



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)

06.10.2024

(Подпись, дата)

Л.И. Заушников

(И.О. Фамилия)

Преподаватель

06.10.2024

(Подпись, дата)

В.Д. Шульман

(И.О. Фамилия)

Москва, 2024

**Цель работы:** изучение основ асинхронного программирования с использованием языка Golang.

**Ход работы**

1. Первым делом было проведено ознакомление с разделом "3. Map, файлы, интерфейсы, многопоточность и многое другое" курса <https://stepik.org/course/54403/info>
2. Далее был сделан форк данного репозитория в GitHub, клонирована получившаяся копия локально, создана от мастера ветка dev и было произведено переключение на неё.

Далее было выполнено 5 заданий.

## Задание 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.

Код, решающий задачу выше представлен ниже.

```
package main
```

```
import "fmt"
```

```

func calculator(firstChan, secondChan <-chan int, stopChan <-chan struct{}) <-chan int
{
output := make(chan int)
go func(ch chan int) {
defer close(ch)
select {
case x := <-firstChan:
ch <- x * x
fmt.Println("Первый канал")
case x := <-secondChan:
ch <- x * 3
fmt.Println("Второй канал")
case <-stopChan:
fmt.Println("Стоп")

}

}(output)
return output
}

```

Результаты тестирования программы представлены на рисунках 1-3.

