UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Tutorato di programmazione

Piano Lauree Scientifiche - Progetto di Informatica

Scheda 4: composizione di piani

In questa scheda si lavorerà sulla combinazione di piani iterativi, di cui si è parlato nella scheda "Goal che richiedono la composizione di piani". Gli esercizi servono a sviluppare diverse abilità: scrivere codice che combina due piani tra loro in maniera opportuna; analizzare le specifiche di un programma per individuare i goal che devono essere realizzati, e come questi goal si relazionano tra loro; impostare un programma a partire da questa analisi; completare e mettere a punto il programma così impostato, implementando i vari piani e la loro combinazione.

Autori:
Violetta Lonati (coordinatrice del gruppo), Anna Morpurgo e Umberto Costantini

Hanno contribuito alla revisione del materiale gli studenti tutor:
Alessandro Clerici Lorenzini, Alexandru David, Andrea Zubenko, Davide Cernuto, Francesco Bertolotti, Leonardo Albani, Luca Tansini, Margherita Pindaro, Umberto Costantini, Vasile Ciobanu, Vittorio Peccenati.

Si ringraziano per i numerosi spunti:
Carlo Bellettini, Paolo Boldi, Mattia Monga, Massimo Santini e Sebastiano Vigna.

1 Implementare combinazioni di piani

1.1 Prime vocali

Si vuole scrivere un programma che legga una sequenza di stringhe e stampi, per ogni stringa, la sua prima vocale (per semplicità assumiamo che le stringhe contengano solo lettere minuscole). Nel caso in cui una stringa non contenga vocali il programma stampa "la parola non contiene vocali".

Per individuare la prima vocale all'interno di una stringa, usiamo il piano per la *ricerca della prima occorrenza*: ad ogni iterata del ciclo esamineremo una lettera della stringa per verificare se è una vocale.

```
prima_vocale := -1
for _, c := range parola {
    if .... {
        prima_vocale = c
        break
    }
}
```

Poiché dobbiamo ripetere questa stessa operazione su ogni stringa, useremo il piano per la *ripetizione*. Osservate che il piano per la ricerca del prima occorrenza va *annidato* nel piano per la ripetizione, poiché va applicato a ciascuna stringa. Completate la funzione completando e combinando i due piani.

```
func primeVocali (lista []string ) {
    for _, parola := range lista {
        for _, c := range parola {

            ...

        }

        ...

    }
}
```

1.2 Inizia con 'a', inizia con 'b'

Si vuole scrivere un programma che legga una sequenza di dieci stringhe e stampi il numero di stringhe che cominciano con la lettera `a` e il numero di stringhe che cominciano con `b`.
Per calcolare il numero di stringhe che cominciano per `a`, usiamo il piano del *conteggio*; lo stesso piano serve anche a calcolare il numero di stringhe che cominciano per `b`.

```
counta := 0
for i:=0; i<10; i++ {
    fmt.Scan(&parola)
    if parola[0] == 'a' {
        counta++
    }
}
```

```
countb := 0
for i:=0; i<10; i++ {
    fmt.Scan(&parola)
    if parola[0] == 'b' {
        countb++
    }
}
```

Non è possibile combinare questi piani mettendoli semplicemente uno dopo l'altro, perché il piano di conteggio delle parole con `a` "consumerebbe" le parole lette prima dell'inizio del piano di conteggio delle parole con `b`.

Per ovviare a questo problema si potrebbe memorizzare l'intera sequenza di parole in input, ma questo è sconsigliato, perché si occuperebbe inutilmente spazio di memoria.

È possibile invece *fondere* i due piani in un unico ciclo, secondo lo schema seguente:

```
package main

import "fmt"

func main() {
    var parola string
    ...

    for i:=0; i<10; i++ {
        fmt.Scan(&parola)
        ...
    }

    fmt.Println(counta)
    fmt.Println(countb)
}
```

Completate le parti mancanti del programma e salvatelo con il nome `count_a_and_b.go`

1.3 Elettricità

Vogliamo leggere una sequenza di N interi (almeno 3), che descrivono il consumo di elettricità in N giorni, e stampare i giorni in cui il consumo è stato inferiore rispetto sia al giorno prima che al giorno dopo. I giorni sono numerati a partire da 1.

