



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

(Группа)

(Подпись, дата)

А.С.Авдеева

(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 calculator– 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.

Ход работы и результаты:

1) Написание кода на Golang:

```
package main  
  
import "fmt"  
  
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan
```

```

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

    return a
}

func main() {
    ch1, ch2 := make(chan int), make(chan int)
    stop := make(chan struct{})
    r := calculator(ch1, ch2, stop)
    ch1 <- 3
    //ch2 <- 4
    //close(stop)
    fmt.Println(<-r)
}

```

2)Результат, если поступает из первого канала число 3(рис.№1)

```

19
20     return a
21 }
22
23 func main() {
24     ch1, ch2 := make(chan int), make(chan int)
25     stop := make(chan struct{})
26     r := calculator(ch1, ch2, stop)
27     ch1 <- 3
28     //ch2 <- 4
29     //close(stop)
30     fmt.Println(<-r)
31 }
32
calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int

```

Run go build main.go x

```

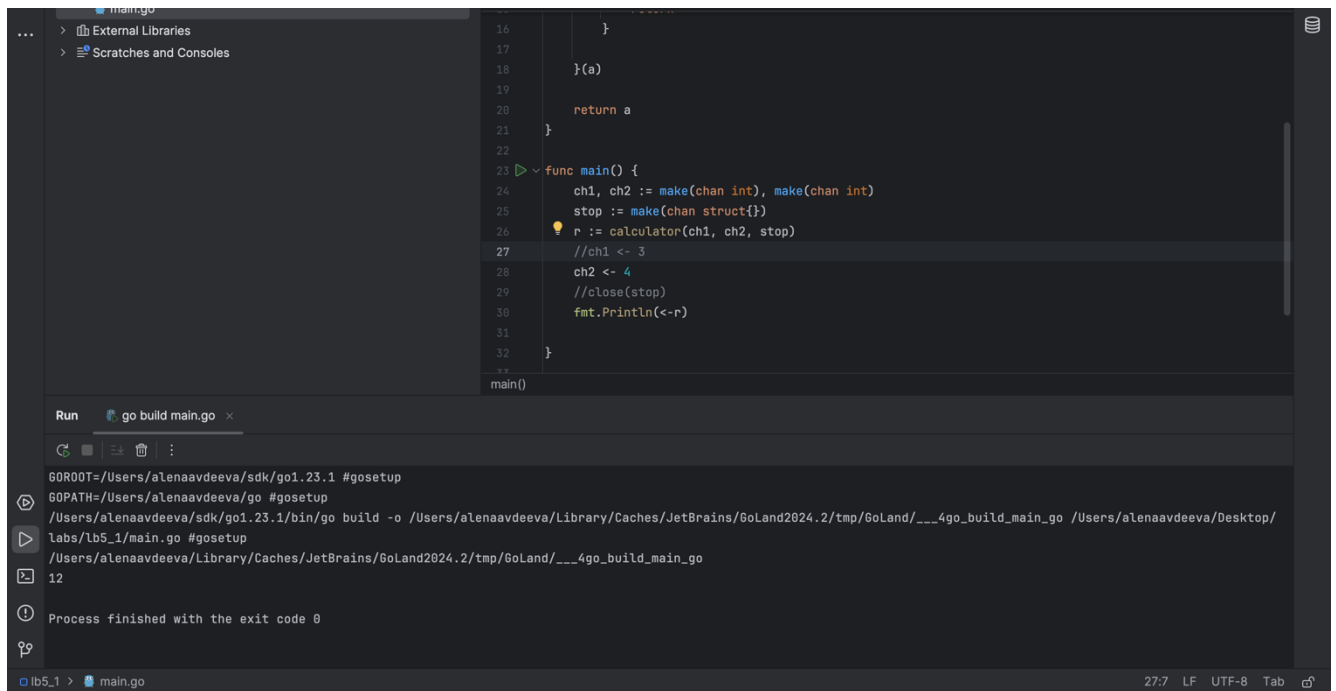
GOROOT=/Users/alenaavdeeva/sdk/go1.23.1 #gosetup
GOPATH=/Users/alenaavdeeva/go #gosetup
/Users/alenaavdeeva/sdk/go1.23.1/bin/go build -o /Users/alenaavdeeva/Library/Caches/JetBrains/GoLand2024.2/tmp/GoLand/___4go_build_main_go /Users/alenaavdeeva/Desktop/Labs/lb5_1/main.go #gosetup
/Users/alenaavdeeva/Library/Caches/JetBrains/GoLand2024.2/tmp/GoLand/___4go_build_main_go
9
Process finished with the exit code 0

```

lb5_1 > main.go 18:9 LF UTF-8 Tab

(рис.№1)

3)Результат, если поступает из второго канала число 4(рис.№2)



The screenshot shows an IDE with a Go file named `main.go`. The code defines a `calculator` function that takes two channels and a stop signal, and a `main` function that uses them. The `Run` panel shows the command `go build main.go` and the output indicating a successful build with exit code 0.

