

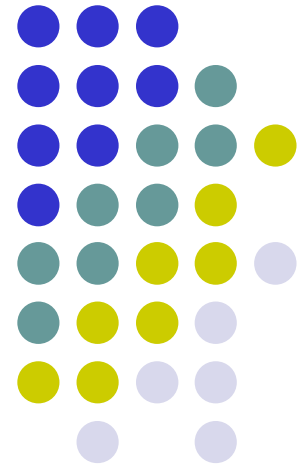
Laboratorio di Calcolatori 1

Corso di Laurea in Fisica

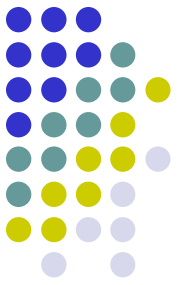
A.A. 2007/2008

Dott. Davide Di Ruscio

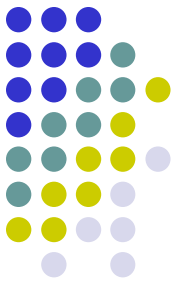
Dipartimento di Informatica
Università degli Studi di L'Aquila



Nota



Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele



Sommario (II parte)

Il Linguaggio C

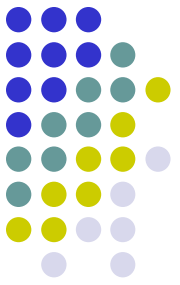
- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
 - tipi
 - definizioni di tipi
 - definizioni di variabili
- Parte esecutiva
 - istruzione di assegnamento
 - istruzioni (funzioni) di input-output
 - istruzioni di selezione
 - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata
- Algoritmi elementari
 - ricerca sequenziale e binaria
 - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
 - ricorsione
 - strutture dati dinamiche

RIFERIMENTI

Ceri, Mandrioli, Sbattella

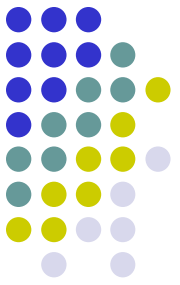
[Informatica arte e mestiere](#)

McGraw-Hill



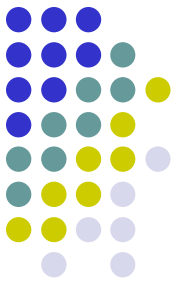
Gestione file

- Il file system in C
- Principali operazioni sui file (dalla Standard Library)
- Esempi



File System in C

- Un programma C prima di poter utilizzare un file deve aprire un flusso di comunicazione.
- Al termine dell'utilizzo di un file il flusso di comunicazione viene chiuso
- Informazioni sul file ed il suo stato:
 - Posizione del file
 - Tipo del file (***di testo*** o ***binario***)
 - Operationi possibili
 - Posizione corrente del cursore nel file
 - Indicatore di end of file (EOF)
 - Indicatore di errore (ERROR)



