UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Tutorato di programmazione

Piano Lauree Scientifiche - Progetto di Informatica

Scheda 2: compiti ricorrenti e soluzioni tipiche

Questa scheda serve a riflettere sul fatto che problemi apparentemente diversi spesso condividono una struttura simile, dunque possono essere affrontati con strategie simili. Imparare a individuare queste similarità è una abilità fondamentale da sviluppare per imparare a programmare. Infatti, quando si deve scrivere un programma in base a specifiche date, un approccio utile è quello di individuare quali strategie già conosciute (nel senso di "mattoni" di base già pronti) si possono utilizzare per impostare il programma in questione.

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.

Nota sull'utilizzo di questo materiale:
L'utilizzo del materiale contenuto in questa scheda è consentito agli studenti iscritti al progetto di tutorato; ne è vietata qualsiasi altra utilizzazione, ivi inclusa la diffusione su qualsiasi canale, in assenza di previa autorizzazione scritta degli autori.

1 Frammenti di codice

Analizzate i frammenti di codice qui sotto, poi rispondete alle domande che trovate nella pagina seguente.

Frammento 1:

```
var maxM rune = -1

for _, r := range testo {
  if unicode.IsUpper(r) && r > maxM {
    maxM = r
  }
}
```

Frammento 2:

```
prodotto := 1

for i := 0; i < DIM; i++ {
  prodotto *= numeri[i]
}
```

Frammento 3:

```
ultimoGiornoPioggia := -1

for i, mis := range mmPioggia {
  if mis > 0 {
    ultimoGiornoPioggia = i
  }
}
```

Frammento 4:

```
somma := 0

for i := 0; i < DIM; i++ {
  if numeri[i]%3 == 0 {
    somma += numeri[i]
  }
}
```

Frammento 5:

```
for i := 1; i < DIM; i++ {
  fmt.Println(numeri[i] - numeri[i-1])
}
```

Frammento 6:

```
somma := 0

for i := 0; i < DIM; i++ {
  somma += numeri[i]
  if somma >= TARGET {
    fmt.Println(i,"superato",TARGET)
    break
  }
}
```

Frammento 7:

```
var min byte = testo[0]

for i := 1; i < DIM; i++ {
  if testo[i] < min {
    min = testo[i]
  }
}
```

Frammento 8:

```
alternata := true

for i := 1; i < DIM; i++ {
  if (numeri[i] > 0) == (numeri[i-1] > 0) {
    alternata = false
    break
  }
}
```

Frammento 9:

```
var precedente rune = testo[0]

for _, corrente := range testo {
  if corrente < precedente {
    fmt.Println()
  }
  fmt.Print(string(corrente), " ")
  precedente = corrente
}
```

Dopo aver analizzato i frammenti di codice, rispondete per iscritto alle seguenti domande.

1. A cosa servono questi frammenti di codice? Associate ciascun frammento ad una delle seguenti descrizioni:
   a) Individua l'ultimo giorno piovoso
   b) Somma i multipli di 3
   c) Trova la massima maiuscola (quella con la posizione maggiore nell'alfabeto)
   d) Divide una sequenza di rune in sottosequenze crescenti
   e) Trova il minimo carattere (quello rappresentato dal byte più piccolo
   f) Calcola le differenze tra i numeri che appaiono consecutivamente in una sequenza
   g) Calcola il prodotto di una sequenza di numeri
   h) Verifica se in una sequenza si alternano numeri positivi e negativi
   i) Calcola quando la somma di una sequenza di numeri supera un limite fissato
2. Provate a raggruppare i frammenti di codice in base alle loro affinità. Quali affinità avete individuato? (ovvero: quale criterio –o criteri– avete usato per raggrupparli)
3. Cosa hanno in comune
   a) i frammenti 1, 3 e 9?
   b) i frammenti 2, 4 e 5?
   c) i frammenti 6 e 8?
   d) i frammenti 1 e 9?
   e) i frammenti 4 e 8?
4. Cosa hanno in comune
   a) i frammenti 5, 8 e 9?
   b) i frammenti 1 e 7?
   c) i frammenti 2, 4 e 6?
5. Le seguenti coppie di frammenti hanno strutture e svolgono compiti in qualche modo simili, ma non uguali. Che differenze strutturali e quindi semantiche presentano:
   a) i frammenti 2 e 4?
   b) i frammenti 1 e 3?

