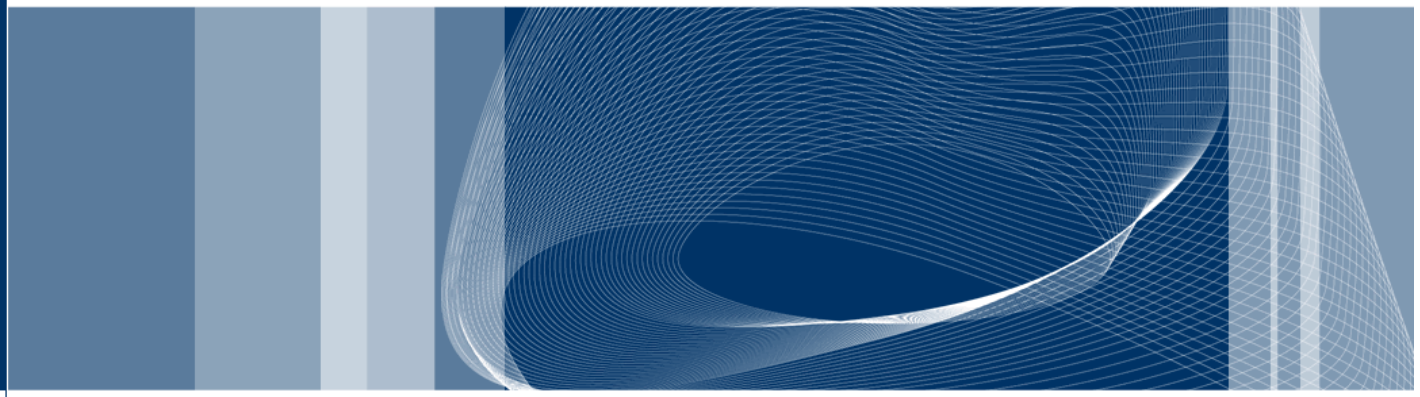
POLITECNICO DI MILANO



Gestione dei file in C

Fondamenti di Informatica

Daniele Loiacono

Che cos’è un file e a cosa serve?

Memoria di massa vs memoria centrale

- La memoria di massa (disco fisso) è un dispositivo di memorizzazione generalmente presente in un calcolatore.
- Si differenzia dalla memoria centrale nei seguenti aspetti:
  - dimensioni e costo
  - persistenza
  - prestazioni
- I **file** sono lo strumento che consente di memorizzare delle informazioni nella memoria di massa del calcolatore.

File come flussi

- Un flusso (*stream*) di dati è un meccanismo i cui principali usi sono:
  - lo scambio di dati con le periferiche (es. terminale, stampanti, etc.);
  - **lettura e scrittura dei dati memorizzati sul disco fisso.**
- In C i file sono quindi rappresentati come flussi e vengono fornite diverse funzionalità nella libreria stdio.h per la loro gestione:
  - il flusso di informazione può essere testuale (flusso di caratteri) o binario (flusso di byte);
  - consente di avere file ad *accesso sequenziale* e ad *accesso diretto* (o *casuale*).

Lavorare con i file in C

Apertura e chiusura di un file

- Per poter utilizzare un file in un programma C è prima necessario *aprirlo*:

```
FILE *fopen(nomefile, modalità)
```

  - `nomefile` è il nome del file da aprire (incluso il path)
  - `modalità` è una stringa che serve a specificare come verrà utilizzato il file
  - la funzione ritorna un puntatore di tipo `FILE`, una particolare struttura dati che consente di tenere traccia dell'ultima posizione letta/scritta nel file, del suo stato, etc; se l'operazione non ha successo, viene ritornato NULL.

