U.D. 8.1

Files

- Concetti generali - File sequenziali e a indice
- I FILE sequenziali in C
- FILE di testo e binari

Fonti:

Antola Dispense del Corso
A. Ciampolini – UNI_BO – Lucidi del corso

I File

- Generalità sui File
- I File Sequenziali
- I File in C
- Apertura e Chiusura
- Manipolazione
- R/W a carattere, a stringa e a blocchi

I File

Il file e` l'unita` logica di memorizzazione dei dati su memoria di massa.

- Consente una ***memorizzazione persistente*** dei dati, ***non limitata*** dalle dimensioni della memoria centrale.
- Generalmente un file e` una sequenza di componenti omogenee (***record logici***).
- I file sono gestiti dal Sistema Operativo. Per ogni versione C esistono funzioni per il trattamento dei file (***Standard Library***) che tengono conto delle funzionalita` del S.O ospite.

In C i file vengono distinti in due categorie:

- ***file di testo***, trattati come sequenze di caratteri. organizzati in linee (ciascuna terminata da '\n')
- ***file binari***, visti come sequenze di bit

File di testo

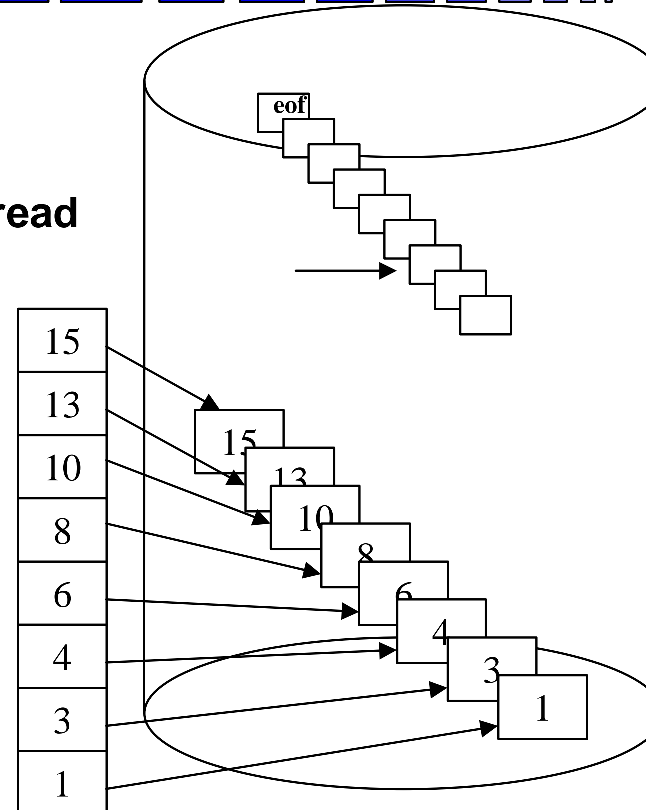
Sono file di caratteri, organizzati in linee.
Ogni linea e` terminata da una marca di fine linea (***newline***, carattere '\n').

```
Egregio Sig. Rossi,▮
            con la presente  ▮
le rendo noto di aver provveduto ▮
...
```

- Il ***record logico*** puo` essere il singolo carattere oppure la singola linea.

File Access Methods

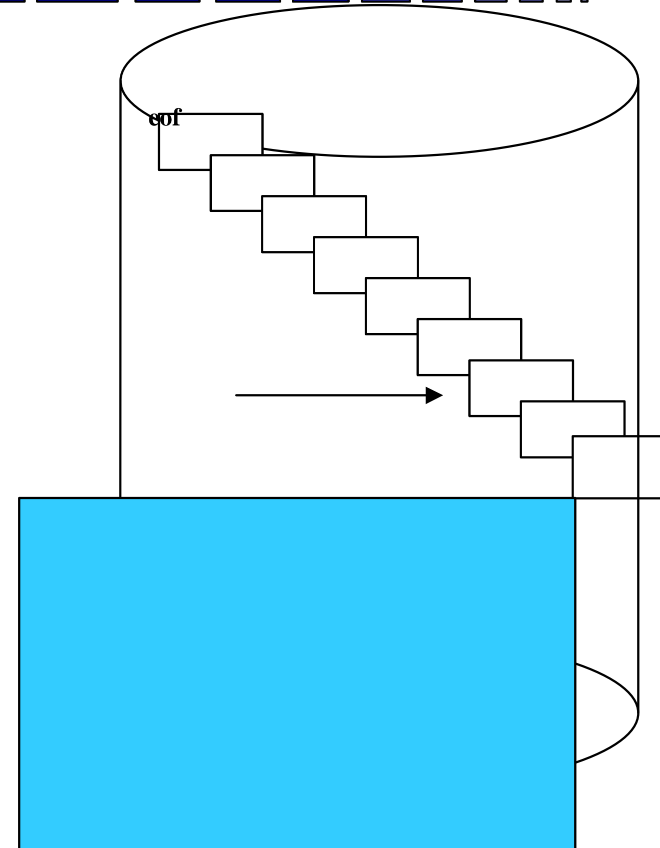
- **Sequential**
  - process from first to last
  - typically for ordered input (read only)
  - can do ordered updates
- **Random**
  - via relative record #
  - Hashing
  - via key field (index)
    - » primary (unique)
    - » secondary
    - » duplicate keys



©M.L.Mick 1994

File Sequenziali

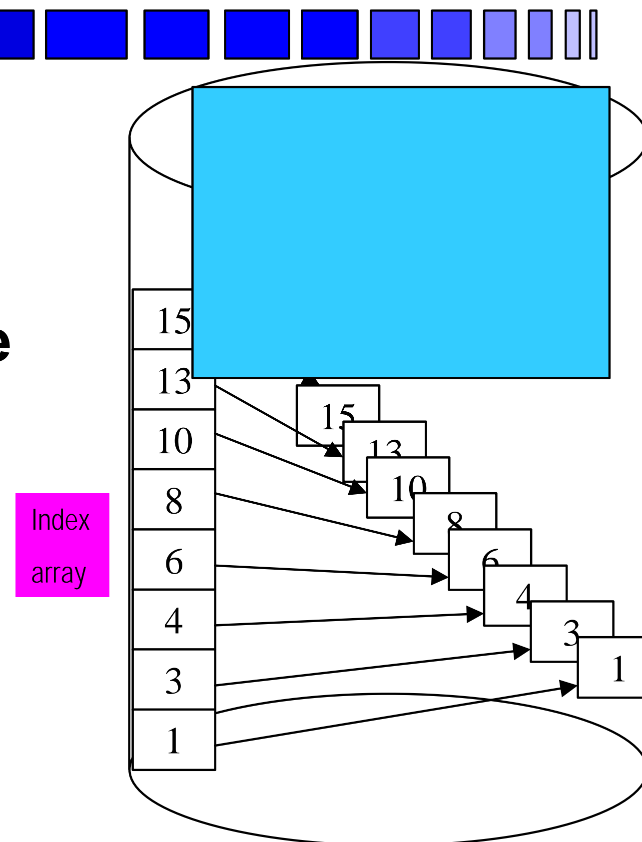
- Tutti gli elementi (record) di ugual tipo
- Scrittura sempre in coda
- lettura dall'inizio sino all'EOF
- Termine determinato dall'End Of File (tappo)



