Struttura di un Programma, Package e Variabili

Docente: Dragan Ahmetovic
mail: dragan.ahmetovic@unimi.it
Ricevimento: su appuntamento

Tutor: Alexandru David

Sito del corso: `http://dragan.ahmetovic.it/?p=teaching`

Informazioni sul corso (cont.)

Gruppo Telegram

- Telegram del corso, iscrivetevi TUTTI, usatelo per chiedere info, se avete dubbi, etc
  `http://dragan.ahmetovic.it/t.php`
  (indirizzo anche sul sito)
- Chi ha problemi di accesso mi scriva appunto su TL, taggandomi!
- Per cose tecniche tipo installazione go, accesso pc lab etc scrivete ad Alex

Questionario sulla volta scorsa

`dragan.ahmetovic.it/r.php`
(indirizzo anche sul sito)

hello world (primo programma in go)

Struttura di un programma di go

```
//package main stampa "hello world". Ah, questo è un commento e inizia per //
package main

//dopo c'è l'import dei pacchetti che il vostro pacchetto usa. fmt è il pacchetto standard per scrivere e leggere da riga di comando
import "fmt"

//Pippo è la variabile alla quale dico ciao
var Pippo string = "world"

//main è la funzione principale di un ESEGUIBILE, ma ce ne possono essere molte altre. Pacchetto non eseguibile = no main!
func main() {
        /*
                Sotto c'è il codice del metodo. Se voglio fare commenti estesi posso scriverli così invece di usare // per ogni riga
        */

        //pluto è la variabile che dice ciao
        var pluto string = "Hello"

        fmt.Println(pluto, Pippo) //stampa il valore delle variabili. Usatelo così, quando faremo le funzioni capirete meglio
                                                                                //commento inutilmente indentato
}
```


go tools

go tools: strumenti base per l'utilizzo di go

L'installazione di go contiene gli strumenti base per lavorare con go. eccone alcuni:

- Compilazione: go build
- Esecuzione: go run
- Formattazione: go fmt
- Documentazione: go doc

go tools

go build [-o output][-i][build flags][files]

Compila i file specificati. Se il pacchetto è main (e ha la funzione main) produce un eseguibile con il nome del primo file.

- Se nomi file omessi compila tutti i file della cartella corrente (produce un eseguibile con il nome della cartella).
- ATTENZIONE, dovete compilare tutti i file usati dal pacchetto!
- -o nome_output //compila specificando il nome dell'eseguibile prodotto
- Gli altri argomenti servono per installare il binario prodotto in una cartella e per settare variabili avanzate di compilazione (non saranno usati ma teneteli presenti)

go tools

go run [build flags][-exec xprog]package [arguments...]

ESEGUE il pacchetto (se eseguibile) //Non può essere omesso, non crea il file compilato

- [arguments...] //Come qualsiasi altro programma, anche il vostro può avere argomenti, se lo eseguite con run, potete anche dare gli argomenti desiderati
- Gli altri argomenti servono per eseguire il codice con un simulatore (se compilati per un'architettura diversa), o per settare variabili avanzate di compilazione (non saranno usati ma teneteli presenti)

go tools

go fmt [-n][-x][file]

Formatta il codice sorgente secondo gli standard go

- Questo migliora la leggibilità
- fatelo sempre
  (QUESTO SARÀ VALUTATO)

go tools

go doc [-u][-c][package | [package.]symbol [.methodOrField]]

Mostra la documentazione del pacchetto, simbolo, metodo o "Field" (variabile, costante...)

- I commenti sono indispensabili, scriveteli sempre, scriveteli bene
  (QUESTO SARÀ VALUTATO)
- se non è specificato nulla mostra la documentazione del pacchetto nella cartella corrente
- -all mostra tutta la documentazione del pacchetto (anche i suoi metodi e field)
- -u mostra anche la documentazione relativa a parti non "esportate"

ESERCIZI - go tools

build

- create una nuova cartella hello nella vostra cartella ~/go/src/
- scaricate hello.go (`dragan.ahmetovic.it/hello.go`) e compilatelo.
- Che file ha prodotto?
- Altri modi per compilare i file senza specificarli?

```
go build .
```

- cosa ha prodotto?
- Potete usarlo per includere altre cartelle
- Altri modi ancora?

```
go build *
```

- cosa ha prodotto?
- Eseguite l'eseguibile. Come si fa?

