Capitolo 16

Iterazione

Accesso a sequenze

- I problemi esaminati in questo capitolo hanno lo scopo di accedere ed elaborare delle sequenze
  - sequenze immesse dall'utente – lette dalla tastiera
    - di lunghezza nota
    - con un metodo che consente di sapere quando è stata letta completamente
    - di cui è noto l'ultimo elemento o una sua proprietà
  - sequenze generate con una variabile di controllo
  - sequenze ottenute scandendo gli elementi di una sequenza con accesso posizionale
  - file di testo
- Queste tecniche possono essere applicate anche per altre tipologie di sequenze
  - array e collezioni

Accumulazione

- **Accumulazione**
  - calcolare una proprietà sintetica (cumulata) da una sequenza di valori

- Tecnica dell'accumulazione
  - una variabile di controllo (**accumulatore**)
  - una **operazione di accumulazione**, binaria

Lettura e somma di una sequenza di numeri interi


```
int numero;    // elemento corrente della sequenza
int somma;    // somma degli elementi della sequenza

/* leggi una sequenza di numeri interi e
 * calcolane la somma */
/* inizialmente somma vale zero */
somma = 0;
/* finché ci sono altri elementi nella sequenza,
 * leggili e sommali a somma */
while (scanner.hasNextInt()) { // finché ci sono altri elementi
    /* leggi un elemento della sequenza */
    numero = scanner.nextInt();
    /* incrementa somma di numero */
    somma = somma + numero;
}
/* visualizza somma */
System.out.println(somma);
```

Fattoriale di un numero naturale


```
... calcola il fattoriale f di n ...
int f;              // il fattoriale di n
int i;              // per iterare tra 1 e n

/* calcola il fattoriale di n */
f = 1;
/* moltiplica f per ogni numero intero i
 * compreso tra 1 e n */
for (i=1; i<=n; i++)
    f = f*i;
... il fattoriale di n è f ...
```

Invarianti di ciclo

- In una istruzione ripetitiva, un **invariante di ciclo** è una proprietà che risulta verificata
  - ad ogni esecuzione del corpo dell'istruzione ripetitiva
  - in un particolare punto del corpo dell'istruzione ripetitiva

- Nel caso della lettura e somma di una sequenza di numeri
  - un invariante di ciclo è che **somma** è uguale alla somma degli elementi letti finora dalla tastiera
  - il punto in cui la proprietà risulta verificata ad ogni esecuzione del corpo dell'istruzione ripetitiva è al termine dell'esecuzione del corpo

Accumulazione


```
... dichiarazione della variabile accumulatore ...

accumulatore = elemento neutro dell'operazione
               di accumulazione ;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi al prossimo elemento ...
    accumulatore = applica l'operazione di accumulazione
                   all'accumulatore e all'elemento
                   corrente ;
}

... altre elaborazioni ...
```

Calcolo di una sottostringa

- Scrivere un metodo di classe **String sottostringa(String s, int inizio, int fine)** che calcola la sottostringa di **s** che consiste dei caratteri compresi tra quello di posizione **inizio** e quello di posizione **fine**-1
  - **sottostringa("automobile", 2, 6)** deve restituire **"tomo"**
  - usa solo le seguenti operazioni sulle stringhe
    - **int length()**
    - **char charAt()**
    - **+** (concatenazione)
  - motivazioni
    - accesso posizionale su stringhe

Calcolo di una sottostringa


```
/* Calcola la sottostringa di s compresa tra i
 * caratteri di posizione da inizio a fine-1. */
public static String sottostringa(String s, int inizio, int fine) {
    // pre: s!=null &&
    //      inizio>=0 && inizio<=s.length() &&
    //      fine>=inizio && fine<=s.length()
    String ss;    // la sottostringa di s tra inizio e fine
    int i;        // indice per la scansione di s
    /* calcola la sottostringa, come concatenazione
     * dei caratteri tra la posizione inizio (inclusa)
     * e fine (esclusa) */
    ss = "";        // la stringa vuota è l'elemento neutro
                    // della concatenazione
    for (i=inizio; i<fine; i++)
        ss = ss + s.charAt(i);  // concatenazione a destra
                                // e conversione automatica
    return ss;
}
```

Conteggio

- I problemi di **conteggio** sono un caso particolare dei problemi di accumulazione
  - l'accumulatore è un **contatore**
  - l'operazione di accumulazione è un incremento unitario
  - l'aggiornamento deve essere eseguito sempre (conteggio), oppure condizionatamente (conteggio condizionale) al soddisfacimento di una proprietà da parte dell'elemento corrente della sequenza

Conteggio (condizionale)

```
int contaProprietà;    // numero di elementi che
                       // soddisfano la proprietà

contaProprietà = 0;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi al prossimo elemento ...
    if (l'elemento corrente soddisfa la proprietà)
        contaProprietà++;
}

... altre elaborazioni ...
```