Siccome si deve confrontare il consumo di ogni giorno con quello del giorno successivo è utile usare il piano per l'*elaborazione su valori adiacenti*. Lo stesso confronto deve avvenire anche tra un giorno e il precedente, per cui lo stesso piano andrà applicato due volte.

I due piani possono essere *fusi* e realizzati con un solo ciclo, come nella funzione `main` seguente. Servono due coppie di variabili con ruolo di *prev-curr*; poiché le due coppie di variabili non sono indipendenti tra loro, ma si sovrappongono, il programma usa tre variabili `oggi`, `ieri` e `dueGiorniFa`.

La funzione `main` contiene un errore. Correggete il programma andando a rivedere, se necessario, quali sono le componenti del piano per l'elaborazione su valori adiacenti; quindi compilate il programma e testatelo.

```
func main() {
  const N = 10
  var dueGiorniFa, ieri, oggi int

  fmt.Scan(&ieri)
  fmt.Scan(&oggi)

  for i := 3; i <= N; i++ {
    ieri = oggi
    fmt.Scan(&oggi)
    fmt.Println("oggi: ", oggi, "ieri: ", ieri, "dueGiorniFa: ", dueGiorniFa)
    if ieri < dueGiorniFa && ieri < oggi {
      fmt.Println("il consumo nel giorno", i-1, "   stato inferiore",
        "rispetto sia al giorno prima che al giorno dopo")
    }
  }
}
```

2 Analizzare le specifiche per individuare i *goal* e le modalità di composizione

Per ciascuna delle seguenti specifiche, rispondete a queste domande:

1. Quali sono i *goal* principali da realizzare nell'implementazione del programma?
2. Come vanno combinati i piani che realizzano i *goal* individuati?

1. **Esempio svolto**
   Specifiche: Scrivere un programma che legga da standard input una sequenza di numeri terminata da zero e conti quante sono le coppie di numeri naturali consecutivi uguali.
   - Siccome devo confrontare ciascun numero con il consecutivo, il *goal* da realizzare è quello dell'*elaborazione di valori adiacenti*. Inoltre, per contare quante coppie di numeri soddisfano una certa condizione, il *goal* da realizzare è quello del *conteggio*.
   - I piani che realizzano i due *goal* devono essere *fusi* tra loro.
2. **Specifiche**: Scrivere un programma che legga una sequenza di interi terminata da zero e stampi il più piccolo intero letto e il numero di volte che compare all'interno della sequenza.
3. **Specifiche**: Scrivere un programma che riceve da riga di comando una serie di lunghezza arbitraria di parole e stampa la parola più lunga e la parola più corta. Nel caso in cui vi siano più parole di lunghezza massima o minima, il programma stampa l'ultima incontrata.
4. **Specifiche**: Scrivere un programma che legge da standard input una sequenza di interi terminata da -1 e stampa il numero che contiene il maggior numero di 0. Nel caso in cui vi siano più numeri che contengono il massimo numero di 0, il programma stampa l'ultimo incontrato. Ad esempio, se la sequenza letta è `3040 145 80 1707 105002 78 1970 6005 -1` il programma stampa `105002`.

3 Individuare i piani, progettare il programma, implementarlo

In ciascuno dei prossimi esercizi si analizzano le specifiche di un programma, si individuano i *goal* da realizzare e i relativi piani, si stabilisce come questi debbano essere combinati (come nell'esercizio precedente di analisi delle specifiche). Si entra poi nel dettaglio della progettazione del programma: spesso i piani vanno scomposti e adattati, per poter essere poi combinati tra loro. Negli esercizi è richiesto di implementare i programmi sulla base di queste indicazioni progettuali.

3.1 Conta massimi

Si vuole scrivere un programma che legga una sequenza di 10 interi positivi e stampi il massimo intero letto e quante volte il massimo compare nella sequenza.

