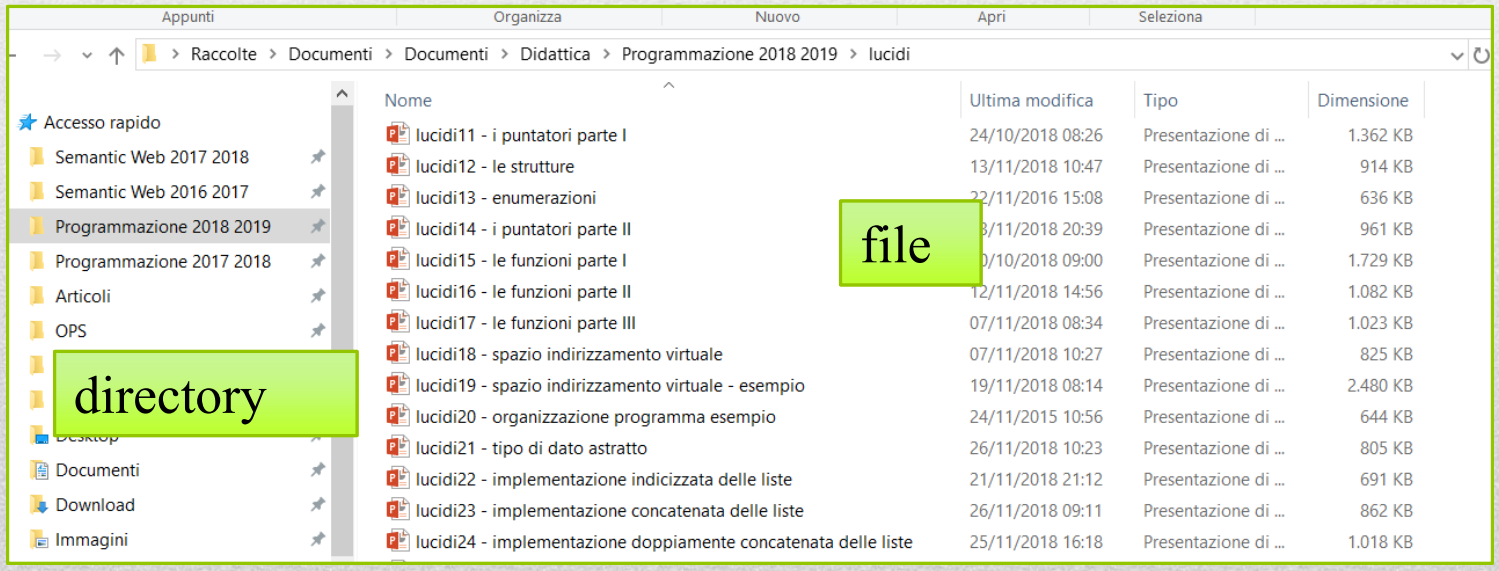
FILE

Directory e file

Tutti i sistemi operativi permettono di organizzare le informazioni su hard disk secondo la metafora di cartelle (**directory**) e **file.**

Il **file** è l'unità logica di memorizzazione dei dati su memoria di massa che consente una memorizzazione persistente dei dati, non limitata dalle dimensioni della memoria centrale. E' caratterizzato da uno specifico nome ed è contenuto all'interno di una specifica cartella.

La **directory** è un contenitore di file e altre directory caratterizzata da uno specifico nome e contenuta all'interno di un'altra directory



Identificazione di file

Per identificare un file occorre conoscere:

- Il nome assegnato al file
- Il nome della directory in cui esso è memorizzato

Una volta identificato un file è possibile accedere al suo contenuto:

- Leggere la sequenza di byte di cui è composto
- Modificare la sequenza di byte di cui è composto

Operazioni sui file

Operazioni **generiche**

- Cancellazione di un file esistente
- Rinomina di un file esistente
- Copia di un file, creando un nuovo file con lo stesso nome in una diversa directory
- Spostamento di un file, equivalente alla copia con cancellazione dell'originale

Operazioni **specifiche**

- Creazione di un nuovo file
- Modifica del contenuto del file

Le operazioni generiche sono solitamente svolte interagendo con il sistema operativo, e si possono applicare a qualsiasi file.
Le operazioni specifiche invece coinvolgono il contenuto del file, pertanto richiedono programmi specifici per ogni tipologia di file.

Tipi di file

Il significato dei byte memorizzati all'interno dei file è noto, solitamente, solamente al programma che lo ha creato

Si parla di **tipi di file**, che sono solitamente indicati **dall'estensione** (ultime lettere del nome del file)

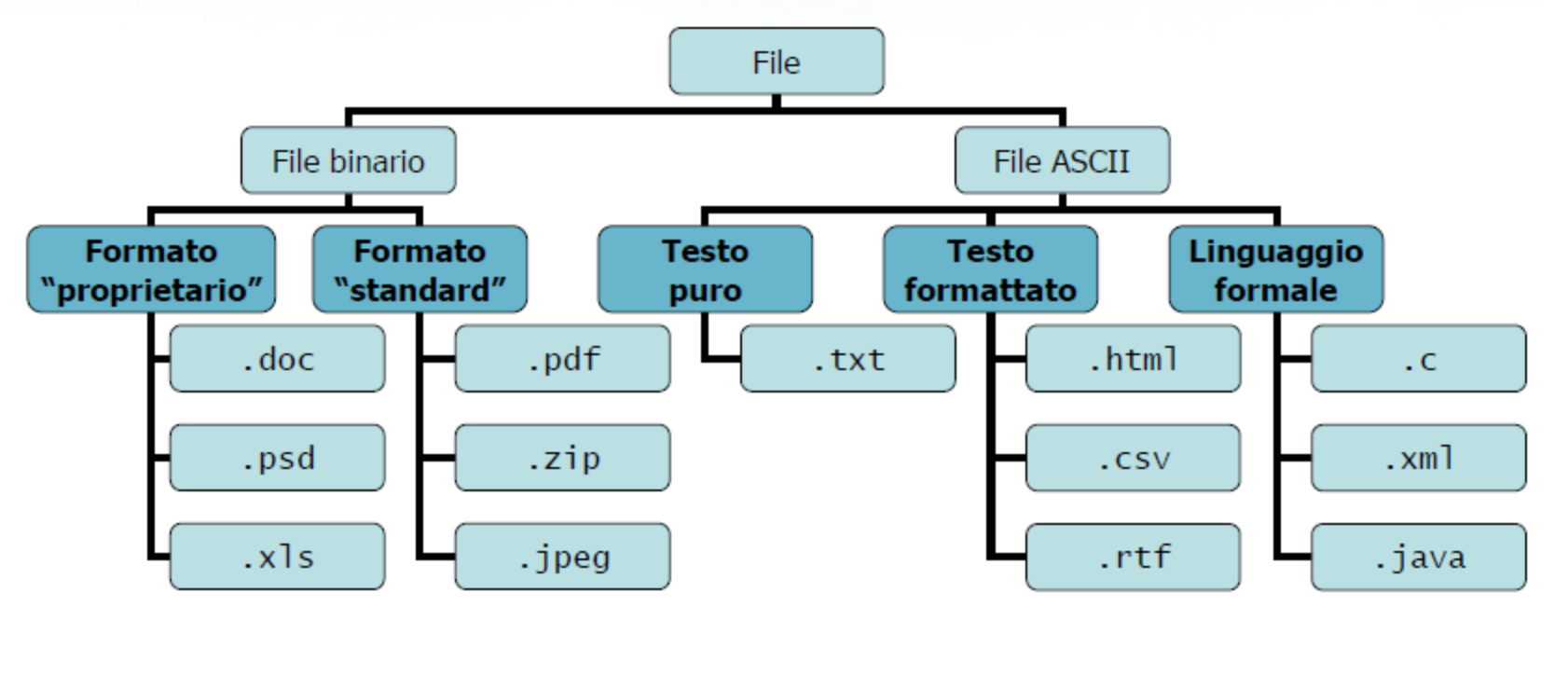
- File prodotto da Microsoft Powerpoint → .ppt
- File prodotto da Microsoft Word → .doc
- File prodotto da Openoffice.Org Writer → .odt
- ...