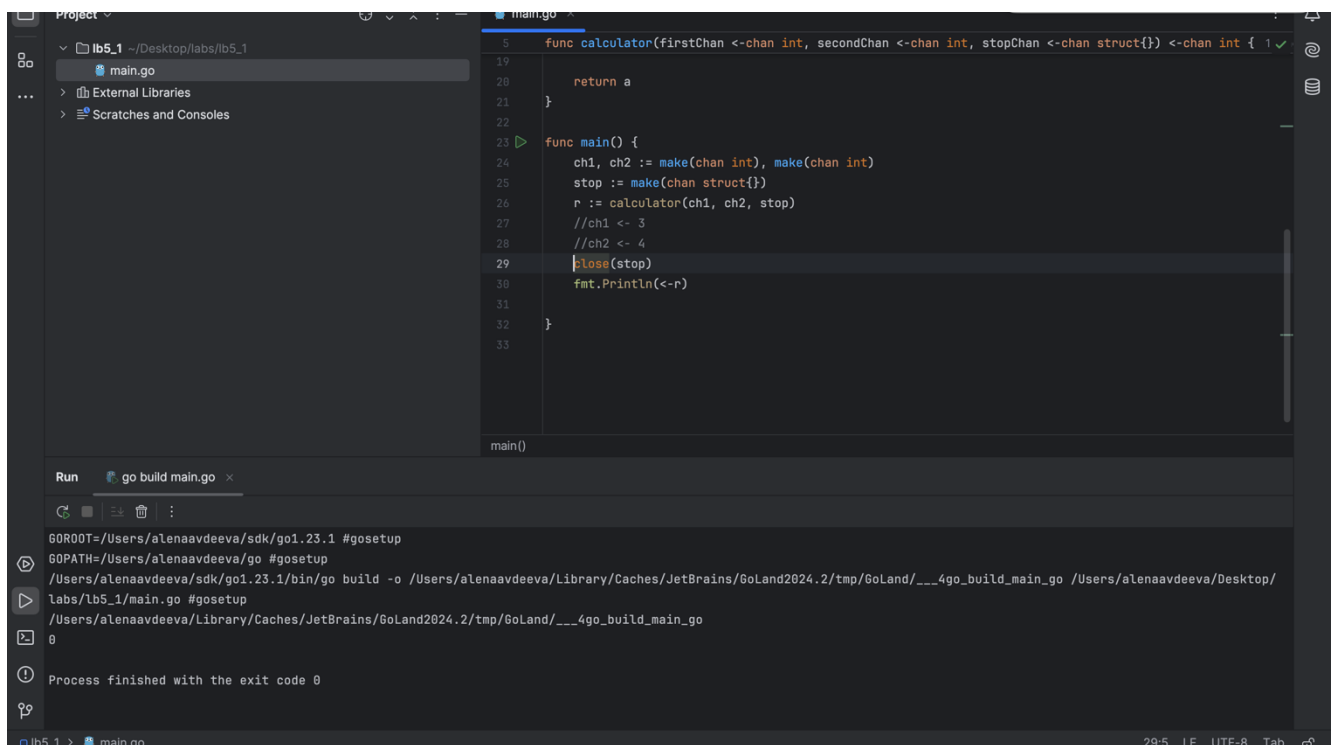
```
16 }
17
18 }(a)
19
20 return a
21 }
22
23 func main() {
24     ch1, ch2 := make(chan int), make(chan int)
25     stop := make(chan struct{})
26     r := calculator(ch1, ch2, stop)
27     //ch1 <- 3
28     ch2 <- 4
29     //close(stop)
30     fmt.Println(<-r)
31 }
32
33 main()
```