Per individuare il massimo intero usiamo il piano per la *ricerca del valore estremo*, usando il primo numero in input per inizializzare la variabile `max` fuori dal ciclo. Per calcolare quante volte questo valore compare nella sequenza usiamo il piano per il *conteggio*. I due piani sono mostrati in questi frammenti di codice:

```
fmt.Scan(&numero)
max := numero

for i := 1; i < 10; i++ {
  fmt.Scan(&numero)
  if numero > max {
    max = numero
  }
}
```

```
count := 0
for i := 0; i < 10; i++ {
    fmt.Scan(&numero)
    if numero == n {
        count++
    }
}
```

Come già visto per altri casi, non è possibile combinare questi piani mettendoli semplicemente uno dopo l'altro (a meno di salvare inutilmente tutti i dati in input), perché il piano per la ricerca del massimo "consumerebbe" i numeri letti prima dell'inizio del piano di conteggio.

È possibile invece *fondere* i due piani in un unico ciclo. Tuttavia in questa fusione di piani è necessario prestare attenzione ad alcuni aspetti critici, che dipendono dalla relazione che lega i due *goal* del problema.

- Il valore `n` con cui confrontare `numero` nel piano del conteggio è proprio il valore corrente di `max`, che cambia durante l'esecuzione.
- Il ciclo del piano per la ricerca del valore estremo è più breve di quello del piano del conteggio, perché il primo numero è usato per inizializzare la variabile `max` e quindi viene letto prima del ciclo; usando questo ciclo *breve* per la fusione dei piani, è necessario inizializzare diversamente anche la variabile `count`.
- La variabile `count` va re-inizializzata anche ogni volta che si trova un massimo *migliore*, cioè ogni volta che si aggiorna la variabile `max` nel piano per la ricerca del massimo.

Completate il programma seguente inserendo, nell'ordine corretto, le istruzioni che trovate a destra

```
package main

import "fmt"

func main() {
  var numero, i int
  fmt.Scan(&numero)
  max := numero

    ...

  fmt.Println(max)
  fmt.Println(count)
}
```

```
}
if numero == max {
for i := 1; i<10; i++ {
count++
max = numero
count = 1
count := 1
fmt.Scan(&numero)
} else if numero > max {
}
```

3.2 Massimo numero di consonanti

Si vuole scrivere una funzione che, presa come argomento una slice di stringhe, restituisca il numero massimo di consonanti contenute in una stringa (assumiamo per semplicità che le stringhe contengano solo caratteri minuscoli).

Per contare il numero di consonanti in ogni stringa usiamo il piano del *conteggio*; in ogni iterata del ciclo analizziamo una lettera della stringa e verifichiamo se è una consonante:

```
count := 0
for _, c := range parola {
    if strings.IndexRune("bcdfghjklmnpqrstvwxyz", c) != -1 {
        count++
    }
}
```

Per trovare il massimo usiamo il piano della *ricerca del valore estremo*; in questo caso ad ogni iterata del ciclo prendiamo in considerazione una stringa:

```
for _, parola := range lista {
    /* n = numero di consonanti in parola */
    if max < n {
        max = count
  }
}
```

Il piano del conteggio va applicato ad ogni stringa, mentre il piano per la ricerca dell'estremo va applicato ai valori calcolati sulle varie stringhe mediante il piano del conteggio; per questo motivo il piano per il conteggio va *annidato* nel piano per la ricerca del valore estremo.

Completate il programma inserendo, nell'ordine corretto, le righe riportate in fondo al listato.

```
func maxConsonanti ( lista []string ) int {
  var max int

  for _, parola := range lista {
    /*  inserisci  qui le righe in ordine */
  }
    return max
}

/*************************
    righe da riordinare :
*************************/
if strings.IndexRune("bcdfghjklmnpqrstvwxyz", c) != -1 {
}
for _, c := range parola {
max = count
}
}
count++
count := 0
if max < count {
```

NOTA BENE: osservate che negli ultimi due esercizi sono utilizzati, in entrambi i casi, un piano per il conteggio e un piano per la ricerca del valore estremo; il modo in cui i due piani sono combinati è però molto diverso nei due casi!

