Programmazione B
Ingegneria e Scienze Informatiche - Cesena
A.A. 2020-2021

I FILE

Andrea Piroddi - andrea.piroddi@unibo.it
Credit: Pietro Di Lena

Introduzione

- In C le operazioni di input e output (I/O) sono gestite come operazioni su **stream di dati** (flusso di dati).
- Uno stream di dati viene gestito in modo uniforme tramite l'**astrazione file**, indipendentemente dal fatto che si lavori con device fisici (come ad esempio tastiera, monitor, stampante) oppure con dati strutturati su memoria di massa (come disco rigido).
  - L'astrazione file permette di gestire in maniera uniforme le operazioni di I/O senza considerare le varie caratteristiche fisiche dei device con cui sono effettuate queste operazioni.

Introduzione

- In C le operazioni di input e output (I/O) sono gestite come operazioni su **stream di dati** (flusso di dati).
- Uno stream di dati viene gestito in modo uniforme tramite l'**astrazione file**, indipendentemente dal fatto che si lavori con device fisici (come ad esempio tastiera, monitor, stampante) oppure con dati strutturati su memoria di massa (come disco rigido).
  - L'astrazione file permette di gestire in maniera uniforme le operazioni di I/O senza considerare le varie caratteristiche fisiche dei device con cui sono effettuate queste operazioni.
- Sono supportate due forme di stream.
  - Stream di dati **testuali**: consiste di una sequenze ordinata di caratteri organizzati in righe, terminate da un carattere di nuova linea.
  - Stream di dati **binari**: consiste di una sequenza di caratteri.
  - Nei sistemi Unix i due stream sono equivalenti. Su altre piattaforme lo stream testuale permette di gestire in modo trasparente la conversione di caratteri *speciali* che sono inseriti in file di testo.

Introduzione

- In C le operazioni di input e output (I/O) sono gestite come operazioni su **stream di dati** (flusso di dati).
- Uno stream di dati viene gestito in modo uniforme tramite l'**astrazione file**, indipendentemente dal fatto che si lavori con device fisici (come ad esempio tastiera, monitor, stampante) oppure con dati strutturati su memoria di massa (come disco rigido).
  - L'astrazione file permette di gestire in maniera uniforme le operazioni di I/O senza considerare le varie caratteristiche fisiche dei device con cui sono effettuate queste operazioni.
- Sono supportate due forme di stream.
  - Stream di dati **testuali**: consiste di una sequenze ordinata di caratteri organizzati in righe, terminate da un carattere di nuova linea.
  - Stream di dati **binari**: consiste di una sequenza di caratteri.
  - Nei sistemi Unix i due stream sono equivalenti. Su altre piattaforme lo stream testuale permette di gestire in modo trasparente la conversione di caratteri *speciali* che sono inseriti in file di testo.
- La struttura dati che rappresenta uno stream è chiamata `FILE`.
  - Nonostante il nome, un oggetto di tipo `FILE` non rappresenta necessariamente uno stream di dati associato ad un file sulla memoria di massa.
- Le dichiarazioni del tipo `FILE` ed i prototipi delle funzioni di I/O sono definiti nell'header di libreria standard `stdio.h`.

Il tipo di dato FILE

- Il tipo di dato `FILE` è una struttura (identificatore dichiarato con `typedef`).
- Contiene una lista di campi che permettono di gestire le operazioni di **apertura, chiusura, lettura, scrittura** del file.
- La struttura interna del tipo di dato `FILE` dipende dall'implementazione.
- Il programmatore dovrebbe utilizzare il tipo `FILE` soltanto tramite le funzioni di libreria fornite, senza accedere direttamente ai campi interni della struttura.
- Codice C che si affidi ad una particolare organizzazione interna del tipo `FILE` non è portabile su architetture differenti.
- Le funzioni di I/O che lavorano su file prendono come argomenti formali **puntatori** alla struttura `FILE`.

