UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Tutorato di programmazione

Piano Lauree Scientifiche - Progetto di Informatica

Scheda 1: input

Questa scheda serve a riflettere e lavorare su aspetti legati alla gestione dell'input
che si presentano costantemente nella progettazione di programmi.

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.

Gestione dell'input

Come prima attività di questa sessione vi proporremo alcuni programmi da analizzare rispetto alla gestione dell'input. In particolare vogliamo farvi soffermare sui seguenti aspetti, che si presentano costantemente nella progettazione di programmi:

raggruppamento I dati da elaborare sono "dati sparsi" (cioè separati, ciascuno con un suo nome) o "dati in serie" che possono essere elaborati in maniera uniforme?

memoria Quanti e quali dati serve tenere in memoria contemporaneamente? Se sono una serie, occorre tenere in memoria tutta la serie, solo l'ultimo valore letto o solo gli ultimi due (o un numero fisso di valori)? I dati sono già in memoria (in `os.Args`, passati come parametri, ecc.) o occorre salvarli?

tipo I dati di che tipo sono? E come vanno memorizzati? In una o più variabili separate o in strutture e quali (stringa, array, slice, ecc.)?

pronti I dati ricevuti da input sono pronti da elaborare o è necessario manipolarli prima in qualche modo (calcolarne/derivarne/estrarne altri dati)? Ad esempio, se ho memorizzato un numero ma ne devo elaborare le cifre, sarà necessario estrarre le cifre dal numero. Oppure, se ho memorizzato una riga in una stringa e devo lavorare sulle parole, dovrò identificare le singole parole all'interno della stringa.

canale I dati attraverso quale canale arrivano? Input standard, linea di comando, file, come parametri a una funzione o come valori restituiti da una funzione?

stop Nella lettura dei dati, quando mi devo fermare? Ne devo leggere un numero prefissato, devo leggere fino a un terminatore/sentinella o devo leggere fino al verificarsi di un evento? Devo leggere un dato "complesso" (stringa, riga di testo, file) tutto insieme o per parti (per caratteri, per stringhe, . . .)?

1 Array

Considerate il seguente programma `array.go` che legge da standard input 10 numeri ed un ulteriore numero e stampa la posizione dell'ultima occorrenza di quest'ultimo numero nella sequenza precedentemente letta. Il programma è stato già proposto nell'esercizio 5 della scheda di azzeramento.

Codice 1:

```go
package main

import "fmt"

func main() {
  var s [10]int
  var n int

  fmt.Println("Inserisci 10 numeri:")
  for i := 0; i < 10; i++ {
    fmt.Scan(&s[i])
  }
  fmt.Println("Inserisci un numero:")
  fmt.Scan(&n)

  var i int;
  for i = 9; i >= 0; i-- {
    if s[i] == n {
      break;
    }
  }
  fmt.Println(i);
}
```

Analizzate il codice sorgente e rispondete per iscritto alle domande che seguono.

1. Il programma `array.go` attraverso quale canale riceve i dati da elaborare?
2. Di che tipo/i sono i dati in input?
3. I dati ricevuti da input sono pronti da elaborare o vengono manipolati prima in qualche modo (per calcolarne/derivarne/estrarne altri dati)?
4. Quali, fra i dati in input, sono memorizzati in variabili singole e quali in strutture (array, slice, ecc.)?
5. Quando si ferma il programma nella lettura della sequenza di numeri?
6. Perché è necessario salvare alcuni dati in una struttura?

2 Maiuscole

Considerate il seguente programma `maiuscole.go` che legge da standard input una riga di testo ed emette in output il testo stesso, ma tutto in maiuscole. Il programma è stato già proposto nell'esercizio 7 della scheda di azzeramento.

Codice 2:

```go
package main

import (
  "bufio"
  "fmt"
  "os"
)

func main() {
  var s string
  const BASE_MINU = 'a'
  const BASE_MAIU = 'A'

  scanner := bufio.NewScanner(os.Stdin)
  fmt.Println("Inserisci la stringa da mettere in maiuscolo:")
  if scanner.Scan() {
    s = scanner.Text()
  }

  fmt.Println("La stringa e` lunga", len(s))

  for _, char := range s {
    if char >= 'a' && char <= 'z' {
      n := char - BASE_MINU
      char = BASE_MAIU + n
    }
    fmt.Print(string(char))
  }
  fmt.Println()
}
```

Analizzate il codice sorgente e rispondete per iscritto alle domande che seguono.

1. Il programma `maiuscole.go` attraverso quale canale riceve i dati da elaborare?
2. Di che tipo sono i dati ricevuti in input dal programma?
3. I dati ricevuti da input sono pronti da elaborare o vengono manipolati prima in qualche modo (per calcolarne/derivarne/estrarne altri dati)?

