Fondamenti di Informatica

Allievi Automatici

A.A. 2015-16

Gestione dei File

Perché i file?

- Sono strutture dati **persistenti** (dischi, etc ...)
  - Si leggono e si scrivono con opportune istruzioni
- Grazie ai file, i dati possono sopravvivere al termine dell'esecuzione del programma
- N.B.: si usano anche per memorizzare i programmi !!
  - Quando se ne chiede l'esecuzione, il sistema operativo copia il programma (eseguibile, conservato in un file) in memoria centrale, e inizia a eseguirlo

File binari, file di testo

- I file sono strutture dati **sequenziali**
  - Sequenziale significa: si leggono (e scrivono) gli elementi del file in sequenza
- Un file ***binario*** è una sequenza di byte che non è "interpretata" in alcun modo
- Un file ***di testo*** è una sequenza di caratteri "interpretata":
  - Alcuni caratteri rappresentano separatori
    - Esempio: il carattere di "newline" è interpretato dalla stampante come "salto alla riga successiva"

File e sistema operativo

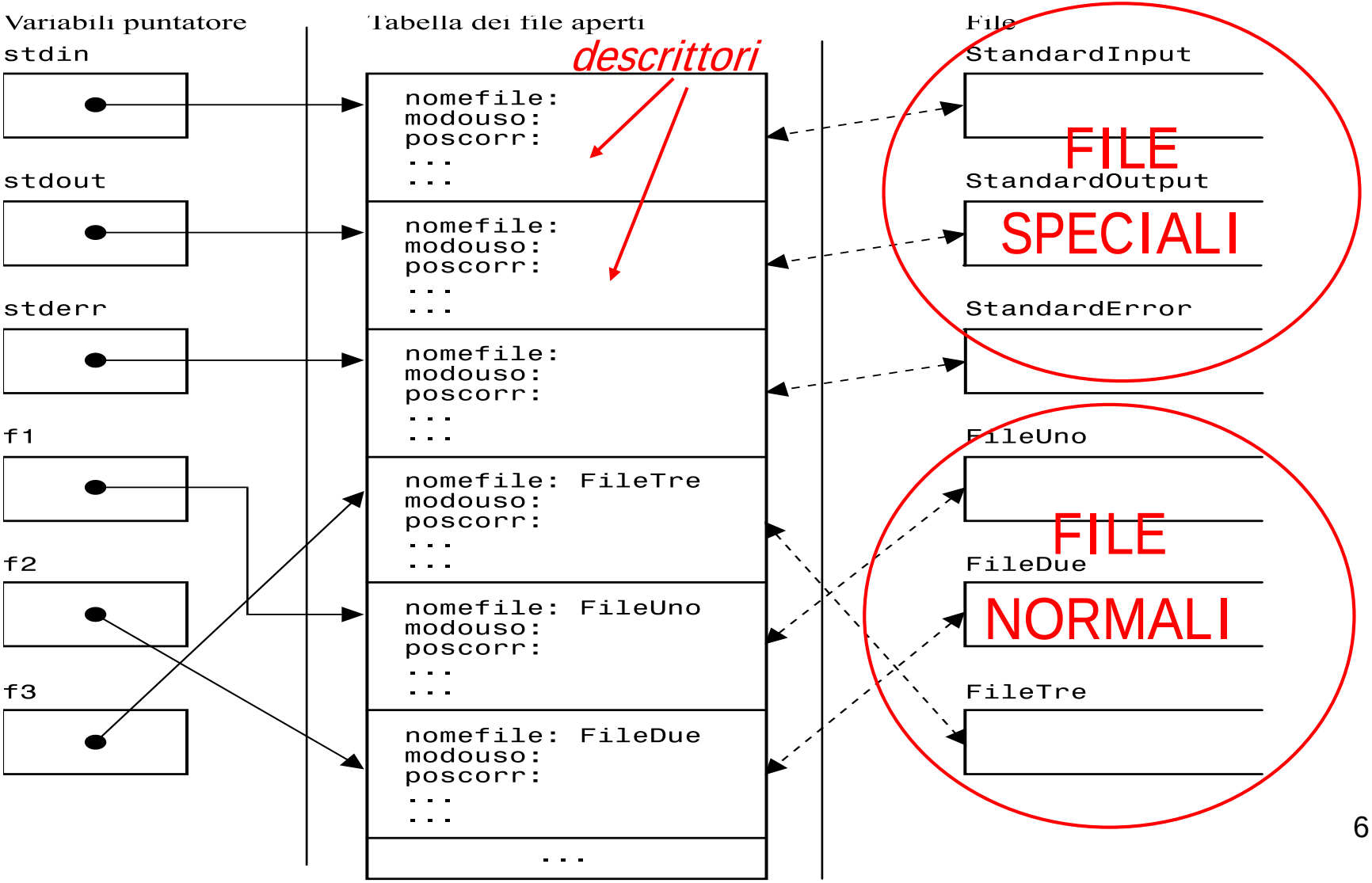
- I file sono gestiti dal S.O.
  - Sono resi visibili all'interno del linguaggio per essere manipolati attraverso opportune funzioni di libreria
- Per essere usato, un file deve essere prima ***aperto***, e dopo l'uso andrà ***chiuso***
  - *Aprire e chiudere il **"flusso di comunicazione"** tra il programma e il file*
- In C tutte le periferiche sono viste come file (chiamati "file speciali“, ad esse associati)
  - **stdin** e **stdout** (terminali, stampanti, ecc)
  - **In C possiamo "leggere" e "scrivere" da/su ogni periferica di I/O con le stesse modalità (quelle dei file)**

