Osnove strukture i sintakse Goa – polje (array)

- Tip [n]T je polje od n vrijednosti tipa T
- Izraz ***var a [10]int*** označava varijablu ***a*** kao polje od 10 integera
- Duljina polja je dio njegovog tipa, pa se poljima ne može mijenjati vrijednost
- Iako je ovo vrlo ograničavajuće, Go omogućava drugačiji način rada s poljima

```
package main

import "fmt"

func main() {
var a [2]string
a[0] = "Hello"
a[1] = "World"
fmt.Println(a[0], a[1])
fmt.Println(a)

primes := [6]int{2, 3, 5, 7,
11, 13}
fmt.Println(primes)
}
```

Osnove strukture i sintakse Goa – slice

- Kako je polje vrlo ograničavajuće, Go nudi tip objekt tipa ***slice***
- Slice je dinamički određen, fleksibilan način reprezentacije polja
- U praksi, slice se koristi gotovo uvijek na uštrb polja
- Slice u suštini ne sadrži nikakav podatak, samo opisuje polje u pozadini
- Slice ima **duljinu** i **kapacitet**
- Duljina slicea je broj elemenata kojeg slice opisuje
- Kapacitet slicea je broj elemenata polja u pozadini, počevši od prvog elementa slicea
- Duljina i kapacitet slicea može se dobiti putem izraza ***len()*** i ***cap()***

Osnove strukture i sintakse Goa – slice

```
package main
import "fmt"
func main() {
	s := []int{2, 3, 5, 7, 11, 13}
	printSlice(s)
	s = s[:0] // slice kapaciteta 0
	printSlice(s)
	s = s[:4] // proširi slice
	printSlice(s)
	s = s[2:] // izbaci prve dvije vrijednosti
	printSlice(s)
}

func printSlice(s []int) {
	fmt.Printf("len=%d cap=%d %v\n", len(s), cap(s), s)
}
```

Osnove strukture i sintakse Goa – slice

- Slice može biti napravljen putem ugrađene ***make*** funkcije
- To je način izrade polja dinamičke duljine
- ***make*** funkcija alocira polje s nul-vrijednostima i vraća slice koji se referencira na to polje
- ***make*** podržava eksplicitno definiranje kapaciteta polja
- To je bitno zbog nekih kasnijih optimizacija i internog načina rada Goa
- Slice može sadržavati drugi slice kao tip podatka
- Sliceu dodajemo novi element putem ugrađene ***append*** funkcije
- Prvi parametar ***append*** funkcije je slice tipa T, a drugi parametri su vrijednosti koje dodajemo tom sliceu
- Povratna vrijednost funkcije append je slice koji sadrži originalne elemente slicea uz one dodane u argumentu

Osnove strukture i sintakse Goa – slice

```
package main
import "fmt"
func main() {
	s := make([]int, 0)
	printSlice(s)
	s = append(s, 0)
	printSlice(s)
	s = append(s, 1) // slice raste kako je potrebno
	printSlice(s)
	s = append(s, 2, 3, 4) //možemo dodati više elemenata
	printSlice(s)
}

func printSlice(s []int) {
	fmt.Printf("len=%d cap=%d %v\n", len(s), cap(s), s)
}
```

Osnove strukture i sintakse Goa – slice range

- Postoji specijalna sintaksa za iteriranje sliceom
- Koristi ***for*** i ***range*** ključne riječi
- Prilikom iteriranja slicea, dvije povratne vrijednosti se vraćaju - indeks i kopija elementa na tom indeksu
- Obe varijable su opcionalne

```
package main

import "fmt"

var pow = []int{1, 2, 4, 8, 16, 32,
64, 128}

func main() {
        for i, v := range pow {
fmt.Printf("2**%d = %d\n", i, v)
        }
```

```
        for _, v := range pow {
fmt.Printf("2**%d = %d\n", i, v)
        }

        for i, _ := range pow {
fmt.Printf("2**%d = %d\n", i, v)
        }
}
```

Osnove strukture i sintakse Goa - vježbe

Definicija: Neka su $a \neq 0$ i b cijeli brojevi. Kažemo da je b djeljiv s a, odnosno da a dijeli b, ako postoji cijeli broj x dakav da je $b = ax$. To zapisujemo s $a \mid b$. Ako b nije djeljiv s a, onda pišemo a ∤ b. Ako $a \mid b$, onda još kažemo da je a djeljitelj od b te da je b višekratnik od a.