©M.L.Mick 1994

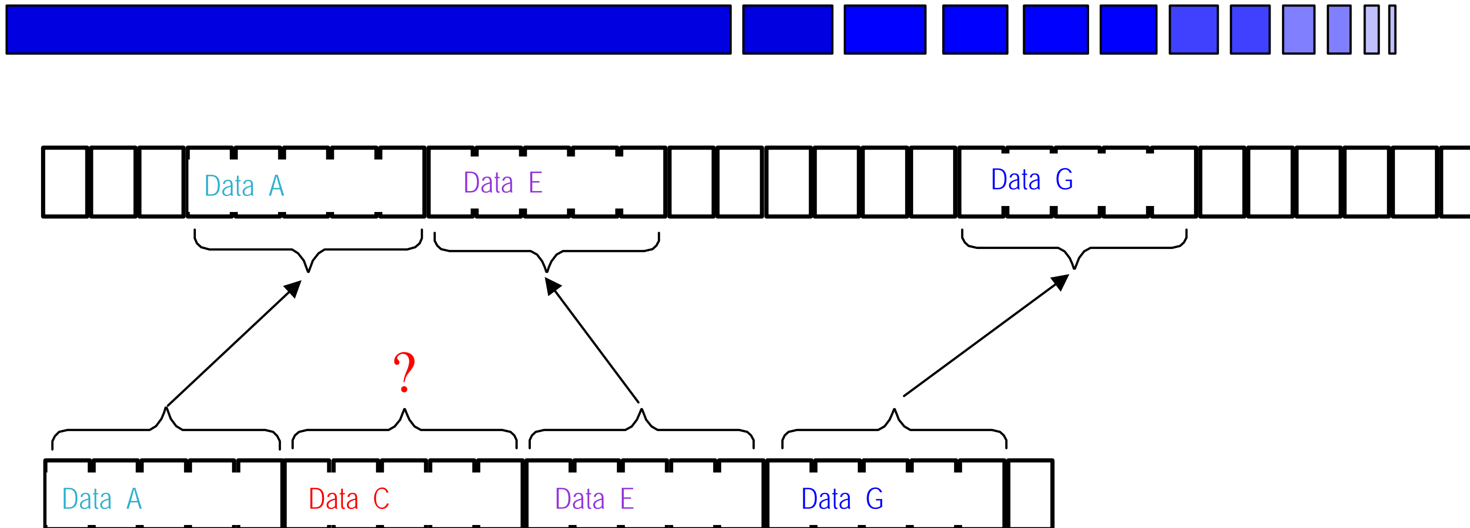
File Random

- Elementi (record) anche diversi (dipende)
- Uso di uno o più INDICI in tabelle esterne
- Non necessario l'EOF
- Lettura e scrittura in qualunque posizione