Possiamo "leggere" e "scrivere" con le stesse modalità (quelle dei file) in ogni operazione di I/O

QUINDI SAPPIAMO GIÀ USARLI !!

Rappresentazione interna dei file

- Ogni file aperto da un prog. ha un **descrittore**
  - Risiede nella *tabella dei file aperti*, una delle strutture dati che il S.O. associa ai programmi in esecuzione
- Il descrittore memorizza:
  - la **modalità** d'uso (read, write)
  - un indicatore della **posizione** corrente nel file (testina)
  - l'indicatore di eventuale **errore**
  - l'indicatore di **eof** (end-of-file)
- L'**apertura** del file restituisce un descrittore
  - Per la precisione, un **puntatore** a un descrittore


Variabili puntatore
Tabella dei file aperti
File
descrittori
stdin
stdout
stderr
f1
f2
f3
nomefile:
modouso:
poscorr:
...
...
nomefile: FileTre
nomefile: FileUno
nomefile: FileDue
StandardInput
StandardOutput
StandardError
FileUno
FileDue
FileTre
FILE
SPECIALI
FILE
NORMALI

Dichiarare e aprire un file

- Puntatore al descrittore**: FILE * fp**
  - FILE * **fopen**(char * *nomefile*, char * *modalità*)

    nomefile e *modalità* sono stringhe

    nomefile dà il percorso (path), o il file è cercato nella cartella da cui si è eseguito il programma

    apre il file (oppure ne **crea** uno, se non lo trova)
  - modalità di apertura

    "r" lettura modalità testo, posizionamento inizio file (**read**)

    "w" scrittura modalità testo, posizionamento inizio file (**write**)

    "a" scrittura in modalità testo, posizionamento fine file (**append**)

    "rb", **"wb"** e **"ab"** (idem, ma considerando il file come **binario**)
- Se si verifica un errore, fopen() restituisce **NULL**

Cancellare, ridenominare, chiudere

int **remove**(char * nomefile)

- cancella file nomefile
- restituisce 0 se buon fine, != 0 altrimenti

int rename(char *oldname, char *newname)

- cambia nome al file
- restituisce 0 se buon fine, !=0 altrimenti

int **fclose**(FILE * fp)

- fp diventa NULL, descrittore di tipo FILE rilasciato
- restituisce 0 se buon fine, altrimenti EOF

Gestione degli errori

int ferror(FILE * fp)

- restituisce 0 se non si verificano errori, altrimenti un codice specifico dell'errore

int **feof**(FILE * fp)

- restituisce 0 (falso) se NON si è alla fine

void clearerr(FILE * fp)

- riporta al valore normale gli indicatori di errore e eof

Lettura e scrittura

- Si opera sui file in quattro modi possibili
- Tre modi per i file di testo:
  - Precisando la **formattazione** dell' I/O ("à-la-scanf“)
  - Un **carattere** alla volta ("à-la-getc")
  - Per **linee** di testo ("à-la-gets“ - fino ad ogni prossimo '\n')
- Un modo per i file binari:
  - Per **blocchi** di byte
    - "à-la-sizeof" – indico solo il numero di byte da leggere

Lettura / scrittura formattata

- *scanf e printf fanno riferimento a stdin e stdout*
  - *Non serve specificare su quale file agiscono (è implicito)*
- **f**printf e **f**scanf fanno riferimento a file generici, ma si usano esattamente come scanf e printf

int fprintf(FILE * **fp**, *str_di_controllo*, *espressioni*)

int fscanf(FILE * **fp**, *str_di_controllo*, *indirizzi_variabili*)

- Restituiscono il numero di elementi effettivamente letti/scritti, o zero in caso di errore
  - fscanf() restituisce **-1** (costante **EOF**) se l'indicatore di posizione è già oltre la fine del file e non può leggere alcun valore valido

Lettura carattere per carattere

- *int getc(void)*
  - *legge un carattere **da standard input**, restituendolo come intero*
- *int putc(int c)*
  - *scrive un carattere **su standard output***
- int **fgetc**(FILE * fp)
- int **fputc**(int c, FILE * fp)
  - leggono/scrivono un carattere dal/sul **f**ile descritto da *fp, restituendolo come intero

Se fp è stdin/stdout è identico scrivere getc() e putc(c)