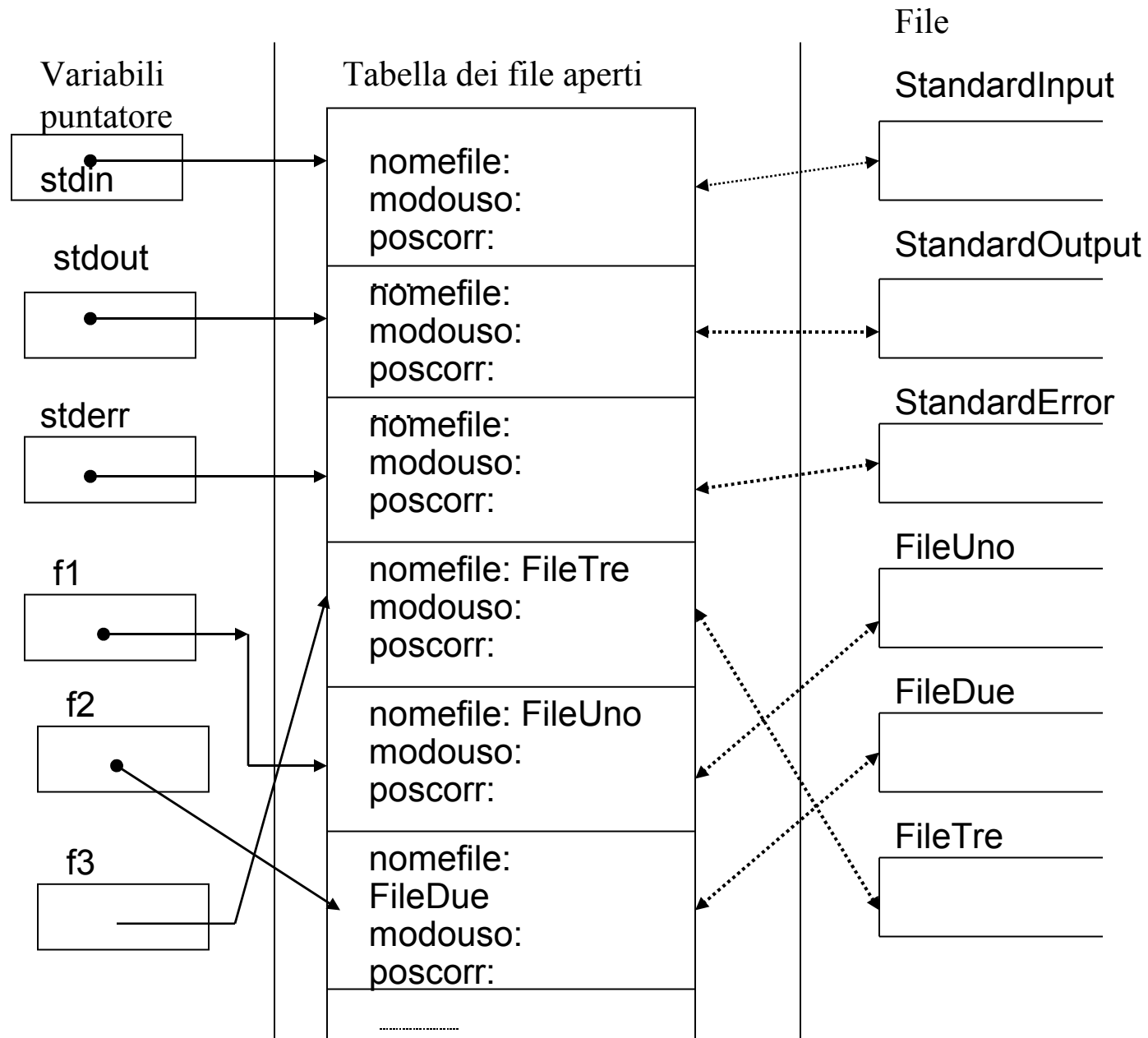
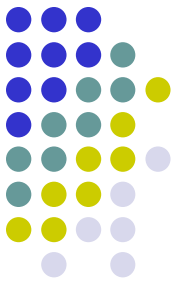
File system in C

- Queste informazioni sono contenute in una struttura. Il suo tipo è **FILE**, ed è definita nell'header file `stdio.h`

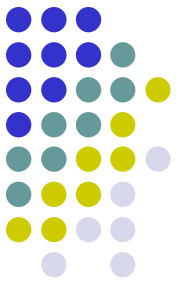
```
FILE *ifp, *ofp; /*pointers for input and output file*/
```

- Non abbiamo bisogno di conoscere nel dettaglio questa struttura, in quanto viene creata e gestita automaticamente dalle librerie di input/output del C
- Nel linguaggio C un file è semplicemente una sequenza di byte, ogni dispositivo fisico viene visto come un file

Tabella file aperti

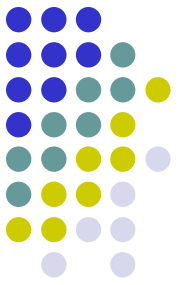


Operazioni di gestione dei file in ANSI C



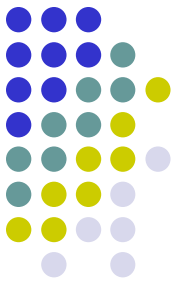
- fopen, fclose;
- fgetc, fputc;
- fgets, fputs;
- fprintf, fscanf;
- fread, fwrite;
- fseek, fsetpos;
- ftell, fgetpos;
- ...

Contenute nella libreria: `stdio.h`



Apertura e chiusura di file

- **FILE *fopen(nomefile, modalità)**
 - "apre" un file ovvero inizializza una area di memoria attraverso la quale vengono scambiati i dati tra memoria principale e secondaria
 - Deve essere effettuata sempre prima di ogni tipo di elaborazione sul file
 - Se l'apertura del file non ha successo la funzione restituisce NULL.
- **int fclose (FILE *fp)**
 - "chiude" un file chiudendo il relativo canale di comunicazione
 - Se la chiusura del file non ha successo la funzione restituisce NULL
 - Deve essere effettuata sempre alla fine delle elaborazione sul file



Apertura e chiusura di file

Come aprire e chiudere i files.