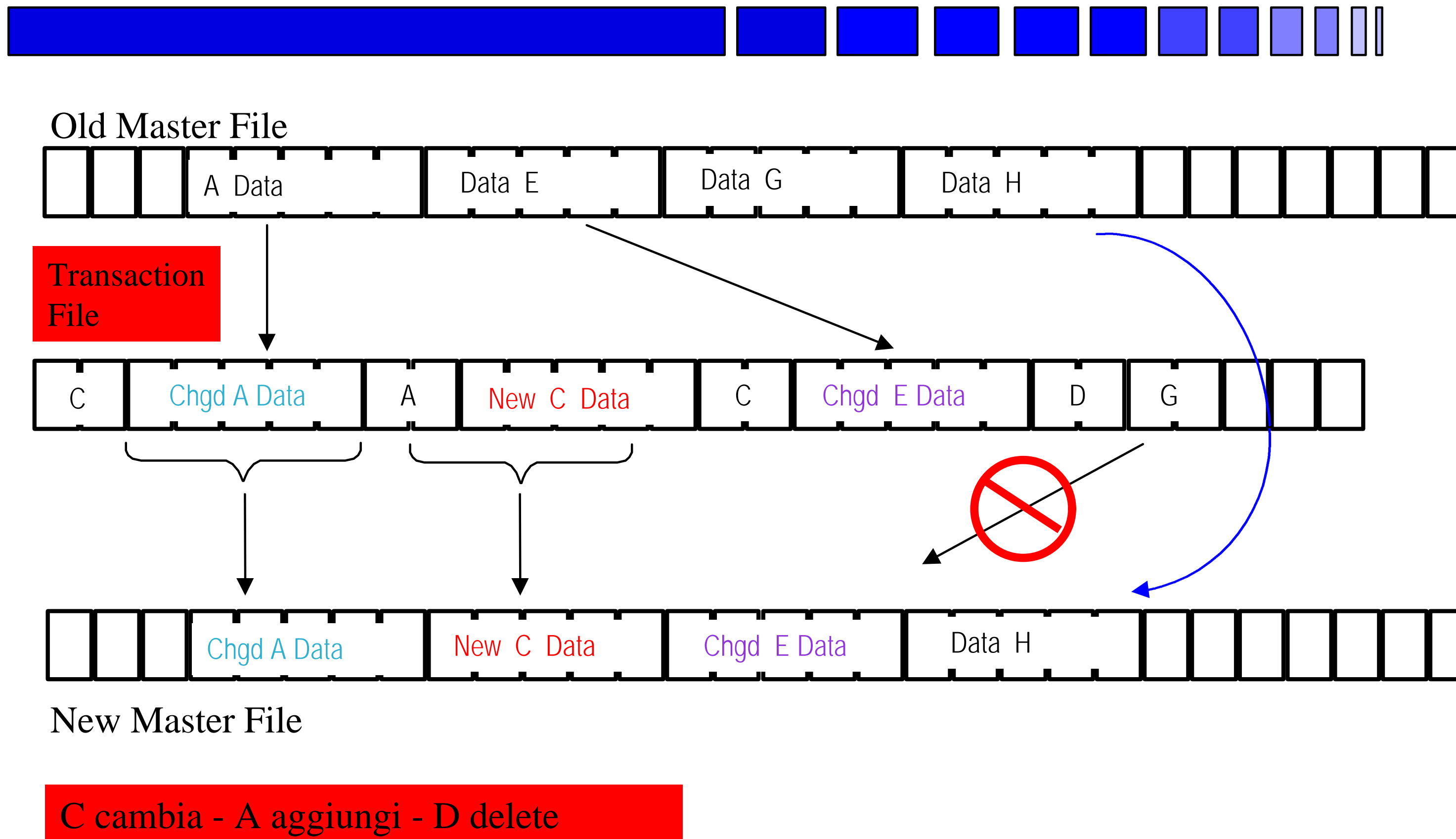
©M.L.Mick 1994

Sequential File Updating



Updating in place is **possible**, but it presents problems!

Sequential File Updating



C cambia - A aggiungi - D delete

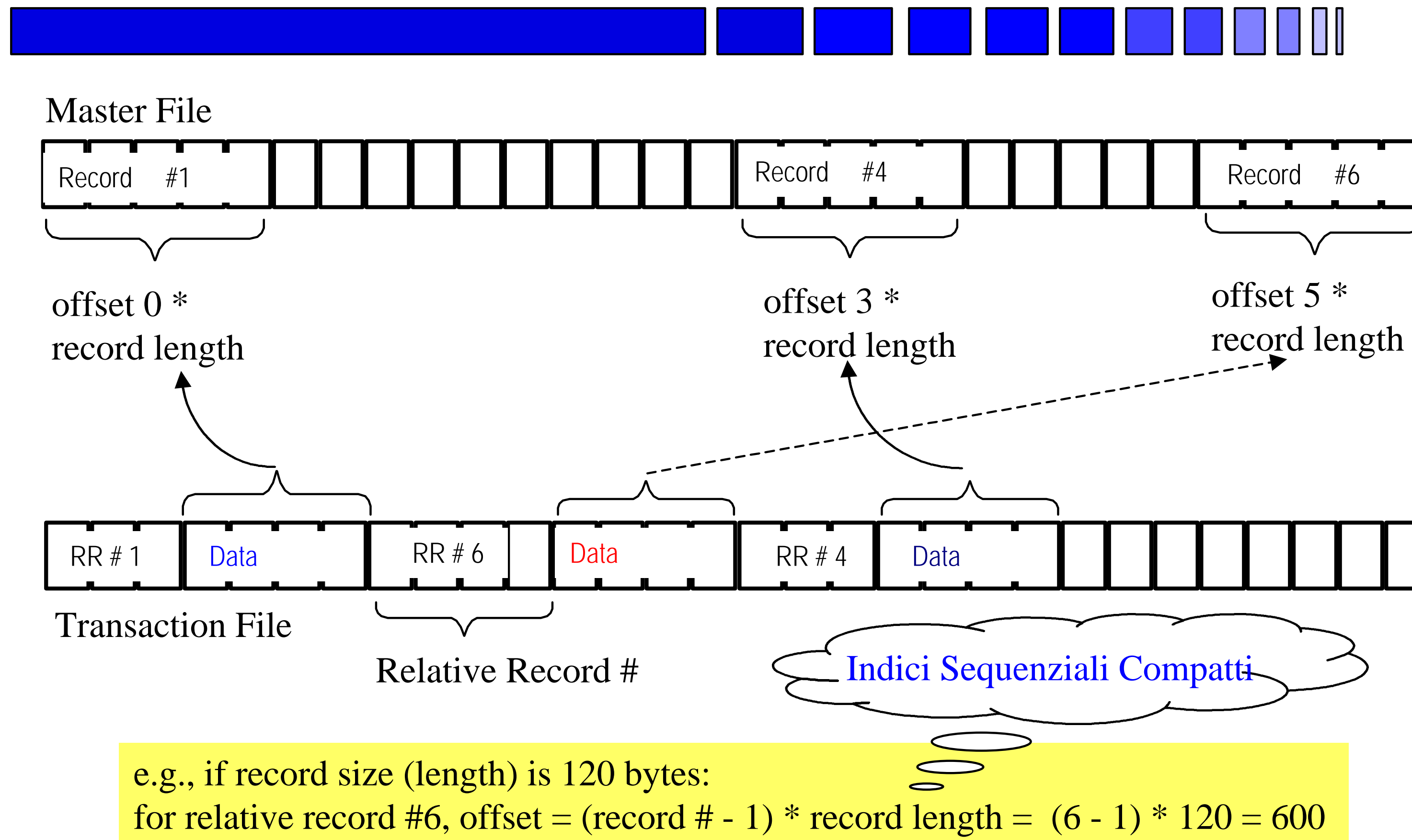
©M.L.Mick 1994

Random Access (1)- Relative Records

- simple form of random access
- Need to know size of “record”
  - Used with only fixed-length records
- Relative record number
  - Used to compute offset into file
  - represents multiple of record length
  - record # “relative” to beginning of file
  - need not be stored as data item in record
- may be used in conjunction with true “key” to access data record

©M.L.Mick 1994

Random Access (1) - Relative Records



e.g., if record size (length) is 120 bytes:
for relative record #6, offset = (record # - 1) * record length = (6 - 1) * 120 = 600

©M.L.Mick 1994

Random Access (2) - Hashing

- What is a "Hash"?
  - operation to transform larger data number into a smaller number, usually a record #
- Why is Hashing used?
  - used for tables in memory & disk files
    - » need efficient search for random access
  - convert key data item into location/number
    - » used to facilitate searching & reduce space
  - variety of functions used to compute address/location
- Collisions and overflows
  - typically # of possible identifiers > # locations
  - want function that minimizes collisions

Hashing - an example

Given an array of structs containing customer name, acct balance, address, etc., and we want to use customer phone number as an index to directly access customer information:

	acctbal	name	address	other ...
cust[0000000]				
cust[0000001]				
cust[9999999]				

e.g., cust[9895555].balance

Using the customer phone number as an index requires 10 million array entries!

