Dipartimento di Informatica e Sistemistica
Antonio Ruberti

“Sapienza” Università di Roma

Gestione dinamica della memoria
Stringhe
File

Corso di Fondamenti di Informatica

Laurea in Ingegneria Informatica

(Canale di Ingegneria delle Reti e dei Sistemi Informatici - Polo di Rieti)

Anno Accademico 2007/2008

Prof. Paolo Romano

Si ringrazia il Prof. Alberto Finzi per aver reso
disponibile il materiale didattico sul quale si basano queste slides

Allocazione dinamica della memoria

Un puntatore deve puntare ad una zona di memoria

- a cui il sistema operativo permette di accedere
- che non viene modificata inaspettatamente

Finora abbiamo visto un modo per soddisfare questi requisiti: assegnare ad un puntatore l'indirizzo di una delle variabili del programma.

Metodo alternativo: *allocazione dinamica della memoria*, attraverso una chiamata di funzione che crea una nuova zona di memoria e ne restituisce l'indirizzo iniziale.

- la zona di memoria è accessibile al programma
- la zona di memoria non viene usata per altri scopi (ad esempio variabili in altre funzioni)
- ad ogni chiamata della funzione viene allocata una nuova zona di memoria

Funzione `malloc`

La funzione `malloc` è dichiarata in `<stdlib.h>` con prototipo:

```
void * malloc(size_t);
```

• prende come parametro la dimensione (numero di byte) della zona da allocare (`size_t` è il tipo restituito da `sizeof` e usato per le dimensioni in byte delle variabili --- ad esempio potrebbe essere `unsigned long`)

• alloca (riserva) la zona di memoria

•restituisce il puntatore iniziale alla zona allocata (è una funzione che restituisce un puntatore)

N.B. La funzione `malloc` restituisce un puntatore di tipo `void*`, compatibile con tutti i tipi puntatore

Esempio:
```
float *p;
p = malloc(4);
```

Funzione `malloc`

Uso tipico di `malloc` è con `sizeof(`*tipo*`)` come parametro.

Esempio

```
#include <stdlib.h>
int *p; p = malloc(sizeof(int)); *p = 12;
(*p)++;
/* N.B. servono le parentesi */
printf("*p vale %d\n", *p);
```

- attivando `malloc(sizeof(int))` viene allocata una zona di memoria adatta a contenere un intero; ovvero viene creata una nuova variabile intera
- il puntatore restituito da `malloc` viene assegnato a `p`
  `*p` si riferisce alla nuova variabile appena creata

Lo heap (o memoria dinamica)

La zona di memoria allocata attraverso `malloc` si trova in un'area di memoria speciale, detta *heap* (o memoria dinamica).

Abbiamo *4 aree di memoria*:

- zona programma: contiene il codice macchina
- stack: contiene la pila dei RDA
- statica: contiene le variabili statiche
- heap: contiene dati allocati dinamicamente

Funzionamento dello heap:
- gestito dal sistema operativo
- le zone di memoria sono *marcate libere o occupate*
    a. marcata libera: può venire utilizzata per la prossima malloc
    b. marcata occupata: non si tocca

Lo heap (o memoria dinamica)

Potrebbe *mancare la memoria* per allocare la zona richiesta. In questo caso `malloc` restituisce il puntatore `NULL`: bisogna sempre verificare cosa restituisce `malloc`.

Esempio:

```
p = malloc(sizeof(int));

if (p == NULL) {
   printf("Non ho abbastanza memoria per l'allocazione\n");
  exit(1); } ...
```

La costante `NULL`

- è una costante di tipo `void*` (quindi compatibile con tutti i tipi puntatore)
- indica un puntatore che non punta a nulla: non può essere dereferenziato
- ha tipicamente valore 0
- definita in `<stdlib.h>` (ed in altri file header)

Deallocazione della memoria dinamica

Le celle di memoria allocate dinamicamente devono essere deallocate (o rilasciate) quando non servono più.

Si utilizza la funzione `free`, che è dichiarata in `<stdlib.h>`:

```
void * free(void *);
```

Esempio

```
int *p; ... p = malloc(sizeof(int)); ... free(p);
```

- il parametro `p` *deve* essere un puntatore ad una zona di memoria allocata precedentemente con `malloc` (altrimenti il risultato non è determinato)
- la zona di memoria *viene resa disponibile* (viene marcata libera)
- `p` non punta più ad una locazione significativa (valore arbitrario)

Esempio:
Gestione della dimensione di un array

Vedi codice

Stringhe

Stringhe

Una stringa è un vettore di caratteri. Contiene la sequenza di caratteri che forma la stringa, seguita dal *carattere speciale di fine stringa*: `'\0'`

Esempio

```
char stringa1[10] = {'p', 'i', 'p', 'p', 'o', '\0'};
```

Il seguente vettore di caratteri non è una stringa perchè non termina con `'\0'`

Esempio

```
char non_stringa1[2] = {'p', 'i'};
```

Inizializzazione di stringhe

Una stringa può anche essere *inizializzata* utilizzando una *stringa letterale*:

```
char stringa2[] = "pippo"; oppure
char stringa2[6] = "pippo";
```

`stringa2` è un array statico di 6 caratteri:

`‘p’,'i', 'p', 'p', 'o', '\0'`

E’ possibile memorizzare una stringa in un *array di caratteri dinamico*

Esempio. `char *buffer = malloc(80*sizeof(char));`

In questo caso (come per tutti gli array dinamici) non possiamo inizializzare l'array contestualmente alla sua creazione.