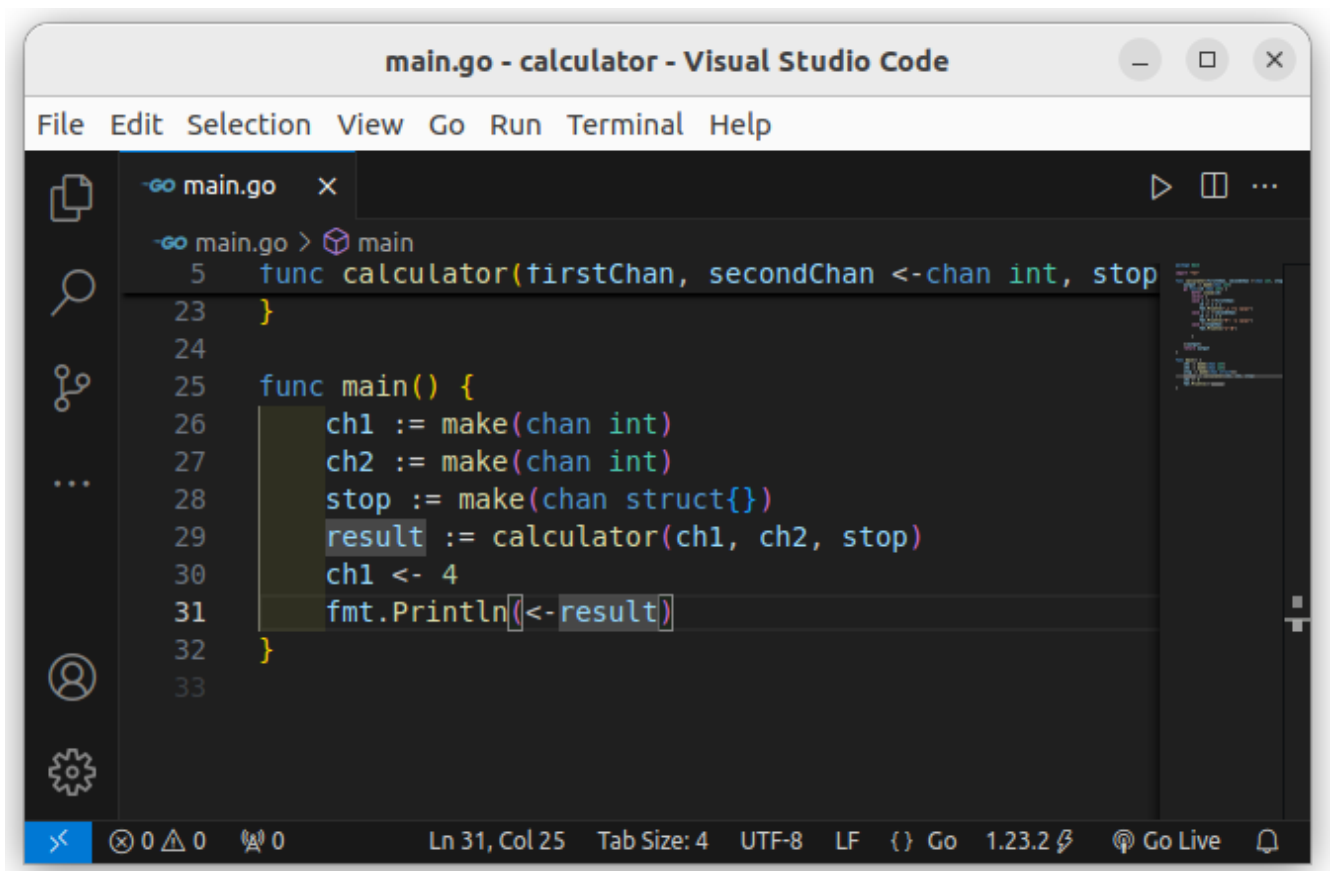