ESERCIZI - go tools

run

```
go run hello.go
```

- in cosa differisce da build?
- secondo voi produce un file eseguibile?
- si, ma è temporaneo! per vederlo:

```
go run --work hello.go
```


ESERCIZI - go tools

fmt

- copiate hello.go in hello.old

```
go fmt hello.go
```

- cosa ha fatto?
- perchè?
- comparate i due file usando il comando diff

ESERCIZI - go tools

doc

```
go doc
```

- cosa restituisce?
- per fare la documentazione del pacchetto bisogna creare un commento che ne contiene la descrizione, prima della dichiarazione del pacchetto (vedi esempio).
- mostrate la documentazione di Pippo. Come si fa?

```
go doc Pippo
```

- per fare la documentazione delle variabili, costanti, funzioni bisogna fare un commento prima della loro dichiarazione. dovrebbe iniziare col loro nome (vedi esempio)
- come faccio a mostrare la documentazione di pluto?
- non posso! è all'interno di una funzione, la documentazione mostra le definizioni a livello di pacchetto!

ESERCIZI - go tools

doc

`go doc` cosa restituisce?

- per fare la documentazione del pacchetto bisogna creare un commento che ne contiene la descrizione, prima della dichiarazione del pacchetto (vedi esempio).
- mostrate la documentazione di Pippo. Come si fa?

```
go doc Pippo
```

- per fare la documentazione delle variabili, costanti, funzioni bisogna fare un commento prima della loro dichiarazione. dovrebbe iniziare col loro nome (vedi esempio)
- come faccio a mostrare la documentazione di pluto?
- non posso! è all'interno di una funzione, la documentazione mostra le definizioni a livello di pacchetto!

pacchetti e sorgenti

pacchetti

- contenitori di funzioni (e variabili, costanti...) di un programma go
- DEVONO avere ciascuno una loro cartella
  (meglio se chiamata con loro stesso nome)
- POSSONO avere più file, basta che ogni file dichiara il nome del pacchetto
- ATTENZIONE: import ha valore solo nel file in cui è fatto!!!
- la documentazione del pacchetto dovrebbe essere in un solo file

pacchetti e sorgenti

pacchetto main

- come dicevamo il pacchetto main è speciale, perchè produce eseguibili
- cosa succederebbe se cambiassimo il nome del pacchetto main?
- se proviamo a fare build?
- compila perchè il pacchetto non è errato, ma non è eseguibile!
- e se proviamo a fare run?
- non esegue perchè non è main!
- (ricordatevi di rimettere main)
- e se cambiassimo il nome della funzione main in un pacchetto main? provate!
- non va perchè un pacchetto main DEVE avere una (e una sola) funzione main
- (ricordatevi di rimettere main)

pacchetti e sorgenti

altri pacchetti

- perchè potrei volere fare un pacchetto che non sia main?
- per definire funzionalità che mi potrebbero tornare utili in più programmi!
- non ripetere lo stesso codice, ma riusarlo se possibile (QUESTO SARÀ VALUTATO)
- pacchetti di sistema definiti in go: `https://golang.org/pkg/`
- pacchetti vengono caricati chiamando "import "nomepacchetto"" es:

```
import "fmt"
```

- posso rinominarli con "import nuovonome "nomepacchetto"", es:

```
import f "fmt"
```

- ora invece di fmt.Println() devo fare f.Println()
- perchè vorrei una cosa del genere?
- Brevità! Ma sopratutto addattabilità
  (es: prevedo di sostituire fmt con altro pacchetto)

altri pacchetti (cont.)