Quindi non è possibile lavorare con file gestiti da altri programmi, a meno di non conoscere il formato del file

Eccezione: se il formato del file è pubblicamente documentato, allora sarà possibile leggerlo e scriverlo interpretando correttamente i byte

- **Esempio**: file di testo (ASCII): la codifica ASCII è utilizzata in molti campi: testi (.txt), programmi in C (.c), pagine HTML (.html), ...
- **Esempio**: file Acrobat (.pdf): struttura complessa, ma documentata

File binary e di testo



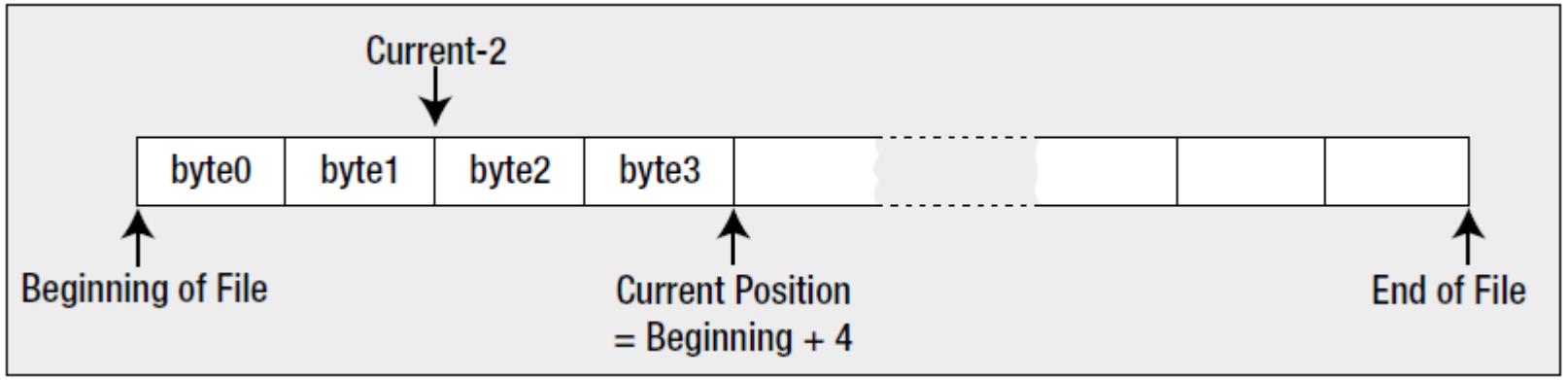
File binary e di testo

A livello di programmazione, esistono:

- Funzioni generali per accedere ai file
- Funzioni specifiche per la lettura e la scrittura del contenuto di **file binari**
- Funzioni specifiche per la lettura e la scrittura del contenuto di **file ASCII**

Il corretto funzionamento di un programma dipende dalla perfetta conoscenza del formato del file stesso.

Struttura di un file in C



La **posizione corrente** indica il punto in cui ogni azione sul file (la lettura **da** un file o la scrittura **su** un file) avviene. La posizione corrente può indicare un punto qualsiasi del file e può essere specificata come offset rispetto al suo punto iniziale, oppure, in certe circostanze, come offset positivo o negativo rispetto alla posizione corrente. La posizione corrente può anche indicare la fine del file.

Accesso ai file di testo in C

Un programma C può accedere ad alcuni file presenti sui dischi del calcolatore

- File **aperto**: file al quale attualmente il programma ha accesso
- File **chiuso**: file residente su disco al quale attualmente il programma non ha accesso

All'atto dell'apertura di un file, il programma deve dichiarare la modalità di accesso:

- Modalità di **lettura**: il programma può leggere il contenuto del file, ma non modificarlo
- Modalità di **scrittura**: il programma può riscrivere da zero il contenuto del file
- Modalità di **aggiunta**: il programma può aggiungere nuove informazioni al file
- Modalità di **lettura/scrittura**: tutte le precedenti

I successivi accessi al file devono essere compatibili con la modalità di accesso dichiarata

Accesso ai file di testo in C

L'accesso ai file di testo è rigorosamente **sequenziale:**

- La **lettura** avviene dalla prima riga all'ultima, dal primo carattere all'ultimo
- In **scrittura**, ogni riga o carattere scritto vengono posizionati dopo le righe o caratteri scritti in precedenza:
  - a partire dal primo carattere, in modalità di **scrittura**
  - a partire dall'ultimo carattere esistente, in modalità di **aggiunta**

All'atto dell'apertura di un file, il programma deve dichiarare se il file è di tipo binario oppure di testo.
La differenza consiste solamente nel trattamento "speciale" del carattere '\n' nel caso dei file di testo.

Stream associato ad un file

In un programma C, esiste un **tipo di dato specifico** per rappresentare le informazioni relative ad un file aperto: **file stream** (flusso associato ad un file)

Tipo di dato: FILE * (definito in <stdio.h>)

Aprire un file significa quindi creare un nuovo stream ed associarlo ad uno specifico file sul disco.

Una volta che il file è aperto, il suo stream rappresenta

- Un "collegamento" mediante il quale poter compiere delle operazioni sul contenuto del file
- Le modalità di accesso scelte (testo/binario, lettura/scrittura/...)
- La posizione attuale a cui si è arrivati nello scrivere o nel leggere il file

Ogni operazione sul file avviene chiamando una funzione che riceve lo stream come parametro.

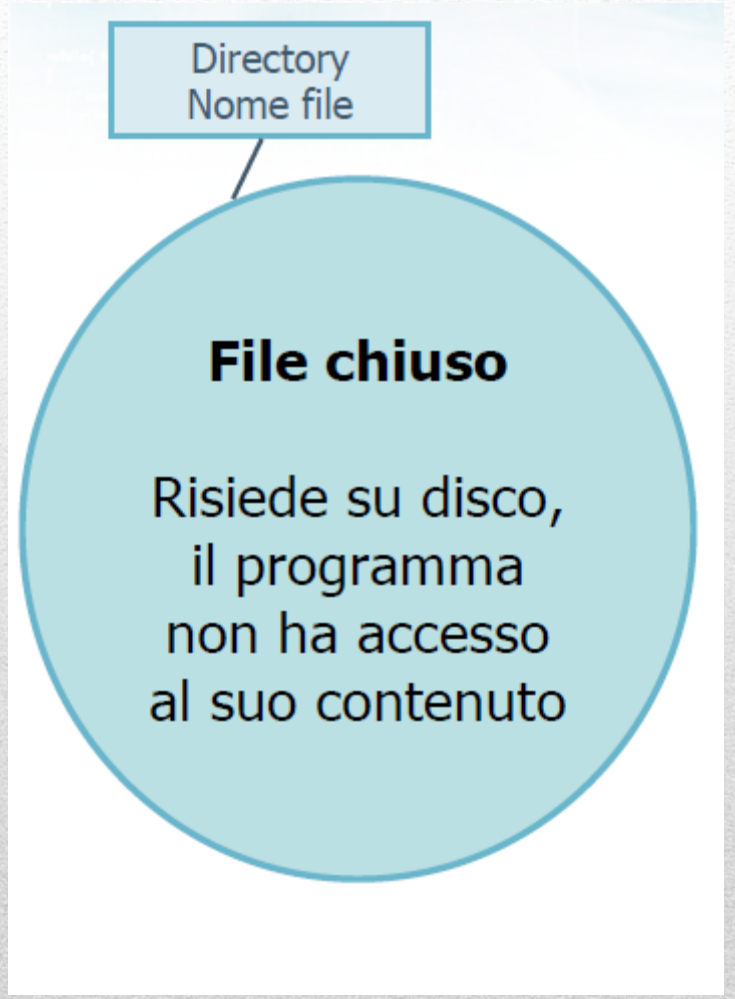
Stream predefiniti

La funzione fopen() permette di aprire nuovi stream, associati a file esistenti sui dischi dell'elaboratore.
All'avvio di un programma in C, sono già stati aperti in modo automatico 3 stream predefiniti:

