1 Esercizi in pseudocodice

Questa dispensa propone esercizi sulla scrittura di algoritmi in un linguaggio semi–formale, utile all'acquisizione delle abilità essenziali per implementare algoritmi in qualsiasi linguaggio di programmazione.

1.1 Algoritmi ed esecutori

Dato un problema ed un opportuno metodo risolutivo, la *risoluzione* di tale problema è quel processo che trasforma i *dati in ingresso* nei corrispondenti *dati finali*. Affinché la risoluzione di un problema possa essere realizzata attraverso l'uso di un calcolatore, il metodo risolutivo deve poter essere definito come una sequenza di azioni o istruzioni elementari, ovvero occorre codificare la risoluzione del problema in un algoritmo. Si dice *esecutore* quella macchina astratta capace eseguire le istruzioni specificate dall'algoritmo.

Un *algoritmo* è una sequenza finita di istruzioni, definita con precisione, che portano alla risoluzione di un compito. Un algoritmo è tale se possiede le seguenti proprietà:

- **Eseguibilità**: ogni istruzione deve essere eseguibile dal calcolatore in tempo finito;
- **Non–ambiguità**: ogni istruzione deve essere univocamente interpretabile dal calcolatore;
- **Finitezza**: il numero totale di azioni da eseguire, per ogni insieme di dati in ingresso, deve essere finito.

1.2 Il linguaggio "SIMITA"

Introduciamo uno pseudolinguaggio di programmazione, che chiameremo SIMITA[1]. Vogliamo che questo linguaggio sia in grado di gestire:

- Dati singoli (sia quelli in ingresso, sia quelli ottenuti durante l'esecuzione del programma, sia quelli finali);
- Scelte tra alternative;
- Ripetizioni;
- Gruppi di dati.

Immaginiamo di poter gestire i valori dei dati su dei "foglietti". Sui di essi possiamo *scrivere*, *leggere*, *cancellare* e *riscrivere* (come nella memoria dei calcolatori). Per indicare scrittura di un numero N su foglietto f:

$f \leftarrow N$

Allo stesso modo indichiamo la scrittura del contenuto del foglietto g nel foglietto f come:

$f \leftarrow g$

In entrambi i casi l'operazione di scrittura cancellerà tutto ciò che era scritto sul foglietto f. Inoltre, disponiamo di altre espressioni per codificare delle funzionalità di cui dispone l'esecutore (nel nostro caso l'elaboratore):

- $leggi(f)$: indica l'operazione di lettura di un valore introdotto dall'utente (ad esempio, da tastiera) e la scrittura di tale valore nel foglietto f;
- $stampa(f)$: indica l'operazione di lettura del valore contenuto nel foglietto f e la stampa (ad esempio, a video) di tale valore. L'istruzione $stampa$ permette anche di stampare dei caratteri, ad esempio $stampa$("ciao come stai") stampa a video i caratteri "ciao come stai".

Gestiamo nel linguaggio SIMITA il costrutto condizionale, capace di esprimere il concetto di scelta tra alternative, nel seguente modo:

```
Istruzione 0

Se condizione allora
    Istruzione 1
```

[1] Se invece vi volete divertire con un linguaggio differente `https://github.com/esseks/monicelli`

```
    Istruzione 2
Altrimenti
    Istruzione 3
    Istruzione 4
chiudi Se

Istruzione 5
```

Nell'esempio precedente l'algoritmo prescrive al calcolatore di eseguire innanzitutto "Istruzione 0". Successivamente, esso valuta "condizione": se "condizione" risulta vera, allora vengono eseguite "Istruzione 1" ed "Istruzione 2", in caso contrario vengono eseguite "Istruzione 3" ed "Istruzione 4". Al termine viene eseguita "Istruzione 5". Il blocco "**Altrimenti**" è opzionale: se assente, nel caso in cui "condizione" risulti falsa l'esecuzione passa direttamente ad "Istruzione 5". Non vi è limite al numero di istruzioni specificate in ogni blocco del costrutto condizionale.