- ATTENZIONE: quello che non serve non bisogna importarlo o ERRORE (se non uso fmt non importo fmt!)
- posso caricare anche pacchetti "locali" dal loro percorso
- chiamando "import "percorso_pacchetto"" es:

```
import "./nomepacchetto"
```

- posso installare i pacchetti "locali" con "go install" (vanno in ~/go/pkg/$ARCH/)
- dopo li potete caricare come i pacchetti di sistema

pacchetti di sistema

- importate il pacchetto di sistema time, rinominato come t
- stampate a schermo il tempo corrente
- suggerimento: funzione Now()
- se non va assicuratevi di avere fatto l'import giusto

```
import t "time"
```

- se non va ancora assicuratevi di avere fatto correttamente la chiamata alla funzione

```
fmt.Println(t.Now())
```


pacchetti utente

Create un pacchetto paperino (nella cartella src) di go con il seguente contenuto:

```
//package paperino contiene solamente una stringa chiamata HelloWorld
package paperino

//HelloWorld contiene il testo "hello world" nella lingua di paperino
var HelloWorld string = "hwewo wowd"
```

- importate nel pacchetto main il pacchetto paperino usando il percorso locale
- stampate a schermo la variabile HelloWorld
- se non va assicuratevi che il percorso sia corretto

```
import "../paperino"
```

- se non va assicuratevi di aver chiamato "NomePacchetto.NomeVariabile" es:

```
fmt.Println(paperino.HelloWorld)
```

- installate paperino e ora importate la versione installata

nomi

- nomi file: minuscolo, no spazi, underscore per separare le parole
- Così evitiamo il problema di case (in)sensitive OS
- nomi var func, const...: brevi ma significativi, iniziano per lettera
- no spazi o underscore, ogni parola in maiuscolo (es: "currentFile") (QUESTO SARÀ VALUTATO)
- la prima lettera in MAIUSCOLO solo se voglio che sia visibile da import (vedi sotto)

variabili

Go è un linguaggio coi tipi di dati statici (una variabile ha un tipo fisso). Le variabili possono essere dichiarate globalmente (fuori dalle funzioni) o localmente (dentro)

- dichiarazione:

```
var biscotti int //crea variabile di tipo int chiamata biscotti
```

- dichiarazione + assegnamento:

```
var biscotti int = 3 //crea var int biscotti e ci assegna il valore 3
```

- dichiarazione + assegnamento senza tipo (il tipo è intuito dal valore assegnato)

```
var biscotti = 5 //crea var biscotti = 5, che è int, quindi diventa int!
```

- assegnamento (può essere fatto solo localmente):

```
biscotti = 7 //assegna alla variabile biscotti il valore 7
```

- dichiarazione breve (può essere fatto solo localmente)

```
frolle := 9 // crea var frolle = 9, che è int, quindi diventa int!
```


costanti

- Una volta assegnate non cambiano MAI.
- Perchè si usano?
- Appunto, come costanti (es: pi greco), o stringhe di testo che non cambiano (es: localizzazione di un app)
- DEVONO avere un valore! quindi assegnamento obbligato
- Come var possono essere globali o locali.

- dichiarazione + assegnamento:

```
const giorniSettimana int = 7 //come per le variabili
```

- dichiarazione + assegnamento senza tipo (il tipo è intuito dal valore assegnato)

```
const giorniSettimana = 7 //come per le variabili
```


dichiarazioni ed assegnamenti multipli

- dichiarazione:

```
var pippo, pluto int //variabili dello stesso tipo
```

- dichiarazione + assegnamento:

```
var pippo, pluto int = 1, 2 //va bene sia per variabili che per costanti
```

- dichiarazione + assegnamento senza tipo (il tipo è intuito dal valore assegnato)

```
var pippo, pluto = 1, "lorem ipsum" //sia per variabili che per costanti
```

- assegnamento (può essere fatto solo localmente):

```
pippo, pluto = 1, "lorem ipsum" //solo variabili
```

- dichiarazione breve (può essere fatto solo localmente)

```
pippo, pluto := 1, "lorem ipsum" //solo variabili
```


tipi

Anche i tipi possono essere definiti.