La struttura FILE: esempio

Esempio di dichiarazione della struttura FILE: implementazione Apple derivata dalla versione 8.5 di `stdio.h` (4/29/95) sviluppata a Berkeley (University of California).

```
1  /* Copyright (c) 2000, 2005, 2007, 2009 Apple Inc. All rights reserved. */
2  typedef struct __sFILE {
3    unsigned char *_p;  /* current position in (some) buffer */
4    int _r;    /* read space left for getc() */
5    int _w;    /* write space left for putc() */
6    short _flags;    /* flags, below; this FILE is free if 0 */
7    short _file;     /* fileno, if Unix descriptor, else -1 */
8    struct  __sbuf _bf; /* the buffer (at least 1 byte, if !NULL) */
9    int _lbfsize; /* 0 or -_bf._size, for inline putc */
10
11   /* operations */
12   void  *_cookie; /* cookie passed to io functions */
13   int (*_close)(void *);
14   int (*_read) (void *, char *, int);
15   fpos_t  (*_seek) (void *, fpos_t, int);
16   int (*_write)(void *, const char *, int);
17
18   /* separate buffer for long sequences of ungetc() */
19   struct  __sbuf _ub; /* ungetc buffer */
20   struct __sFILEX *_extra; /* additions to FILE to not break ABI */
21   int _ur;     /* saved _r when _r is counting ungetc data */
22
23   /* tricks to meet minimum requirements even when malloc() fails */
24   unsigned char _ubuf[3]; /* guarantee an ungetc() buffer */
25   unsigned char _nbuf[1]; /* guarantee a getc() buffer */
26
27   /* separate buffer for fgetln() when line crosses buffer boundary */
28   struct  __sbuf _lb; /* buffer for fgetln() */
29
30   /* Unix stdio files get aligned to block boundaries on fseek() */
31   int _blksize; /* stat.st_blksize (may be != _bf._size) */
32   fpos_t  _offset;  /* current lseek offset (see WARNING) */
33 } FILE;
```

I FILE *standard*

- Il C gestisce tutti i *device* tramite l'interfaccia unica `FILE`.
- In particolare, il device di input *tastiera* e il device di output *monitor* sono interfacciati esattamente come i file di dati su memoria di massa.

I FILE *standard*

- Il C gestisce tutti i *device* tramite l'interfaccia unica `FILE`.
- In particolare, il device di input *tastiera* e il device di output *monitor* sono interfacciati esattamente come i file di dati su memoria di massa.
- Esistono tre *stream standard* che sono aperti di default in un programma in esecuzione per poter comunicare con tastiera e monitor:

Tipo	**Nome**	**Device**
Standard Input	`stdin`	Tastiera
Standard Output	`stdout`	Monitor
Standard Error	`stderr`	Monitor

- Le variabili `stdin`, `stdout`, `stderr` sono di tipo puntatore alla struttura `FILE`.
- Lo *standard input* è uno stream aperto con il device di input *tastiera*. Viene utilizzato solo in lettura.
- Lo *standard error* è lo stream generalmente utilizzato per stampare messaggi di errore.
- I dati inviati allo stream `stderr` sono mostrati sul monitor, così come quelli inviati allo stream `stdout`.
- I due stream `stderr` e `stdout` possono essere *rediretti* separatamente, permettendo così di distinguere l'output del programma dai messaggi di errore.

Operazioni su FILE

- La libreria standard di I/O, oltre alla definizione della struttura `FILE`, contiene diverse funzioni per poter lavorare con stream di dati.
- **Cosa vediamo**.
  - Funzioni per l'apertura e chiusura di uno stream di dati.
  - Funzioni per la scrittura di dati su uno stream.
  - Funzioni per la lettura di dati da uno stream.

Operazioni su FILE

- La libreria standard di I/O, oltre alla definizione della struttura `FILE`, contiene diverse funzioni per poter lavorare con stream di dati.
- **Cosa vediamo**.
  - Funzioni per l'apertura e chiusura di uno stream di dati.
  - Funzioni per la scrittura di dati su uno stream.
  - Funzioni per la lettura di dati da uno stream.
- **Cosa non vediamo** (tra gli argomenti principali)
  - Funzione per la *sincronizzazione* di uno stream di output con il device associato
    - `fflush()`
  - Funzioni di *posizionamento* all'interno di un file
    - `ftell()`, `fseek()`, `fgetpos()`, `fsetpos()`, `rewind()`
  - Funzioni per la gestione di errori
    - `ferror()`, `perror()`, `feof()`, `clearerr()`

Apertura di un file

- Per poter lavorare con un file su memoria di massa è necessario creare uno stream associato al file.

```
1 FILE *fopen(const char *filename, const char *mode);
```

- La funzione `fopen()` apre e associa uno stream con file il cui nome è specificato nella stringa `filename`.
- Restituisce un puntatore all'oggetto che controlla lo stream, oppure `NULL` se l'operazione di apertura dello stream non è possibile.

Apertura di un file

- Per poter lavorare con un file su memoria di massa è necessario creare uno stream associato al file.

```
1 FILE *fopen(const char *filename, const char *mode);
```

- La funzione `fopen()` apre e associa uno stream con file il cui nome è specificato nella stringa `filename`.
- Restituisce un puntatore all'oggetto che controlla lo stream, oppure `NULL` se l'operazione di apertura dello stream non è possibile.
- Le modalità di apertura sono specificate nella stringa `mode`.