If we only have 1000 customers, 99.99% of vector is unused!

Solution: need a vector with a length approximately equal to # of customers, then hash the search key (phone number) into a vector index, mapping a long int in the range of 0 - 9999999 onto an int from 0 - 999.

©M.L.Mick 1994

Hashing - an example

ES. Divide phone number by 10,000 to extract 1st 3 digits. If every phone number is unique in the 1st 3 digits, then our hash function is a **perfect hash function** in that it maps each potential search key onto a unique position (“bucket”) in the hash table.

H(9892375) = 989, and H(4749384) = 474, etc.

More likely, we will get a **collision**, as in H(4642312) = H(4648473) = 464!
A collision occurs when 2 distinct keys has to the same “bucket” in the hash table.
The colliding key are called **synonyms**.

e.g., cust[989].balance

	acctbal	name	address	other ...
cust[000]				
cust[001]				
cust[999]				

©M.L.Mick 1994

Hashing & Collisions

❑ To minimize collisions

- hash function should disperse data uniformly
- each “bucket” should have equal likelihood of being selected
- 1st 3 digits of phone is poor hash function!
  - » would like probability to be .001 (1/# slots)
  - » probability of 000 is 0.0!
  - » if 70% of customers have ph # 474-###, then probability is .7 for 474!
- even reasonably good hash functions need a good collision resolution technique

❑ Overflow

- sometimes a “bucket” has slots for synonym data
- when the # of synonyms a given bucket can accomodate has been reached
- when a bucket has 1 slot (can hold only 1 synonym) collisions and overflow occur simultaneously

©M.L.Mick 1994

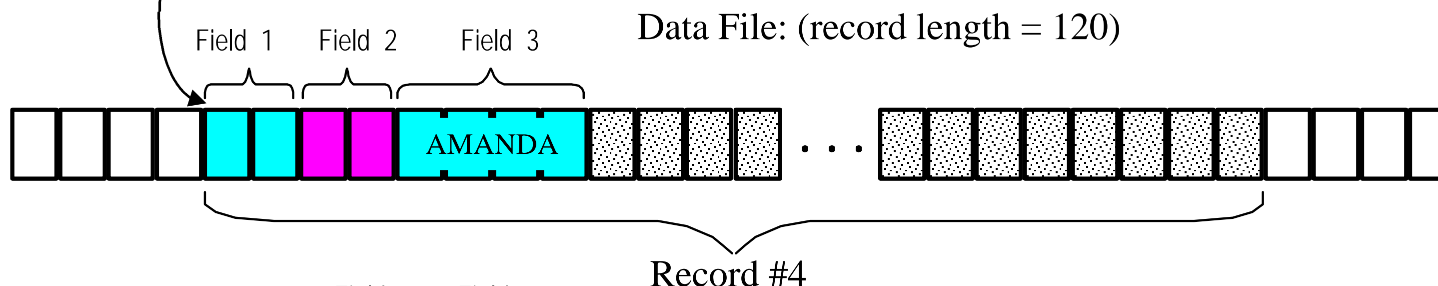
Random Access (3) - Indexed Records

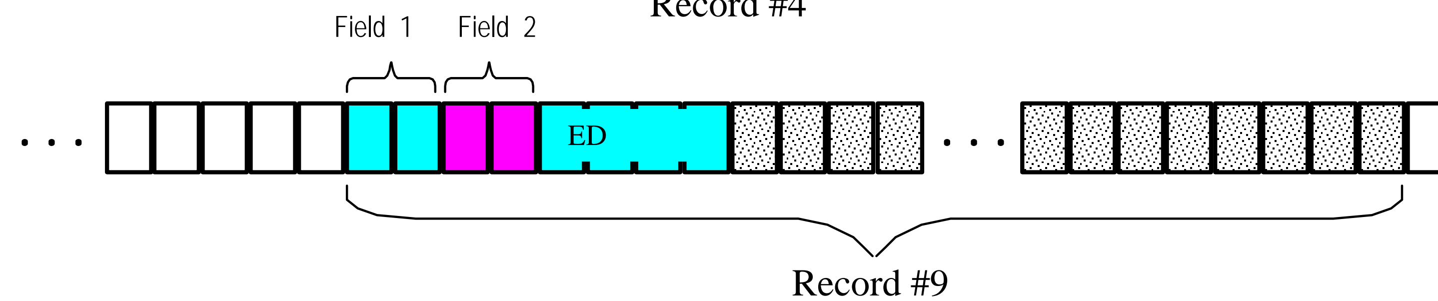
- Locate via key or "index"
  - one or more fields
  - must have unique key
- No "automatic" support in C
  - unlike COBOL, other languages
- Usually stored in separate file
  - MAY store additional keys in same file (MIX)
  - may have separate file for each index (dBase)
- Requires "link" to record
  - usually field giving offset
  - could be relative record number

©M.L.Mick 1994

Random Access (3) - Indexed Records

Index file:

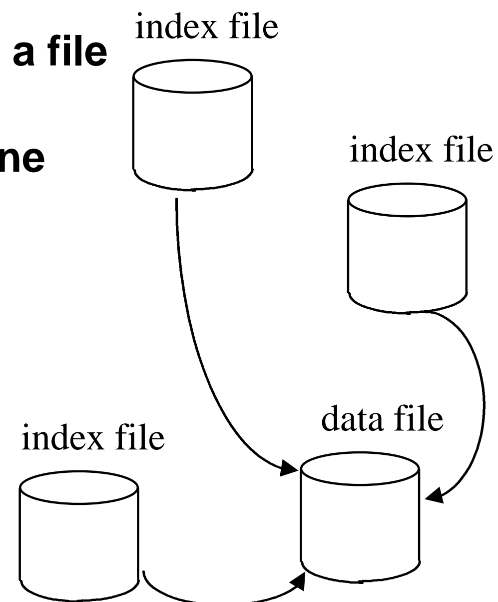




©M.L.Mick 1994

Index Files

- **Separate Index file**
  - **typically index values are stored in a file separate from the data file**
  - **index file may contain more than one index**
- **Multiple index files**
  - **may put each index in its own file**