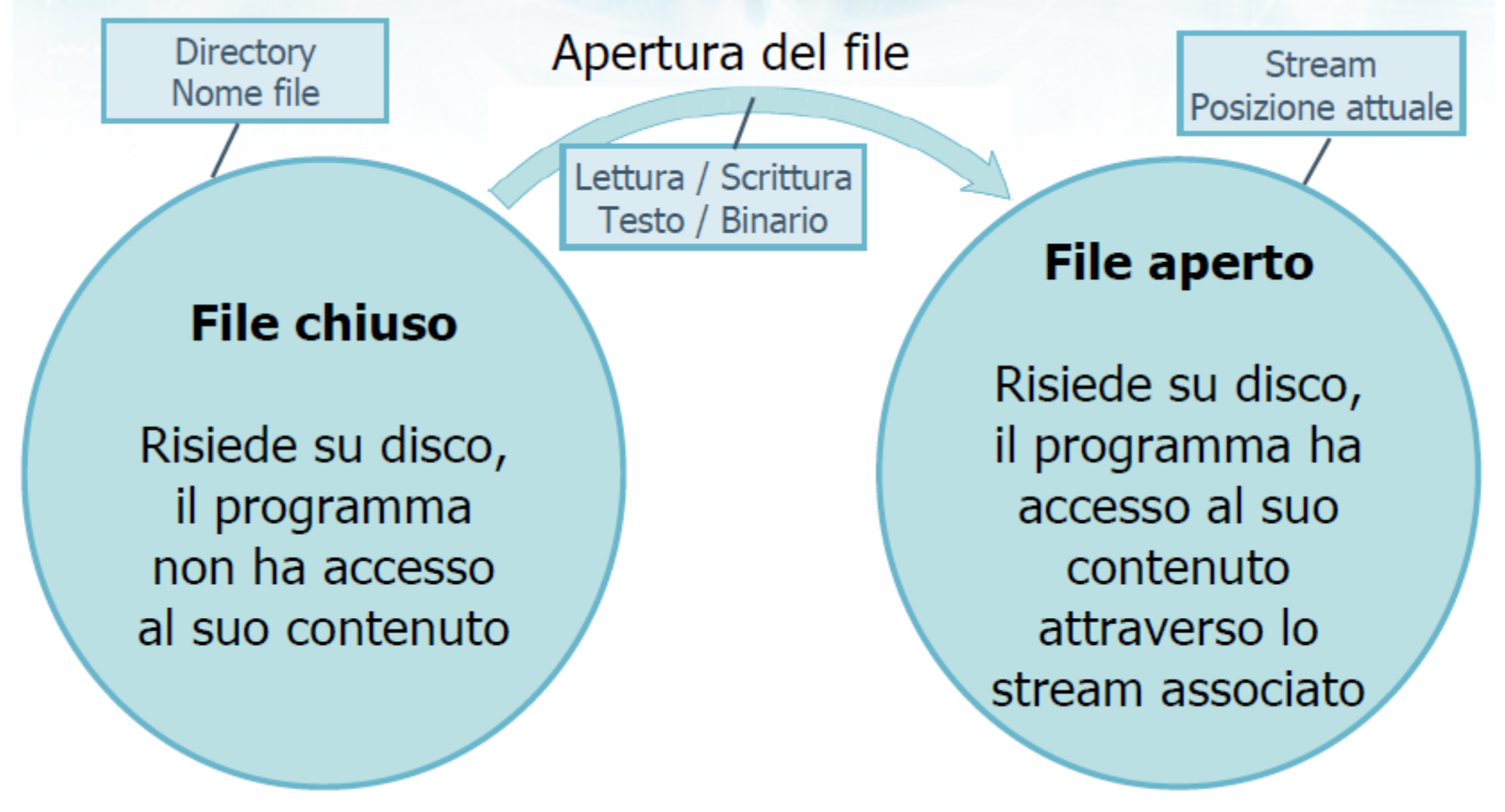
- **stdin** (associato al canale di input del terminale nel quale il programma è avviato), aperto in lettura
- **stdout** (associato al canale di output del terminale nel quale il programma è avviato), aperto in scrittura
- **stderr** (associato al canale di output del terminale nel quale il programma è avviato), aperto in scrittura

Stdin, stdout, stderr sono variabili di tipo puntatore a file automaticamente ed implicitamente definite.