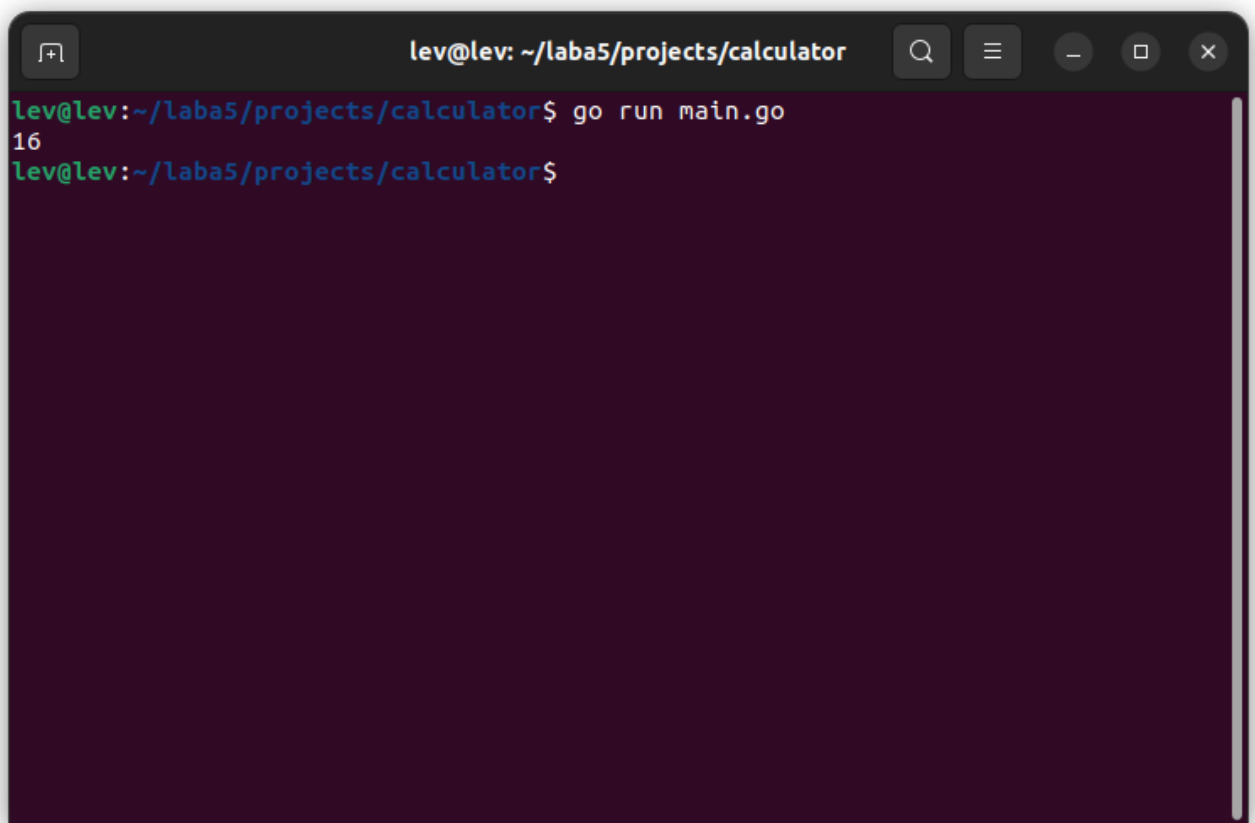
