

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

Название: Основы асинхронного программирования на Golang

Дисциплина: Языки Интернет-программирования

(Подпись, дата)

(И.О. Фамилия)

Оглавление

Основы асинхронного программирования на Golang.....	1
Цель работы.....	3
Ход работы.....	3
Задание 1.....	3
Условие.....	3
Решение.....	3
Задание 2.....	4
Условие.....	4
Решение.....	4
Задание 3.....	5
Условие.....	5
Решение.....	5
Вывод.....	7
Источники информации.....	7

Цель работы

Знакомство с Go, компилируемым многопоточным языком программирования.

Ход работы

Задание 1

Условие

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.

Решение

Код программы:

```
package main
```

```
import "fmt"
```

```
// реализовать calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{ }) <-chan int
```

```
func main() {  
    chan1, chan2 := make(chan int), make(chan int)  
    stop := make(chan struct{ })  
    r := calculator(chan1, chan2, stop)
```

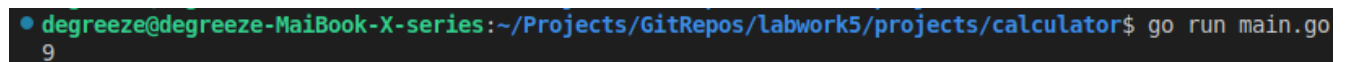
```

    go func() {
        chan1 <- 3
        chan2 <- 3
        close(stop)
    }()
    fmt.Println(<-r)
}

func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{})
<-chan int {
    res := make(chan int)
    go func() {
        select {
            case num := <-firstChan:
                res <- num * num
            case num := <-secondChan:
                res <- num * 3
            case _ = <-stopChan:
        }
    }()
    close(res)

    return res
}

```



```

• degreeze@degreeze-MaiBook-X-series:~/Projects/GitRepos/labwork5/projects/calculator$ go run main.go
9

```

Рисунок 1 - результат работы программы

Задание 2

Условие

Дана строка, содержащая только арабские цифры. Найти и вывести наибольшую цифру.

Входные данные

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!

Решение

```
package main
```

```
import (  
    "fmt"  
)
```

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

```
func main() {  
    inputStream := make(chan string)  
    outputStream := make(chan string)  
    go removeDuplicates(inputStream, outputStream)
```

```
    go func() {  
        inputStream <- "a"  
        inputStream <- "a"  
        inputStream <- "b"  
        inputStream <- "b"  
        inputStream <- "c"  
        close(inputStream)
```

```
    }()
```


```
    for x := range outputStream {
```

```

        fmt.Print(x)
    }
    fmt.Print("\n")
}

func removeDuplicates(inputStream chan string, outputStream chan string) {
    prev_str := ""
    cur_str := ""
    for value := range inputStream {
        prev_str = cur_str
        cur_str = value
        if cur_str != prev_str {
            outputStream <- cur_str
        }
    }
    close(outputStream)
}

```



```

• degreeze@degreeze-MaiBook-X-series:~/Projects/GitRepos/labwork5/projects/pipeline$ go run main.go
abc

```

Рисунок 2 - результат работы программы

Задание 3

Условие

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.

Решение

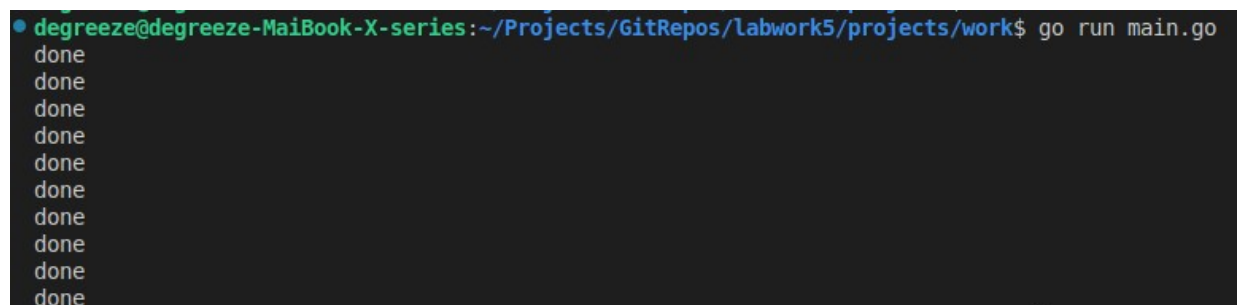
```
package main

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()
}
```



```
degreeeze@degreeeze-MaiBook-X-series:~/Projects/GitRepos/labwork5/projects/work$ go run main.go
done
done
done
done
done
done
done
done
done
done
done
```

Рисунок 3 - результат работы программы

Вывод

При выполнении заданий лабораторной работы мы познакомились с асинхронностью в Golang: решили несколько задач, используя горутины и каналы.

Источники информации

- [Курс Golang на Stepik](#) — источник информации и условий задач