

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

по лабораторной работе № 5

Дисциплина: Языки интернет-программирования

Студент	<u>ИУ6-33б</u>	<u>(Подпись, дата)</u>	<u>А.В. Архипов</u>
	(Группа)		(И.О. Фамилия)
Преподаватель		<u>(Подпись, дата)</u>	<u>В.Д. Шульман</u>
			(И.О. Фамилия)

Москва, 2024

Основы асинхронного программирования в Golang

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

Задание — выполнить поставленные задачи.

1 задание: Калькулятор.

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`.

1. в случае, если аргумент будет получен из канала `firstChan`, в выходной (возвращенный) канал вы должны отправить квадрат аргумента.
2. в случае, если аргумент будет получен из канала `secondChan`, в выходной (возвращенный) канал вы должны отправить результат умножения аргумента на 3.
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.

```

func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
    outputChan := make(chan int)

    go func() {
        defer close(outputChan)
        select {
            case val, ok := <-firstChan:
                if ok {
                    outputChan <- val * val
                }
            case val, ok := <-secondChan:
                if ok {
                    outputChan <- val * 3
                }
            case <-stopChan:
                return
        }
    }()
    return outputChan
}

func main() {
    firstChan := make(chan int)
    secondChan := make(chan int)
    stopChan := make(chan struct{})

    go func() {
        firstChan <- 2
        secondChan <- 4
        stopChan <- struct{}{}
    }()
    outputChan := calculator(firstChan, secondChan, stopChan)

    for result := range outputChan {
        fmt.Println("Результат:", result)
    }
}

```

Рисунок 1. Код программы.

```

PS C:\Users\artem\sites\golang> go run second.go
Результат: 4
PS C:\Users\artem\sites\golang> go run second.go
Результат: 12
PS C:\Users\artem\sites\golang> go run second.go
PS C:\Users\artem\sites\golang> |

```

Рисунок 2. Результаты выполнения программы при передаче данных через каналы firstChan (2), secondChan (4) и stopChan(struct{} {}).

Задание 2: Удаление последовательных дубликатов.

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

import (
    "fmt"
)

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

func main() {
    inputStream := make(chan string)
    outputStream := make(chan string)

    go removeDuplicates(inputStream, outputStream)

    go func() {
        inputs := []string{"apple", "banana", "apple", "orange", "banana", "grape", "grape"}
        for _, input := range inputs {
            inputStream <- input
        }
        close(inputStream)
    }()

    for unique := range outputStream {
        fmt.Println(unique)
    }
}
```

Рисунок 3. Код программы.

```
PS C:\Users\artem\sites\golang> go run third.go
apple
banana
apple
orange
banana
grape
PS C:\Users\artem\sites\golang> |
```

Рисунок 4. Результаты вывода программы (был удален «grape» в конце строки).

Задание 3: 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.

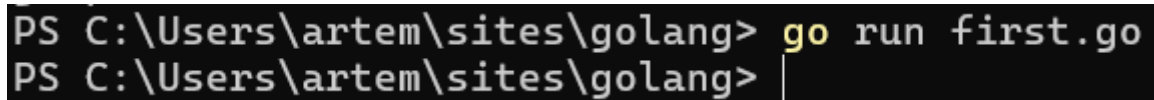
```
package main

import "sync"

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

func work() {
    // ничего
}
```

Рисунок 5. Код программы.

A screenshot of a Windows command prompt window. The background is black, and the text is white. The first line shows the command 'go run first.go' being entered at the prompt 'PS C:\Users\artem\sites\golang>'. The second line shows the prompt 'PS C:\Users\artem\sites\golang>' with a vertical cursor bar at the end, indicating the command has been executed.

```
PS C:\Users\artem\sites\golang> go run first.go
PS C:\Users\artem\sites\golang> |
```

Рисунок 6. Результат работы программы.

Заключение: Были изучены основы асинхронного программирования на языке Go. Были выполнены поставленные задачи.