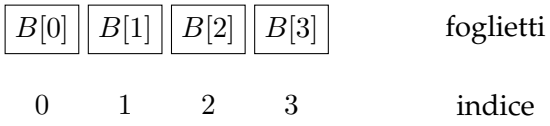
Per poter eseguire più volte una stessa operazione, in SIMITA, abbiamo un costrutto ciclico (o iterativo), capace di esprimere il concetto di ripetitività:

```
Istruzione 0        //questo è un commento

Finché condizione esegui
    Istruzione 1
    Istruzione 2
chiudi Finché

Istruzione 3
```

Nell'esempio precedente la sequenza di istruzioni 1 e 2 viene eseguita un numero *finito* di volte. L' "Istruzione 0" viene eseguita una sola volta. Dopodiché viene valutata "condizione": se risulta vera, allora vengono eseguite "Istruzione 1" ed "Istruzione 2". Viene poi valutata nuovamente "condizione": se vera, si ripete l'esecuzione di "Istruzione 1" ed "Istruzione 2". L'iterazione *termina*, e quindi non vengono più eseguite "Istruzione 1" ed "Istruzione 2", solo quando "condizione" diventa falsa. Quando questo avviene, si esce dal ciclo e si esegue "Istruzione 3". Solitamente, al fine di evitare che il costrutto ciclico non abbia fine, è buona norma che le istruzioni all'interno del ciclo modifichino i foglietti che vengono valutati in "condizione".

Sempre nell'esempio precedente è stata inserita una porzione di istruzione, detta *commento*, che non verrà considerata dall'esecutore, ossia "questo è un commento". Solitamente un commento viene inserito per descrivere in linguaggio naturale ciò che le istruzioni specificano e quindi per chiarificare a chi legge le istruzioni il compito svolto dall'algoritmo.

SIMITA ha anche la nozione di *blocchi di foglietti*:



$B[0]$ indica il primo foglietto del blocco;

$B[1]$ indica il secondo foglietto del blocco;

...

$B[n-1]$ indica l'n-simo foglietto del blocco.

Si noti che ogni foglietto $B[n]$ è, a tutti gli effetti, un normale foglietto. Quindi, ad esempio, le seguenti istruzioni sono valide:

```
i ← 1
f ← B[i]        //scrive il contenuto del secondo foglietto
                //B[1] in f

B[i] ← 3        //scrive 3 nel secondo foglietto B[1]

stampa(B[4])      //stampa il contenuto del quinto foglietto

leggi(B[i])      //legge un dato
                  //e lo scrive nel secondo foglietto
```

1.3 Esercizi

Esercizio 1.1

Scrivere in linguaggio SIMITA l'algoritmo per descrivere il problema di svegliarsi ed uscire di casa, sapendo che:

- Una volta sveglio si deve spegnere la sveglia (`spegniSveglia()`);
- Viviamo in una casa di 3 piani e prima di uscire dobbiamo chiudere tutte le finestre (se sono aperte) ad ogni piano;
- Dormiamo al secondo piano, dove c'è il bagno, in cui ci dobbiamo fare la doccia (`doccia()`);
- Al primo piano c'è la cucina dove dobbiamo fare colazione (`colazione()`), se è domenica facciamo una colazione sostanziosa (`colazioneSostanziosa()`);
- Al piano terreno c'è la porta di uscita che dobbiamo aprire e chiudere.

L'utente ci comunica all'inizio quali sono le finestre aperte e se è domenica. Ogni volta che eseguiamo un'azione dobbiamo comunicarla all'utente stampandola a video.

Esercizio 1.2

Calcolare il prodotto di due numeri interi positivi e stamparlo a video. L'esecutore è in grado di eseguire solamente somme e sottrazioni.

Esercizio 1.3

Calcolare il fattoriale di un numero intero e positivo e stamparlo a video. L'esecutore è in grado di eseguire moltiplicazioni.

Esercizio 1.4

1. Scrivere un algoritmo per valutare e stampare a schermo se un numero intero positivo è pari o dispari utilizzando solo le 4 operazioni fondamentali.
2. Scrivere il medesimo algoritmo supponendo di avere un operatore div che restituisce il risultato intero della divisione (ad esempio, $7\ div\ 3$ ha come risultato 2).

Esercizio 1.5

Determinare se una funzione continua ha uno zero nell'intervallo $[a, b]$ (i cui estremi sono forniti dall'utente). Si faccia ricorso al seguente teorema (teorema di Bolzano):

- Ipotesi: $f : [a, b] \mapsto \mathbb{R}$, continua in $[a, b] \subseteq \mathbb{R}$, tale che $f(a) \cdot f(b) < 0$
- Tesi: $\exists c \in [a, b]$ tale che $f(c) = 0$ (zero di f)

Per valutare la funzione f in un punto generico a potete utilizzare l'operatore

$$g \leftarrow calcola(f, a)$$

che scrive il risultato di $f(a)$ sul foglietto g.

Esercizio 1.6

1. Scrivere un programma che data una sequenza di numeri (positivi) in ingresso restituisce il maggiore e la sua posizione nella sequenza. Supponiamo che la sequenza abbia 10 numeri.