3 Precipitazioni

Considerate il seguente programma `precipitazioni.go` che legge una serie di numeri non negativi da standard input (mm di pioggia caduti giorno per giorno), fino a incontrare il numero 9999, e stampa la media delle precipitazioni.

Codice 3:

```go
package main

import "fmt"

const STOP = 9999

func main() {
  var dayRain, totRain, count int

  fmt.Print("mm di pioggia: ")
  for {
    fmt.Scan(&dayRain)
    if dayRain == STOP {
      break
    }

    count++
    totRain += dayRain
  }

  if count == 0 {
    fmt.Println("nessun dato disponibile")
  } else {
    averageRainfall := float64(totRain) / float64(count)
    fmt.Println("media:", averageRainfall)
  }
}
```

Analizzate il codice sorgente e rispondete per iscritto alle seguenti domande.

1. Il programma `precipitazioni.go` attraverso quale canale riceve i dati da elaborare?
2. Di che tipo sono i dati ricevuti in input dal programma?
3. È necessario memorizzare l'intera sequenza di dati ricevuti in input dal programma? Giustificate la risposta.
4. Cosa deve avvenire affinché il ciclo for termini?
5. I dati ricevuti da input sono pronti da elaborare o è necessario manipolarli prima in qualche modo (calcolarne/derivarne/estrarne altri dati)?
6. Se si vuole un programma che stampi il numero di giorni in cui le precipitazioni sono state sopra la media e quale è stato l'ultimo giorno di pioggia, per quali delle domande qui sopra deve cambiare la risposta e come cambia?
7. Modificate ora il programma in modo che stampi il numero di giorni in cui le precipitazioni sono state sopra la media e quale è stato l'ultimo giorno di pioggia, salvatelo con nome `precipitazioni_bis.go` e caricatelo sul sito di upload.

4 Conversione di temperature

Considerate il seguente programma `fahr_to_cels.go` che legge una serie di temperature da riga di comando e stampa le temperature equivalenti in gradi centrigradi.

Codice 4:

```go
package main

import "fmt"
import "os"
import "strconv"

func main() {
  const FATTORE_SCALA, ZERO = 5.0 / 9.0, 32.0

  if len(os.Args) < 2{
      fmt.Println("Nessun dato in input")
      return
  }

  for _, temp := range os.Args[1:] {
    fahr, _ := strconv.ParseFloat(temp, 64)
    fmt.Println(( fahr- ZERO) * FATTORE_SCALA)
  }
}
```

Analizzate il codice sorgente e rispondete per iscritto alle seguenti domande.

1. Il programma `fahr_to_cels.go` attraverso quale canale riceve i dati da elaborare?
2. Di che tipo sono i dati ricevuti in input dal programma?
3. Dove vengono salvati i dati in input?
4. I dati in input vengono elaborati direttamente o è necessario manipolarli prima in qualche modo (calcolarne/derivarne/estrarne altri dati)?
5. Se si vuole un programma che legga le temperature da standard input, terminando la lettura quando l'utente inserisce 9999, per quali delle domande qui sopra deve cambiare la risposta e come cambia?
6. Modificate ora il programma in modo che legga le temperature da convertire da standard input, terminando la lettura quando l'utente inserisce 9999. Salvate il programma con nome `fahr_to_cels_bis.go` e caricatelo sul sito di upload.

5 Strana sillabazione

Considerate il seguente programma `sillabe.go`. Questo programma implementa un nuovo e originale metodo di sillabazione. Il metodo consiste in questo: una sillaba è una sequenza massimale di caratteri consecutivi che rispettano l'ordine alfabetico. Per esempio, la parola `ambire` viene sillabata come `am-bir-e`: infatti la lettera a precede la lettera `m`, e le lettere `b`, `i` e `r` rispettano anch'esse l'ordine alfabetico. Il programma è dotato di una funzione `dividiInSillabe` che riceve come parametro una stringa e restituisce le sue sillabe sotto forma di una slice di stringhe. La funzione `main` legge da riga di comando una parola e stampa le sillabe che la compongono.

Codice 5:

```go
package main

import (
  "fmt"
  "os"
)

func main() {
  fmt.Println(dividiInSillabe(os.Args[1]))
}

func dividiInSillabe(parola string) (sliceSillabe []string) {
  var letteraPrecedente rune
  var sillaba string

  for _, letteraCorrente := range parola {
    if letteraCorrente < letteraPrecedente {
      sliceSillabe = append(sliceSillabe, sillaba)
      sillaba = string(letteraCorrente)
    } else {
      sillaba += string(letteraCorrente)
    }
    letteraPrecedente = letteraCorrente
  }
  sliceSillabe = append(sliceSillabe, sillaba)
  return
}
```

Analizzate il codice sorgente e rispondete per iscritto alle domande che seguono.