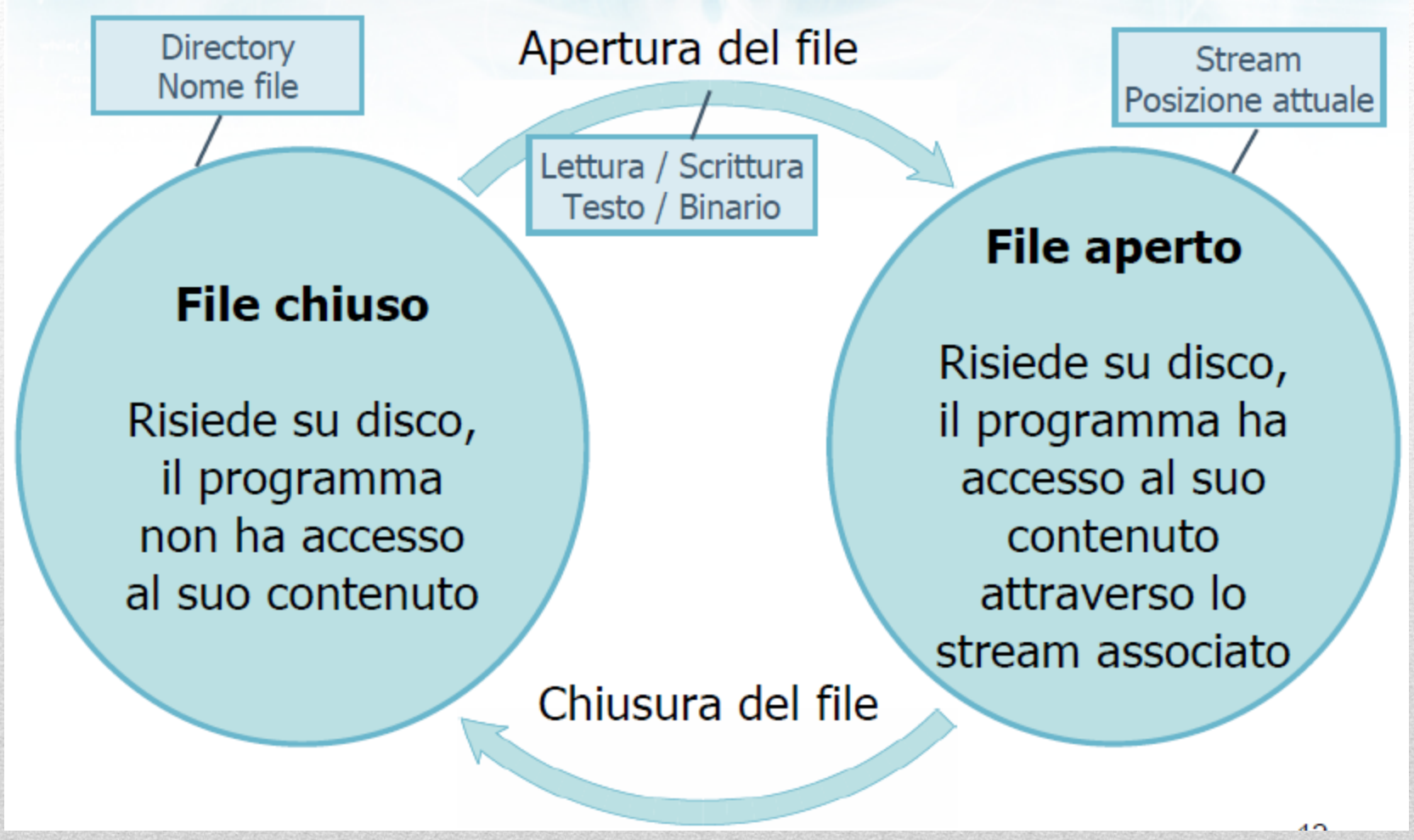
Stati di un file



Stati di un file



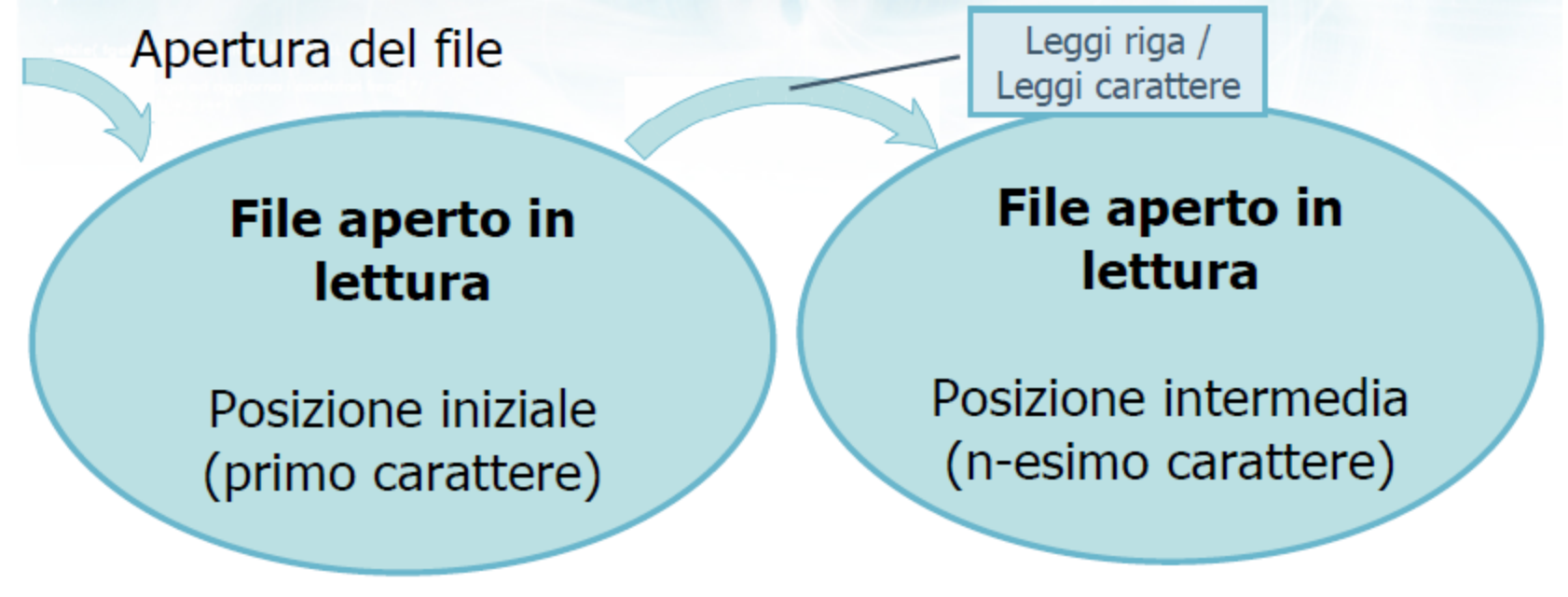
Stati di un file



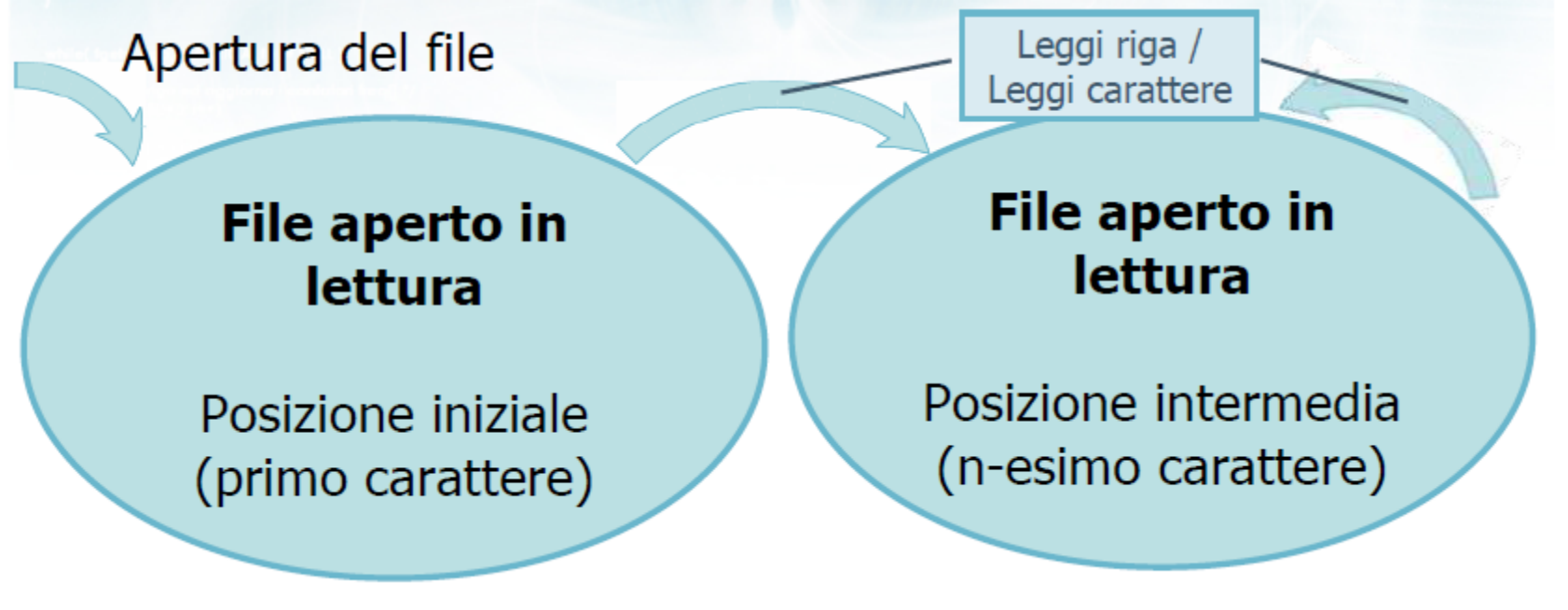
Lettura di un file



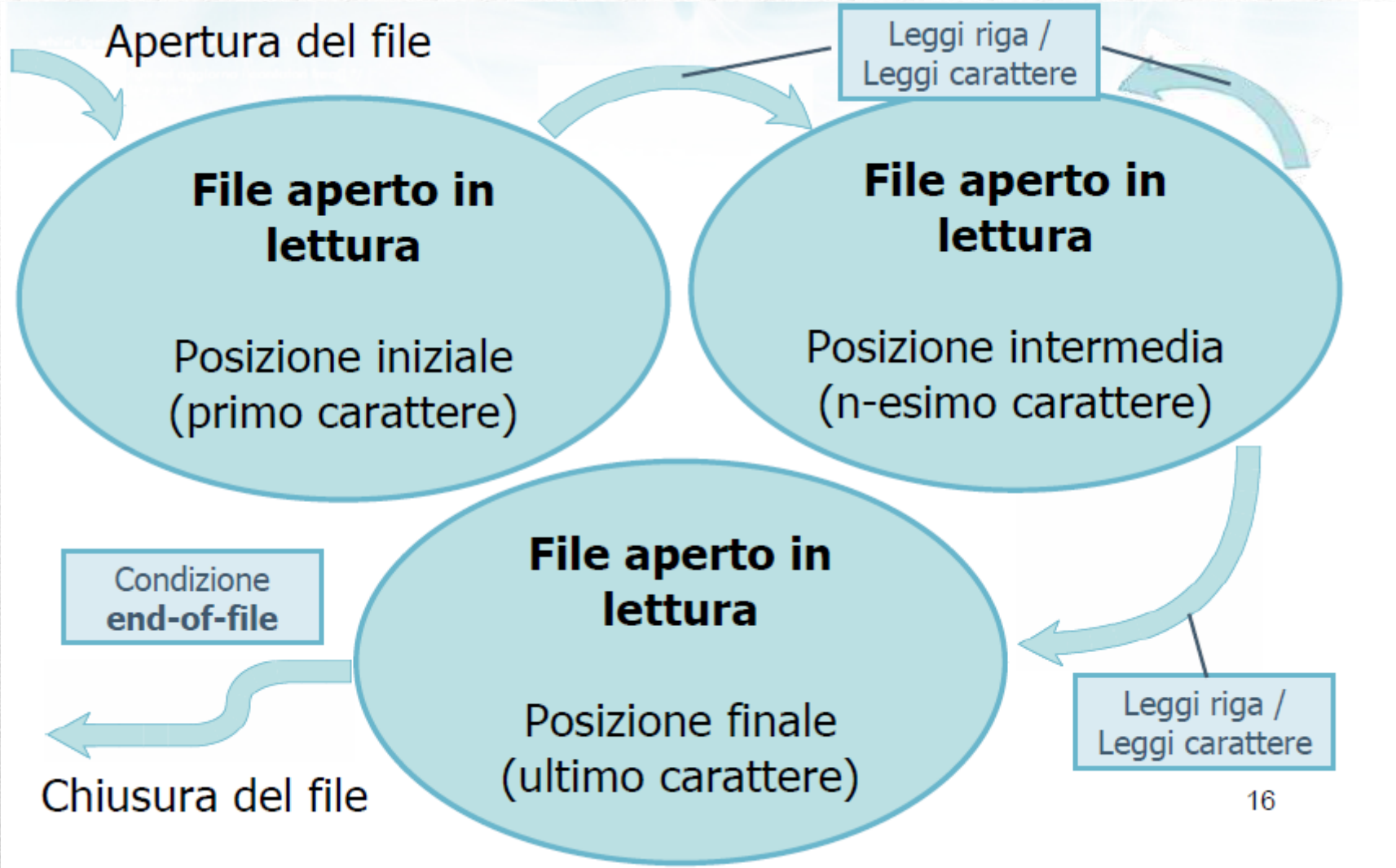
Lettura di un file



Lettura di un file



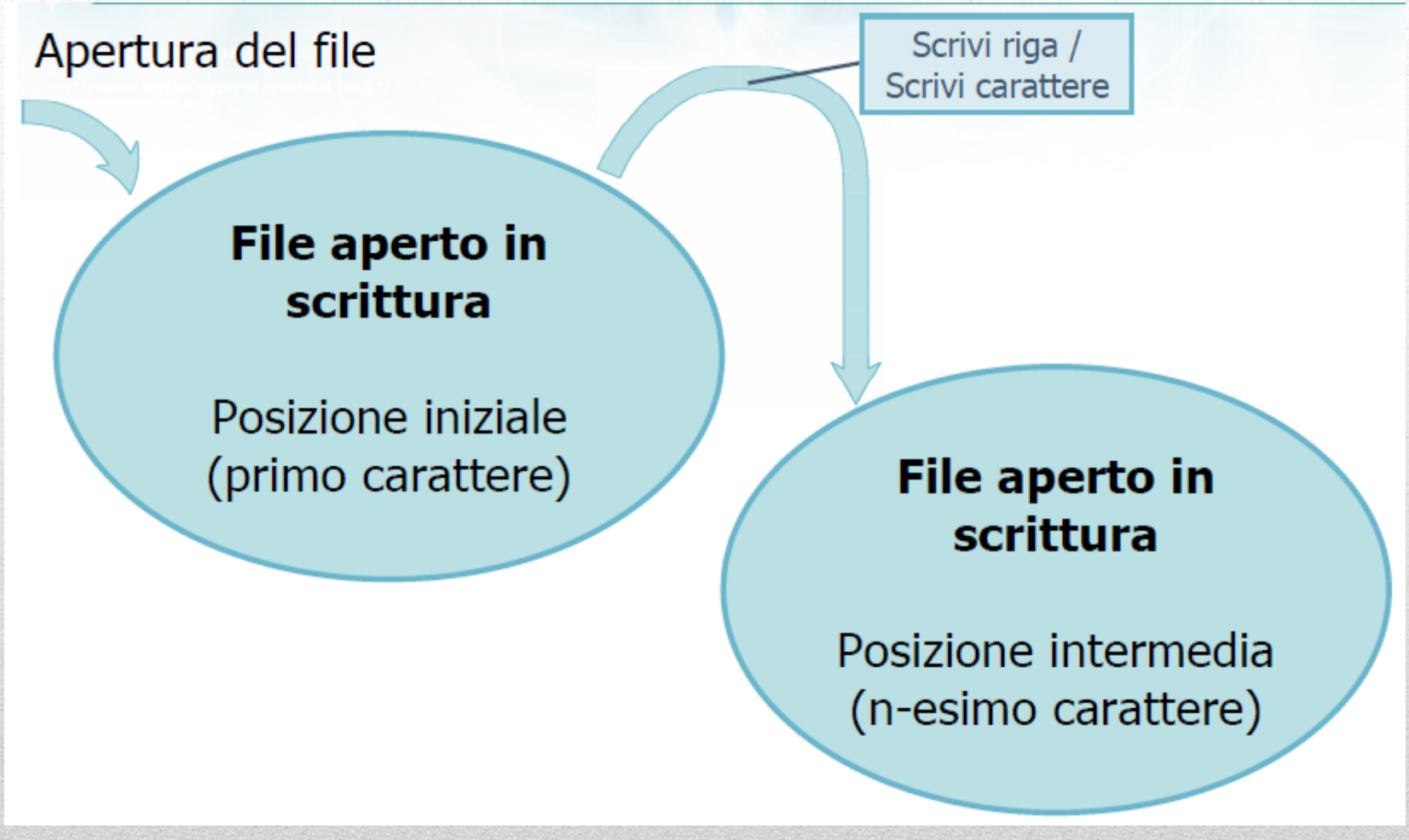
Lettura di un file



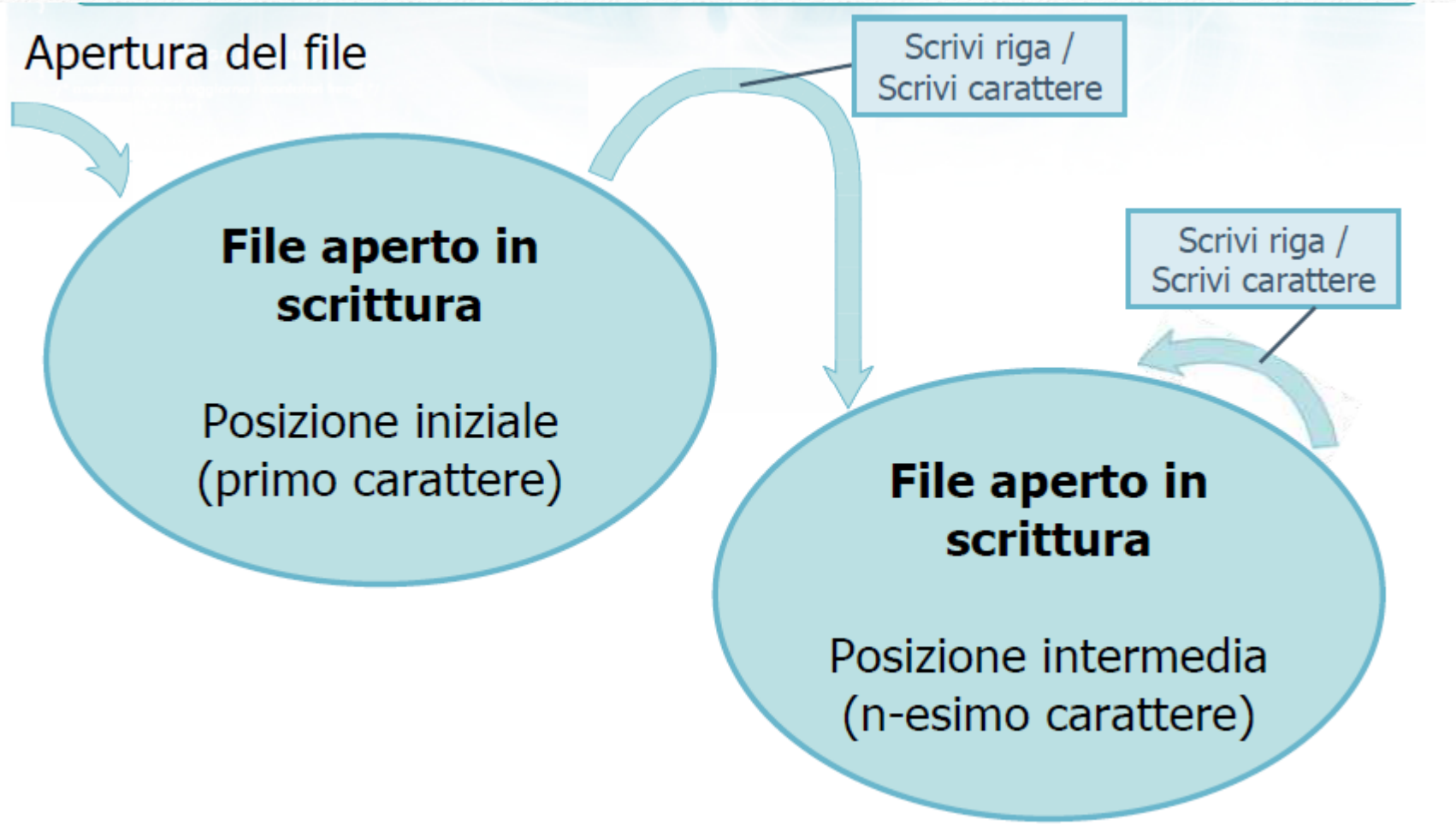
Scrittura di un file



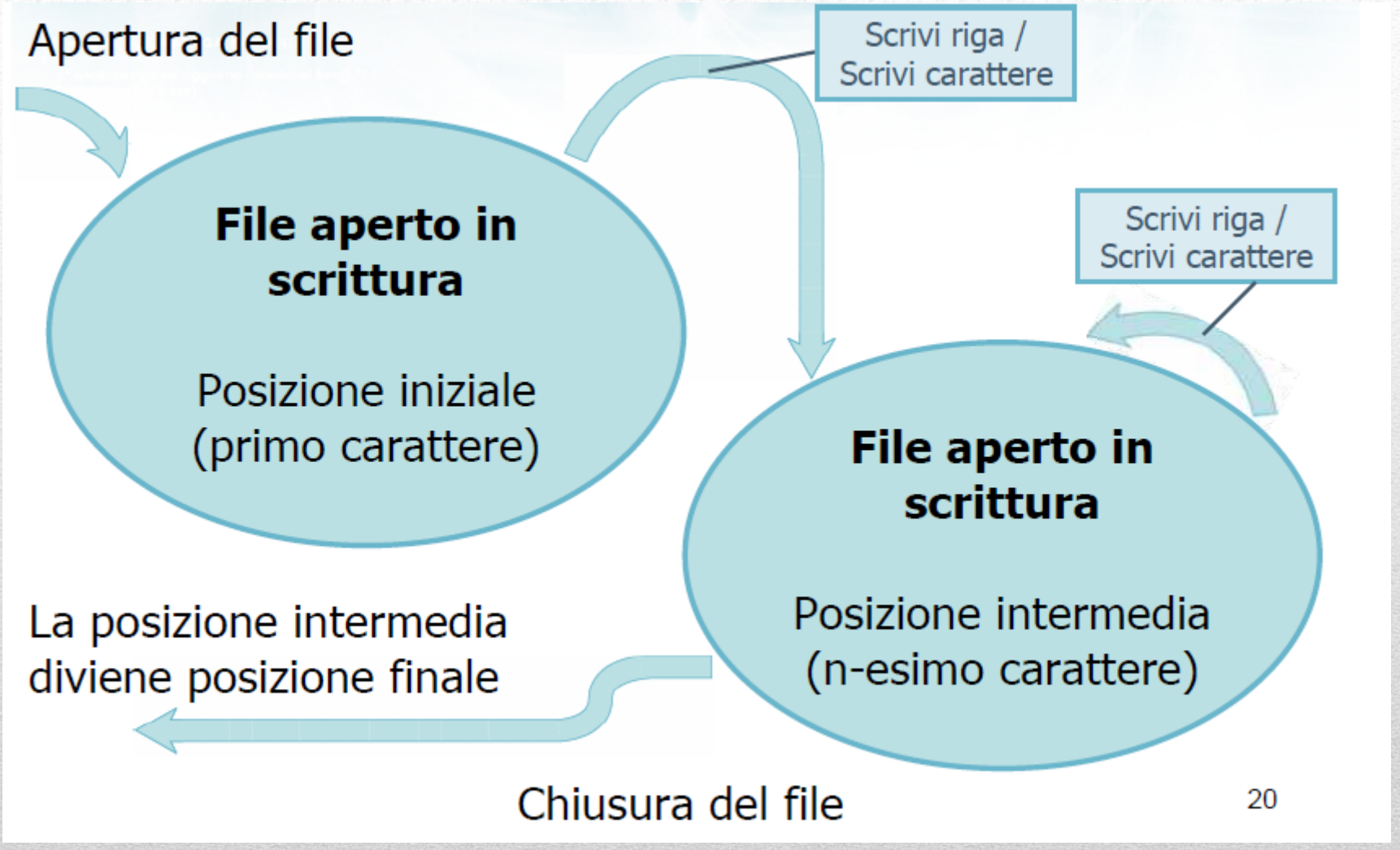
Scrittura di un file



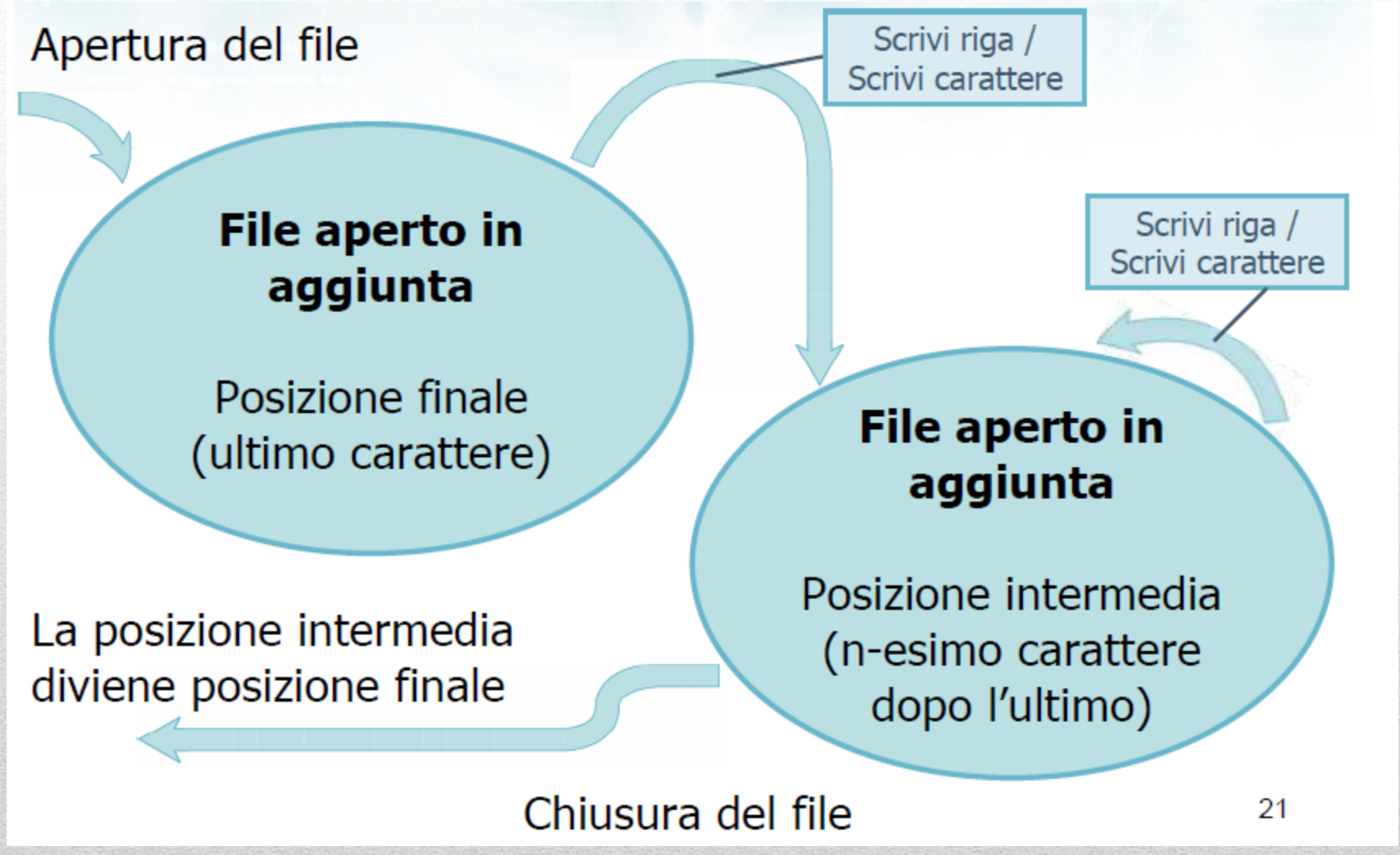
Scrittura di un file



Scrittura di un file



Aggiunta ad un file



Funzioni C

Tutte le funzioni per l'accesso ai file sono contenute in <stdio.h>

Funzioni per **apertura** e **chiusura**: fopen(), fclose()
Funzioni per la **lettura**: fgetc(), fgets(), fscanf()
Funzioni per la **scrittura**: fputc(), fputs(), fprintf()
Funzioni per lo **stato del file**: feof()

fopen(), fclose()

Apertura del file:

- La funzione fopen() apre un file e restituisce una variabile stream
- Richiede il nome del file e le modalità di apertura
- Restituisce una nuova variabile di tipo FILE *

Chiusura del file:

- Quando il file non è più richiesto, si chiama la funzione fclose() che chiude il file
- Richiede come parametro lo stream, precedentemente restituito da fopen()

fopen()

```
FILE *f;                          //variabile stream di tipo FILE *
…
f= fopen(“nomefile”, “modo”);     //nomefile è la stringa contenente il nome del file
                                  //modo è la stringa che specifica la modalità di apertura
```

Esempi:

```
f = fopen( "c:\\prog\\dati.txt", "modo" ) ;

char nome[20];
gets(nome);
f = fopen( nome, "modo" );