Esercizio

- Scrivere una applicazione che legge dalla tastiera una sequenza di numeri interi e ne conta e visualizza, separatamente, il numero degli elementi positivi e il numero degli elementi negativi (gli zeri non vanno contati)

```
Scrivi una sequenza di numeri interi
10  20  0  -10  4 -8
La sequenza contiene 3 elementi positivi e 2 negativi
```

Verifica esistenziale

- I problemi di **verifica esistenziale** sono un altro caso particolare dei problemi di accumulazione
  - bisogna determinare se una sequenza di elementi contiene almeno un elemento che soddisfa una certa proprietà
  - come accumulatore può essere usata una variabile booleana che indica se la sequenza contiene almeno un elemento che soddisfa la proprietà

Verifica se una sequenza contiene almeno uno zero


```
... legge una sequenza di numeri interi e
    verifica se contiene almeno uno zero ...
int numero;       // elemento corrente della sequenza
boolean contieneZero;    // la sequenza contiene almeno
                         // un elemento uguale a zero
... visualizza un messaggio ...
/* legge la sequenza e verifica
 * se contiene almeno uno zero */
contieneZero = false;
while (scanner.hasNextInt()) {
    /* legge il prossimo elemento della sequenza */
    numero = scanner.nextInt();
    /* se numero vale zero, allora la sequenza
     * contiene almeno uno zero */
    if (numero==0)
        contieneZero = true;
}

... il risultato è contieneZero ...
```

Verifica esistenziale


```
int contieneProprietà;    // almeno un elemento
                          // soddisfa la proprietà

contieneProprietà = false;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi al prossimo elemento ...
    if (l'elemento corrente soddisfa la proprietà)
        contieneProprietà = true;
}

... altre elaborazioni ...
```

Un errore comune

- Un errore comune nella verifica esistenziale

```
contieneZero = false;
while (scanner.hasNextInt()) {
    /* legge il prossimo elemento della sequenza */
    numero = scanner.nextInt();
    /* se numero vale zero, allora la sequenza
     * contiene almeno uno zero */
    if (numero==0)
        contieneZero = true;
    else
        contieneZero = false;
}
```

Verifica universale

- Un problema di **verifica universale** consiste nel verificare se tutti gli elementi di una sequenza soddisfano una certa proprietà
  - una variante (duale) dei problemi di verifica esistenziale

Verifica se una sequenza di dieci elementi è crescente


```
... legge una sequenza di dieci numeri interi e
    verifica se è crescente ...
int numero;         // elemento corrente della sequenza
int precedente;     // elemento che precede numero
                    // nella sequenza
int i;              // per contare i numeri letti
boolean crescente;    // la sequenza è crescente

/* il primo elemento della sequenza è il
 * precedente del prossimo che sarà letto */
precedente = scanner.nextInt();
/* finora la sequenza letta è crescente */
crescente = true;
/* legge e elabora gli altri elementi della sequenza */
for (i=1; i<10; i++) {
    /* legge il prossimo numero e verifica che
     * la sequenza sia ancora crescente */
    numero = scanner.nextInt();
    if (precedente>=numero)
        crescente = false;
    /* prepara la prossima iterazione */
    precedente = numero;
}
... il risultato della verifica è crescente ...
```

Verifica l'uguaglianza tra stringhe

- Scrivere un metodo **boolean uguali(String s, String t)** che verifica se le stringhe **s** e **t** sono uguali
  - **uguali("alfa", "alfa")** deve restituire **true**, mentre **uguali("alfa", "beta")** deve restituire **false**
  - due stringhe **s** e **t** sono uguali se
    - **s** e **t** hanno la stessa lunghezza
    - ciascun carattere della stringa **s** è uguale al carattere della stringa **t** che occupa la stessa posizione

Uguaglianza tra stringhe


```
/* Verifica se le stringhe s e t sono uguali. */
public static boolean uguali(String s, String t) {
    // pre: s!=null && t!=null
    boolean uguali;    // s e t sono uguali
    int i;             // indice per la scansione di s e t

    /* verifica se s e t sono uguali */
    if (s.length()==t.length()) {
        /* s e t hanno la stessa lunghezza: s e t
        /* possono essere uguali, ma sono diverse
         * se contengono almeno un carattere diverso */
        uguali = true;
        for (i=0; i<s.length(); i++)
            if (s.charAt(i) != t.charAt(i))
                uguali = false;
    } else
        /* s e t hanno lunghezza diversa,
         * e quindi sono diverse */
        uguali = false;
    return uguali;
}
```

Uguaglianza tra stringhe

- Una variante

```
/* verifica se s e t sono uguali */
if (s.length()==t.length()) {
    /* s e t hanno la stessa lunghezza: s e t
    /* possono essere uguali, ma sono diverse
     * se contengono almeno un carattere diverso */
    uguali = true;
    for (i=0; uguali && i<s.length(); i++)
        if (s.charAt(i) != t.charAt(i))
            uguali = false;
} else
    /* s e t hanno lunghezza diversa,
     * e quindi sono diverse */
    uguali = false;
```