2. Si provi ad implementare un algoritmo analogo a quello dell'esercizio precedente, utilizzando un solo ciclo ed eliminando l'assunzione che i numeri inseriti siano positivi.

3. Si estenda l'algoritmo per calcolare la media m dei valori inseriti.

4. Si estenda l'algoritmo per calcolare e stampare a video la differenza $N[i] - m$ tra ogni elemento della sequenza e la media della sequenza.

5. È possibile implementare questi algoritmi senza l'uso di blocchi di foglietti?

6. Si estenda l'algoritmo per calcolare anche la deviazione standard della sequenza, supponendo di avere a disposizione una funzione che calcola la radice quadrata di un numero (`sqrt()`). È possibile risolvere questo problema senza ricorrere ai blocchi di fogli?

Esercizio 1.7

Determinare almeno uno zero di una funzione continua f nell'intervallo $[a, b]$.

Soluzioni

Soluzione dell'esercizio 1.1

Risoluzione del problema: prima di tutto dobbiamo acquisire e memorizzare i dati relativi alle finestre e al fatto che sia o meno domenica. Dopodichè segnaleremo all'utente che abbiamo iniziato la procedura di uscita e spegneremo la sveglia. Per ogni piano controlleremo se le finestre sono chiuse e, solo se sono aperte, le chiuderemo. Ad ogni piano faremo un'azione differente (doccia al secondo piano, colazione al primo, aprire e chiudere la porta al pian terreno). Per quanto riguarda l'azione della colazione dovremo controllare se è domenica e, solo in quel caso, fare una colazione abbondante. Non appena usciti, comunicheremo all'utente che abbiamo finito la procedura di uscita.

```
leggi(domenica)        //1 se domenica, 0 altrimenti
nPiano ← 2
Finché nPiano > 0 esegui
    leggi(fin[nPiano])        //1 se aperte, 1 se chiuse
    nPiano ← nPiano − 1
chiudi Finché
porta = 0        //0 se chiusa, 1 se aperta
stampa("Inizio procedura uscita")
spegniSveglia()
nPiano ← 2
Finché nPiano > 0 esegui
    Se fin[nPiano] = 1 allora
        fin[nPiano] = 0
        stampa("Chiuse le finestre al piano nPiano")
    chiudi Se
    Se nPiano = 2 allora
        doccia()
        stampa("Doccia fatta")
    chiudi Se
    Se nPiano = 1 allora
        Se domenica = 1 allora
            colazioneSostanziosa()
            stampa("Colazione sostanziosa fatta")
        Altrimenti
            colazione()
            stampa('Colazione fatta')
        chiudi Se
    chiudi Se
    Se nPiano = 0 allora
        porta = 1
        stampa("Porta aperta")
        porta = 0
        stampa("Porta chiusa")
    chiudi Se
    nPiano ← nPiano − 1
chiudi Finché
stampa("Fine procedura uscita")
```

Soluzione dell'esercizio 1.2

Risoluzione del problema: dopo aver letto e memorizzato i due numeri, calcoliamo il prodotto in modo cumulativo utilizzando un ciclo. Infine stampiamo il risultato a video.

Soluzione 1

```
leggi(f)        //leggi il primo numero
leggi(g)        //leggi il secondo numero

risultato ← 0       //inizializzo il risultato

Finché f > 0 esegui
    risultato ← risultato + g       //aggiungi g per f volte
    f ← f − 1
chiudi Finché

stampa(risultato)
```

Soluzione 2

```
leggi(f)        //leggi il primo numero
leggi(g)        //leggi il secondo numero

Se g < f allora       //assicuriamoci che f < g
    tmp ← f       //metto f su un foglietto temporaneo
    f ← g       //g contiene il numero minore
    g ← tmp
Altrimenti
    tmp ← g       //l'altro lo scrivo in g
chiudi Se

Finché f > 1 esegui       //ripeti la somma f − 1 volte
    tmp ← tmp + g       //aggiungi g per f volte
    f ← f − 1
chiudi Finché

stampa(tmp)
```

In alcuni casi (ad esempio, $f = 10000$ e $g = 2$). questo algoritmo è più efficiente del primo in quanto, se eseguito, userebbe meno istruzioni in totale. L'efficienza di un algoritmo rispetto ad un altro avviene rispetto a due caratteristiche:

- **Complessità temporale**: uso meno istruzioni in totale;
- **Complessità spaziale**: uso meno foglietti in totale;

In questo caso entrambi gli algoritmi hanno la stessa complessità spaziale, in quanto usano entrambi 3 foglietti in tutto. Il secondo algoritmo "spreca" 4 istruzioni all'inizio per poter poi risparmiarne nel caso in cui $f \gg g$.