```

La modalità di apertura può essere:

- modalità lettura
- modalità scrittura
- modalità append

fopen() in lettura

Modalità "**r**"

- Se il file esiste, viene aperto e la variabile f punta allo stream relativo, posizionato in lettura al primo carattere
- Se il file non esiste, non viene creato nessuno stream e f==NULL

fopen() in scrittura

Modalità "**w**"

- Se il file non esiste, viene creato da zero e la variable f punta allo stream relativo, posizionato in scrittura al primo carattere
- Se il file esiste già, viene innanzitutto cancellato e poi ricreato da zero, f punta allo stream relativo, posizionato in scrittura al primo carattere
- Se non è possibile creare il file (perché la directory non esiste, il disco è protetto in scrittura, ...), non viene creato alcuno stream e f==NULL

fopen() in append

Modalità "**a**"

- Se il file non esiste, viene creato da zero e la variabile f punta allo stream relativo, posizionato in scrittura al primo carattere
- Se il file esiste già, non viene modificato, f punta allo stream relativo, posizionato in scrittura dopo l'ultimo carattere
- Se non è possibile creare o modificare il file (perché la directory non esiste, il disco è protetto in scrittura, ...), non viene creato alcuno stream e f==NULL

fopen(), controllo dell'errore

```
FILE * f ;
...
f = fopen( "nomefile", "r" ) ;
if( f == NULL )
{
   printf("Impossibile aprire file\n");
   exit(1) ;
}
```

Quando viene chiamata la funzione fopen(), è indispensabile ricordarsi di controllare se il valore ritornato è NULL

fclose()

La funzione fclose() può essere chiamata solamente su stream correttamente aperti. Quindi mai chiamare fclose() se f==NULL