Apertura e chiusura di un file (2)

- Modalità di apertura di un file:

Modalità	Significato
"r"	Apre un file testuale in lettura
"w"	Crea un file testuale in scrittura
"a"	Apre o crea un file testuale e si posiziona alla fine del file
"rb"	Apre un file binario in lettura
"wb"	Crea un file binario in scrittura
"ab"	Apre o crea un file binario e si posiziona alla fine del file
"r+"	Apre un file testuale in lettura/scrittura
"w+"	Crea un file testuale in lettura/scrittura
"rb+"	Apre un file binario in lettura/scrittura
"wb+"	Crea un file binario in lettura/scrittura

Apertura e chiusura di un file (3)

- Una volta utilizzato un file in un programma, è necessario chiuderlo:

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

  - `fp` è il puntatore al file da chiudere
  - la funzione ritorna 0 se l'operazione viene eseguita correttamente, altrimenti la funzione restituisce un valore speciale `EOF` definito in `stdio.h` (che viene generalmente utilizzato come carattere terminatore di un file).

Operazioni di lettura e scrittura

- Le operazioni di lettura e scrittura su file possono essere effettuate in quattro modi diversi:
  - precisando il formato dei dati in ingresso e in uscita
  - accedendo ai dati carattere per carattere
  - linea per linea
  - blocco per blocco
- Generalmente si adotta l'accesso linea per linea nel caso di flussi di testo e l'accesso carattere per carattere o blocco per blocco in presenza di flussi binari

Lettura e scrittura formattata

- Le funzioni **fprintf** e **fscanf** consentono operazioni formattate analoghe a quelle di scanf e printf
- Restituiscono il numero degli elementi effettivamente letti o stampati o restituiscono un numero negativo in caso di errore

```
int  fprintf(fp, str_controllo, …)
int  fscanf(fp, str_controllo, …)
```

  - `fp` è il puntatore al file dal quale leggere o dove scrivere, gli argomenti successivi sono analoghi a quelli usati con scanf e printf
  - le funzioni restituiscono l'effettivo numero di elementi letti (fscanf) o di caratteri scritti (fprintf)

Lettura e scrittura di caratteri

- Lettura di un carattere

```
int fgetc (FILE *fp)
```

  - riceve come argomento il file da cui leggere
  - restituisce il codice del carattere letto o `EOF` in caso di errore

- Scrittura di un carattere

```
int fputc (int c, FILE *fp)
```

  - riceve come argomenti il codice del carattere da scrivere e il file su cui scrivere
  - restituisce il codice del carattere scritto o `EOF` in caso di errore

Esempio

```
#include <stdio.h>

int main()
{
  FILE *fp;
  char c;
  if ((fp = fopen(”car.txt", "r")) != NULL) {
    while ((c = fgetc(fp)) != EOF)
       printf(“***%c***\n”,c);
    fclose(fp);
  }
  else
    printf("Il file non può essere aperto\n");
  return 0;
}
```

Lettura e scrittura di stringhe

- Lettura di una stringa

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

  - legge dal file puntato da `fp` fino a n-1 caratteri (si interrompe nel caso raggiunga prima il carattere `\n` o la fine del file) e mette i caratteri letti nella stringa `s` (aggiungendo il carattere terminatore `\0`);
  - La funzione restituisce l'indirizzo di `s` se ha successo o `NULL` in caso di errore.

- Scrittura di una stringa

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

  - scrive nel file puntato da `fp` il contenuto della stringa `s` (fino a raggiungere il carattere terminatore `\0`);
  - restituisce 0 in caso l'operazione abbia avuto successo.

Esempio

```
#include <stdio.h>
#include <strings.h>
#define MAXLINE 1000

int copiaselettiva(char refstr[], char in[], char out[]) {
        char    line[MAXLINE];
        FILE    *fin, *fout;

        if ((fin = fopen(in, "r")) == NULL)
                return -1;
        if ((fout = fopen(out, "w")) == NULL) {
                fclose(fin);
                return -1;
        }
        while (fgets(line,MAXLINE,fin) != NULL)
                if (strstr (line,refstr) != NULL)
                        fputs(line,fout);
        fclose(fin);
        fclose(fout);
        return 0;
}
```

Operazioni di gestione degli errori

- `int ferror(FILE *fp)`
  - Controlla se è stato commesso un errore nella precedente operazione di lettura o scrittura
  - Restituisce 0 se nessun errore è stato commesso, un valore diverso da 0 in caso contrario
- `int feof(FILE *fp)`
  - Controlla se è stata raggiunta la fine del file nella precedente operazione di lettura o scrittura
  - Restituisce 0 se la condizione di fine file non è stata raggiunta, un valore diverso da 0 in caso contrario