3.3 Sequenze di bit

Vogliamo leggere una sequenza di N bit (interi `0` o `1`) e stampare le lunghezze delle sottosequenze di `1` e di quelle di `0`. Ad esempio, su input `1 1 1 0 0 1 0 0` l'output deve essere `3 2 1 2`. Assumiamo per semplicità che l'utente inserisca solo `0` o `1`.

Per calcolare la lunghezza di una (sotto)sequenza di bit, usiamo il piano del *conteggio*, con il contatore `leng` che viene incrementato ad ogni bit della (sotto)sequenza.

Una (sotto)sequenza di bit finisce quando il bit cambia valore; per individuare quando questo avviene è opportuno analizzare coppie di bit consecutivi: se i bit sono diversi la sequenza si è conclusa. Dunque, per individuare le posizioni in cui le sottosequenze terminano, posso usare il piano per l'*elaborazione su valori adiacenti*: ad ogni iterata si verifica se la sequenza corrente di bit si è conclusa o prosegue ancora.

I due piani vanno *fusi*: il contatore `leng` va incrementato quando siamo all'interno delle sottosequenze di bit e va re-inizializzato quando inizia una nuova sottosequenza.

Riordinate le seguenti righe per scrivere una porzione di codice che implementa il progetto sopra descritto. Osservate l'indentazione: vi potrà guidare nella ricostruzione dell'ordine corretto

```
fmt.Scan(&bit)
    bitPrev = bit
const N = 8
        leng++
    fmt.Scan(&bit)
for i := 1; i < N; i++ {
    }
        fmt.Println(leng)
        leng = 1
    if bit == bitPrev {
}
var bit, bitPrev, leng int
    } else {
fmt.Println(leng)
leng = 1
```

4 Dalle specifiche al progetto

Nei seguenti esercizi sarete guidati nella fase di progettazione di un programma. Dovrete individuare i *goal* da realizzare e i corrispondenti piani, stabilire come vadano combinati e descrivere nel dettaglio come vanno implementati i piani nel caso specifico in questione.

4.1 Prima parola con doppia

Specifiche: Scrivere un programma che riceve da riga di comando una serie di lunghezza arbitraria di parole e stampa la prima parola che contiene una doppia (una lettera ripetuta due volte consecutivamente). Nel caso nessuna parola contenga una doppia, il programma deve stampare un messaggio opportuno.
Progettazione: Completate ciascuna affermazione scegliendo l'opzione corretta.

Per stabilire se una parola contiene una doppia (*goal* A)
1. devo contare le lettere
2. devo trovare la prima consonante della parola
3. devo analizzare le lettere della parola singolarmente
4. devo analizzare coppie di lettere vicine

Per realizzare il *goal* A che piano è opportuno usare? Il piano (piano A) per:
1. la ripetizione
2. il conteggio
3. il calcolo di un totale
4. la ricerca della prima occorrenza
5. la ricerca del valore estremo
6. l'elaborazione su valori adiacenti

Ad ogni iterata del piano A si analizza:
1. una stringa
2. una coppia di lettere
3. una sequenza di stringhe
4. una coppia di numeri
5. una sequenza di numeri

Per trovare la prima parola che ha la proprietà desiderata usiamo un piano B. Che piano è opportuno usare come piano B? Il piano per:
1. la ripetizione
2. il conteggio
3. il calcolo di un totale
4. la ricerca della prima occorrenza
5. la ricerca del valore estremo
6. l'elaborazione su valori adiacenti

Ad ogni iterata del piano B si analizza:
1. una stringa
2. una coppia di lettere
3. una sequenza di stringhe
4. una coppia di numeri
5. una sequenza di numeri

Quali di queste affermazioni è corretta?
1. ad ogni lettera devo applicare il piano B
2. ad ogni parola della serie devo applicare il piano A
3. per poter applicare il piano A serve aver completato il piano B
4. devo applicare il piano B solo alla prima parola della serie

Quindi, come devono essere combinati il piano A e il piano B?
1. il piano A va annidato nel piano B
2. il piano B va annidato nel piano A
3. il piano A e il piano B vanno fusi
4. il piano A va messo prima del piano B
5. il piano B va messo prima del piano A

Quanti cicli servono per realizzare i due piani?