Dopo la chiusura del file, non è più possibile accedere allo stream.

```
FILE * f ;
...
f = fopen( "nomefile", "modo" ) ;

...

fclose(f) ;
```

La funzione fclose() ritorna un **intero**:

- 0 se l'operazione di chiusura è eseguita correttamente
- Costante EOF, diversamente

Fare i dovuti controlli, come per la funzione fopen()

Lettura e scrittura su file di testo

Distinguiamo funzioni che lavorano sul singolo carattere:

- fgetc(), fputc()

e funzioni che lavorano su stringa:

- fgets(), fputs()

```
int fgetc (FILE *f):
int fputc(int c, FILE *f);
```

Funzioni simili a getchar() e putchar() ma con un parametro in più che rappresenta il puntatore al file sul quale si vuole leggere o scrivere

In caso di esecuzione corretta restituiscono il carattere letto o scritto come intero, altrimenti EOF

Lettura e scrittura su file di testo

Distinguiamo funzioni che lavorano sul singolo carattere:

- fgetc(), fputc()

e funzioni che lavorano su stringa:

- fgets(), fputs()

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

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

Per lavorare in maniera più **sicura** perché c'è parametro *n* per limitare numero caratteri letti ed evitare overflow buffer

fputs trasferisce la stringa *s* (terminata da \0) nel file puntato da *f*.
Restituisce l'ultimo carattere scritto in caso di terminazione corretta, EOF altrimenti

fgets() trasferisce nella stringa *s* i caratteri letti dal file puntato da *f* fino a quando ha letto n-1 caratteri, oppure ha incontrato newline, oppure la fine del file.
fgets() mantiene il carattere newline nella stringa *s*.
