Scheme ha una gestione particolare dei file. Per prima cosa, i flussi di file, che negli altri linguaggi sono dei *file handle* o semplicemente *stream*, in Scheme prendono il nome di *port*: ***porte***. Scheme distingue quindi tra porte in ingresso, in grado di «consegnare» dei caratteri, e porte in uscita, in grado di «accettare» caratteri.

Apertura e chiusura

Scheme distingue tra flussi di file in ingresso e in uscita, per cui le funzioni per aprire i file e trasformarli in porte, sono due, uno per l'apertura in lettura (ingresso) e l'altra per l'apertura in scrittura (uscita). La tabella u130.1 riassume le funzioni utili per aprire, controllare e chiudere i file. Gli esempi successivi, dovrebbero aiutare a comprenderne l'utilizzo.

Tabella u130.1. Elenco di alcune funzioni per l'apertura e la chiusura dei file, oltre che per il controllo dei flussi di file predefiniti.

Funzione	Descrizione
`(open-input-file` ***str_nome_file*** `)`	Apre il file nominato e restituisce la porta in ingresso.
`(open-output-file` ***str_nome_file*** `)`	Apre il file nominato e restituisce la porta in uscita.
`(port?` ***oggetto*** `)`	*Vero* se si tratta di una porta.
`(input-port?` ***oggetto*** `)`	*Vero* se si tratta di una porta in ingresso.
`(output-port?` ***oggetto*** `)`	*Vero* se si tratta di una porta in uscita.
`(close-input-port` ***porta*** `)`	Chiude la porta in ingresso.
`(close-output-port` ***porta*** `)`	Chiude la porta in uscita.

```
(define porta-i (open-input-file "mio_file"))

(port? porta-i)                         ===> #t
(output-port? porta-i)                  ===> #f
(input-port? porta-i)                   ===> #t

(close-input-port porta-i)
```

In condizioni normali, sono sempre disponibili una porta in ingresso e una in uscita, in modo predefinito. Si tratta generalmente di standard input e standard output. Questi flussi di file predefiniti potrebbero essere diretti verso altri file. Tuttavia questo non viene mostrato; eventualmente si può approfondire il problema leggendo *R*^{5}*RS*.

Ingresso dei dati

L'ingresso dei dati, ovvero la lettura, avviene attraverso due funzioni fondamentali: '**`read-char`**' e '**`read`**'. La prima legge un carattere alla volta, la seconda interpreta ciò che legge in forma di dati Scheme. In pratica, '**`read`**' legge ogni volta ciò che riesce a interpretare come un oggetto per Scheme.

Tabella u130.3. Elenco di alcune funzioni per la gestione dei dati in ingresso.

Funzione	Descrizione
`(read-char)`	Legge e restituisce il carattere successivo dalla porta predefinita.
`(read-char` ***porta*** `)`	Legge e restituisce il carattere successivo dalla porta indicata.
`(peek-char)`	Restituisce una copia del carattere successivo dalla porta predefinita.

Funzione	Descrizione
`(peek-char` ***porta*** `)`	Restituisce una copia del carattere successivo dalla porta indicata.
`(read)`	Legge un oggetto dalla porta predefinita.
`(read` ***porta*** `)`	Legge un oggetto dalla porta indicata.
`(eof-object` ***porta*** `)`	*Vero* la lettura dalla porta ha raggiunto la fine.

L'esempio seguente mostra in che modo potrebbe essere utilizzata la funzione '**`read-char`**'. Si inizia aprendo il file '`/etc/passwd`', dal quale vengono letti i primi caratteri. Si suppone che il primo record a essere letto sia quello di definizione dell'utente '**`root`**':

```
; apre il file e gli associa la porta «utenti»
(define utenti (open-input-file "/etc/passwd"))

; legge un carattere alla volta
(read-char utenti)                          ===> #\r
(read-char utenti)                          ===> #\o
(read-char utenti)                          ===> #\o
(read-char utenti)                          ===> #\t
(read-char utenti)                          ===> #\:
;…

; chiude il file
(close-input-file utenti)
```

Nell'esempio seguente si vuole mostrare l'uso della funzione '**`read`**'. Prima si suppone di avere preparato il file seguente:

```
; prova_lettura.scm

; somma
(+ 1 2)

; moltiplicazione
(* 2 5)

; stringa
"ciao"

; valore numerico
123

; fine
```

Supponendo che il file si chiami '`prova_lettura.scm`', si osservi la sequenza di istruzioni Scheme seguente, assieme a ciò che si ottiene dalla lettura del file:

```
; apre il file e gli associa la porta «prova»
(define prova (open-input-file "prova_lettura.scm"))

; legge il primo oggetto
(read utenti)                               ===> (+ 1 2)

; legge il secondo oggetto
(read utenti)                               ===> (* 2 5)

; legge il terzo oggetto
(read utenti)                               ===> "ciao"

; legge il quarto oggetto
(read utenti)                               ===> 123

; chiude il file
(close-input-file prova)
```

Si intende l'importanza della funzione '**`read`**' per facilitare l'inserimento di dati nei programmi in modo interattivo.

Uscita dei dati

«

L'emissione dei dati, ovvero la scrittura, avviene in maniera simile alla lettura, con la stessa distinzione tra le funzioni '**`write-char`**' e '**`write`**'. Anche in questo caso, la prima scrive un carattere alla volta, mentre la seconda emette la rappresentazione di un oggetto alla volta. Tuttavia, si aggiunte un'altra funzione fondamentale: '**`output`**'. Questa funzione viene usata preferibilmente per mostrare dei messaggi senza codici di escape, soprattutto per non lasciare le virgolette di delimitazione delle stringhe.

Tabella u130.7. Elenco di alcune funzioni per la gestione dei dati in ingresso.

Funzione	Descrizione
`(write-char` ***carattere*** `)`	Scrive il carattere indicato attraverso la porta predefinita.
`(write-char` ***carattere porta*** `)`	Scrive il carattere indicato attraverso la porta indicata.
`(write` ***oggetto*** `)`	Scrive la rappresentazione dell'oggetto attraverso la porta predefinita.
`(write` ***oggetto porta*** `)`	Scrive la rappresentazione dell'oggetto attraverso la porta indicata.
`(display` ***oggetto*** `)`	Mostra l'oggetto attraverso la porta predefinita.
`(display` ***oggetto porta*** `)`	Mostra l'oggetto attraverso la porta indicata.
`(newline)`	Emette un codice di interruzione di riga attraverso la porta predefinita.
`(newline` ***porta*** `)`	Emette un codice di interruzione di riga attraverso la porta indicata.

L'esempio seguente dovrebbe chiarire la differenza tra la funzione '**`write`**' e '**`display`**'. Gli oggetti vengono emessi attraverso lo standard output, ovvero la porta predefinita:

```
(write (+ 1 2))                         ; visualizza «3»
(write "ciao")                          ; visualizza «"ciao"»
(write "ciao, come \"stai\"?")          ; visualizza «"ciao, come \"stai\""»
(write #\A)                             ; visualizza «#\A»
(display (+ 1 2))                       ; visualizza «3»
(display "ciao")                        ; visualizza «ciao»
(display "ciao, come \"stai\"?")        ; visualizza «ciao, come "stai"»
(display #\A)                           ; visualizza «A»
```

È già stato descritto l'uso di '**`newline`**', che è indispensabile per ottenere l'avanzamento alla riga successiva. In linea di principio, non è possibile inserire un carattere di controllo nella stringa emessa da '**`write`**' o da '**`display`**'.