1. un solo ciclo
2. due cicli, uno successivo all'altro
3. due cicli, uno dentro l'altro

Implementazione: sulla base delle scelte di progettazione appena fatte, impostate e mettete a punto un programma che implementi le specifiche date.

4.2 Massima somma delle cifre

Specifiche: Scrivere un programma che legge 10 interi positivi e stampa l'intero la cui somma delle cifre è maggiore. Ad esempio, se la sequenza letta è `1 5 14 27 111 234 87 14 6 1221` il programma stampa `87` (la cui somma delle cifre è 15).
Progettazione: Completate ciascuna affermazione scegliendo l'opzione corretta.

Per calcolare la somma delle cifre di un numero usiamo un piano A. Che piano è opportuno usare come piano A? Il piano per:
1. la ripetizione
2. il conteggio
3. il calcolo di un totale
4. la ricerca della prima occorrenza
5. la ricerca del valore estremo
6. l'elaborazione su valori adiacenti

Per il piano A servirà una variabile VAR con ruolo
1. accumulatore
2. ricercato
3. one-way flag
4. inseguitore
5. passo-passo

Ad ogni iterata del piano A si analizza:
1. una coppia di lettere
2. un numero
3. una cifra
4. una coppia di cifre

Per trovare la somma maggiore usiamo un piano B. Che piano è opportuno usare come piano B? Il piano per:
1. la ripetizione
2. il conteggio
3. il calcolo di un totale
4. la ricerca della prima occorrenza
5. la ricerca del valore estremo
6. l'elaborazione su valori adiacenti

Per il piano B servirà una variabile con ruolo
1. accumulatore
2. ricercato
3. one-way flag
4. inseguitore
5. passo-passo

Ad ogni iterata del piano B si considera
1. una coppia di lettere
2. un numero
3. una cifra
4. una coppia di cifre
5. una stringa

Quali di queste affermazioni è corretta?
1. ad ogni numero della serie devo applicare il piano B
2. per poter applicare il piano A serve aver completato il piano B
3. ad ogni numero della serie devo applicare il piano A
4. devo applicare il piano B a tutte le cifre del numero

Quindi, come devono essere combinati il piano A e il piano B?

1. il piano A va annidato nel piano B
2. il piano B va annidato nel piano A
3. il piano A e il piano B vanno fusi
4. il piano A va messo prima del piano B
5. il piano B va messo prima del piano A

Quanti cicli servono per realizzare i due piani?

1. un solo ciclo
2. due cicli, uno successivo all'altro
3. due cicli, uno dentro l'altro

La variabile VAR

1. dovrà essere inizializzata una sola volta prima del ciclo
2. dovrà essere re-inizializzata a 0 per ogni numero, quindi all'interno del ciclo più esterno
3. dovrà essere re-inizializzata a 1 per ogni cifra, quindi all'interno del ciclo più interno
4. dovrà essere inizializzata una sola volta fuori dall'unico ciclo del programma

Implementazione: sulla base delle scelte di progettazione appena fatte, impostate e mettete a punto un programma che implementi le specifiche date.

4.3 Differenza massima tra numeri consecutivi

Specifiche: Scrivere un programma che legga una sequenza di N interi (almeno due) e stampi la più grande differenza (in valore assoluto) che c'è tra due numeri consecutivi.
Progettazione: Completate ciascuna affermazione scegliendo l'opzione corretta.
Serve
1. trovare il primo numero in valore assoluto
2. calcolare la differenza tra ogni numero e il successivo
3. sommare tutti gli N numeri della serie
4. contare quante lettere ci sono nella serie

Per realizzare questo *goal* usiamo un piano A. Che piano è opportuno usare come piano A? Il piano per:
1. la ripetizione
2. il conteggio
3. il calcolo di un totale
4. la ricerca della prima occorrenza
5. la ricerca del valore estremo
6. l'elaborazione su valori adiacenti