Soluzione dell'esercizio 1.3

Risoluzione del problema: si legge e memorizza il numero dato. Si inizializza il risultato al numero dato. Si esegue un ciclo moltiplicando il risultato per numeri descrescenti partendo dal dato meno 1 fino ad 1. Si stampa il risultato a video.

```
leggi(f)
fattoriale ← f

Finché f > 2 esegui
    f ← f − 1
    fattoriale ← fattoriale ∗ f
chiudi Finché

stampa(fattoriale)
```

Soluzione dell'esercizio 1.4

1. Risoluzione del problema: dopo aver letto e memorizzato il numero, sottraiamo 2 finché il risultato non è 1 o 0. Se il risultato è 1, il numero di partenza è dispari, altrimenti è pari.

```
leggi(f)        //leggi il numero

Finché f > 2 esegui
    f ← f − 2
chiudi Finché

Se f = 1 allora
    stampa(dispari)
Altrimenti
    stampa(pari)
chiudi Se
```

2. Risoluzione del problema: dopo aver letto e memorizzato il numero, dividiamo per 2 usando l'operatore div e successivamente moltiplichiamo per 2. Se il risultato è uguale al numero di partenza, il numero è pari, altrimenti è dispari.

```
leggi(f)        //leggi il numero

g ← 2 * (f div 2)

Se f = g allora
    stampa(pari)
Altrimenti
    stampa(dispari)
chiudi Se
```

Soluzione dell'esercizio 1.5

Risoluzione del problema: leggiamo e memorizziamo gli estremi dell'intervallo forniti dall'utente.

```
leggi(a)        //lettura estremo inferiore
leggi(b)        //lettura estremo superiore

A ← calcola(f, a)
B ← calcola(f, b)

Se A * B < 0 allora
    stampa("esiste almeno uno zero")
Altrimenti
    stampa("potrebbe non esistere alcuno zero")
chiudi Se
```

Soluzione dell'esercizio 1.6

1. Risoluzione del problema: innanzitutto procediamo all'acquisizione dei 10 numeri. All'inizio il massimo è zero e la sua posizione è meno 1. Confrontiamo questo massimo con il primo numero della sequenza dei numeri acquisiti. Se il numero della sequenza è maggiore lo sostituiamo al massimo e registriamo la sua posizione. Ripetiamo l'operazione con tutti i numeri della sequenza e infine stampiamo a video valore e posizione del massimo.

```
max ← 0
i ← 0
pos ← −1

Finché i < 10 esegui
    leggi(N[i])
    i ← i + 1
chiudi Finché

i ← 0
Finché i < 10 esegui
    Se N[i] > max allora
        max ← N[i]
        p ← i
    chiudi Se
    i ← i + 1
chiudi Finché

stampa("il valore massimo è:")
stampa(max)

stampa("ed è in posizione:")
stampa(p)
```

2. Risoluzione del problema: prima di tutto notiamo che il primo ciclo, quello di acquisizione, memorizza i numeri nel blocco di foglietti. Tuttavia, il secondo ciclo, non fa altro che esaminarli (nello stesso ordine). Quindi, è possibile eliminare del tutto la necessità di memorizzare i numeri, e svolgere le istruzioni per il controllo "maggiore di" subito dopo l'acquisizione.

   Inoltre, eliminando l'assunzione di numeri positivi, non abbiamo un estremo inferiore (zero). Perciò è necessario inizializzare il massimo al primo valore incontrato (che potrebbe essere inferiore a zero).

```
i ← 0
p ← i
n ← 0      //foglietto per il numero letto

Finché i < 10 esegui
    leggi(n)

    Se i = 0 ∨ n > max allora      //primo numero o maggiore
        max ← n      //salvo il valore massimo
        p ← i      //salvo la posizione
    chiudi Se

    i ← i + 1      //incremento il contatore
chiudi Finché

stampa("il valore massimo è:")
stampa(max)

stampa("ed è in posizione:")
stampa(p)
```

3. Risoluzione del problema: il calcolo della media richiede che, durante l'acquisizione, si calcoli anche la somma incrementale di tutti i numeri letti. Al termine del ciclo, tale somma verrà divisa per il numero di input immessi.

```
i ← 0
p ← i
media ← 0
n ← 0      //foglietto per il numero letto

Finché i < 10 esegui
    leggi(n)

    Se i = 0 ∨ n > max allora      //primo numero o maggiore
        max ← n      //salvo il valore massimo
        p ← i      //salvo la posizione
    chiudi Se

    media ← media + n      //somma incrementale
    i ← i + 1      //incremento il contatore
chiudi Finché

media ← media/(i + 1)

stampa("il valore massimo è:")
stampa(max)

stampa("ed è in posizione:")
stampa(p)

stampa("il valore medio è:")
stampa(media)
```