Leggere e mostrare a video un file

```
#include <stddef.h>
int main ()  {
    FILE * fp;
    char c;
    fp = fopen ("filechar", "r");  /* file lettura, modalità testo */
    if (fp != NULL) {
        c = fgetc (fp);
        while (c != EOF)  {      /* oppure while (! feof (fp)) */
                putc (c);
                c = fgetc (fp);   /* oppure c=(char)fgetc (fp); */
        }
        fclose (fp);
    } else printf ("Il file non può essere aperto.\n");
    return 0;
}
```

```
while ((c=fgetc(fp)) != EOF)
        putc(c);
```

Lettura / scrittura per linee di testo

- Su **stdin** e **stdout**:
  - char * **gets**(char * s)
    - s è l'array in cui copiare la stringa letta da stdin
      - s risulta terminata da un '\0', aggiunto in automatico
    - Non si può limitare la dimensione dei dati in input
      - Non controlla che la stringa s sia sufficientemente grande
    - In caso di errore, restituisce **NULL**
  - int **puts**(char * s)
    - scrive la stringa s, escluso il '\0'
      - al posto del '\0' che si trova nella stringa scrive un '\n'
    - Restituisce n>=0 se OK, EOF in caso di errore

Lettura / scrittura per linee di testo

- Su **file qualunque** (fp):
  - char * **fgets**(char * s, **int n**, FILE * fp)
    - legge al più n-1 caratteri, fino a '\n' o EOF
    - se incontra '\n' lo inserisce tra gli n-1, e mette alla fine **anche** il terminatore '\0'
    - In caso di errore, restituisce **NULL**
  - int **fputs**(char * s, FILE * fp)
    - come puts
      - Ma **non** aggiunge il '\n', si limita a non scrivere il '\0'
    - Restituisce 0 se OK, EOF in caso di errore

```
int copiaselettiva (char * refstr, char * filein, char * fileout) {
    char line [MAXLINE];
    FILE * fin, * fout;
    fin = fopen( filein, "r" );
    if ( fin == NULL )  return ERROR;
    fout = fopen( fileout, "w" );       /* aperto in scrittura, modalità testo */
    if ( fout == NULL ) {
          fclose( fin );  return ERROR;
    }
    while( fgets(line, MAXLINE, fin) != NULL )     /*  fgets legge da filein al più */
          if ( strstr(line, refstr) != NULL)        /*  MAXLINE–1 caratteri       */
                fputs (line, fout);
     /* strstr rest. la posiz. della prima occorrenza di refstr in line; se non c'è, NULL */
    fclose( fin ); fclose( fout );
    return OK;
}
```

```
#define OK 1
#define ERROR 0
# define MAXLINE 100
```

Esercizietto

```
#define MAXLINE 100
int main() {
        char *temp, line[MAXLINE], match[MAXLINE]; FILE * cfPtr; int countMatch = 0;
        if( (cfPtr = fopen("D:\\prova1.txt", "r")) != NULL ) {
                printf("stringa da cercare--> ");
                if (gets(match) != NULL) {
                        printf("match = %s\n",match);
                        while (!feof(cfPtr))
                                if ( fgets(line, MAXLINE, cfPtr) != NULL ) {
                                        temp = strstr(line, match);
                                        if ( temp !=NULL )   countMatch++;   }
                        printf("numero match--> %d", countMatch);
                }
        }
        else printf("errore apertura file");
        return 0;
}
```

Che cosa fa?

Se non si capisce...

provare per credere!

char * strstr(char* s, char* p)
restituisce NULL, o un puntatore al carattere di s da cui inizia la sottostringa p (se presente)

Lettura / scrittura per blocchi di byte

- Ci sono funzioni che consentono di scrivere o leggere un intero blocco di dati testuali o binari
  - Utili, per esempio, quando si vuole scrivere su file un'intera struct
- Funzioni di libreria (non descritte qui):
  - fread(...)
  - fwrite(...)
- Consultare i manuali del C!

Accesso diretto (I)

- Si può accedere ad uno specifico byte come se il file fosse un array di blocchi di byte:
  - int fseek(FILE * fp, long offset, int refpoint)
    - imposta la posizione corrente a un valore pari a uno **spostamento** (positivo o negativo) pari a offset, calcolato **rispetto** a uno dei seguenti punti di partenza:
      - L'inizio del file, se refpoint vale SEEK_SET (costante di stdio.h)
      - L'attuale posizione corrente, se refpoint vale SEEK_CUR (altra costante di stdio.h)
      - La fine del file, se refpoint vale SEEK_END (costante di stdio.h)
    - restituisce 0 se l'operazione di spostamento va a buon fine, un valore diverso altrimenti

Accesso diretto (II)

- long int **ftell**(FILE * fp)
  - restituisce la posizione corrente della “testina":
    - per file binari è il numero di byte dall'inizio
    - per file testuali il numero dipende dall'implementazione
- void rewind(FILE * fp)
  - definita dall'equivalenza:
  - void rewind(f) ≡ fseek (f, 0, SEEK_SET);
    - "riavvolge" il file (la pos. corrente torna all'inizio)