Za proizvoljan prirodni broj a i cijeli broj b postoji jedinstveni cijeli brojevi q i r takvi da je $b = qa + r$, $0 \leq r < a$.

Broj r nazivamo ostatkom, a broj q kvocjentom pri dijeljenju b sa a. $r = 0$ ako i samo ako a dijeli b, odnosno, a dijeli b ako i samo ako je ostatak pri dijeljenju b sa a jednak 0.

Neka su b i c cijeli brojevi. Cijeli broj a zovemo zajednički djeljitelj od b i c ako $a \mid b$ i $a \mid c$. Ako je barem jedan od brojeva b i c različit od nule, onda postoji samo konačno mnogo zajedničkih djeljitelja od b i c. Najveći menju njima zove se *najveći zajednički djeljitelj* od b i c i označava se s **nzb(b,c)** odnosno **gcb(b,c)** - *greatest common divisor*.

Osnove strukture i sintakse Goa - vježbe

Opišimo sada **Euklodov algoritam**, jedan od najstarijih ali i najvažnijih algoritama u teoriji brojeva. Dobio je ime po znamenitom grčkom matematičaru Euklidu (~300. PNE, - 275. PNE), koji ga je opisao u svojim Elementima. Bez ulaženja u detaljnu teoriju, Euklidov algoritam možemo zapisati kao:

```
while (c>0)
    (b,c) = (c, b mod c)
return b
```

Recimo nešto i o učinkovitosti Euklidovog algoritma, onajprije o broju koraka (dijeljenja) u algoritmu. Može se pokazati da je broj koraka proporcionalan broju znamenaka od c. Takve algoritme, u kojima je broj operacija proporcionalan nekoj potenciji broja znamenaka ulaznog podatka nazivamo **polinomijalnim** ili **učinkovitim** algoritmima.

Osnove strukture i sintakse Goa - vježbe

Zadatak 1.

Izračunajte gcd(252, 198), te pritom koristeći sliceove zapišite svaki korak algoritma u odgovarajući slice (međurješenje, djeljitelj, ostatak).

Zadatak 2.

Izračunajte gcd(3587, 1819), te pritom koristeći sliceove zapišite svaki korak u jedan slice koristeći strukturu.

Osnove strukture i sintakse Goa - mape

- Mapa mapira (sic!) ključ na vrijednost
- Nulvrijednost mape je *nil*
- *make* funkcija vraća mapu zadanog tipa, inicijaliza ju i priprema za korištenje

```
package main

import "fmt"

type Tocka struct {
        Lat, Long float64
}

var m map[string]Tocka

func main() {
        m = make(map[string] Tocka)
        m["FER Zagreb"] = Tocka{
                45.8011247, 15.9710522,
        }
        fmt.Println(m["FER Zagreb"])
}
```

Osnove strukture i sintakse Goa - mape

- Operacije na mapama mogu se podijeliti na unos/ažuriranje, dohvaćanje, brisanje i provjeru
- *m[key] = elem* unosi ili mijenja vrijednost pridodanu ključu *key*
- *elem = m[key]* dohvaća vrijednost pridodanu ključu *key*
- *delete(m, key)* briše ključ i pripadnu vrijednost
- *elem, ok = m[key]* testira postojanje ključa *key* u dvije povratne vrijednosti; ako ključ *key* postoji, *ok* je *true*, u suprotnom *false*; ako ključ ne postoji u mapi, *elem* je nulelement tipa mape

```
package main
import "fmt"
func main() {
        m := make(map[string]int)
        m["Answer"] = 42
        fmt.Println("The value:",
m["Answer"])
        m["Answer"] = 48
        fmt.Println("The value:",
m["Answer"])
        delete(m, "Answer")
        fmt.Println("The value:",
m["Answer"])
        v, ok := m["Answer"]
        fmt.Println("The value:", v,
"Present?", ok)
}
```

