Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa E03
Esercizi su ricorsione

F. Gasparetti

Marzo 2008

Contenuti

Esercizi su ricorsione

- Massimo di interi positivi letti da file
- Confronto tra lettura iterativa e lettura ricorsiva
- Gli ultimi saranno i primi
- Conteggio di elementi usando la ricorsione
- Conteggio condizionato di elementi
- Calcolo di valori usando la ricorsione
- Sequenza simmetrica di interi

Massimo di interi positivi letti da file . . .

- **Formalizzazione ricorsiva dell'operazione di ricerca valore massimo all'interno di un file che contiene interi positivi:**
  - **se il file è terminato, restituisci *0* – passo base**
  - **altrimenti – passo induttivo**
    - **leggi un intero *i* dal file**
    - **trova il massimo *m* tra i valori rimanenti nel file**
    - **restituisci il maggiore tra *i* ed *m***
- **Supponiamo che ad ogni linea del file di input corrisponde un valore salvato in formato testo**

La classe Lettore del package fiji

- Per elaborare dei dati in ingresso solitamente li inseriamo direttamente nel codice, oppure utilizziamo la libreria fiji.io per semplificare la lettura
- La classe Lettore permette di leggere sequenze di caratteri da diverse sorgenti:
  - **Lettore():** Crea un Lettore corrispondente al flusso standard di ingresso (cioè System.in).
  - **Lettore(String s):** Crea un Lettore per leggere da una stringa s
  - **Lettore(java.io.Reader r):** Crea un Lettore per uno stream d'ingresso preesistente.

Eccezioni . . .

- Durante l'esecuzione possono nascere molte situazioni di errore con i file, ad esempio:
  - File inesistente, Spazio insufficiente, Indici array fuori dai range, . . .
- Per segnalare tali eventualità, i metodi possono generare (throws) particolari eccezioni, ovvero istanziare una classe Exception (o una sua estensione) che rappresenta una condizione anomala

. . .Eccezioni . . .

- Documentazione della classe java.io.FileReader

```
public FileReader(String fileName)
          throws FileNotFoundException

Creates a new FileReader, given the name of the file to
read from.

Parameters:
    fileName - the name of the file to read from
Throws:
    FileNotFoundException - if the named file does not
exist, is a directory rather than a regular file, or
for some other reason cannot be opened for reading.
```

. . . Eccezioni

- Che succede se eseguo il seguente codice?

```
class Prova {
   . . .
   public static void main(String[] args){
    int[] array = new int[10];
    for (int i = 0; i < 20; i++)
      array[i] = i;
   }
}
```

. . . Eccezioni . . .

- E' opportuno trattare ogni eccezione che può scaturire nel ramo try agganciandola nel ramo catch

```
try {
  /* apertura di uno stream in lettura da un
     file di nome readme.txt */
   FileReader reader = new FileReader("readme.txt");
  ...
} catch (Exception ex) {
  /* stampo su schermo la rappresentazione testuale
  dell'eccezione */
   System.out.println(ex.toString());
}
```

- Se nessuno tratta l'eccezione, si arriva al main e l'esecuzione termina

Massimo di interi positivi letti da file . . .

- Implementazione ricorsiva:

```
import java.io.*;
import fiji.io.*;

public class MassimoRicorsivo {

  public MassimoRicorsivo() { }

  public static int massimo(Lettore fin) {
    int ret = 0;
    if (!fin.eof()) {
      /* legge una stringa (parola senza spazi) */
      String s = fin.leggiString();
      int i = Integer.parseInt(s);
       . . .
```

. . .Massimo di interi positivi letti da file. . .

```
      int m = massimo(fin);
      if (m > i) ret = m;
      else       ret = i;
    }
    return ret;
}

public static void main(String[] args) {
    try {
      final String NOME_FILE = "lettura.txt";
      /* creiamo un oggetto FileReader per leggere
         dal file lettura.txt */
      FileReader fr = new FileReader(NOME_FILE);
      . . .
```

…Massimo di interi positivi letti da file…

```
        . . .
        /* istanziamo Lettore con il Reader fr */
        Lettore fin = new Lettore(fr);
        System.out.println(massimo(fin));
         /* chiusura streams aperti */
        fin.close();
        fr.close();
      } catch(Exception ex) {
        System.out.println("eccezione:"+
                                        ex.toString());
      }
    }
}
```

…Massimo di interi positivi letti da file

- Invece di leggere una linea e convertirla in intero con parseInt della classe Integer:

```
String s = fin.leggiLinea();
...
int i = Integer.parseInt(s);
```

si poteva utilizzare la leggiInt di Lettore:

```
int i = fin.leggiInt();
```

ma il metodo non indica chiaramente come si comporta in caso di assenza di caratteri numerici da leggere oppure quando si arriva al *end-of-file*.