1. Il programma `sillabe.go` attraverso quale canale riceve i dati da elaborare?
2. E le sillabe da stampare?
3. La funzione `dividiInSillabe` attraverso quale canale riceve i dati da elaborare?
4. Quanti e di che tipo sono i dati da elaborare che riceve la funzione `dividiInSillabe`?

I prossimi esercizi sono di scrittura, modifica e debugging di programmi. Negli esempi di esecuzione useremo la convenzione di rappresentare l'input inserito dall'utente in grassetto e l'output prodotto dal programma in carattere monospace non grassetto.

6 Sequenza di interi

Considerate il seguente programma `sequenza_interi_bug.go` implementato in modo errato. Il programma legge da standard input una sequenza di numeri. La lettura termina quando il numero letto è minore del precedente. Il programma alla fine stampa la lunghezza della sequenza non decrescente letta.

Codice 6:

```go
package main

import "fmt"

func main() {
    var previous, current int

    count := 0
    for previous <= current {
        fmt.Scan(&previous)
        fmt.Scan(&current)
        count++
        previous = current
    }
    fmt.Println("Numeri validi inseriti:", count)
}
```

Analizzate le specifiche del programma e rispondete per iscritto alle seguenti domande.

1. Il programma quando deve terminare di leggere dati?
2. Il programma deve memorizzare i dati letti in variabili singole o in strutture (array, slice, ecc.)?
3. Quanti e quali dati deve tenere in memoria contemporaneamente ai fini dell'elaborazione?
4. Quali aspetti non sono gestiti correttamente dal programma `sequenza_interi_bug.go`?

Scrivete ora un programma `sequenza_interi.go` che rispetta le specifiche e produce l'output corretto.

7 Sequenza di lettere

Modificate il programma precedente in modo che legga da standard input una sequenza di lettere minuscole[1] separate da spazi o a-capo. La lettura deve terminare quando la lettera letta è minore (ovvero viene prima nell'alfabeto) della precedente. Il programma alla fine deve stampare la lunghezza della sequenza "non decrescente" e la massima differenza positiva tra lettere consecutive riscontrata.

[1] Si ricorda che con la funzione `fmt.Scan` non è possibile leggere caratteri, occorre utilizzare la funzione `fmt.Scanf`, la sintassi è `fmt.Scanf("%c", &carattere)` (si veda anche la documentazione).

Esempio di funzionamento

```
a b
e g f
Lunghezza: 4
Massima differenza riscontrata: 3
```

8 Prodotto dei pari

Specifiche: Scrivere un programma che legge da standard input una sequenza di interi terminata da zero e stampa il prodotto dei numeri pari inseriti. Ad esempio, se l'utente inserisce la sequenza `1 2 3 4 5 0` il programma stampa `8`.

Osservate che:

1. i dati arrivano da input standard
2. abbiamo una sequenza di dati che possono essere elaborati in serie, in modo uniforme
3. si tratta di numeri interi
4. non è necessario memorizzare prima tutti i numeri, poiché il prodotto può essere calcolato man mano che si leggono i numeri; basta quindi tenere in memoria in una variabile singola l'ultimo numero letto
5. la lettura dei dati deve interrompersi quando si legge 0

Prima di iniziare a scrivere il programma `prodotto_pari.go` leggete attentamente le specifiche e le osservazioni. Poi progettate e scrivete il programma.

9 Parola più lunga

Specifiche: Scrivere un programma che riceve da riga di comando una serie di lunghezza arbitraria di parole e stampa la parola più lunga. Nel caso in cui vi siano più parole di lunghezza massima, il programma stampa l'ultima. Ad esempio, se l'utente inserisce la sequenza di parole `uno due quattro tredici` il programma stampa `tredici`.

Osserva che:

1. i dati arrivano da riga di comando
2. abbiamo una sequenza di dati che possono essere elaborati in serie, in modo uniforme
3. si tratta di stringhe
4. tutti i dati inseriti all'avvio del programma sono già memorizzati nella slice os.Args

Prima di iniziare a scrivere il programma `parola_lunga.go` leggete attentamente le specifiche e le osservazioni. Poi progettate e scrivete il programma.

10 Parola più lunga Bis

Modificate il programma dell'esercizio precedente in modo che, invece di leggere i dati da riga di comando, legga una serie di lunghezza arbitraria[2] di parole da standard input.

Prima di iniziare a scrivere il programma `parola_lunga_bis.go`, rispondete per iscritto alle seguenti domande. Poi progettate e scrivete il programma.

1. Attraverso quale canale devono essere forniti i dati al programma?
2. Di che tipi sono i dati in input?
3. I dati sono "sparsi" o sono una "serie"?
4. I dati ricevuti da input sono pronti da elaborare o è necessario manipolarli prima in qualche modo (calcolarne/derivarne/estrarne altri dati)?
5. Quanti e quali dati è indispensabile avere in memoria contemporaneamente ai fini dell'elaborazione richiesta?
6. Il programma quando deve fermarsi nella lettura dei dati?