Run go build main.go

GOROOT=/Users/alenaavdeeva/sdk/go1.23.1 #gosetup
GOPATH=/Users/alenaavdeeva/go #gosetup
/Users/alenaavdeeva/sdk/go1.23.1/bin/go build -o /Users/alenaavdeeva/Library/Caches/JetBrains/GoLand2024.2/tmp/GoLand/___4go_build_main_go /Users/alenaavdeeva/Desktop/labs/lb5_1/main.go #gosetup
12
Process finished with the exit code 0

(рис.№2)

4)Результат, если поступает из третьего канала(рис.№3)



The screenshot shows the same IDE with the `main.go` file. The code is identical to the previous one. The `Run` panel shows the command `go build main.go` and the output indicating a successful build with exit code 0.

```
5 func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int { 1 ✓
19
20 return a
21 }
22
23 func main() {
24     ch1, ch2 := make(chan int), make(chan int)
25     stop := make(chan struct{})
26     r := calculator(ch1, ch2, stop)
27     //ch1 <- 3
28     //ch2 <- 4
29     //close(stop)
30     fmt.Println(<-r)
31 }
32
33 main()
```

Run go build main.go

GOROOT=/Users/alenaavdeeva/sdk/go1.23.1 #gosetup
GOPATH=/Users/alenaavdeeva/go #gosetup
/Users/alenaavdeeva/sdk/go1.23.1/bin/go build -o /Users/alenaavdeeva/Library/Caches/JetBrains/GoLand2024.2/tmp/GoLand/___4go_build_main_go /Users/alenaavdeeva/Desktop/labs/lb5_1/main.go #gosetup
0
Process finished with the exit code 0

(рис.№3)

Задание 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. Не забудьте закрыть канал ;)

Функция **должна** называться removeDuplicates()

Выводить или вводить ничего не нужно!

Ход работы и результаты:

1) Написание кода на Golang:

```
package main

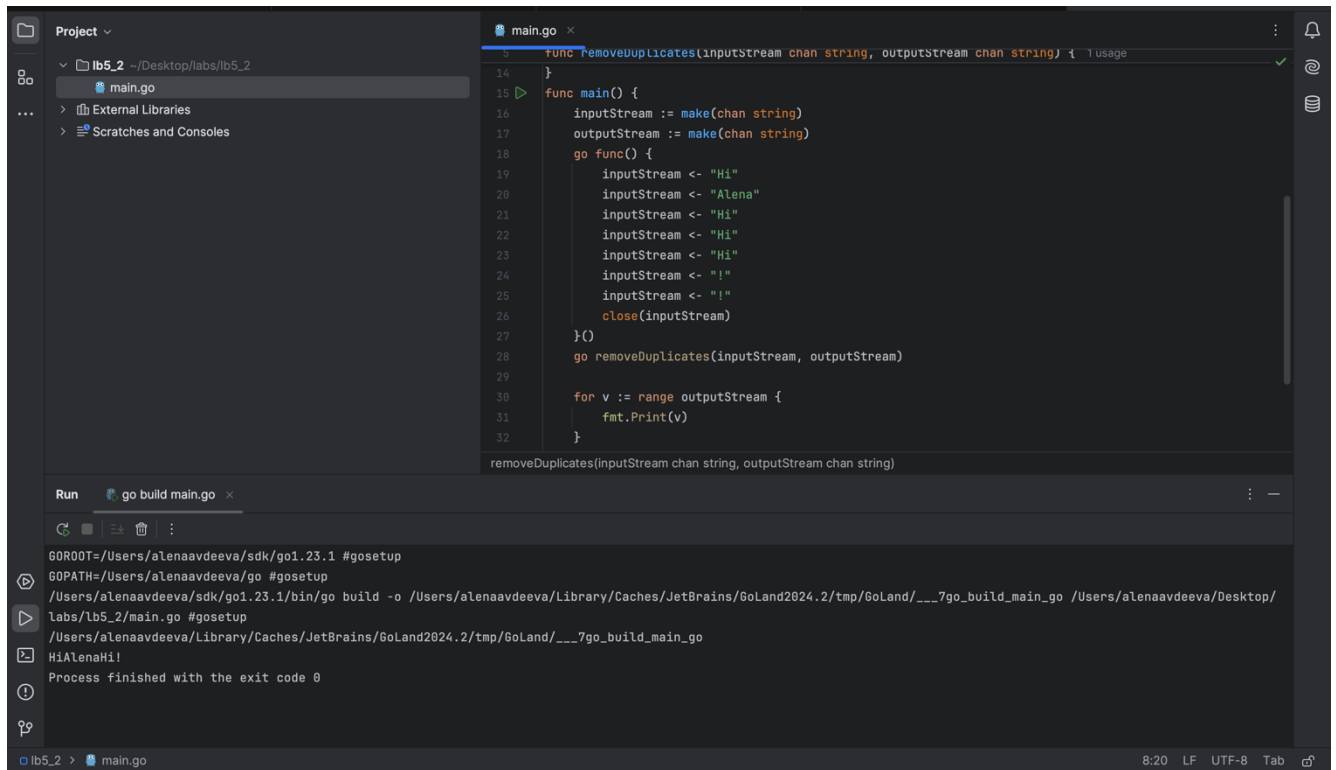
import "fmt"

func removeDuplicates(inputStream chan string, outputStream chan string) {
    var a string
    for i := range inputStream {
        if i != a {
            outputStream <- i
            a = i
        }
    }
    close(outputStream)
}

func main() {
    inputStream := make(chan string)
    outputStream := make(chan string)
    go func() {
        inputStream <- "Hi"
        inputStream <- "Alena"
        inputStream <- "Hi"
        inputStream <- "Hi"
        inputStream <- "Hi"
        inputStream <- "!"
        inputStream <- "!"
        close(inputStream)
    }()
    go removeDuplicates(inputStream, outputStream)

    for v := range outputStream {
        fmt.Print(v)
    }
}
```