Confronto tra lettura iterativa e lettura ricorsiva

- Ciclo di lettura non ricorsiva:

```
leggi primo elemento ;
while (elemento valido) {
  elabora elemento letto ;
  leggi elemento successivo ;
}
```

- Lettura ricorsiva:

```
leggi un elemento ;
if (elemento valido) {
  elabora elemento letto ;
  chiama ricorsivamente lettura ;
}
```

Confronto tra lettura iterativa e lettura ricorsiva: stampa di un file . . .

- Esempio: stampa su console (schermo) di un file letto in input attraverso un oggetto Lettore
- Implementazione con un metodo ricorsivo:

```
public static void stampa(Lettore in) {
  if (!in.eof()) {
    String s = in.leggiLinea();
    System.out.println(s);
    stampa(in);
  }
  /* else -> non fare nulla */
}
```

Confronto tra lettura iterativa e lettura ricorsiva: stampa di un file . . .

- Il seguente frammento di codice manda in output su schermo il contenuto del file readme.txt:

```
public class Stampa {
  public static void stampa(...) { ... };

  public static void main(String[] args) {
    try {
      final String FILENAME = “readme.txt";
      FileReader fr = FileReader(FILENAME);
      stampa(new Lettore(fr));
      fr.close();
    } catch(Exception ex) { ... }
  }
}
```

. . .Confronto tra lettura iterativa e lettura ricorsiva: copia di un file

- Implementazione del metodo stampa in versione iterativa:

```
public static void stampaIterativa(Lettore in) {
  while (!in.eof()) {
    System.out.println(in.leggiLinea());
  }
}
```

Esempio: gli ultimi saranno i primi . . .

- Leggiamo le linee di un file e le stampiamo su schermo, invertendo l'ordine delle linee del file
- La lettura avviene tramite Lettore, mentre l'output mediante stream System.out:

```
public static void stampaInversa(Lettore in) {
  if (!in.eof()) {
    String s = in.leggiLinea();
    stampaInversa(in);
    System.out.println(s);
  }
}
```

Esempio: gli ultimi saranno i primi . . .

- E' chiaro che le linee del file vengono memorizzate in delle stringhe a cui si accede tramite occorrenze successive della variabile s nei RDA delle successive chiamate ricorsive di stampaInversa
- Quindi, la pila dei RDA viene usata come una "struttura dati" temporanea nella quale memorizzare le linee del file prima di poterle stampare
- Una implementazione iterativa di stampaInversa richiede la lettura e la memorizzazione in una struttura dati addizionale di tutte le linee dal file prima di poter iniziare a stampare, ad esempio un array di stringhe

. . . Esempio: gli ultimi saranno i primi

- In generale è possibile convertire facilmente un metodo ricorsivo quando l'ultima istruzione corrisponde alla chiamata ricorsiva - tail recursion
- Il codice ricorsivo è generalmente meno efficiente poiché ogni chiamata genera molte istruzioni macchina di supporto.
- Per questo alcuni compilatori riconoscono la tail recursion e convertono la ricorsione in codice iterativo

Conteggio di elementi usando la ricorsione . . .

- Si vogliono contare gli elementi di una sequenza in modo ricorsivo

```
leggi un elemento;
if (elemento valido)
  return 1 + risultato-chiamata-ricorsiva;
return 0;
```

. . .Conteggio di elementi usando la ricorsione

- Esempio: si vogliono conteggiare le linee presenti in un file:

```
public static int conta(Lettore in) {
   int ret = 0;
   if (!in.eof()) {
      String s = in.leggiLinea();
      ret = 1 + conta(in);
   }
   return ret;
}
```

Conteggio condizionato di elementi . . .

- Si vogliono contare gli elementi di una sequenza in modo ricorsivo

```
leggi un elemento;
if (elemento valido){
  if (condizione)
    return 1 + risultato-chiamata-ricorsiva;
  else
    return risultato-chiamata-ricorsiva;
}
return 0;
```

. . .Conteggio condizionato di elementi

- **Esempio: si vogliono conteggiare le linee presenti in un file che iniziano con un dato carattere ch:**

```
public static int contaLinee(Lettore in, char ch){
  int ret = 0;
  if (!in.eof()) {
    String s = in.leggiLinea();
    if (s.charAt(0) == ch)
      ret = 1 + contaLinee(in, ch);
    else
      ret = contaLinee(in, ch);
  }
  return ret;
}
```

Calcolo di valori usando la ricorsione . . .

- Supponiamo di voler effettuare un'operazione *op* (ad esempio, la somma) tra tutti gli elementi di una struttura definita induttivamente

```
leggi un elemento;
if (elemento valido)
  return val-elemento op ris-chiamata-ricorsiva;
return elemento-neutro-di-op;
```

dove *elemento-neutro-di-op* è l'elemento neutro rispetto all'operazione da effettuare (0 per la somma, 1 per il prodotto, "" per la concatenazione tra stringhe)

Calcolo di valori usando la ricorsione . . .