4. Risoluzione del problema: per calcolare lo scarto dalla media di ogni elemento è necessario memorizzare tutti gli elementi, perché la media sarà calcolabile solo dopo aver letto tutti i numeri. Quindi:

```
i ← 0
p ← i
media ← 0
n ← 0      //foglietto per il numero letto

Finché i < 10 esegui
    leggi(n)

    Se i = 0 ∨ n > max allora      //primo numero o maggiore
        max ← n      //salvo il valore massimo
        p ← i      //salvo la posizione
    chiudi Se

    N[i] ← n      //memorizzazione elemento i-esimo
    media ← media + n      //somma incrementale
    i ← i + 1      //incremento il contatore
chiudi Finché

media ← media/(i + 1)
i ← 0

Finché i < 10 esegui      //ciclo su tutti i foglietti
    stampa("lo scarto dalla media di ")
    stampa(N[i])
    stampa("è: ")
    stampa(N[i] − media)      //scarto dalla media
    i ← i + 1
chiudi Finché

stampa("il valore massimo è:")
stampa(max)

stampa("ed è in posizione:")
stampa(p)

stampa("il valore medio è:")
stampa(media)
```

5. Risoluzione del problema: per il calcolo del massimo e della media non è necessario memorizzare i valori inseriti, quindi non sono necessario blocchi di foglietti. Per quanto riguarda lo scarto della media, non è possibile implementare un algoritmo incrementale, poiché il valore della media è noto solo alla fine dell'acquisizione dei valori.

6. Risoluzione del problema: la deviazione standard è definita come la media degli scarti quadratici dalla media, ovvero:

$$\mathrm{devstd}(\{x_1, \ldots, x_M\}) = \sqrt{\frac{\sum_{i=1}^{M}(x_i - m)^2}{M}}$$

si deve quindi calcolare la somma degli scarti quadratici $N[i] - m$, dividere tale numero per il contatore ed estrarne la radice quadrata.

```
i ← 0
p ← i
media ← 0
devstd ← 0
n ← 0      //foglietto per il numero letto

Finché i < 10 esegui
    leggi(n)

    Se i = 0 ∨ n > max allora      //primo numero o maggiore
        max ← n      //salvo il valore massimo
        p ← i      //salvo la posizione
    chiudi Se

    N[i] ← n      //memorizzazione elemento i-esimo
    media ← media + n      //somma incrementale
    i ← i + 1      //incremento il contatore
chiudi Finché

media ← media/(i + 1)

i ← 0
Finché i < 10 esegui      //ciclo su tutti i foglietti
    scarto ← (N[i] − media)
    scarto ← scarto ∗ scarto      //scarto quadratico
    devstd ← devstd + scarto      //somma scarti
    i ← i + 1
chiudi Finché

devstd ← devstd/(i + 1)      //media somma scarti
devstd ← sqrt(devstd)      //deviazione standard

stampa("il valore massimo è:")
stampa(max)

stampa("ed è in posizione:")
stampa(p)

stampa("il valore medio è:")
stampa(media)

stampa("la deviazione standard è:")
stampa(devstd)
```

Diversamente dal caso degli scarti esiste una formula per il calcolo incrementale della deviazione standard, che può essere derivato analiticamente dalla sua definizione. Nell'algoritmo precedentemente descritto non ha senso utilizzare tale formula in quanto vogliamo poi stampare gli scarti.

Soluzione dell'esercizio 1.7

```
leggi(a)        //lettura estremo inferiore
leggi(b)        //lettura estremo superiore

A ← calcola(f, a)
B ← calcola(f, b)

Se A * B > 0 allora
    stampa("potrebbe non esistere alcuno zero")
Altrimenti
    K ← 1

    Finché K ≠ 0 esegui
        c ← (b + a)/2
        K ← calcola(f, c)

        Se A * K < 0 allora
            b ← c
            B ← K
        chiudi Se
        Se B * K < 0 allora
            a ← c
            A ← K
        chiudi Se

    chiudi Finché
chiudi Se

stampa(c)
```

Attenzione: Il programma potrebbe non terminare nel caso in cui lo zero non sia in un valore razionale ($c \in \mathbb{R} \setminus \mathbb{Q}$), poiché K è sempre un razionale ($K \in \mathbb{Q}$).

Inoltre, si osservi che il programma non gestisce il caso in cui lo zero sia $c = a$ o $c = b$.