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

```
int main(void)  
{
```

```
    int a, sum = 0;
```

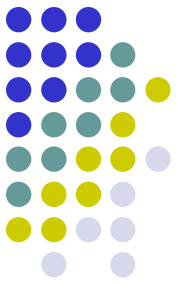
```
    FILE *ifp, *ofp    /* input file pointer,  
                        output file pointer */
```

```
    ifp = fopen("my_file", "r");    /* aperto in lettura */  
    ofp = fopen("outfile", "w");    /* aperto in scrittura */  
    .....
```

```
    fclose(ifp);  
    fclose(ofp);
```

Puntatori a strutture di tipo FILE

La modalità di apertura del file



Modalità di apertura dei files

```
FILE *ifp;  
ifp = fopen("filename", "r");
```

←
"r" - Lettura (testo)

"w" - Scrittura (testo)

"a" - Scrittura a fine file (testo)

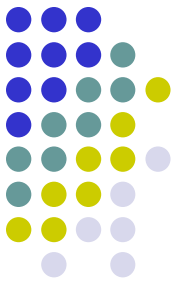
"rb" - Lettura (binario)

"wb" - Scrittura (binario)

"ab" - Scrittura a fine file (binario)

"r+", "w+", "a+" – Lettura e Scrittura (testo)

"rb+", "wb+", "ab+" – Lettura e Scrittura (binario)



Lettura e scrittura formattata

Per scrivere su file: funzione `fprintf()` analoga a `printf()`
Per leggere da file: funzione `fscanf()` analoga a `scanf()`

Deve essere specificato un puntatore a FILE:

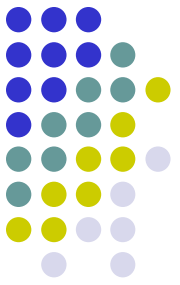
```
fprintf(ifp, string_controllo, lista_arg);  
fscanf (ifp, string_controllo, lista_var);
```

Esistono tre speciali puntatori predefiniti

esempio: `"%s %12.2f\n"`

<code>stdin</code>	punta a: tastiera
<code>stdout</code>	punta a: schermo
<code>stderr</code>	punta a: schermo

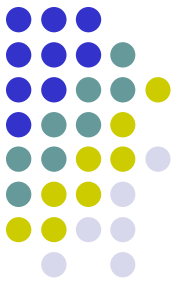
<code>printf(...)</code>	è equivalente a	<code>fprintf(stdout,)</code>
<code>scanf(...)</code>	è equivalente a	<code>fscanf(stdin,)</code>



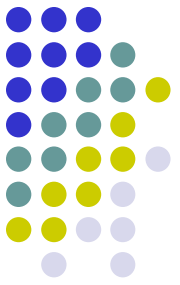
Lettura e scrittura formattata

- `int fprintf(FILE *pf, str_cont, elementi)`
 - scrive caratteri in un file secondo il formato specificato
 - Restituisce il numero di caratteri scritti se l'operazione è andata a buon fine, altrimenti restituisce un valore negativo
- `int fscanf(FILE *pf, str_cont, ind_elementi)`
 - legge caratteri da un file secondo il formato specificato e assegna i valori letti ai suoi successivi argomenti
 - Restituisce il numero di caratteri letti se l'operazione é andata a buon fine, altrimenti restituisce EOF

Lettura e scrittura di caratteri

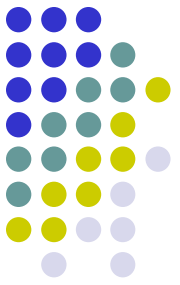


- `int fputc(int c, FILE *pf)`
 - scrive un singolo carattere in un file
 - Restituisce il carattere stesso se l'operazione è andata a buon fine, altrimenti restituisce EOF
- `int fgetc(FILE * pf)`
 - legge un carattere dal file specificato
 - Restituisce il carattere letto (come intero) se l'operazione è andata a buon fine, altrimenti restituisce EOF se si è raggiunta la fine del file



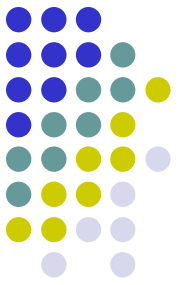
Lettura e scrittura di stringhe

- `char * fgets(char *s, int n, FILE *fp)`
 - legge un numero specificato di caratteri da un file e li memorizza in una stringa aggiungendo il carattere di fine stringa
 - Restituisce la stringa se l'operazione é andata a buon fine, altrimenti restituisce NULL
- `int fputs(char *s, FILE *fp)`
 - Scrive una stringa nel file specificato senza aggiungere il carattere di fine stringa
 - Restituisce 0 se l'operazione é andata a buon fine, altrimenti restituisce EOF .