©M.L.Mick 1994

Primary Key

- Data field(s) used to access file
  - may be only one field
  - may be 2 or more concatenated fields
- Must be unique
  - direct one-to-one ratio of keys to data
  - duplicate keys not allowed
- Indexed sequential file is kept in primary order
  - not entirely necessary
  - requires overflow handling and periodic reorganization (true ISAM) or reservation of space (VSAM)/splitting tracks

©M.L.Mick 1994

Alternate (Secondary) Keys

- Keys other than primary key
  - may include primary key data, but not typical
- Indexed on other field(s)
  - when access in alternate sequence is needed
  - usually in place of having to sort file
  - may concatenate with other fields to make unique
- Duplicate keys may be allowed
  - may be actual duplicate
  - may be partially duplicate
  - may be unique

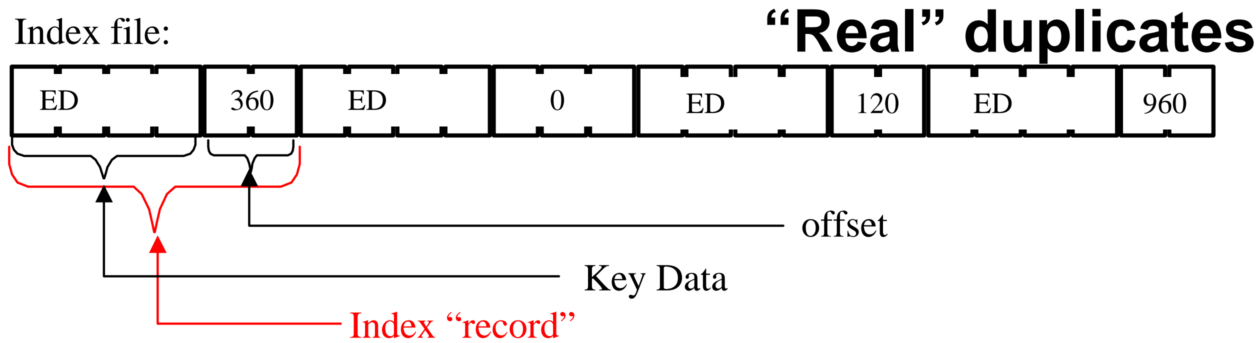
©M.L.Mick 1994

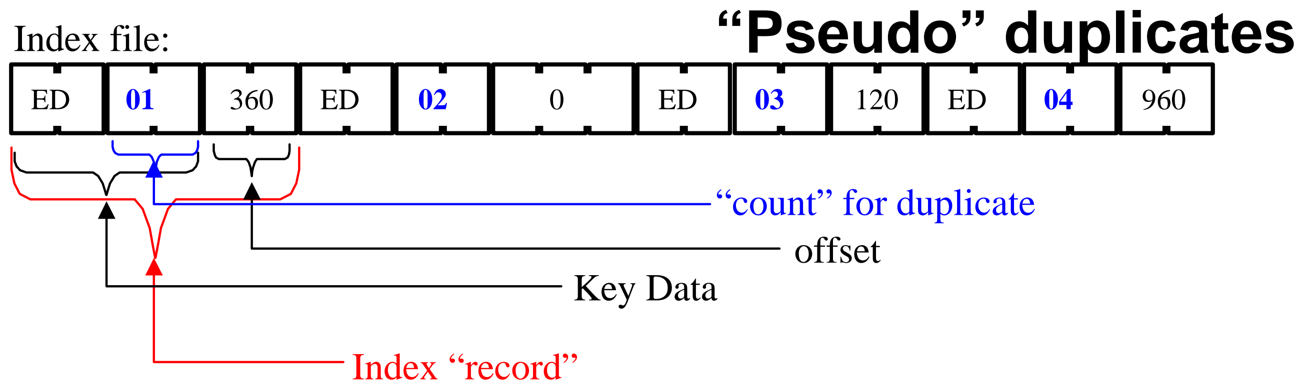
Duplicate Keys

- “Real” duplicates
  - key id portion is valid for more than one record
  - requires selection of desired record
    - » visually by user
    - » by checking other data fields
  - requires extra processing & reads to select
- “Pseudo” duplicates
  - key portion may be real duplicate
  - has additional portion which makes key unique
  - processing easier, more efficient
  - preferred method

©M.L.Mick 1994

Duplicate Keys





©M.L.Mick 1994

Creating & Maintaining Indexes

- Creating index with new data
  - write index file concurrent with data file
  - must have code to handle duplicates, lack of space, etc. & be able to “back out” transaction
  - must consider power loss, concurrent access
- Creating index for existing data
  - sequentially process the data file
  - write an index record for each data record
- Correcting corrupt indexes
  - recreate the index as for existing data
  - relatively fast process

©M.L.Mick 1994

Creating & Maintaining Indexes

- How is index accessed?
  - depends on usage of data file
  - may be updated on disk or in memory
- Update on disk
  - if interactive program
  - if multiple concurrent users
  - if not low potential for outage, corruption
- Update in memory
  - if speed absolute requirement
  - if single user
  - if low outage, corruption potential

©M.L.Mick 1994

Dynamic Data Structures

- Created with malloc()
- Use to create data structure dynamically
  - linked lists, trees, etc.
- Storage released with free()
- Decreased access time over disk
- Increased risk of loss of updates
- Not advisable in shared environment

©M.L.Mick 1994

Using DDS for Indexes

- Store keys in linked list, tree, etc.
- For existing index
  - read entire index into memory
  - add, change, delete as required
  - write entire index to disk
- Can use to store partial index
  - if too large for memory
  - read portion into memory; access keys
  - load other portion(s) as required - paging!
  - can also use for index to index!

I File in C

Faremo SOLO i File Sequenziali ☺

FILE E PROGRAMMI C

In C l'accesso a file da parte di un programma avviene tramite le funzioni disponibili nella Standard Library (come tutta l'ingresso e uscita). In particolare, le funzioni su file sono definite in `stdio.h`.