Modalità	**Stringa**	**Descrizione**
Lettura	`"r"`	Apre un file esistente in sola lettura
Scrittura	`"w"`	Apre un file in scrittura. Se il file esiste, viene troncato a zero. Altrimenti, viene creato.
Append	`"a"`	Apre un file in scrittura. Se il file non esiste, viene creato. Se esiste, il contenuto attuale non viene modificato e i nuovi dati vengono scritti a partire dalla fine del file.
Lettura e scrittura	`"r+"`	Apre un file in lettura e scrittura, senza troncarne il contenuto.
Scrittura e lettura	`"w+"`	Apre un file in scrittura e lettura, troncandone il contenuto.
Lettura e append	`"a+"`	Apre un file in lettura (a partire dall'inizio del file) e scrittura (a partire dalla fine).

- Per aprire uno stream binario è sufficiente aggiungere il carattere `b` nella stringa `mode`.

Apertura di un file: esempi

- Come aprire un file testuale in modalità di sola lettura.

```
1 char *file = "data.txt";
2 FILE *fp   = fopen(file, "r");
3
4 if(fp == NULL)
5   printf("%s: file does not exist or cannot be opened\n",file);
6 else
7   printf("%s: file successfully opened\n",file);
```

Se il file `data.txt` non esiste nella directory corrente o se non si possiedono i permessi di lettura, `fopen()` ritorna `NULL`.

Apertura di un file: esempi

- Come aprire un file testuale in modalità di sola lettura.

```
1 char *file = "data.txt";
2 FILE *fp   = fopen(file, "r");
3
4 if(fp == NULL)
5   printf("%s: file does not exist or cannot be opened\n",file);
6 else
7   printf("%s: file successfully opened\n",file);
```

Se il file `data.txt` non esiste nella directory corrente o se non si possiedono i permessi di lettura, `fopen()` ritorna `NULL`.

- Come aprire un file testuale in modalità di sola scrittura.

```
1 char *file = "data.txt";
2 FILE *fp  = fopen(file, "w");
3
4 if(fp == NULL)
5   printf("%s: file cannot be created\n",file);
6 else
7   printf("%s: file successfully created\n",file);
```

In questo caso il file `data.txt` viene creato se non esiste nella directory corrente. Se esiste il suo contenuto viene perso. Se non è possibile creare il file, `fopen()` ritorna `NULL`.

Chiusura di un file

- Ad eccezione degli stream standard, tutti gli stream aperti dovrebbero essere chiusi prima della terminazione del programma.
- Nel caso di stream in scrittura, la chiusura assicura che tutte le operazioni di scrittura siano effettuate.
- La funzione di libreria per chiudere file è la seguente.

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

Chiusura di un file

- Ad eccezione degli stream standard, tutti gli stream aperti dovrebbero essere chiusi prima della terminazione del programma.
- Nel caso di stream in scrittura, la chiusura assicura che tutte le operazioni di scrittura siano effettuate.
- La funzione di libreria per chiudere file è la seguente.

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

- La funzione `fclose()` ritorna 0 se l'operazione di chiusura è andata a buon file, oppure `EOF` se ci sono stati problemi.

```
1 char *file = "data.txt";
2 FILE *fp   = fopen(file, "w");
3
4 if(fclose(fp) != EOF)
5   printf("%s: file successfully closed\n",file);
6 else
7   printf("%s: file cannot be closed\n",file);
8
9 printf("%d\n",fclose(fp)==EOF);
```

La `printf()` a riga 8 stampa 1 poichè li stream associato a `data.txt` è già stato chiuso a riga 3.

Scrittura testuale su file

- Per poter stampare dati in formato testuale su un file possiamo utilizzare le seguenti funzioni della libreria standard I/O.

```
1 int fprintf(FILE *stream, const char *format, ...);
2
3 int fputc(int c, FILE *stream);
4
5 int fputs(const char *s, FILE *stream);
```

- Sono perfettamente equivalenti alle funzioni `printf()`, `putc()` e `puts()`.
- Per poterle utilizzare con stream di output su monitor è sufficiente passare `stdout` o `stderr` come argomento `stream`.