2)Результат(рис.№4)



(рис.№4)

Задание work – 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.

Ход работы и результаты:

1) Написание кода на Golang:

```
package main

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

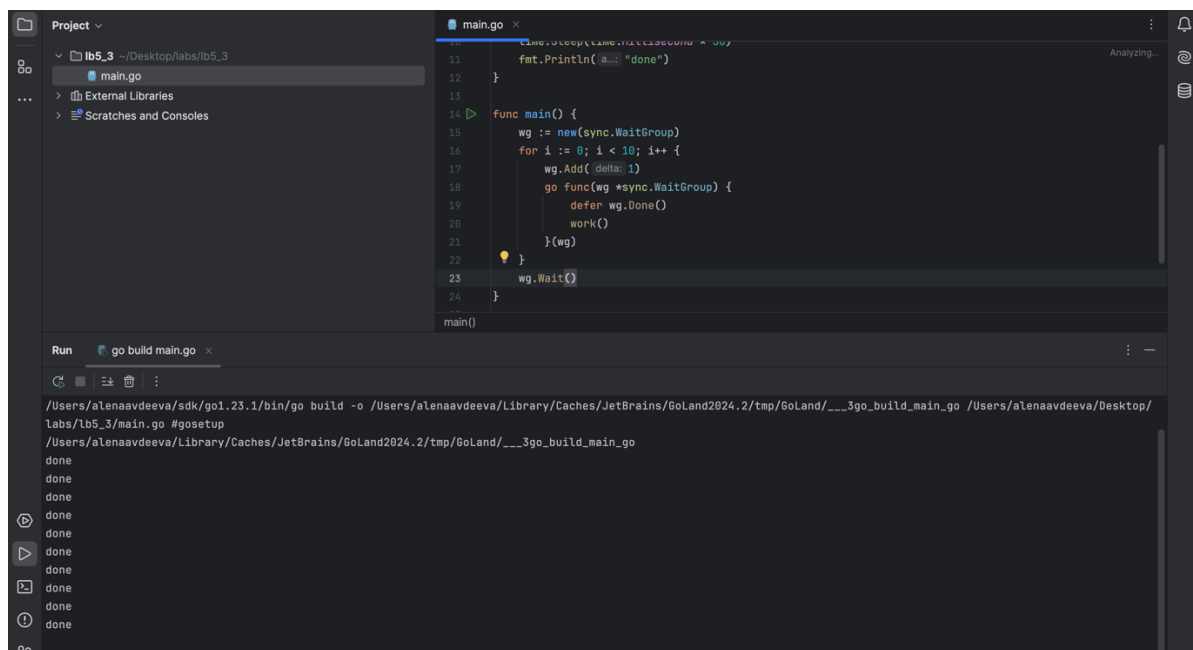
func work() {
    time.Sleep(time.Millisecond * 50)
    fmt.Println("done")
}
```

```

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

```

2)Результат(рис.№5)



(рис.№5)

Заключение – проделана успешная работа в понимании работы с асинхронным программированием на языке Golang!