Le funzioni di libreria della `stdio.h` interagiscono con il sistema operativo per consentire l'accesso a file. Questo vuol dire che al loro interno le **funzioni di libreria** contengono delle chiamate a **funzioni di sistema operativo**.

programma C → funzioni di stdio.h → funzioni di Sistema Operativo

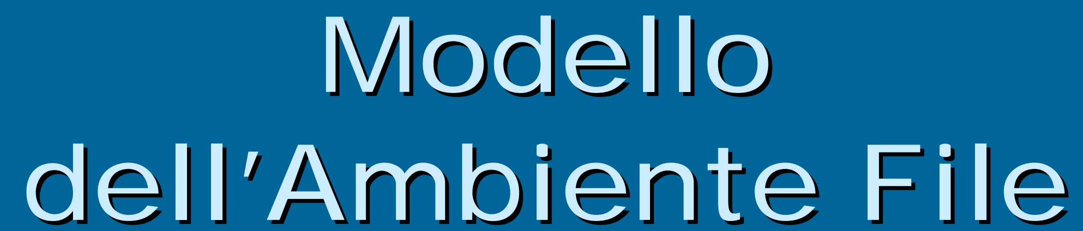
In ambiente C per **utilizzare un file** all'interno di un programma è necessario:

1. aprire un «flusso di comunicazione» (aprire il file)
2. accedere a file in lettura e/o scrittura
3. chiudere il «flusso di comunicazione» (chiudere il file)

Tipi di flussi di comunicazione («tipi» di file)

- **binari**: sequenza di byte
- **testuali**: sequenza di caratteri: ciascun byte è la codifica ASCII di una carattere alfanumerico (in particolare, alcuni caratteri possono essere *non printable* e quindi di controllo)

Se un flusso di comunicazione viene aperto in un certo modo (binario o testuale), le operazioni sul file corrispondente devono essere effettuate in modo congruente, e l'utilizzo successivo deve tener conto del tipo flusso.


Modello
dell'Ambiente File

Files in C

- A file descriptor, or file pointer, is an abstraction in C. The library function fopen returns a file pointer, which you can then use in your program. Afterward, you must “fclose” the file pointer.

Declare a variable to be of type **`FILE *,`** for example:

```
FILE * fd;

fd = fopen(“filename”, “mode”);

if (NULL == fd) /* if fopen fails, it returns NULL */
{
   printf(“error message”);
   exit(EXIT_FAILURE);
}

reads and writes happens here.

fclose(fd);
```

APERTURA E CHIUSURA DI UN FLUSSO DI COMUNICAZIONE (FILE) IN C

L'insieme dei file aperti da un programma in ambiente C può essere rappresentato da una struttura dati gestita dal Sistema Operativo. Questa stuttura dati può essere pensata come

FILE TabFileAperti[NumMaxFile]

cioè come un array (tabella) di NumMaxFile elementi dove ciascun elemento è di **tipo** FILE.

FILE è da intendersi come un tipo strutturato che rappresenta il tipo del **descrittore del file**.I campi più significativi sono:

- nome del file
- modalità di utilizzo
- posizione corrente sul file (prossimo byte a cui accedere)
- indicatore di fine file
- indicatore di errore

In `stdio.h` sono definiti tra gli altri:

- l'identificatore simbolico FILE, che viene usato da un programma per indicare il tipo associato (al descrittore di) file
- l'identificatore simbolico EOF per indicare la fine del file
- l'identificatore simbolico NULL (***spostato in*** **`stddef.h`**)
- i prototipi di tutte le funzioni per accesso a file

©M.L.Mick 1994

File Control Block (DOS)

```
typedef struct
{
        short           level;      /* fill/empty level of buffer    */
        unsigned        flags;      /* File status flags             */
        char            fd;         /* File descriptor               */
        unsigned char   hold;       /* Ungetc char if no buffer      */
        short           bsize;      /* Buffer size                   */
        unsigned char   *buffer;    /* Data transfer buffer          */
        unsigned char   *curp;      /* Current active pointer        */
        unsigned        istemp;     /* Temporary file indicator      */
        short           token;      /* Used for validity checking */
}       FILE;                       /* This is the FILE object       */
```

1. APERTURA DI UN FILE

```
FILE   * fopen(<nome_file>, <modo>)
```

- è una funzione che riceve in ingresso il nome del file da aprire (specificato secondo le convenzioni del SO) e il modo con cui si vuole aprire il file.

- restituisce il puntatore al descrittore di file creato

FUNZIONAMENTO:
alla chiamata, il SO (che viene attivato in modo opportuno) crea un nuovo elemento nella struttura dati che costituisce la tabella dei file aperti (crea un nuovo descrittore di file), inizializza i campi del descrittore e restituisce il puntatore a tale elemento. Quindi:

- è il SO che gestisce la struttura dati e i contenuti di tutti gli elementi

- a livello di programma C un file viene visto come un puntatore (è rappresentato da un puntatore). Un programma C che utilizza un file deve dichiarare una variabile puntatore, ad esempio

  ```
  FILE   * fp;
  ```

  e far riferimento al file nel suo complesso tramite tale variabile (**fp**).

All’inizio dell’esecuzione di un programma C vengono automaticamente aperti **3 (4) flussi di comunicazione standard** rappresentati da 3 (4) «variabili implicite» di tipo puntatore a FILE