Lettura e scrittura per blocchi

- `int fread(ptr, dim, num, FILE *fp)`
  - legge al più `num*dim` byte di dati binari o testuali dal file cui fa riferimento `fp` e li memorizza nel vettore identificato da ptr (la lettura termina prima se viene raggiunta la fine del file o si verifica un errore);
  - la funzione ritorna il numero di elementi effettivamente letti.
- `int fwrite(ptr, dim, num, FILE *fp)`
  - scrive `num*dim` byte di dati binari o testuali sul file cui fa riferimento `fp` prelevandoli dal vettore identificato da `ptr`;
  - la funzione ritorna il numero di elementi effettivamente scritti.

Esempio

```
typedef struct
{
        float x;
        float y;
} punto;

int carica(FILE *fp, punto p[])
{
        int n=0;
        while (!feof(fp))
        {
                if (fread(&p[n],sizeof(punto),1,fp)==1)
                        n++;
                else
                        break;
        }
        return n;
}
```

Accesso diretto

- Spostare l'indicatore di posizione corrente di un file:
  `int fseek(FILE *fp, long offset, int refpoint)`
  - Lo spostamento `offset` (in byte) può essere positivo o negativo e si riferisce a `refpoint`;
  - `refpoint` può assumere tre diversi valori
    - `SEEK_SET` indica l'inizio del file
    - `SEEK_CUR` indica la posizione corrente
    - `SEEK_END` indica la fine del file
  - la funzione fseek restituisce zero se l'operazione ha avuto successo
- *Riavvolgere* un file
  `rewind(FILE *fp)`
  - equivale a `fseek` con `offset=0` e `refpoint=SEEK_SET`
- `long` **`ftell`**`(FILE *fp)`
  - restituisce il valore corrente dell'indicatore di posizione del file specificato: pari al numero di byte dall'inizio del file nel caso di un file binario, mentre un valore dipendente dal sistema per i file testuali