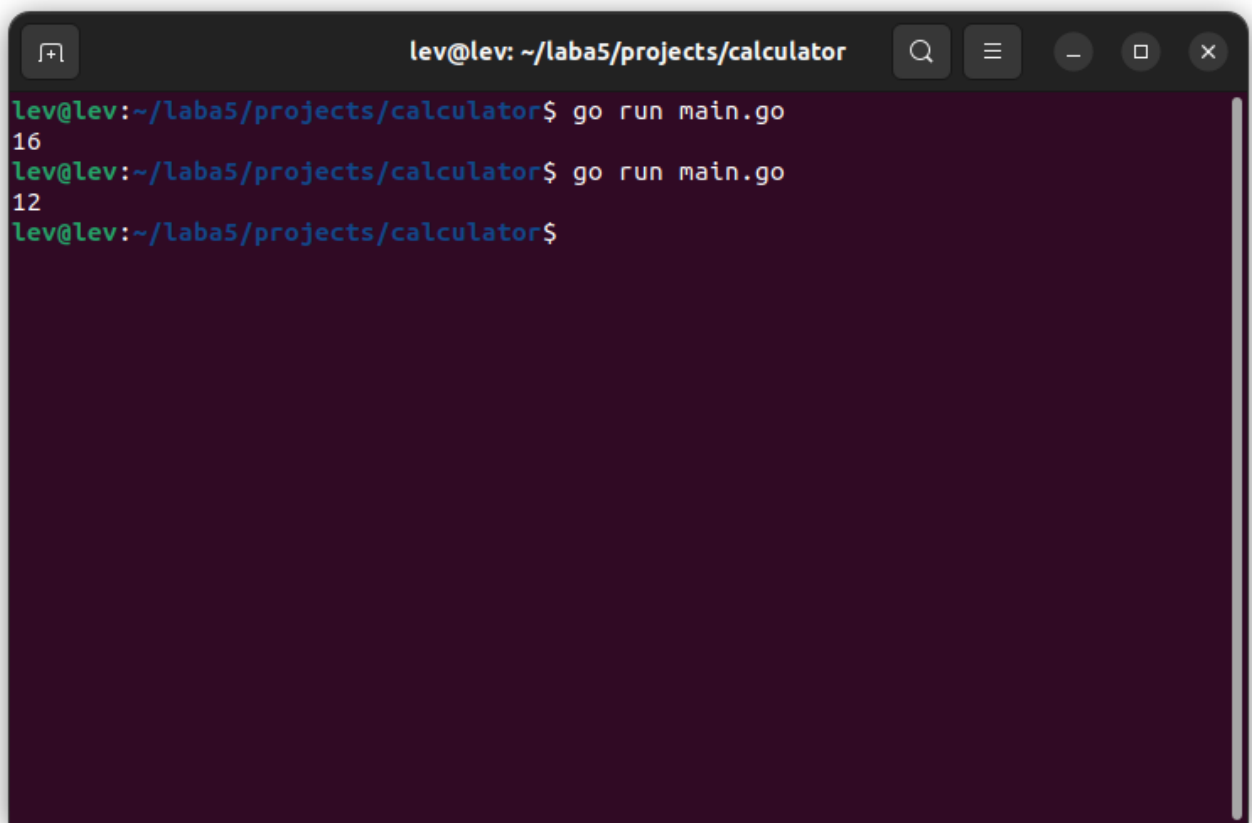