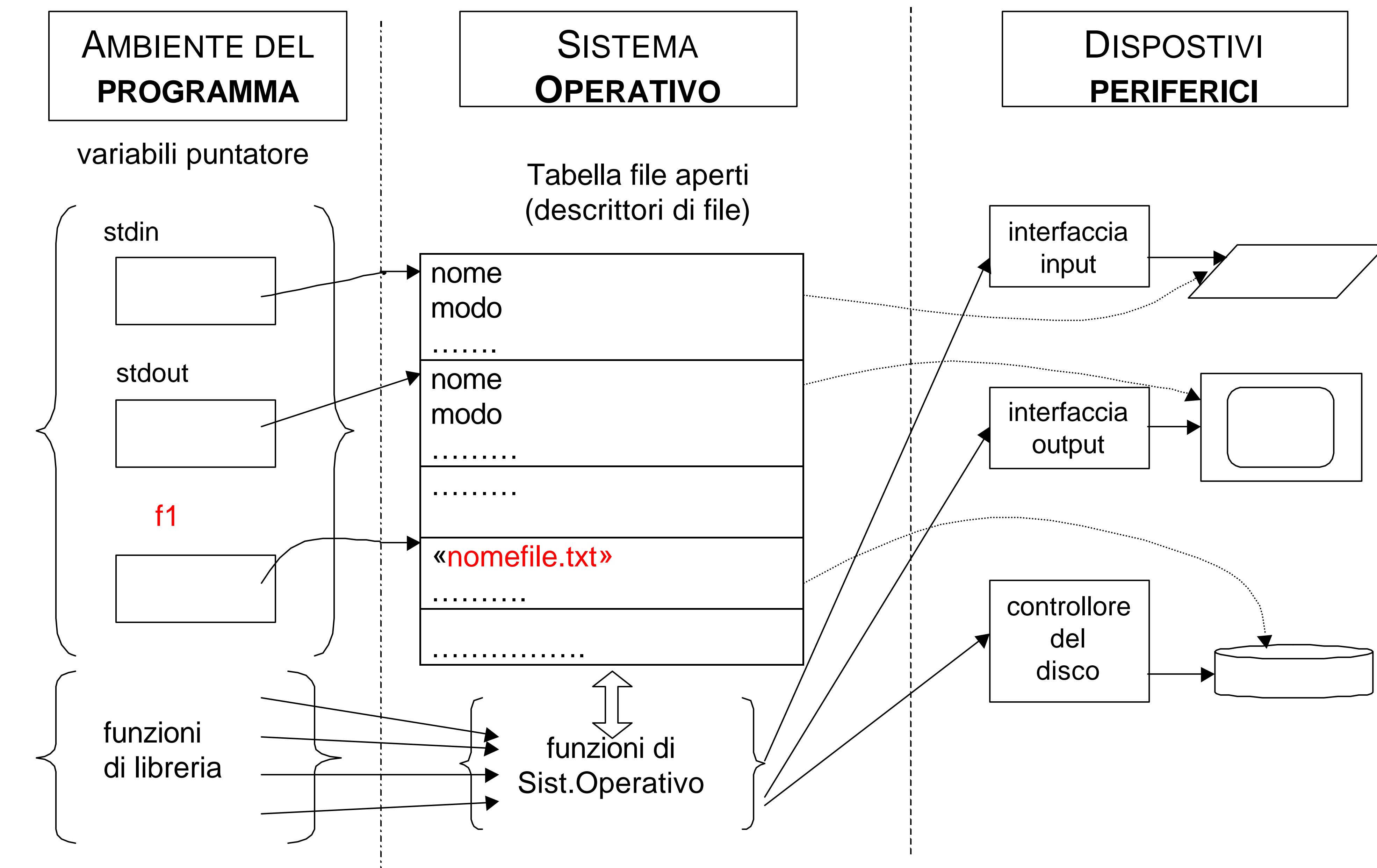
- **stdin** associato al «file» che rappresenta il dispositivo di ingresso standard (tastiera)
- **stdout** associato al «file» che rappresenta il dispositivo di uscita standard (video)
- **stderr** associato al «file» che rappresenta il dispositivo di uscita standard (video)
- **stdprn** associato al «file» che rappresenta il dispositivo di uscita su carta standard (stampante: porta di uscita lpt1: o prn)

©M.L.Mick 1994

Opening a File

- **each file needs a pointer**
  - **program will reference each file by a pointer**
  - **declare a pointer to FILE**
  - **FILE *fptr;**
  - **FILE *in_file, *out_file;**
  - **(stdin, stdout, and stderr are all file pointers!)**

- **use fopen() with file name and mode**
  - **returns pointer of type FILE**
  - **creates File Control Block (FCB)**
    - an area of memory where details of data transfer are kept
    - a struct of type FILE



AMBIENTE DEL PROGRAMMA
SISTEMA OPERATIVO
DISPOSTIVI PERIFERICI
variabili puntatore
stdin
stdout
f1
Tabella file aperti
(descrittori di file)
nome
modo
…….
nome
modo
………
………
«nomefile.txt»
……….
…………….
funzioni
di libreria
funzioni di
Sist.Operativo
interfaccia
input
interfaccia
output
controllore
del
disco

Apertura File

```
FILE    * fopen(<nome_file>, <modo>)
```

`<nome_file>`

è una stringa di caratteri (racchiusa tra « e «) che rappresenta il nome (pathname) del file specificato secondo le convenzioni del SO («anna.txt»).

`<modo>`

è una stringa di caratteri (racchiusa tra « e «) che rappresenta il modo con cui si intende aprire il file: lettura (r), scrittura inizio file (w), scrittura fine file (a), lettura e scrittura, .. e se il file è binario (b) o testuale.

Se il file esiste, lo apre; se il file non esiste lo crea e lo apre.

Restituisce:

- il puntatore al descrittore, se l'operazione è andata a buon fine
- il valore NULL se l'operazione non è andata a buon fine (apertura in lettura di un file che non esiste,......)