Scrittura testuale su file: esempio

- Il seguente esempio mostra come scrivere su file un array di `int` in formato testuale.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int *randvect(unsigned int n) {
  int i,*v = (int *)malloc(n*sizeof(int));
  if(v!=NULL) for(i=0; i<n; i++) printf("Value: %d\n",v[i] = random());
  return v;
}

int main(int argc, char *argv[]) {
  int *v, i, n;
  FILE *out;

  if(argc!=3) {
    fprintf(stderr,"Usage: fprintf <n> <filename>\n");
    return 1;
  }
  if((n=atoi(argv[1]))<=0) {
    fprintf(stderr,"Error: n should be greater than zero\n");
    return 1;
  }
  if((out=fopen(argv[2],"w"))==NULL) {
    fprintf(stderr,"Error: %s cannot be opened\n",argv[2]);
    return 1;
  }
  srandom(time(0));
  if ((v = randvect((unsigned int)n)) != NULL) {
    // Salva il contenuto del vettore v su file testuale.
    for(i=0; i<n; i++) fprintf(out,"%d\n",v[i]);
    free(v);
  }
  fclose(out);
  return 0;
}
```

Scrittura di dati binari su file

- La scrittura di un blocco di byte su file può essere effettuata con la seguente funzione di libreria standard I/O.

```
1 size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- La `fwrite()` scrive sullo stream puntato da `stream` fino a `nmemb` oggetti di dimensione `size` a partire dall'indirizzo di memoria puntato da `ptr`.
- Può essere utilizzata per salvare direttamente in binario su file una porzione contigua della memoria RAM.
- Ritorna il numero di oggetti effettivamente scritti. Questo valore è differente dal valore passato `nmemb` soltanto se la scrittura non è stata effettuata con successo.

Scrittura di dati binari su file: esempio

- Il seguente esempio mostra come scrivere su file un array di `int` in formato binario.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int *randvect(unsigned int n) {
  int i,*v = (int *)malloc(n*sizeof(int));
  if(v!=NULL) for(i=0; i<n; i++) printf("Value: %d\n",v[i] = random());
  return v;
}

int main(int argc, char *argv[]) {
  int *v, n;
  FILE *out;

  if(argc!=3) {
    fprintf(stderr,"Usage: fwrite <n> <filename>\n");
    return 1;
  }
  if((n=atoi(argv[1]))<=0) {
    fprintf(stderr,"Error: n should be greater than zero\n");
    return 1;
  }
  if((out=fopen(argv[2],"wb"))==NULL) {
    fprintf(stderr,"Error: %s cannot be opened\n",argv[2]);
    return 1;
  }
  srandom(time(0));
  if((v = randvect((unsigned int)n)) != NULL) {
    // Salva il vettore v su file. Non controlla eventuali errori
    fwrite(v,sizeof(int),n,out);
  }
  free(v);
  fclose(out);
  return 0;
}
```

Lettura da file testuali

- Per poter leggere il contenuto di un file testuale possiamo utilizzare le seguenti funzioni della libreria standard I/O.

```
1 int    fscanf(FILE *stream, const char *format, ...);
2
3 int    fgetc(FILE *stream);
4
5 char *fgets(char *s, int n, FILE *stream);
```

- Sono (quasi) perfettamente equivalenti alle funzioni `scanf()`, `getc()` e `gets()`.
- Per poterle utilizzare con stream di input da tastiera è sufficiente passare `stdin` come argomento `stream`.
- Notiamo che la `fgets()` è *sicura* anche su stream di input da tastiera (a differenza della `gets()`) perché permette di limitare il numero massimo di caratteri letti (parametro formale `n`).

Lettura da file testuali: esempio

- Il seguente esempio mostra come leggere un array di `int` da un file testuale.
- E' complementare all'esempio che mostra come scrivere su file un array di `int` in formato testuale.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[]) {
5    int *v, n, i;
6    FILE *in;
7
8    if(argc!=3) {
9      fprintf(stderr,"Usage: fscanf <n> <filename>\n");
10     return 1;
11   }
12   if((n=atoi(argv[1]))<=0) {
13     fprintf(stderr,"Error: n should be greater than zero\n");
14     return 1;
15   }
16   if((in=fopen(argv[2],"r"))==NULL) {
17     fprintf(stderr,"Error: %s cannot be opened\n",argv[2]);
18     return 1;
19   }
20
21   if ((v = (int *)malloc(n*sizeof(int))) != NULL) {
22     // Legge n interi e li salva in v.
23     for(i=0; i<n; i++) fscanf(in,"%d",&v[i]);
24   }
25   // Stampa i valori letti
26   for(i=0; i<n; i++) printf("Value: %d\n",v[i]);
27
28   fclose(in);
29   return 0;
30 }
```