Restituisce la stringa letta nel caso di corretta terminazione, \0 in caso di errore o fine del file

Lettura e scrittura su file di testo: esempi

```
#include  < stdio.h >
int main()
{
   char colour[20];
   printf("Enter your favorite colour\n");
   fgets(colour,20,stdin);
   printf("My favorite colour is %s.\n",name);
   return(0);
}
```

```
#include  < stdio.h >
int main ()
{
   FILE *fp;
   fp = fopen("proverb.txt", "w");
   fputs("Cleanliness is next to godliness.", fp);
   fputs("Better late than never.", fp);
   fputs("The pen is mightier than the sword.", fp);
   fclose(fp);
   return(0);
}
```

Esercizio scrivere su file e poi leggere il file

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define LENGTH 81           // Maximum input length
int main(void)
{
  char mystr[LENGTH];       // Input string
  int mychar = 0;           // Character for output
  FILE *pfile = NULL;       // File pointer
  char *filename = "myfile.txt";

  printf("Enter an interesting string of up to %d characters:\n", LENGTH - 1);
  if(!fgets(mystr, LENGTH, stdin)) {
    printf("Input from keyboard failed.\n");
    exit(1);
  }
```

Esercizio scrivere su file e poi leggere il file

```
// Create a new file we can write
if((pfile=fopen(filename, "w"))==NULL) {
   printf("Error opening %s for writing. Program terminated.\n", filename);
   exit(1);
}

for(int i = strnlen(mystr) - 1 ; i >= 0 ; --i)
   fputc(mystr[i], pfile); // Write string to file backward

fclose(pfile); // Close the file
```

Esercizio scrivere su file e poi leggere il file

```
// Open the file for reading
if((pfile=fopen(filename, "r"))==NULL) {
   printf("Error opening %s for reading. Program terminated.", filename);
   exit(1);
}