remove e rename di file

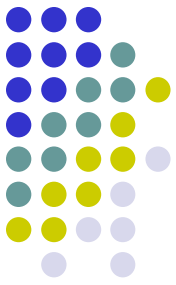
- **int remove(<nomefile>)**
 - Cancella il file identificato da <nome file>
- **int rename(<vecchionome>, <nuovonome>)**
 - Modifica il nome del file identificato da <vecchionome> a <nuovonome>
- Le funzioni restituiscono 0 se le operazioni avvengono correttamente, altrimenti restituiscono un valore diverso da zero



Gestione indicatori di stato

- Quando si rileva un errore in una operazione di accesso a file, vengono aggiornati degli "indicatori di stato"
- `int ferror(FILE *fp)`
 - Restituisce un valore diverso da 0 se si è verificato un errore sul file, altrimenti restituisce 0
- `int feof(FILE *fp)`
 - Restituisce un valore diverso da 0 se è stata raggiunta la fine del file, altrimenti restituisce 0
- `void clearerr(FILE *fp)`
 - Resetta gli indicatori di stato
- In caso di errore la variabile interna `errno` (in `<errno.h>`) registra un numero che da indicazioni sul tipo di errore

Funzioni di posizionamento su file



- Quando leggiamo o scriviamo in un file, il sistema che tiene traccia della posizione corrente nel file effettua l'operazione di lettura /scrittura a partire da quella posizione (e quindi in modo sequenziale).
- Abbiamo a disposizione delle funzioni che ci permettono di accedere in una posizione specifica del file:

```
int rewind(FILE *fp) ;
```

- setta la posizione corrente all'inizio del file

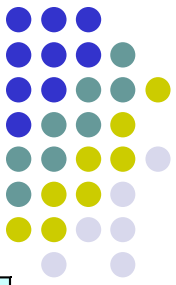
```
int ftell(FILE *fp) ;
```

- restituisce il valore della posizione corrente

```
int fseek (FILE *fp, long offset, int place) ;
```

- sposta la posizione corrente di `offset` bytes rispetto a `place`, dove `place` può essere `SEEK_SET`, `SEEK_CUR` or `SEEK_END` rispettivamente per l'inizio, la posizione corrente o la fine del file
- L'offset può essere anche negativo.

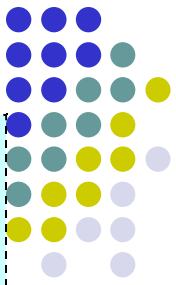
Esempio



```
/* Legge un file al contrario */
#include <stdio.h>
#define MAXSTRING 100
main()
{
    char fname[MAXSTRING];
    int c;
    FILE *ifp;
    fprintf(stdout, "\nInput a filename: ");
    scanf("%s", fname);
    ifp = fopen(fname, "r");
    fseek(ifp, 0, SEEK_END);
    fseek(ifp, -1, SEEK_CUR);
    while (ftell(ifp) > 0) {
        c = fgetc(ifp);
        putchar(c);
        fseek(ifp, -2, SEEK_CUR); }
    fclose(ifp);
}
```

Dobbiamo spostare la posizione corrente indietro di 2 caratteri, poiché il fgetc l'aveva incrementata di 1.

Altri esempi



```
/*Legge e mostra sul video il contenuto del file di testo filechar*/
#include <stdio.h>

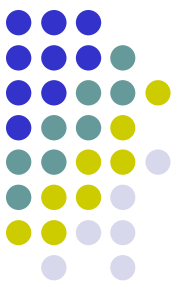
/* Contiene la definizione di EOF, del tipo FILE
e le testate delle funzioni che operano su file */
#include <stddef.h>

/* Contiene la definizione di NULL */

main()
{
    FILE    *fp;
    int      c;

    fp = fopen("filechar", "r");_
    if (fp != NULL)
        /* Il file viene aperto in lettura con modalità testo */
        {
            while ((c = fgetc(fp)) != EOF)
                /* Viene letto e stampato un carattere per volta sino a fine file */
                putchar(c);
            fclose(fp);
        }
    else
        printf("Il file non può essere aperto\n");
}
```

Altri esempi (2)