Lettura da file binari

- Per poter leggere il contenuto di un file binario possiamo utilizzare le seguenti funzioni della libreria standard I/O.

```
1 size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- La `fread()` legge dallo stream puntato da `stream` fino `nmemb` oggetti, la cui dimensione è `size` byte, e li salva nell'array puntato da `ptr`.
- Ritorna il numero di oggetti effettivamente letti. Questo valore è differente dal valore passato `nmemb` soltanto se la lettura non è stata effettuata con successo o se il file termina prima che siano stati letti esattamente `nmemb` oggetti.

Lettura da file binari: esempio

- Il seguente esempio mostra come leggere un array di `int` da un file binario.
- E' complementare all'esempio che mostra come scrivere su file un array di `int` in formato binario.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[]) {
5    int *v, n, i;
6    FILE *in;
7
8    if(argc!=3) {
9      fprintf(stderr,"Usage: fread <n> <filename>\n");
10     return 1;
11   }
12   if((n=atoi(argv[1]))<=0) {
13     fprintf(stderr,"Error: n should be greater than zero\n");
14     return 1;
15   }
16   if((in=fopen(argv[2],"rb"))==NULL) {
17     fprintf(stderr,"Error: %s cannot be opened\n",argv[2]);
18     return 1;
19   }
20
21   if((v = (int *)calloc(n,sizeof(int))) != NULL) {
22     // Legge n interi e li salva in v. Non controlla eventuali errori
23     fread(v,sizeof(int),n,in);
24   }
25   // Stampa i valori letti
26   for(i=0; i<n; i++) printf("Value: %d\n",v[i]);
27
28   fclose(in);
29   return 0;
30 }
```

Lettura da file di testo: esempio 1

- Conteggio del numero di linee in un file: versione semplificata della utility Unix `wc`, che permette di contare parole, linee, caratteri e byte in un file.

```c
// lc.c: line count
#include <stdio.h>

// Conta il numero di righe in filename
int lc(char *filename) {
  int c, n = 0;
  FILE *in;

  if((in = fopen(filename,"r")) != NULL)
    while((c = fgetc(in)) != EOF)
      n += c == '\n'; // incrementa n se legge un newline

  fclose(in);
  return n;
}

int main(int argc, char *argv[]) {
  // Se non viene passato niente in input stampa 0
  printf("%u %s\n",lc(argv[1]),argv[1]==NULL?"":argv[1]);

  return 0;
}
```

Lettura da file di testo: esempio 2

- Conteggio del numero di caratteri in un file: versione semplificata della utility Unix `wc`, che permette di contare parole, linee, caratteri e byte in un file.

```
1  // cc.c: character count
2  #include <stdio.h>
3
4  // Conta il numero di caratteri in filename
5  int cc(char *filename) {
6    int c, n = 0;
7    FILE *in;
8
9    if((in = fopen(filename,"r")) != NULL)
10     while((c = fgetc(in)) != EOF)
11       n++; // incrementa n per ogni carattere letto
12
13   fclose(in);
14   return n;
15 }
16
17 int main(int argc, char *argv[]) {
18   // Se non viene passato niente in input stampa 0
19   printf("%u %s\n",cc(argv[1]),argv[1]==NULL?"":argv[1]);
20
21   return 0;
22 }
```

Lettura e scrittura da file binari: esempio 3

- Copia binaria di file: versione semplificata della utility Unix cp.

```c
#include <stdio.h>
#include <string.h>
#define N 4096

int copy(FILE *in, FILE *out, char *buf, size_t K) {
  size_t n,m;
  do {
    n =  fread(buf,sizeof(char),K,in);  // Legge massimo K byte
    m = fwrite(buf,sizeof(char),n,out); // Scrive gli n byte letti
  } while((n!=0) && (n==m));
  return n != m;                        // Se n != m c'e' stato un errore di scrittura
}

int main(int argc, char *argv[]) {
  FILE *in, *out;
  char buf[N]; // Buffer di lettura di 4Kb

  if(argc!=3) {
    fprintf(stderr,"Usage: copy <filename> <filename>\n");
    return 1;
  }
  if(strcmp(argv[1],argv[2])==0) {
    fprintf(stderr,"Error: source and destination are the same\n");
    return 1;
  }
  if((in=fopen(argv[1],"rb"))==NULL || (out=fopen(argv[2],"wb"))==NULL) {
    fprintf(stderr,"Error: cannot open %s\n",in==NULL?argv[1]:argv[2]);
    return 1;
  }
  if(copy(in,out,buf,N) != 0) {
    fprintf(stderr,"Error: cannot make a copy\n");
    return 1;
  }
  fclose(in); fclose(out);
  return 0;
}
```