File e sistema operativo

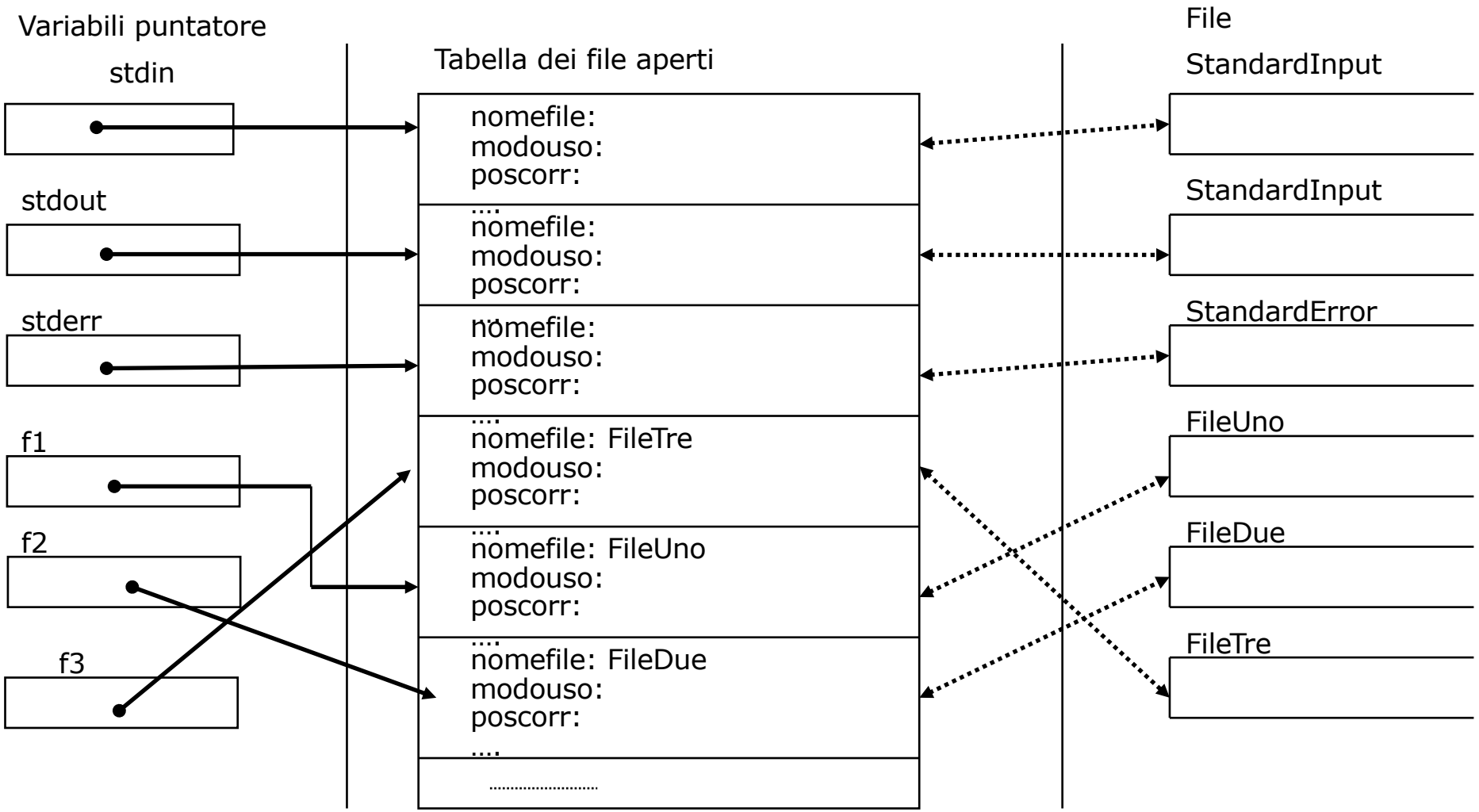
File e programmi C

- Per utilizzare un file in C è necessario prima aprire un flusso di comunicazione per associare il programma al file.
- Il sistema operativo tiene traccia di tutti i file aperti da ogni programma nella **tabella dei file aperti.**
- Per ogni file aperto, si tiene traccia di:
  - modalità di utilizzo del file (lettura, scrittura, o lettura e scrittura);
  - posizione corrente sul file (punta al prossimo byte da leggere o scrivere);
  - indicatore di errore;
  - indicatore di end-of-file (eof).
- Una volta finito di utilizzare un file in un programma, è possibile chiuderlo.

Tabella file aperti e flussi standard

- Per ogni programma, il sistema operativo tiene aggiornata una *tabella dei file aperti.*
- I puntatori FILE restituiti dalla fopen, sono dei riferimenti alla tabella dei file aperti.
- Tre flussi standard vengono automaticamente aperti quando inizia l'esecuzione di un programma: stdin, stdout, e stderr.
- Normalmente questi tre flussi rappresentano:
  - il video del terminale (stdout e stderr);
  - la tastiera del terminale (stdin).
- printf e scanf utilizzano questi flussi standard.

File e sistema operativo



File e terminale non sono l'unico modo di fornire input ad un programma...

Di cosa stiamo parlando?

- Come lanciamo il compilatore gcc?

  gcc sorgente.c –o eseguibile

- gcc è il nome dell'eseguibile che stiamo lanciando (il compilatre gcc), quindi cosa sono sorgente.c –o eseguibile?

- Sono gli argomenti che vengono forniti in input al programma gcc

Programmi con argomenti in C

- In C, per creare un eseguibile a cui sia possibile passare argomenti è necessario definire il prototipo del main così:

```
int main(int argc, char *argv[])
```

  - **argc**, contiene il numero di argomenti con cui il programma viene lanciato (incluso il nome stesso del programma)
  - **argv**, è un array di stringhe che contiene tutti gli argomenti con cui il programma viene lanciato (argv[0] è il nome completo dell'eseguibile lanciato con eventuale percorso)

Utilizzare gli argomenti in C

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;
  for (i=0;i<argc;i++)
    printf("arg%d: %s\n",i,argv[i]);

  return 0;
}
```

Eventuali argomenti numerici possono essere convertiti tramite le funzioni di libreria `sscanf`, `atof` e `atoi`.