NOTA IMPORTANTE
Prima di proseguire con gli esercizi, leggete con attenzione il file di appunti intitolato "*Goal*, *Plan*, e ruoli delle variabili".

Test di verifica della lettura di "Goal, plan e ruoli delle variabili"

Dopo aver letto con attenzione gli appunti, rispondete alle domande che seguono.

Per ciascuna domanda, scegliere la risposta corretta.

1. Che cosa è un *goal*?
   a) una soluzione tipica per un compito dato
   b) un sotto-compito che deve essere affrontato nella soluzione di un problema
   c) un programma realizzato per uno scopo dichiarato
   d) l'obiettivo a cui deve tendere un programma
2. Che cosa è un *plan*?
   a) uno dei sottoproblemi in cui va scomposto il problema da risolvere
   b) una funzione di libreria

   c) la soluzione tipica con cui viene realizzato uno specifico goal
   d) il progetto di cui fa parte il programma da scrivere
3. A cosa ci si riferisce quando si parla di goal/plan per la ripetizione?
   a) al fare una data azione un certo numero di volte
   b) alla ripetizione dello stesso blocco di codice in punti diversi del programma
   c) al ripetere in output l'input letto
   d) a ripetere negli esercizi proposti quanto si è imparato a lezione
4. A cosa ci si riferisce quando si parla di goal/plan per il calcolo di un totale?
   a) all'inizializzazione di una variabile di nome `totale`
   b) al programma `totale` che bisogna implementare
   c) alla memorizzazione di tutto l'input in un'unica variabile
   d) a una computazione, come la somma o il prodotto, che accumula diversi valori in un unico risultato
5. A cosa ci si riferisce quando si parla di goal/plan per il conteggio?
   a) a una computazione del numero di valori in un insieme
   b) all'utilizzo di un ciclo ternario la cui terza espressione è `i++`
   c) alla dichiarazione di una variabile per un conteggio
   d) all'uso di una dell'espressione `+=1`
6. A cosa ci si riferisce quando si parla di goal/plan per l'identificazione della prima occorrenza?
   a) al test per verificare che ci sia almeno un primo elemento in una sequenza
   b) all'identificazione della prima variabile che serve all'interno del programma
   c) alla ricerca del valore che compare più spesso in una sequenza
   d) alla ricerca lineare di un dato valore in una sequenza
7. A cosa ci si riferisce quando si parla di goal/plan per la ricerca di un massimo o di un minimo?
   a) all'identificazione dell'ultimo valore letto
   b) all'identificazione in una sequenza dell'elemento minimo o massimo rispetto a un criterio dato (valore, posizione, ecc.)
   c) alla ricerca del primo/ultimo elemento di un struttura (array, slice, ...)
   d) alle costanti che rappresentano i valori minimi e massimi per un tipo (`int`, `float64`, `...`) messe a disposizione nelle librerie del linguaggio di programmazione
8. A cosa ci si riferisce quando si parla di goal/plan per l'elaborazione su valori adiacenti?
   a) all'incremento di un indice in un ciclo in modo che assuma il valore successivo
   b) alla lettura dell'input due elementi alla volta
   c) a una computazione che si basa sull'elaborazione di un certo numero di elementi consecutivi in una sequenza
   d) al sommare ogni valore letto a quello successivo

2 Riconoscere i *plan* a partire da frammenti di codice

Analizzate i seguenti frammenti di codice, poi completate la tabella che trovate sotto. La variabile *parole* è una slice di stringhe mentre la variabile *numeri* è una slice di interi.