[2] Il programma termina la lettura quando riceve EOF, che da tastiera si produce con `Ctrl+D`

11 Indovina un numero

Specifiche: Scrivere un programma `indovina_numero.go` dotato di:

- una funzione `numeroDaIndovinare` che, dati due parametri interi `lower` e `upper`, genera un numero intero casuale tra `lower` e `upper` inclusi;
- una funzione `main` che utilizza la funzione `numeroDaIndovinare` per generare un numero casuale da indovinare tra 1 e 100; legge numeri interi inseriti dall'utente; nel caso in cui il numero inserito sia maggiore del numero da indovinare il programma stampa '-', se è minore stampa '+', altrimenti stampa 'Hai indovinato!' e termina.

Esempio di esecuzione con `19` come numero generato in modo casuale dalla funzione `numeroDaIndovinare`:

```
Inserisci un numero: 2
+
Inserisci un numero: 30
-
Inserisci un numero: 19
Hai indovinato!
```

Leggete attentamente le specifiche e, prima di iniziare a scrivere il programma `indovina_numero.go`, rispondete per iscritto alle seguenti domande. Poi progettate e scrivete il programma.

1. Il `main` attraverso quale canale riceve i numeri dati dall'utente? E il numero da indovinare?
2. Di che tipi sono i dati che riceve il `main`?
3. I numeri forniti dall'utente sono dati "sparsi" o sono una "serie"?
4. I dati ricevuti da input sono pronti da elaborare o è necessario manipolarli prima in qualche modo (calcolarne/derivarne/estrarne altri dati)?
5. Quanti e quali dei dati forniti dall'utente è indispensabile avere in memoria contemporaneamente ai fini dell'elaborazione richiesta?
6. Il programma quando deve fermarsi nella lettura dei dati forniti dall'utente?

12 Numero cancellato

Specifiche: Data su linea di comando una stringa costituita da cifre, alcune delle quali eventualmente seguite da #, stampare il prodotto ottenuto moltiplicando tra loro le cifre non seguite da #. Ad esempio, se il programma legge da riga di comando la stringa `12#345#6`, il programma stampa `72` (cioè `1 * 3 * 4 * 6`).

Leggete attentamente le specifiche e, prima di iniziare a scrivere il programma `numero_cancellato.go`, rispondete per iscritto alle seguenti domande. Poi progettate e scrivete il programma.

1. Il programma attraverso quale canale riceve i dati?
2. Di che tipo sono i dati in input?
3. I dati ricevuti da input sono pronti da elaborare o è necessario manipolarli prima in qualche modo (calcolarne/derivarne/estrarne altri dati)?
4. I dati da elaborare sono dati "sparsi" o "in serie"?
5. Quanti e quali dei dati in input è necessario considerare contemporaneamente ai fini dell'elaborazione richiesta?

13 Cifrario di Cesare

Specifiche: Svetonio nella *Vita dei dodici Cesari* racconta che Giulio Cesare usava per le sue corrispondenze riservate un codice di sostituzione molto semplice: fissato un numero *k* (detto chiave di cifratura), sostituiva ciascuna lettera con quella *k* posizioni più avanti nell'alfabeto, eventualmente continuando dalla 'a' quando si raggiunge la 'z' e dalla 'A' quando si raggiunge la 'Z'.

Scrivere un programma che implementa il cifrario di Cesare: il programma legge da standard input la chiave di cifratura *k* e una riga di testo da cifrare ed emette su standard output il testo cifrato. I caratteri che non sono lettere non vanno cifrati.

Esempio di funzionamento

```
Chiave: 5
Testo da cifrare: 1, 2, 3. mamma li Turchi
Testo cifrato: 1, 2, 3. rfrrf qn Yzwhmn
```

```
Chiave: 21
Testo da cifrare: rfrrf qn Yzwhmn
Testo cifrato: mamma li Turchi
```

Leggete attentamente le specifiche e, prima di iniziare a scrivere il programma `cifrario.go`, rispondete per iscritto alle seguenti domande. Poi progettate e scrivete il programma. Nella nota a piè pagina [3] trovate un suggerimento su come procedere per cifrare ciascuna lettera *k* posizioni più avanti, nel caso vi occorra.

1. Il programma attraverso quale canale riceve i dati?
2. Di che tipo sono i dati in input?
3. I dati ricevuti in input sono pronti da elaborare o è necessario manipolarli prima in qualche modo (calcolarne/derivarne/estrarne altri dati)?

[3] Data una chiave *k* e la posizione *n* nell'alfabeto di una lettera da cifrare, la posizione nell'alfabeto della corrispondente lettera cifrata è data da:
n = (n + k) % ALPHABET_LEN