Ad ogni iterata del piano A si considera:
1. una stringa
2. una coppia di lettere
3. una sequenza di stringhe
4. una coppia di numeri
5. una sequenza di numeri

Per il piano A servirà una variabile con ruolo
1. accumulatore
2. ricercato
3. one-way flag
4. inseguitore
5. passo-passo

Inoltre si deve individuare
1. il valore più grande
2. il valore più piccolo
3. il primo valore
4. l'ultimo valore
5. la somma dei valori

Per realizzare questo *goal* usiamo un piano B. Che piano è opportuno usare come piano B? Il piano per:
1. la ripetizione
2. il conteggio
3. il calcolo di un totale
4. la ricerca della prima occorrenza
5. la ricerca del valore estremo
6. l'elaborazione su valori adiacenti

Ad ogni iterata del piano B si analizza:
1. una stringa
2. un numero
3. una sequenza di stringhe
4. una coppia di numeri
5. una sequenza di numeri

Come devono essere combinati il piano A e il piano B?
1. il piano A va annidato nel piano B

2. il piano B va annidato nel piano A
3. il piano A e il piano B vanno fusi
4. il piano A va messo prima del piano B
5. il piano B va messo prima del piano A

Quanti cicli servono per realizzare i due piani?

1. un solo ciclo
2. due cicli, uno successivo all'altro
3. due cicli, uno dentro l'altro

Quali di queste affermazioni sono corrette (scegliere tutte quelle corrette):

1. ad ogni iterata del ciclo esterno si aggiorna, quando necessario, la variabile che tiene traccia del valore massimo
2. ad ogni iterata si esamina una cifra del numero in esame
3. ad ogni iterata dell'unico ciclo si calcola la differenza tra due valori adiacenti
4. ad ogni iterata del ciclo interno si esamina un numero e si incrementa la variabile accumulatore
5. ad ogni iterata dell'unico ciclo si aggiorna, quando necessario, la variabile che tiene traccia del valore massimo
6. ad ogni iterata è necessario aggiornare il valore della variabile prev (in modo che assuma il valore di curr) e leggere il prossimo valore da memorizzare in curr.
7. ad ogni iterata si aggiorna la variabile che tiene traccia del primo numero con valore assoluto positivo

Implementazione: sulla base delle scelte di progettazione appena fatte, impostate e mettete a punto un programma che implementi le specifiche date.

5 Senza rete

Concludiamo la scheda con due esercizi di scrittura di programmi. Anche se gli esercizi non sono guidati come i precedenti, provate ad applicare lo stesso approccio per progettare la vostra soluzione: analizzate le specifiche per identificare i *goal* e i relativi piani; stabilite che relazione c'è tra i *goal* e come vanno combinati i piani; riflettete su come devono essere implementati i piani nel caso specifico, su quali variabili servono e quando/come devono essere inizializzate e aggiornate.

5.1 Numero massimo di zeri

Specifiche: Scrivere un programma `massimo_n_zeri.go` che legge da standard input una sequenza di interi terminata da -1 e stampa il numero che contiene il maggior numero di 0. Nel caso in cui vi siano più numeri che contengono il massimo numero di 0, il programma stampa l'ultimo incontrato. Ad esempio, se la sequenza letta è `3040 145 80 1707 105002 78 1970 6005 -1` il programma stampa `105002`.

Per quanto riguarda l'input, osservate che: i dati arrivano da input standard; si tratta di una serie di dati che possono essere elaborati in modo uniforme; non è necessario memorizzare l'intera sequenza; la lettura si interrompe quando viene inserito -1.

5.2 Somme delle sequenze cresenti

Specifiche: Scrivere un programma che legge una sequenza di interi e stampa la somma degli interi in ciascuna sottosequenza crescente. Il programma interrompe la lettura quando riceve un segnale di EOF. Ad esempio, su input `1 2 13 0 7 8 9 -1 0 2` il programma stampa le somme 16, 24 e 1.