Verifica universale


```
int tuttiProprietà;    // tutti gli elementi
                       // soddisfano la proprietà

tuttiProprietà = true;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi al prossimo elemento ...
    if (l'elemento corrente non soddisfa la proprietà)
        tuttiProprietà = false;
}

... altre elaborazioni ...
```

Ricerca

- I problemi di **ricerca** sono un caso più generale dei problemi di verifica esistenziale
  - bisogna verificare se una sequenza contiene almeno un elemento che soddisfa una certa proprietà
  - in caso affermativo, bisogna calcolare delle ulteriori informazioni circa uno degli elementi che soddisfa la proprietà

Indice di un carattere in una stringa

- Scrivere un metodo **int posizioneDi(String s, char car)** che
  - verifica se la stringa **s** contiene almeno un carattere uguale a **car**
  - restituisce la posizione della prima occorrenza del carattere **car** nella stringa **s** oppure il valore **-1**
    - **posizioneDi("alfa", 'f')** deve restituire **2**, **posizioneDi("alfa", 'a')** deve restituire **1**, mentre **posizioneDi("alfa", 'b')** deve restituire **-1**

Indice di un carattere in una stringa


```
/* Verifica se la stringa s contiene il carattere car,
 * restituisce la posizione della prima
 * occorrenza di car in s, oppure -1. */
public static int posizioneDi(String s, char car) {
    // pre: s!=null
    int posizione;    // posizione di car in s
    int i;          // indice per la scansione di s

    /* scandisce i caratteri di s alla ricerca della
     * prima occorrenza di car in s */
    /* finora non è stata trovato nessun car in s */
    posizione = -1;
    for (i=0; posizione==-1 && i<s.length(); i++) {
        if (s.charAt(i)==car)
            /* è stato trovato car in posizione i */
            posizione = i;
    }
    return posizione;
}
```

Ricerca


```
... info;    // informazione associata
             // all'elemento cercato

info = informazione non trovata ;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi al prossimo elemento ...
    if (l'elemento corrente soddisfa la proprietà)
        info = informazione associata
               all'elemento corrente ;
}

... altre elaborazioni ...
```

Indice di una sottostringa in una stringa

- Scrivere un metodo **int posizioneDi(String s, String t)** che
  - verifica se la stringa **s** contiene almeno una sottostringa uguale alla stringa **t**
  - restituisce la posizione in cui inizia la prima sottostringa di **s** uguale a **t** oppure **-1**
    - **posizioneDi("sottostringa", "otto")** deve restituire **1**, **posizioneDi("mamma", "ma")** deve restituire **0**, mentre **posizioneDi("mamma", "pa")** deve restituire **-1**

Indice di una sottostringa in una stringa


```
/* Verifica se la stringa s contiene la sottostringa t
 * e restituisce la posizione della prima
 * occorrenza di t in s, oppure -1. */
public static int posizioneDi(String s, String t) {
    // pre: s!=null && t!=null
    int posizione;    // posizione della sottostringa t in s
    int ls, lt;       // lunghezza di s e lunghezza di t
    int i;            // indice per la scansione
                      // delle sottostringhe di s
    int j;            // indice per la scansione di t
    boolean ssug;     // la sottostringa di s che inizia in
                      // i ed ha lunghezza lt è uguale a t

    ... segue ...
```

Indice di una sottostringa in una stringa


```
/* inizializzazioni */
ls = s.length();
lt = t.length();
/* scandisce le sottostringhe di s
 * alla ricerca di una sottostringa uguale a t */
posizione = -1;
for (i=0; posizione==-1 && i<=ls-lt; i++) {
    /* scandisce i caratteri di t per verificare se la
     * sottostringa di s iniziante in posizione i e lunga lt
     * è uguale a t */
    ssug = true;
    for (j=0; ssug && j<lt; j++)
        if (s.charAt(i+j) != t.charAt(j))
            ssug = false;
    if (ssug)    // sottostringa di s uguale a t trovata
        posizione = i;   // allora t è sottostringa di s
}
return posizione;
}
```

Un’altra soluzione


```
public static int posizioneDi(String s, String t) {
    // pre: s!=null && t!=null
    int posizione;    // posizione della sottostringa t in s
    int ls, lt;       // lunghezza di s e lunghezza di t
    int i;            // indice per la scansione
                      // delle sottostringhe di s

    /* inizializzazioni */
    ls = s.length();
    lt = t.length();

    /* scandisce le sottostringhe di s alla ricerca di
     * una sottostringa uguale a t */
    posizione = -1;
    for (i=0; posizione==-1 && i<=ls-lt; i++) {
        if ( uguali(sottostringa(s,i,i+lt), t) )
            posizione = i;
    }
    return posizione;
}
```