Frammento 10:

```
const N:=10

for i:=N; i>0; i--{
    fmt.Println(i)
}
```

Frammento 11:

```
n := numeri[0]

for i := 1; i < len(numeri); i++{
   fmt.Println(numeri[i] - numeri[i-1])
}
```

Frammento 12:

```
s := ""

for _, p:= range parole{
  s += p
}
fmt.Println(s)
```

Frammento 13:

```
pos := -1

for i := 0; i < len(parole); i ++{
  if parole[i][0] == 'a' {
      pos = i
      break
  }
}
```

Frammento 14:

```
r := -1

for _, n := range numeri {
  if n%2 == 0 && n > r {
    r = n
  }
}
fmt.Println(r)
```

Frammento 15:

```
t := 1

for _, n := range numeri{
    if(isPrime(n)){
        t *= n
    }
}
```

Frammento 16:

```
c:=0

for _, p:= range parole{
  if(p[0]=='a'){
      c++
  }
}
fmt.Println(c)
```

Riconoscete in ciascun frammento di codice il *plan* di iterazione utilizzato tra i seguenti *plan*: ripetizione, calcolo di un totale, conteggio, ricerca lineare, ricerca del valore estremo, elaborazione su valori adiacenti.

Frammento	*Plan*
Frammento 10	
Frammento 11	
Frammento 12	
Frammento 13	
Frammento 14	
Frammento 15	
Frammento 16	

3 Analizzare le specifiche per individuare i *goal*

Per ciascuna delle seguenti specifiche, rispondete a queste domande:

1. Qual è il *goal* principale da affrontare nell'implementazione del programma?
2. Quali variabili occorrono nella realizzazione del relativo *plan*? Come devono essere inizializzate?

1. **Esempio svolto**
   Specifiche: Scrivere un programma che legge da standard input una sequenza di parole e stampa quante delle parole inserite contengono il carattere 'a'.
   a) Per questo problema il *goal* principale da affrontare è quello del *conteggio*.
   b) Per la realizzazione del relativo *plan* occorre una variabile, ad esempio `count` (variabile con il ruolo di *passo-passo*), che tenga conto di quante parole contenenti il carattere 'a' sono state incontrate finora e che venga inizializzata a 0.
2. **Specifiche**: Scrivere una funzione che, data come parametro una slice di interi, trovi quelli che sono maggiori di 7 e multipli di 4 e ne restituisca il prodotto.
3. **Specifiche**: Scrivere un programma che legge da standard input 10 numeri e per ciascun numero letto stampa se è pari o dispari.
4. **Specifiche**: Scrivere un programma legge da standard input una sequenza di parole e stampa la prima parola che contentiene il carattere 'a', o "nessuna parola trovata".
5. **Specifiche**: Scrivere un programma che legge da standard input una sequenza di interi e, per ogni numero letto dopo il primo, stampa se è l'intero successivo a quello inserito in precedenza.
6. **Specifiche**: Scrivere una funzione che riceve come parametro una parola e restituisce `true` se la parola contiene una doppia, `false` altrimenti.
7. **Specifiche**: Scrivere un programma che legge da standard input una serie di parole e restituisce in output la lunghezza della parola più corta.
8. **Specifiche**: Scrivere un programma che legge da riga di comando un intero che rappresenta il saldo di un conto corrente. Il programma legge poi da standard input una serie di spese da addebitare sul conto e stampa il saldo finale.
9. **Specifiche**: Scrivere un programma che legge una sequenza di interi e stampi il più piccolo numero dispari letto. Il programma può leggere sia numeri positivi che negativi.
10. **Specifiche**: Scrivere un programma che legge da standard input una sequenza di interi positivi terminata da -1 e stampa il primo numero che supera 100, se presente; altrimenti stampa "nessun numero maggiore di 100".
11. **Specifiche**: Scrivere un programma che, data da standard input una serie di interi positivi terminata da 0, stampa '+' ogni volta che il nuovo valore è maggiore o uguale al precedente e '-' altrimenti.