Modalità di apertura dei file di testo

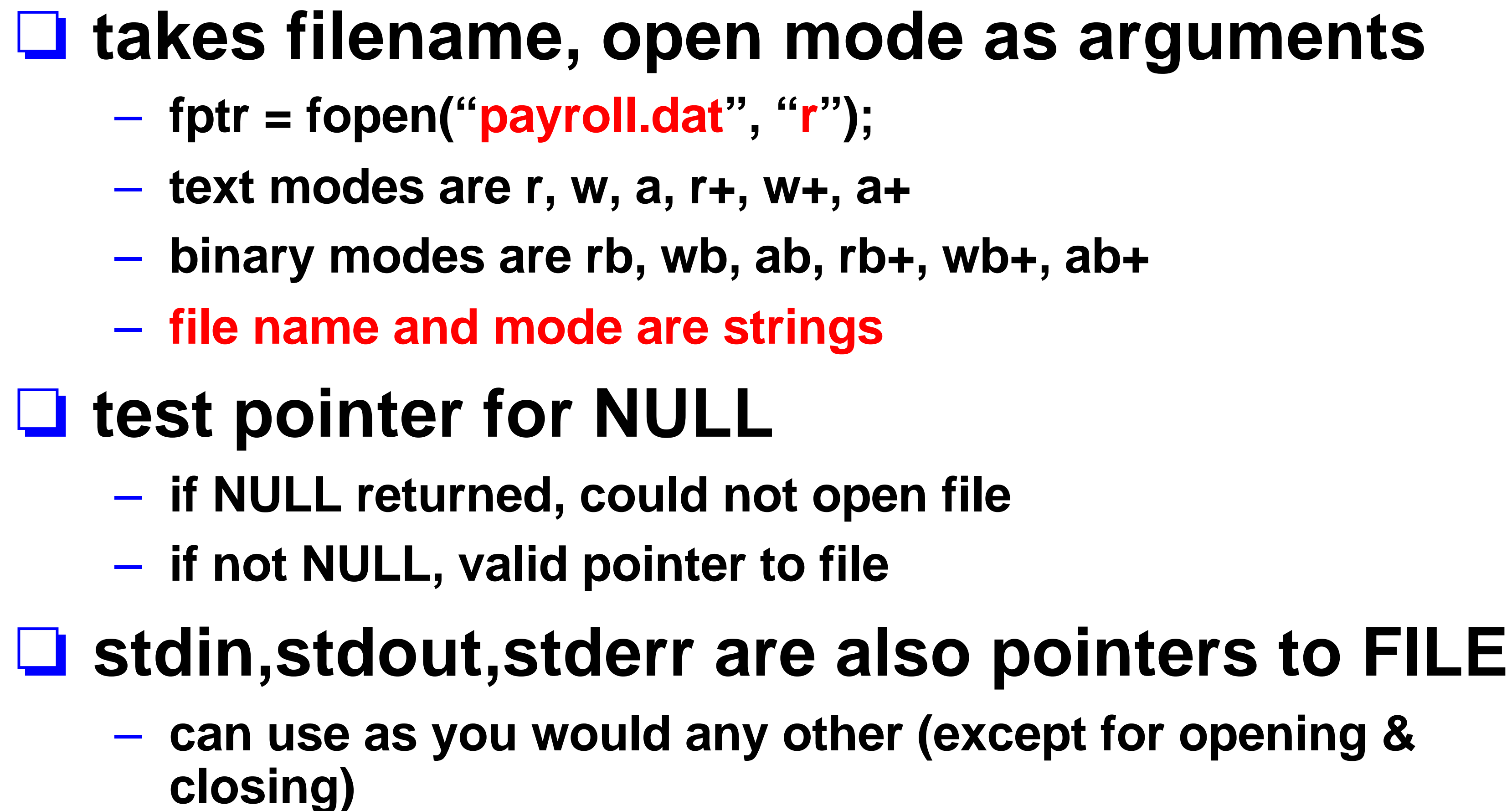
r	Apre un file di testo esistente in modalità lettura, posizionandosi all'inizio del file.
w	Crea un nuovo file di testo e lo apre in modalità scrittura. posizionandosi all'inizio del file (se il file esiste, i dati precedenti vengono eliminati).
a	Apre un file di testo esistente in modalità append (accoda): la scrittura può avvenire solo a partire dalla fine del file; anche se l'indicatore di posizione nel file viene modificato volontariamente. la scrittura avviene comunque alla fine del file.
r+	Apre un file di testo esistente In modalità lettura e scrittura. posizionandosi all'inizio del file.
w+	Crea un nuovo file di testo e lo apre in modalità lettura e scrittura (se il file esiste. i dati precedenti vengono eliminati).
a+	Apre un file di testo esistente o ne crea uno nuovo in modalità append; la lettura può avvenire in una posizione qualsiasi del file, la scrittura solo alla fine.

Proprietà dei file e flussi delle modalità operative di *fopen()*

	r	***w***	***a***	***r+***	***w+***	***a+***
Il file deve esistere prima dell'apertura	✔			✔		
Il vecchio file viene azzerato		✔			✔	
Il flusso può essere letto	✔			✔	✔	✔
Il flusso può essere scritto		✔	✔	✔	✔	✔
Il flusso può essere scritto sola alla fine			✔			✔

©M.L.Mick 1994

Using fopen()



- takes filename, open mode as arguments
  - fptr = fopen("payroll.dat", "r");
  - text modes are r, w, a, r+, w+, a+
  - binary modes are rb, wb, ab, rb+, wb+, ab+
  - file name and mode are strings
- test pointer for NULL
  - if NULL returned, could not open file
  - if not NULL, valid pointer to file
- stdin,stdout,stderr are also pointers to FILE
  - can use as you would any other (except for opening & closing)

©M.L.Mick 1994

File Names

- As used with fopen()
  - filename is a string
  - open mode is a string
- can use hardcoded string, as in

```
fptr = fopen("payroll.dat", "r");
```

- can use variable string, as in

```
char filename[20] = "payroll.dat";
char open_mode[] = "r";
fptr = fopen(filename, open_mode);
```

  - what does this allow?

©M.L.Mick 1994

Examples

- **to open a disk file called “mydata”, read-only**

  fptr = fopen(“mydata”, “r”)

- **to open/create an output file “report.txt”**

  fptr = fopen(“report.txt”, “w”)

- **to open a disk file, “file.dat”, to read binary data**

  fptr = fopen(“file.dat”, “rb”)

- **to open the standard output device**

  you do not open stdout, just use it!

©M.L.Mick 1994

Examples

- to open a binary disk file called “master.dat”, to read old data and write new data, starting at the beginning

```
fptr = fopen(“master.dat”, “rb+”)
```

- to open a disk file, “update.dat”, to read old data and write new data, starting at the end

```
fptr = fopen(“update.dat”, “a+”)
```

- to open/create a disk file to read and write new data, with user input file name & extension

```
strcat(file_name,”.”);           /* file_name is array */
strcat(file_name,file_ext);      /* file_ext is array */
fptr = fopen(file_name, “w+”)
```

Uso dei File

2. CHIUSURA DI UN FILE

```
int  fclose(FILE *fp)
```

- è una funzione che riceve in ingresso il puntatore del file da chiudere.
- restituisce il valore 0 se l'operazione è andata a buon fine, il valore EOF se l'operazione non è andata a buon fine.

FUNZIONAMENTO:
alla chiamata, il SO (che viene attivato in modo opportuno) chiude il file referenziato dal puntatore passato come parametro (assegna al puntatore il valore NULL) e considera disponibile l'elemento che conteneva il descrittore del file chiuso.

3. ALTRE FUNZIONI DI GESTIONE FILE

remove

rename

ridirezione (freopen)

4. GESTIONE DEGLI ERRORI

`int feof(FILE *fp)` **macro**
restituisce 0 se non si è incontrata la fine del file, !=0 se end of file nell'ultima lettura

`int ferror(FILE *fp)` **macro**
restituisce 0 se non si è verificato errore, !=0 se si è verificato errore

©M.L.Mick 1994

Closing a File

- do not need to close stdin, stdout, stderr
  - opened automatically, closed automatically
- if you open it, you close it!
  - could cause problems by terminating a program without closing files
- to close a file:
  - use fclose() with each file pointer

    ```
    fclose(fptr);
    ```