// Read a character from the file and display it
printf("the data read from the file is:\n");
while((mychar = fgetc(pfile)) != EOF)
   putchar(mychar);                  // Output character from the file
putchar('\n');

fclose(pfile);                       // Close the file
return 0;
}
```

Esercizio

Sia dato un file di testo, contenente dei brani scritti da un utente
Si scriva un programma in C che stampi le frequenze con cui compaiono le varie lettere dell'alfabeto
Si considerino equivalenti le maiuscole e le minuscole, e si ignorino i caratteri di spaziatura e punteggiatura.
Es.:

Quel ramo del lago di Como,
che volge a mezzogiorno,
tra due catene non interrotte di monti,
tutto a seni e a golfi,
a seconda dello sporgere e del
rientrare di quelli, vien, quasi
a un tratto, a ristringersi, e a
prender corso e figura di fiume,
tra un promontorio a destra,
e un'ampia costiera dall'altra parte

manzoni.txt

Esercizio

Occorre calcolare un **vettore di frequenze**, in cui ciascuna posizione rappresenti la frequenza di ciascuna delle lettere alfabetiche

Ci sono due approcci possibili alla lettura del file:

- Soluzione per caratteri: il file viene letto un carattere alla volta, usando la funzione fgetc
- Soluzione per righe: il file viene letto una riga alla volta, usando la funzione fgets, e tale riga viene poi esaminata con un ciclo interno al programma

Soluzione per caratteri

```
const int LETT = 26 ;
int freq[LETT] ;
FILE * f ;
int ch, i ;
char nome[30];

for(i=0; i<LETT; i++)
   freq[i] = 0;
gets(nome);
if  (f=fopen( nome, "r" )!=NULL) {
 ch = fgetc(f) ;
 while( ch!=EOF ){
    if(isalpha(ch)){
       i = toupper(ch)-'A' ;   /calcolo posizione 0..25 della lettera /
       freq[i]++ ;
    }
    ch = fgetc(f) ;
 }
}
//CONTINUA
```

```
//CONTINUA
fclose( f ) ;
for(i=0; i<LETT; i++)
   printf("%c : %d\n", i+'A', freq[i]) ;
```

Soluzione per righe

```
const int LUN = 200;
char riga[LUN+1];

...

while( fgets( riga, LUN, f ) != NULL )
   for(i=0; riga[i]!=0; i++)
      if(isalpha(riga[i]))
         freq[toupper(riga[i])-'A']++ ;
```

Lettura e scrittura con formato

Qualora sia necessario scrivere file con più campi nella stessa riga, è scomodo ricorrere alle funzioni fputc() e fputs().
È possibile utilizzare una variante della funzione printf(), operante su uno stream aperto in scrittura.

```
int fprintf(FILE *f, stringa-controllo, elem);
```

Dove:

- **f** è il puntatore al file aperto in scrittura o aggiunta
- **stringa-controllo** indica il formato dei dati da scrivere usando gli stessi specificatori validi per printf
- **elem** è la lista dei valori (espressioni) da scrivere

Restituisce il numero di elementi scritti, oppure un valore negativo in caso di errore.

Lettura e scrittura con formato

Qualora sia necessario leggere file con più campi nella stessa riga, è scomodo ricorrere alla funzione fgetc().
È possibile utilizzare una variante della funzione scanf(), operante su uno stream aperto in lettura.

```
int fscantf(FILE *f, stringa-controllo, ind-elem);
```

Dove:

- **f** è il puntatore al file aperto in lettura
- **stringa-controllo** indica il formato dei dati da leggere usando gli stessi specificatori validi per scanf
- **ind-elem** è la lista dei indirizzi delle variabili a cui assegnare i valori letti

Restituisce il numero di elementi letti, oppure un valore negativo in caso di errore.

Lettura e scrittura con formato: esempi

```
#include  <stdio.h>
int main ()
{
  char s[50];
  float b;
  FILE * fp;
  fp = fopen ("a.txt","w+");
  fprintf (fp, "%s %f", " Value of PI",3.1416);
  rewind (fp);            // This function sets the current position indicator associated with fp to
                          //the beginning of the file .
  fscanf (fp, "%s", s);
  fscanf (fp, "%f", &b);
  fclose (fp);
  printf ("Content of file a: %s is %f \n",s,b);
  return 0;
}
```

Lettura e scrittura con formato: esempi

```
#include <stdio.h>
int main ()
{
  FILE * fp;
  int i;
  char fruit[20];
  fp = fopen ("file1.txt","w");
  for (i=0 ; i < 5 ; i++)
  {
    puts ("Please, enter a fruit name: ");
    gets (fruit);
    fprintf (fp, " Fruit %d \n",m, fruit);
  }
  fclose (fp);
  return 0;
}
```

File binari

L'alternativa alle operazioni in modalità testo sui file è quella delle operazioni in **modalità binaria.**

Un file binario è una pura sequenza di byte, senza alcuna strutturazione particolare. È un'astrazione di memorizzazione assolutamente generale, usabile per memorizzare su file informazioni di qualsiasi natura.

I caratteri come '\n' e '\0' che hanno un significato specifico nella modalità testo sono senza conseguenza nella modalità binaria.

Si specifica la modalità binaria usando *b* con le 3 operazioni di base. Quindi

- "wb" per scrivere un file binario
- "rb" per leggere un file binario
- "ab" per aggiungere dati alla fine di un file binario.

File binari

Le funzioni di I/O usate fino a questo punto sono text oriented, infatti lavorano con caratteri e stringhe.
E se volessimo salvare un dato numerico su un file? Potremmo usare fprintf() e lo specificatore %f, ma lo salveremmo come una sequenza di caratteri:

```
double num = 1./3.;
fprintf(fp,"%f", num);
```

Salva *num* come sequenza di 8 caratteri: 0.333333
Utilizzando %.2f lo salverebbe usando 4 caratteri: 0.33
Utilizzando %.12f lo salverebbe usando 14 caratteri: 0.333333333333
Cioè, **cambiando lo specificatore alteriamo la quantità di memoria necessaria per memorizzare il valore**.
La maniera più accurata e consistente di memorizzare un numero è quella di usare lo stesso pattern di bit che il computer usa. In questo caso diciamo che il **dato è memorizzato in modalità binaria**, senza alcuna conversione. A tal proposito si usano le funzioni fread() e fwrite().

Scrittura/lettura su file binari

Per **scrivere** sul file puntato da f, prelevandoli dal vettore vet, n oggetti ciascuno di dimensione size.
Restituisce un intero che rappresenta il numero di oggetti effettivamente scritti (inferiore a n solo in caso di errore):

```
size_t fwrite(const void * vet, size_t size, size_t n, FILE * f);
```

Per **leggere** dal file puntato da f, al più n oggetti collocandoli nel vettore vet, ciascuno di dimensione size.
Restituisce un intero che rappresenta il numero di oggetti effettivamente letti:

```
size_t fread(void * vet, size_t size, size_t n, FILE * f);
```

Esempio: scrittura/lettura su file binari

```
FILE *pfile = NULL;
char *filename = "myfile.bin";
if((pfile=fopen(filename, "wb"))==NULL){
   printf_s("Error opening %s for writing. Program terminated.\n", filename);
   exit(1);
}
```

Possiamo ora scrivere il file myfile.bin:

```
long data[] = {2L, 3L, 4L};
size_t num_items = sizeof(data)/sizeof(long);
size_t wcount = fwrite(data, sizeof(long), num_items, pfile);
```

E poi leggere dal file myfile.bin:

```
size_t wcount = fread(data, sizeof(long), num_items, pfile);
```

Esercizio

Creare un file di numeri double e poi accedere al contenuto del file

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 1000
int main()
{
   double numbers[SIZE];
   double value;
   const char * file = "numbers.dat";
   int i;
   long pos;
   FILE *iofile;

//CONTINUA
```

Esercizio

```
// create a set of double values
for(i = 0; i < SIZE; i++)
   numbers[i] = 100.0 * i + 1.0 / (i + 1);
   // attempt to open file
   if ((iofile = fopen(file, "wb")) == NULL) {
      fprintf(stderr, "Could not open %s for output.\n", file);
      exit(EXIT_FAILURE);
   }

   // write array in binary format to file
   fwrite(numbers, sizeof(double), SIZE, iofile);
   fclose(iofile);
   if ((iofile = fopen(file, "rb")) == NULL){
      fprintf(stderr, "Could not open %s for random access.\n", file);
      exit(EXIT_FAILURE);
   }
```

Esercizio

```
  // read from file
  while ((fread(&value, sizeof(double), 1, iofile)>0))
     printf("value %f\n", value);

  fclose(iofile);
  puts("Bye!");
  return 0;
}
```