- Esempio: si vuole calcolare la somma degli interi contenuti su un file (dove ogni linea contiene un valore in formato testo)

```
public static int somma(Lettore in) {
   int somma = 0;
   if (!in.eof()) {
      String s = in.leggiString();
      somma = Integer.parseInt(s) + somma(in);
   }
   return somma;
}
```

Calcolo di valori usando la ricorsione

- Esempio: si vuole controllare la presenza di un dato valore val in un file di interi

```
public static boolean presente(int val,
                               Lettore in) {
  boolean ret = false;
  if (!in.eof()) {
    String s = in.leggiString();
    ret = (Integer.parseInt(s) == val) ||
                          presente(val, in);
  }
  return ret;
}
```

Sequenza simmetrica di interi . . .

- Riprendiamo il problema di riconoscere una sequenza simmetrica
- Una sequenza di numeri interi tutti positivi tranne che per uno 0 in posizione centrale si dice simmetrica se la sequenza stessa coincide con la sequenza in ordine inverso
- Caratterizzazione ricorsiva:
  - la sequenza costituita solo da 0 è simmetrica
  - una sequenza *n s m* è simmetrica, se *s* è simmetrica e *n* ed *m* sono due numeri interi positivi uguali
  - nient'altro è una sequenza simmetrica

. . . Sequenza simmetrica di interi . . .

- Supponiamo di avere in input un file di testo contenente una sequenza di numeri interi in formato testo, uno per riga, tutti positivi tranne uno 0 in posizione centrale, e vogliamo verificare se è simmetrica
- Ci sono due alternative:
  - memorizzare tutta la sequenza in un array, e fare la verifica accedendo direttamente agli elementi dell'array, usando un ciclo
  - oppure si può sfruttare la ricorsione per la verifica, senza usare una struttura di dati addizionale ma implicitamente l'RDA
- Qui vedremo la seconda alternativa

. . . Sequenza simmetrica di interi . . .

- **Problema: Si vuole determinare se una data sequenza di numeri è simmetrica o no.**
- **Dati di ingresso:**
  - **Un sequenza di numeri letti da un file**
- **Pre-condizioni:**
  - **Sequenza di almeno una cifra**
- **Dati di uscita:**
  - **True se la sequenza è simmetrica**

. . . Sequenza simmetrica di interi . . .

- **Pseudo-codice dell'algoritmo:**

```
boolean simmetrica(sequenza)
 sim ← FALSE
 n ← estrai un elemento dalla sequenza
 IF (n == 0) OR sequenza finita THEN
   sim ← TRUE
 ELSE
   b ← simmetrica(sequenza senza elemento n)
   m ← estrai un elemento dalla sequenza
   sim ← (n == m) AND b
 END IF
 return sim;
```

. . . Sequenza simmetrica di interi . . .

- Realizzazione utilizzando la libreria fiji:

```
public static boolean simmetrica(Lettore in){
  boolean ret = false;
  if (!in.eof()) {
    /* lettura da sinistra */
    int n = Integer.parseInt
                        (in.leggiString());
    /* al carattere 0 mi fermo */
    if (n == 0)
      ret = true;
    else {
      boolean pal = simmetrica(in);
    . . .
```

. . . Sequenza simmetrica di interi . . .

```
        . . .
        if (!in.eof()) {
          /* lettura da destra */
          int m = Integer.parseInt
                              (in.leggiString());
          ret = (n == m) && pal;
        } else { /* in.eof() = true */
          ret = false;
        }
      }
    }
    return ret;
  }
```

. . . Sequenza simmetrica di interi . . .

```
. . .
public static void testSimmetrica1() {
  try {
    final String FILENAME = "lettura.txt";
    FileReader fr = new FileReader(FILENAME);
    Lettore fin = new Lettore(fr);
    System.out.println(simmetrica(fin));
  } catch (Exception ex) { . . . }
}

public static void testSimmetrica2() {
  Lettore in = new Lettore();
  System.out.println(simmetrica(in));
}
```

. . . Sequenza simmetrica di interi

- Output del programma:

```
Con lettura.txt = “1\n2\n3\n0\n3\n2\n1\n”
true
Con lettura.txt = “1\n1\n3\n0\n3\n7\n1\n”
false

C:>
```

Esercizi

- Scrivere una applicazione che legge un file di testo e calcola il numero di linee e il numero di caratteri presenti nel file di testo
- Scrivere una applicazione che legge un file di testo e determina la più lunga tra le sue linee e la visualizza sullo schermo
- Scrivere una applicazione che copia il contenuto di un file di testo in un altro file di testo