Notatki

Imię i nazwisko

6 marca 2025

Spis treści

1 Wstęp

to są notatki z przedmoitu Programowanie Współbieżne prowadzonym przez drKika na Politechnice Wrocławskiej. Na roku 2, semestrze 4, roku 2025.

2 Wykład 1

2.1 Języki Programowania

Bedziemy programować w Adzie oraz Go.

2.1.1 Lista 1 – Zadania Go

Kożystamy z zadan "Go by Example" oraz Go.dev. Przykład kodu w Go:

1. Hello world

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world")
}
```

2. pętle (wystepuje tylko for)

```
package main

import "fmt"

func main() {
    for i := 0; i < 10; i++ {
        fmt.Println(i)
    }
}
```

3. if

```
package main

import "fmt"

func main() {
    if 7%2 == 0 {
        fmt.Println("7 is even")
    } else {
        fmt.Println("7 is odd")
    }
}
```

4. switch

```
package main

import (
    "fmt"
    "time"
)

func main() {
    switch time.Now().Weekday() {
    case time.Saturday, time.Sunday:
        fmt.Println("weekend")
    default:
        fmt.Println("weekday")
    }
}
```

rowniez mozemy robic swiche po wyrazeniach

```
whatAmI :=func(i interface{}) {
    switch t := i.(type) {
    case bool:
        fmt.Println("I'm a bool")
    case int:
        fmt.Println("I'm an int")
    default:
        fmt.Printf("Don't know type %T\n", t)
    }
}
whatAmI(true)
whatAmI(1)
whatAmI("hey")
```

5. tablice

```
package main

import "fmt"

func main() {
    var a [5]int
    fmt.Println("emp:", a)

    a[4] = 100
    fmt.Println("set:", a)
    fmt.Println("get:", a[4])

    fmt.Println("len:", len(a))

    b := [5]int{1, 2, 3, 4, 5}
    fmt.Println("dcl:", b)

    var twoD [2][3]int
```

```
    for i := 0; i < 2; i++ {
        for j := 0; j < 3; j++ {
            twoD[i][j] = i + j
        }
    }
    fmt.Println("2d: ", twoD)
}
```

- len() zwraca długość tablicy i innych struktur danych dla których jest zdefiniowana.
- append() dodaje element na koniec tablicy.

6. slices

```
s := make([]string, 3)
fmt.Println("emp:", s)

s[0] = "a"
s[1] = "b"
s[2] = "c"
fmt.Println("set:", s)
fmt.Println("get:", s[2])

fmt.Println("len:", len(s))
```

- append() dodaje element na koniec slice'a.
- copy() kopiuje elementy z jednego slice'a do drugiego.
- mozna rowniez tworzyc dwuwymiarowe slice'y.

7. mapy

```
m := make(map[string]int)

m["k1"] = 7
m["k2"] = 13

fmt.Println("map:", m)

v1 := m["k1"]
fmt.Println("v1: ", v1)

fmt.Println("len:", len(m))
```

- delete(mapa, <klucz>) usuwa element z mapy.
- mapy sa referencjami do wartosci.
- normalnie jak javie/cppie.

8. pseudo krotki

```
delete(m, "k2")
fmt.Println("map:", m)
```

```
_, prs := m["k2"]
fmt.Println("prs:", prs)
```

9. range

```
nums := []int{2, 3, 4}
sum := 0
for _, num := range nums {
    sum += num
}
fmt.Println("sum:", sum)

for i, num := range nums {
    if num == 3 {
        fmt.Println("index:", i)
    }
}
```

- range zwraca indeks i wartosc.
- mozna rowniez uzyc range na mapach.

10. funkcje

```
func plus(a int, b int) int {
    return a + b
}

res := plus(1, 2)
fmt.Println("1+2 =", res)



func intSeq() func() int {
    i := 0
    return func() int {
        i++
        return i
    }
}
```

- istnieje cos takiego jak closures. Czyli funkcje wewnatrz funkcji. maja pojebane jakies zmienne lokalne nie wiem czemu to pisze, bo sa te wszystkie slajdy, mozna tam poczytac jak bialy czlowiek.
- funkcje moga zwracac wiele wartosci.
- mozna rowniez przekazywac funkcje jako argumenty.

11. structs

```
type person struct {
    name string
    age  int
```

```
}

fmt.Println(person{"Bob", 20})
fmt.Println(person{name: "Alice", age: 30})
fmt.Println(person{name: "Fred"})
fmt.Println(&person{name: "Ann", age: 40})

s := person{name: "Sean", age: 50}
fmt.Println(s.name)

sp := &s
fmt.Println(sp.age)

sp.age = 51
fmt.Println(sp.age)
```

- mozna tworzyc struktury.
- mozna tworzyc wskazniki do struktur.

12. metody

```
type rect struct {
    width, height int
}

func (r *rect) area() int {
    return r.width * r.height
}

func (r rect) perim() int {
    return 2*r.width + 2*r.height
}
```

- metody mozna tworzyc dla "klas"(struktur).
- nie musza one sie zajdowac w klasie moga byc gdziekolwiek.

13. intefejsy

```
type geometry interface {
    area() int
    perim() int
}

func measure(g geometry) {
    fmt.Println(g)
    fmt.Println(g.area())
    fmt.Println(g.perim())
}
```

- interfejsy sa zbiorami metod.
- mozna je implementowac dla struktur.

14. gorutyny

```
func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}

f("direct")

go f("goroutine")

go func(msg string) {
    fmt.Println(msg)
}("going")

time.Sleep(time.Second)
fmt.Println("done")
```

- gorutyny to lekkie wątki.
- mozna je tworzyc za pomoca go.
- mozna rowniez tworzyc anonimowe gorutyny.

15. channels

```
messages := make(chan string)

go func() { messages <- "ping" }()

msg := <-messages
fmt.Println(msg)
```

- kanaly sluza do komunikacji miedzy gorutynami.
- mozna je tworzyc za pomoca make(chan <typ>).
- mozna rowniez tworzyc kanaly buforowane.

3 Podsumowanie

Podsumowanie lub zakończenie notatek.