- questo torna utile per definire tipi composti (vedremo nelle lezioni future). es: un punto sul piano ha due coordinate: x e y. Si può definire il tipo punto che ha due interi coordinate
- si può anche usare per fare "alias" ovvero cambiare il nome di un tipo

```
type intero int //utile se prevedo cambi di codice (usare diverso tipo)
```

- a questo punto possiamo definire una variabile "intero"

```
var paperino intero = 10
```

variabili, costanti, tipi, scope

factoring

Alcune parole chiave (import, type, var, const) permettono una dichiarazione strutturata di questo tipo:

```
var (
        pippo int
        pluto string
        paperino float
)
```

Per dare struttura, leggibilità, e per un minimo risparmio di scrittura (se ne avete tante)

variabili, costanti, tipi, scope

scope (visibilità delle variabili, costanti, tipi, funzioni)

- **locale**: se dichiarato all'interno di una funzione (o costrutto, vedremo dopo), può essere solo visto dalle cose NELLA funzione (o costrutto) CHE VENGONO DOPO
- **globale**: se dichiarato all'interno del pacchetto (e non inizia per maiuscola), può essere visto da qualsiasi funzione del pacchetto
- **esportato**: se dichiarato all'interno del pacchetto (e inizia per maiuscola), può essere visto anche da qualsiasi funzione di un pacchetto CHE IMPORTA il pacchetto
- È buona pratica rendere visibile da fuori solo lo stretto necessario (QUESTO SARÀ VALUTATO)
- ATTENZIONE, le variabili e costanti globali possono essere ridichiarate localmente (mascherate - per le altre funzioni non cambiano!)
- MA le variabili non possono essere ridichiarate nello stesso scope! (localmente però possono essere assegnate)

ESERCIZI - variabili, costanti, tipi, scope

dichiarazione ed assegnamento

- in hello world, trasformate la dichiarazione di Pippo in "senza tipo"
- trasformate la dichiarazione di pluto in "dichiarazione breve"
- secondo voi cosa succede se provate a trasformare Pippo in "dichiarazione breve"?
- provate! (e tornate indietro eh!)
- fate una dichiarazione (senza assegnamento) multipla di tre variabili di tipo int chiamate "qui", "quo" e "qua"
- secondo voi se ora eseguite il codice cosa succede? perchè?
- secondo voi se li stampate a schermo che valore hanno?
- cambiate la dichiarazione di qui quo qua per usare il factoring ed assegnate a loro i valori 1, 2 e 3 rispettivamente
- trasformate Pippo in una costante
- definite un nuovo alias per string chiamato stringa, trasformate pluto in una stringa

ESERCIZI - variabili, costanti, tipi, scope

scope

- che scope ha Pippo? e pluto?
- torniamo alla documentazione... mostrate la documentazione di main. Come si fa?
- perchè la funzione main non è documentata?
- perchè non è "esportata"!
- come si fa a mostrare la documentazione di main comunque?
- `go doc -u main`
- rendete Pippo di scope globale
- provate a mostrarne la documentazione, come si fa?
- rendete Pippo di scope locale, come si fa?

ESERCIZIONI!

ESERCIZIONE 1: import

- create un nuovo pacchetto main in una cartella a vostro piacere (sempre nel src)
- fate import di fmt con 3 nomi diversi
- stampate i seguenti messaggi, ciascuno con un fmt diverso:

```
hello
hey
hi
```


ESERCIZIONI!

ESERCIZIONE 2: comprensione del codice

- eseguite e capite cosa fa questo codice
- commentatelo e verificate che funziona bene con go doc

```
package main

import "fmt"

func add(x int, y int) int {
        return x + y
}

func main() {
        fmt.Println(add(42, 13))
}
```


ESERCIZIONI!

ESERCIZIONE 3: swapping

- create un nuovo pacchetto main in una cartella a vostro piacere (sempre nel src)
- create due variabili intere a = 3 e b = 5
- stampatele
- ora scambiatele (fate in modo che a prenda il valore di b e b prenda il valore di a)
- stampatele di nuovo

ESERCIZIONI!

ESERCIZI PER CASA

Provate a svolgere gli esercizi proposti dal Prof. Capra:
`https://homes.di.unimi.it/~capra/labprog1920/lezioni/0010/`
Dato che non abbiamo fatto l'inserimento dei dati da riga di comando, dove c'è specificato di inserire dei dati assegnateli semplicemente a delle variabili.
L'inserimento lo faremo la prossima volta. Se non capite qualcosa scrivetemi!