Osnove strukture i sintakse Goa - metode

- Go nema klase
- Moguće je definirati metode na tipovima
- Metoda je funkcija s posebnim *receiver* argumentom
- *Receiver* argument ima svoju listu argumenata između ključne riječi *func* i naziva metode

```
package main

import (
        "fmt"
        "math"
)

type Duljina struct {
        X1, Y1, X2, Y2 float64 }
}
```

```
func (d Duljina) algDuljina() float64 {
        return math.Sqrt(math.Pow(d.X1 -
d.X2, 2.0) + math.Pow(d.Y1 - d.Y2, 2))
}

func main() {
        d := Duljina{1, 1, 5, 7}
        fmt.Println(d.algDuljina())
```

Osnove strukture i sintakse Goa - interface

- Interface (sučelje) služi za uspostavljanje polimorfizma u Gou
- Olakšava posao programeru uvodeći agnostične (ili fleksibilne) funkcije na ulazni tip podatka
- U Gou su intefraceovi maleni, često imaju i samo jednu metodu
- Lako ih je koristiti za testiranje (temu koju ćemo obraditi kasnije na predavanjima)
- Učestali primjer interfacea je **Stringer**, definiran u *fmt* paketu
- Razina korištenja interfacea ovisi o programerskim preferencijama, no uvelike olakšava neke korisničke scenarije
- Drugim riječima, nije nužno – ali je apsolutno ohrabrujuće koristiti ovu značajku Goa

Osnove strukture i sintakse Goa - interface

- Primjer: https://play.golang.org/p/vPp4U2gOf4B
- Postoje mnogi dodatni ugrađeni interfaceovi poput Images, Reader/Writer, Hash, Signer/Decrypter…

Osnove strukture i sintakse Goa – errors (greške)

- Go iskazuje greške (ili njihove slojevite statuse) putem **error** vrijednosti
- **error** je ugrađeni tip interfacea
- Go ne poznaje paradigmu *try/catch/finally* koju možete vidjeti u nekim drugim jezicima, kao ni *exceptione* (iznimke)
- Ne postoji intrinsični razlog zašto je jedan pristup bolji od drugog
- Postoji uvjerenje kako je Go način upravljanja greškama eksplicitniji i pragmatičniji, te ne dovodio do potencijaln „lošeg i neupravljivog koda”
- Objektivno, ako ste loš programer, ne postoji dovoljno dobar programski jezik koji će vas poboljšati
- Posljedično, upravljanje greškama u Gou dovest će vas do mnogo if/else uzoraka, što na prvu može izgledati zaprepaštavajuće

Osnove strukture i sintakse Goa – errors (greške)

- Kao deklarativnu podršku za *try/catch/finally*, Go raspolaže s *panic* i *recovery* metodama
- Iako možete paničariti i sami, češće ćete doživjeti paniku internih Go paketa
- U praksi, rijetko ćete pronaći objektivan razlog za njihovo eksplicitno korištenje; najčešće ih je moguće vidjeti prilikom inicijalizacije same aplikacije
- Postoje neke specifičnosti u Go verziji 1.13 i više vezano za greške
- Za sada ćemo se bazirati na objašnjavanju načina koji su prijemčivi u nižim verzijama Goa
- Primjer: https://play.golang.org/p/2cRHNmeE4Fy
- Pitanje je stila ili akademskih finesa jesmo li mogli ovakav tip pogreške spriječiti drugačije ili elegantnije