания программ с использованием данной концепции программирования.

#### 5) Использованные источники

- <https://github.com/ValeryBMSTU/web-5>
- <https://stepik.org/course/54403/info>