/*Lettura e scrittura di stringhe (1)*/

```
#include <stdio.h>
#include <stddef.h>
#include <string.h>
#define OK 1
#define ERROR 0
#define MAXLINE 100
```

```
int copiasellettiva(char refstr[])
{
```

```
    char line[MAXLINE];
    FILE *fin, *fout;
```

```
    fin = fopen("filein", "r");
```

```
        /* filein viene aperto in lettura con modalità testo */
```

```
    fout = fopen("fileout", "w");
```

```
        /* fileout viene aperto in scrittura con modalità testo */
```

```
    if (fin == NULL || fout == NULL)
```

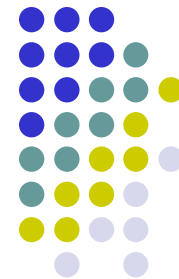
```
        return ERROR;
```

```
    else
```

```
    {
```

```
        ...
```

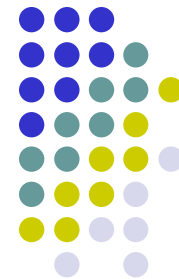
Altri esempi (3)



```
...
while (fgets(line,MAXLINE,fin) != NULL)
    /* fgets legge da filein al più MAXLINE-1 caratteri e
       assegna al vettore line i caratteri letti, incluso l'eventuale carattere di
       newline,
       e termina la stringa con il carattere \0 */
    if (strstr (line,refstr) != NULL)
        /* La funzione strstr restituisce la posizione della prima
           occorrenza della stringa puntata da refstr nella stringa puntata da
           line; se la seconda stringa non è contenuta nella prima viene restituito il
           valore NULL */
        fputs(line,fout);

    fclose(fin);
    fclose(fout);
    return OK;
}
```

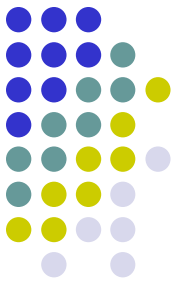
File binari



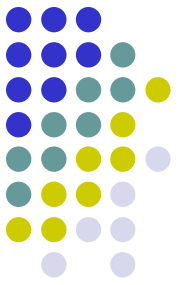
Principali caratteristiche:

- I file binari sono comprensibile all'elaboratore, ma non hanno un formato leggibile dall'uomo come quello testuale
- Un file binario è in grado di memorizzare strutture
- Sono più efficienti dei file testuali in quanto non deve avvenire alcuna conversione tra il formato binario (presente in memoria centrale) e quello testuale
- Non possono essere letti facilmente da altri programmi non sviluppati in C

Lettura e scrittura di strutture



- `int fread(void *ptr, dimelemento, numelementi, FILE *fp) ;`
Legge dal file *fp*, *numelementi* strutture di dimensione *dimelemento* e le memorizza nell'array *ptr*.
- `int fwrite(void *ptr, dimelemento, numelementi, FILE *fp) ;`
Scrive sul file *fp*, *numelementi* strutture di dimensione *dimelemento* presenti nell'array *ptr*.
- Anche nei file binari è possibile usare la funzione **fseek**. Naturalmente l'*offset* rappresenta lo scostamento in bytes, e dunque è utile servirsi della funzione **sizeof()**



Files binari: esempio (1)

```
#include <stdio.h>

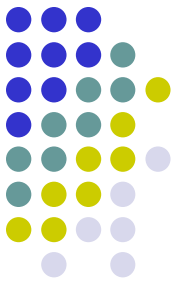
typedef struct { int x,y,z;} rec;

main()
{
    int i,j;

    FILE *f;

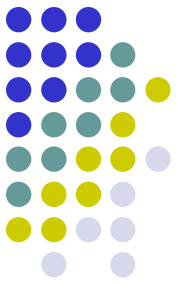
    rec r;

    /* crea il file di 10 strutture */
    f=fopen("pippo","wb");
    if (f!=NULL)
        for (i=1;i<=10; i++)
        {
            r.x=i; r.y=i*2; r.z = i*3;
            fwrite(&r,sizeof(rec),1,f);
        }
    fclose(f);
}
```



Files binari: esempio (2)

```
/* legge le 10 strutture */  
f=fopen("pippo","rb");  
if (f==NULL)  
    return 1;  
  
for (i=1;i<=10; i++)    {  
    fread(&r,sizeof(rec),1,f);  
    printf("%d\n",r.x);  
}  
  
fclose(f);
```



Files binari: esempio (3)

```
/* usiamo fseek per leggere
le 10 strutture in ordine inverso */
f=fopen("pippo","rb");
    if (f!=NULL)
        for (i=9; i>=0; i--)
        {
            fseek(f,sizeof(rec)*i,SEEK_SET);
            fread(&r,sizeof(rec),1,f);
            printf("%d\n",r.x);
        }
    fclose(f);
}_
```