Рисунок 1 - Тестирование программы-калькулятора

A terminal window with a dark background and light-colored text. The title bar at the top reads "lev@lev: ~/laba5/projects/calculator". The terminal shows a command prompt "lev@lev:~/laba5/projects/calculator\$" followed by the command "go run main.go". The output of the command is the number "16". The prompt "lev@lev:~/laba5/projects/calculator\$" appears again on the next line. The window has standard macOS window controls (red, yellow, green buttons) and a search icon on the right side of the title bar.

```
lev@lev: ~/laba5/projects/calculator
lev@lev:~/laba5/projects/calculator$ go run main.go
16
lev@lev:~/laba5/projects/calculator$
```

Рисунок 2 - Тестирование программы-калькулятора

A terminal window with a dark background. The title bar shows 'lev@lev: ~/laba5/projects/calculator'. The terminal content shows three lines of command and output: 'lev@lev:~/laba5/projects/calculator\$ go run main.go' followed by '16', then another 'lev@lev:~/laba5/projects/calculator\$ go run main.go' followed by '12', and finally 'lev@lev:~/laba5/projects/calculator\$' on a new line.

```
lev@lev:~/laba5/projects/calculator$ go run main.go
16
lev@lev:~/laba5/projects/calculator$ go run main.go
12
lev@lev:~/laba5/projects/calculator$
```

Рисунок 3 - Тестирование программы-калькулятора

## Задача 2

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.

Код для решения задачи представлен ниже.

```
package main
```

```
import (
```

```
"fmt"
```

```
"sync"
```

```
"time"
```

```
)
```

```
func work() {
```

```
    time.Sleep(time.Millisecond * 50)
```

```
    fmt.Println("done")
```

```
}
```

```
func main() {
```

```
    // необходимо в отдельных горутинах вызвать функцию work() 10 раз и  
    дождаться результатов выполнения вызванных функций
```

```
    wg := new(sync.WaitGroup)
```

```
    for i := 0; i < 10; i++ {
```

```
        wg.Add(1)
```

```
        go func(wg *sync.WaitGroup) {
```

```
            defer wg.Done()
```

```
            work()
```

```
        }(wg)
```

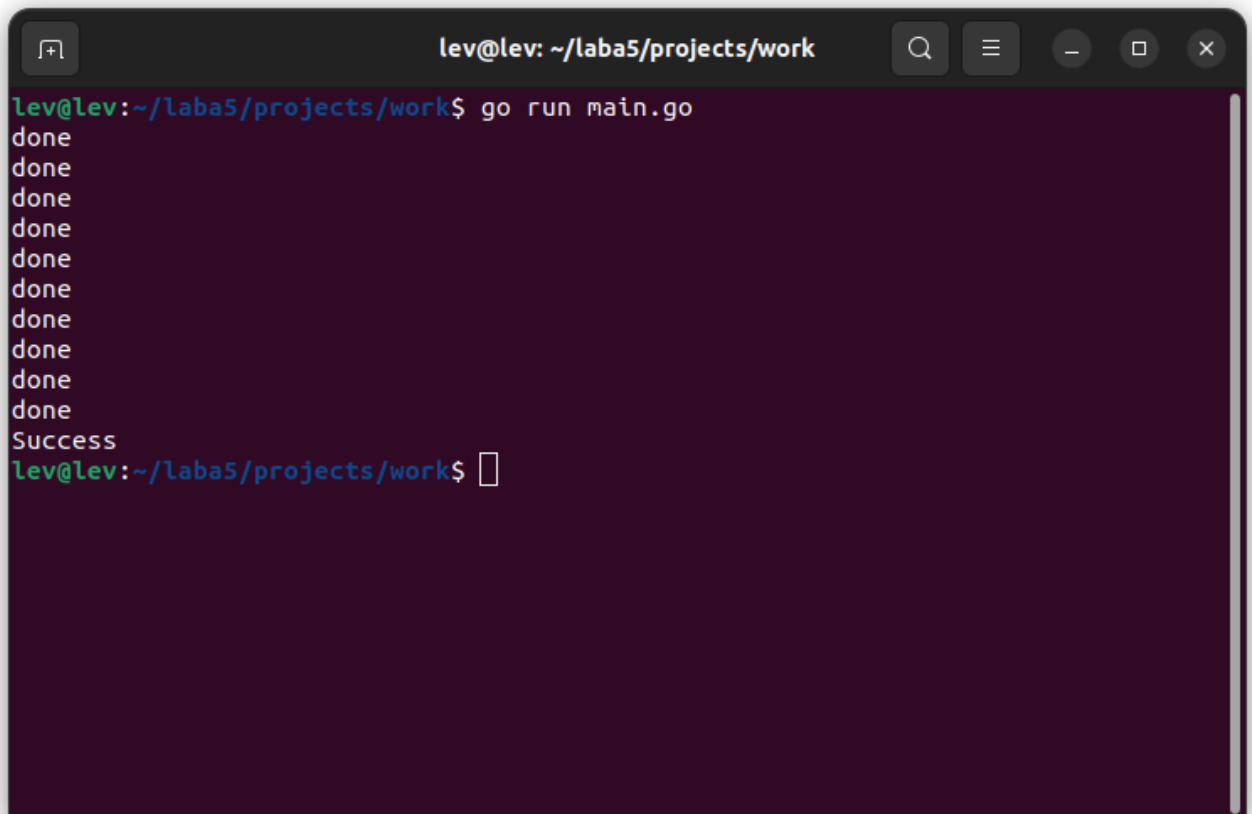
```
    }
```

```
    wg.Wait()
```

```
fmt.Println("Success")
```

