.

Gestione dei file

Stefano Ferrari

Università degli Studi di Milano
stefano.ferrari@unimi.it

Programmazione

anno accademico 2017–2018

Gli *stream*

Si dice *stream* qualsiasi sorgente di dati in ingresso e qualsiasi destinazione per i risultati in uscita

- tastiera
- video
- file su disco, CD, DVD, memorie flash
- dispositivi di comunicazione (porte di rete, stampanti, ecc. . .)

La libreria `stdio.h` tratta tutti gli *stream* allo stesso modo (per quanto possibile)

- rappresentandoli con puntatori a file (`FILE *`)
- su cui operano funzioni simili o identiche

.

Gli *stream* standard

Esistono tre *stream* standard, che non occorre definire, aprire e chiudere

- lo standard input (`stdin`), ovvero la tastiera
- lo standard output (`stdout`), ovvero il video
- lo standard error (`stderr`), ovvero il video

Quando si chiama un programma, il sistema operativo può reindirizzare gli *stream* standard, cioè modificarne il significato

- *programma* < *nomefile* indica che si ricevono i dati dal file *nomefile* anziché da tastiera (`stdin` punta il file *nomefile*)
- *programma* > *nomefile* indica che si stampano i risultati sul file *nomefile* anziché a video (`stdout` punta il file *nomefile*)
- *programma* 2> *nomefile* indica che si stampano i messaggi di errore sul file *nomefile* anziché a video (`stderr` punta il file *nomefile*)

File di testo e binari

Vi sono due tipi di file

- file di testo, costituiti da sequenze di caratteri: sono accessibili all'utente con un editor di testo
- file binari, costituiti da sequenze di byte: occupano meno spazio per rappresentare numeri

I file di testo sono organizzati in righe, separate da appositi caratteri, che sono specifici di ogni sistema operativo

- il C gestisce la differenza automaticamente (è sempre `'\n'`)

Nel seguito considereremo solo i file di testo

.

Apertura di un file (1)

Per usare un file occorre aprirlo specificando

```
FILE *fopen(char *nomefile, char *modo)
```

- il nome del file da aprire e la posizione su disco (*path*)
- il modo in cui usarlo
  - "r": in lettura, ponendosi al principio del file
  - "w": in scrittura, ponendosi al principio del file
  - "a": in accodamento, ponendosi alla fine del file

  Per i file binari si usano "rb", "wb" e "ab"

Il *path* può essere assoluto o relativo (al file eseguibile o al progetto)

Apertura di un file (2)

La funzione `fopen` restituisce un puntatore al file per poterlo usare

- se il file non esiste
  - in lettura, restituisce `NULL`
  - in scrittura e accodamento, ne crea uno vuoto
- se il file non può essere aperto o creato
  - restituisce `NULL`

Aperto un file, la posizione accessibile è

- il principio del file se si è aperto il file in lettura o scrittura
- la fine del file se si è aperto il file in accodamento

.

Chiusura di un file

Dopo l'uso, il file va chiuso con l'istruzione

```
int fclose(FILE *stream)
```

che restituisce

- 0 se la chiusura ha successo
- la costante simbolica `EOF` altrimenti

Si possono usare puntatori diversi per lo stesso file al fine di scorrerlo in maniera differenziata (non ha senso farlo in scrittura)

La funzione

```
void rewind(FILE* stream )
```

riporta la posizione corrente al principio del file

Parsing di *stream*

Tutti gli stream di ingresso sono gestiti allo stesso modo

```
int fscanf(FILE *stream, char *formato, ...)
```

- interpreta il contenuto dello *stream*
- in base alla stringa di formato
- assegna gli oggetti riconosciuti ai puntatori che seguono
- restituisce il numero di oggetti assegnati

Esempio:

```
FILE *fp;
int giorno, mese, anno;
fp = fopen("prova.txt","r");
fscanf(fp,"%d/%d/%d", &giorno, &mese, &anno);
```

Terminazione di un file

Se si arriva al termine di un file

- la funzione `fscanf` restituisce il numero di oggetti assegnati
- se non ne ha assegnati, restituisce la costante simbolica `EOF`

```
int fscanf(FILE *stream, char *formato, ...)
```

N.B.: `fscanf` restituisce `EOF` se si trova esattamente al termine, non se il file termina durante il *parsing*

Dopo il fallimento di un'operazione di lettura

- la funzione `feof` restituisce vero, cioè un valore intero non nullo

```
int feof(FILE *stream)
```

Lettura di righe

```
char *fgets(char *s, int n, FILE *stream)
```

- legge una riga di testo dallo `stream`
  cioè tutto il testo fino al primo `'\n'` incluso
- si ferma dopo al massimo `n` caratteri (meno se compare `'\n'`)
- assegna quanto letto alla stringa `s`
- se fallisce restituisce `NULL`, altrimenti la stringa `s`

```
char *gets(char *s)
```

- opera sullo *stream* `stdin`
- non specifica la lunghezza `n`
- legge `'\n'`, ma non lo include in `s`

.

Scrittura su file

```
int fprintf(FILE *stream, char *formato, ...)
```

funziona esattamente come `printf` e `sprintf`:

- scrive sullo *stream*
- nel *formato* specificato dalla relativa stringa
- il valore degli oggetti che seguono

```
int *fputs(char *s, FILE *stream)
```

scrive la stringa `s` sullo *stream* di uscita senza aggiungere `'\n'`

```
int *puts(char *s)
```

scrive la stringa `s` sullo `stdout` con un `'\n'` aggiuntivo

Restituiscono `EOF` se falliscono, un valore non negativo altrimenti

Buffering

L'accesso ai file non viene eseguito ad ogni operazione (è troppo lento)

- si leggono i dati da file un blocco per volta e si conserva il blocco in un'area di memoria dedicata (*buffer* di lettura)
- si accodano i risultati in un'area di memoria dedicata (*buffer* di scrittura) e si salvano su file quando essa è piena

Il meccanismo di *buffering* è automatico

La funzione

```
int fflush (FILE* stream)
```

svuota completamente il *buffer* associato al file *stream*

Se *stream* vale `NULL`, svuota i *buffer* associati a tutti i file