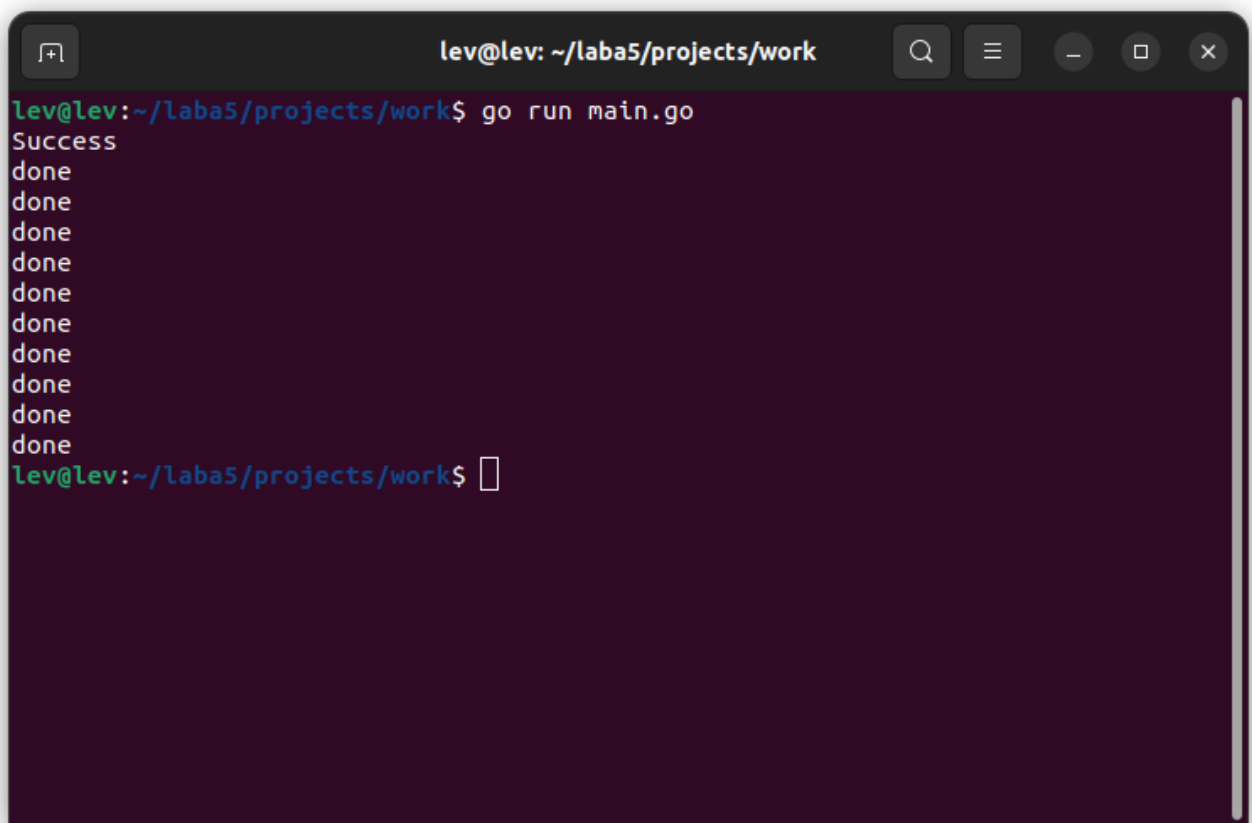
```
}
```

Результаты тестирования программы представлены на рисунках 4-5.

A screenshot of a terminal window with a dark background. The title bar at the top reads "lev@lev: ~/laba5/projects/work". The terminal shows the command "go run main.go" being executed. The output consists of ten "done" lines, followed by a "Success" message, and then the prompt "lev@lev:~/laba5/projects/work\$" with a cursor. The window has standard Linux window controls (minimize, maximize, close) on the right side of the title bar.

```
lev@lev:~/laba5/projects/work$ go run main.go
done
done
done
done
done
done
done
done
done
done
done
Success
lev@lev:~/laba5/projects/work$
```

Рисунок 4 - Тестирование программы для решения 2 задачи

A terminal window with a dark background and light-colored text. The window title is 'lev@lev: ~/laba5/projects/work'. The prompt is 'lev@lev:~/laba5/projects/work\$'. The user has entered 'go run main.go'. The output shows 'Success' followed by ten 'done' lines. The prompt is now 'lev@lev:~/laba5/projects/work\$' with a cursor.

```
lev@lev:~/laba5/projects/work$ go run main.go
Success
done
done
done
done
done
done
done
done
done
done
done
lev@lev:~/laba5/projects/work$
```

Рисунок 5 - Тестирование программы для решения 2 задачи

### Задача 3

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!

```
package main
```



```
// реализовать removeDuplicates(in, out chan string)
```

```
import "fmt"
```

```
func removeDuplicates(in chan string, out chan string) {
```

```
    var stp string
```

```
    defer close(out)
```

```
    for st := range in {
```

```
        if st != stp {
```

```
            out <- st
```

```
            stp = st
```

```
        }
```

```
    }
```

```
}
```

```
func main() {
```

```
    // здесь должен быть код для проверки правильности работы функции  
    removeDuplicates(in, out chan string)
```

```
    inputStream := make(chan string)
```

```
    outputStream := make(chan string)
```

```
    go removeDuplicates(inputStream, outputStream)
```

```
    var str string
```

```
fmt.Print("Text has been captured: ")
```

```
fmt.Scanln(&str)
```

```
go func() {
```

```
defer close(inputStream)
```

```
for _, st := range str {
```

```
inputStream <- string(st)
```

```
}
```

```
()
```

```
fmt.Print("Result: ")
```

```
for st := range outputStream {
```

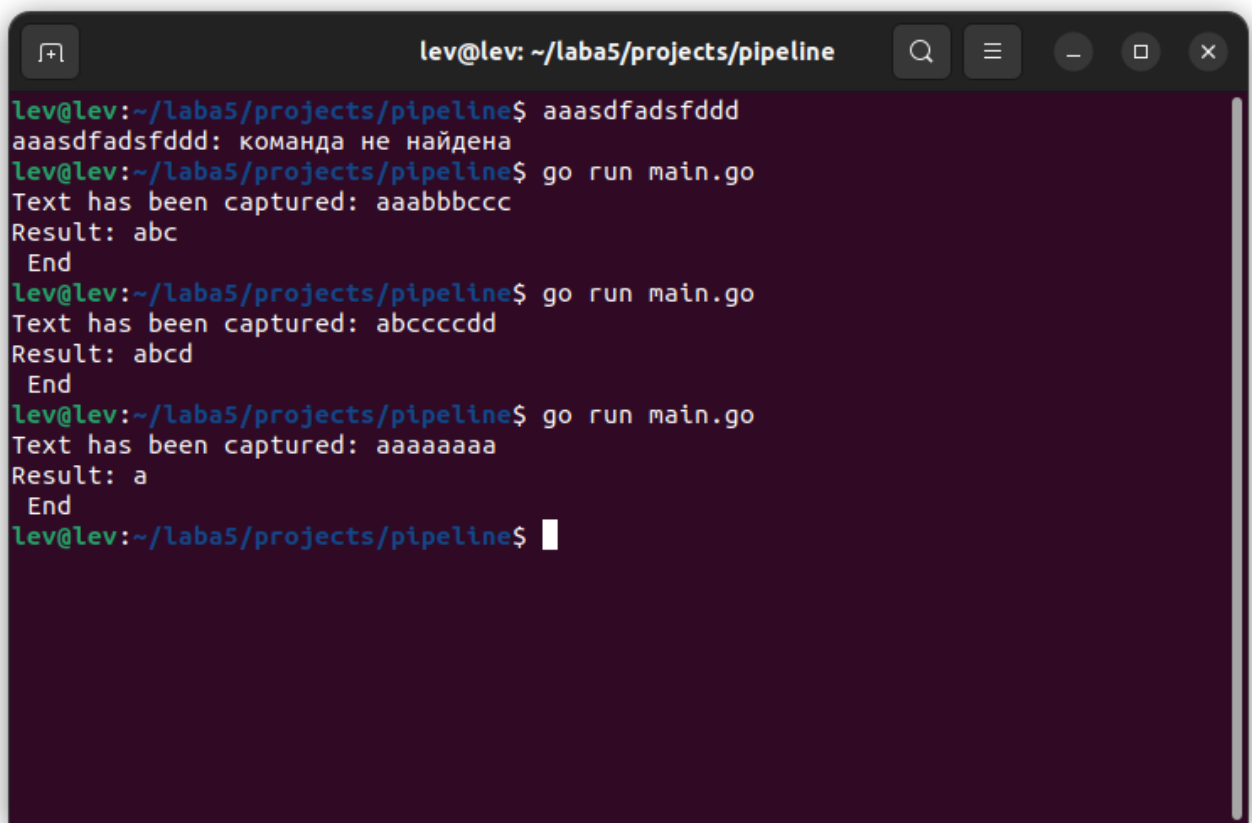
```
fmt.Print(st)
```

```
}
```

```
fmt.Println("\n", "End")
```

```
}
```

Результаты тестирования программы представлены на рисунке 6.

A terminal window with a dark background and light text. The title bar shows the user 'lev' at host 'lev' in the directory '~/laba5/projects/pipeline'. The terminal contains the following text:

```
lev@lev:~/laba5/projects/pipeline$ aaasdfadsfddd
aaasdfadsfddd: команда не найдена
lev@lev:~/laba5/projects/pipeline$ go run main.go
Text has been captured: aaabbbccc
Result: abc
End
lev@lev:~/laba5/projects/pipeline$ go run main.go
Text has been captured: abccccdd
Result: abcd
End
lev@lev:~/laba5/projects/pipeline$ go run main.go
Text has been captured: aaaaaaaa
Result: a
End
lev@lev:~/laba5/projects/pipeline$
```

Рисунок 5 - Тестирование программы для решения задачи 3

## Заключение

В ходе выполнения лабораторной работы были изучены основы асинхронного программирования с использованием языка Golang.

## Список использованных источников

1. <https://stepik.org/course/54403/info>