Inizializzazione di stringhe

Possiamo anche “assegnare" ad una stringa una stringa letterale.

```
char *buffer2;
buffer2 = "pippo";
```

Con questa assegnazione abbiamo assegnato a `buffer2`, di tipo `char*`, la stringa costante `"pippo"`, di tipo `char*` costante: `buffer2` punta al primo carattere della *stringa costante* "pippo".

L'istruzione `buffer2[0] = 't';` non dà errore di compilazione.

Dà però *errore in esecuzione* poichè stiamo cercando di cambiare un carattere dichiarato costante.

N.B. questo è diverso da
`char buffer3[] = "pippo“;` che è equivalente a
`char buffer3[6] = "pippo";` che è equivalente a
`char buffer3[] = {'p', 'i', 'p', 'p', 'o', '\0'};`

In questo caso la stringa constante "pippo" viene usata per inizializzare buffer3.

Inizializzazione di stringhe

Inizializzazione di un vettore di stringhe:

Esempio

```
char *colori[4] = {"rosso", "giallo", "verde", "blu"};
```

E' un vettore di quattro puntatori a quattro stringhe costanti (di lunghezza 6, 7, 6, 4).

E' equivalente ad inizializzare i quattro puntatori separatamente:

```
char *colori[4];
colori[0] = "rosso";     colori[1] = "giallo";
colori[2] = "verde";     colori[3] = "blu";
```

Ingresso/uscita di stringhe

Stampa di una stringa: si deve utilizzare la specifica di formato `%s`

Esempio

```
printf("%s\n", stringa1);
printf("%s\n", stringa2);
printf("%s\n", buffer2);
```

Vengono stampati tutti i caratteri fino al primo `'\0'` escluso.

Esempio

```
printf("%s\n", non_stringa1);
```

stampa:

```
pi^D^H"^D^H"^D^A...
```

Ingresso/uscita di stringhe

Lettura di una stringa: si deve utilizzare la specifica di formato `%s`.

Esempio

```
char buffer[40];
scanf("%s", buffer);
```

- Vengono letti da input i caratteri in sequenza fino a trovare il primo carattere di spaziatura (spazio, tabulazione, interlinea, ecc.).
- I caratteri letti vengono messi dentro il vettore `buffer`.
- Al posto del carattere di spaziatura, viene posto il carattere `'\0'`.

Note:

- il vettore *deve* essere sufficientemente grande da contenere tutti i caratteri letti
- non si usa `&buffer` ma direttamente `buffer` (questo perchè `buffer` è di tipo `char*`, ovvero è già un indirizzo)

Manipolazione di stringhe

Per manipolare una stringa si deve accedere ai singoli caratteri singolarmente.

Esempio

```
for (i = 0; buffer[i] != '\0'; i++) {
  /* fai qualcosa con buffer[i], ad esempio: */
  printf("%c\n", buffer[i]);
}
```

Esempio: Confronto di uguaglianza tra due stringhe.

N.B. *Non* si può usare == perchè questo confronta i puntatori e non le stringhe: si devono necessariamente scandire le due stringhe.

Manipolazione di stringhe: esempio

Confronto di uguaglianza tra due stringhe

```
#include<stdio.h>
#define DIM 80

int uguali1(char * str1, char *str2) {
  int i = 0;
  while(str1[i] == str2[i] && str1[i] != '\0' && str2[i] != '\0')
    i++;  return str1[i] == '\0' && str2[i] == '\0';
 }
 int uguali2(char * str1, char * str2) {
  while (*str1 == *str2 && *str1 != '\0' && *str2 != '\0') {
    str1++; str2++;
  }
  return *str1 == '\0' && *str2 == '\0';
 }
int main() {
  char s1[DIM];  char s2[DIM];
  scanf("%s %s", s1,s2);
  if (uguali2(s1, s2)) printf("uguali\n");
  else  printf("diverse\n");
  return 0;
  }
```

Manipolazione di stringhe: esempio

Calcola la lunghezza della stringa.

```
#include<stdio.h>
#define DIM 80
 int calcolaLunghezza1(char *stringa) {
  int i = 0;
  while (stringa[i] != '\0')
    i++;
  return i;
 }
 int calcolaLunghezza2(char * stringa) {
  char *p;
  p = stringa;
  while (*p != '\0')
    p++;
  return p - stringa;
 }
. . .
```

In realtà queste funzioni, come molte altre, sono disponibili nella libreria sulle stringhe.

Caratteri e stringhe

Caratteri e stringhe sono diversi e *non* vanno confusi:

- un carattere è in realtà un intero
- per denotare una costante di tipo carattere: `'x'`
- una stringa è un vettore di caratteri che termina con il carattere `'\0'`
  per denotare costanti di tipo stringa: `"un esempio di stringa"`
- una variabile di tipo stringa è in realtà un puntatore al primo carattere del vettore

Esempio

`char c = 'a';`	carattere
`char *s = "a";`	puntatore alla stringa costante `"a"`
`char v[] = "a";`	vettore di 2 caratteri inizializzato a `{'a', '\0'}`

```
printf("%d %d %d\n", sizeof(c), sizeof(s), sizeof(v));
```
Stampa: `1 4 2`

Funzioni di libreria per la gestione dei caratteri

Per usare le funzioni per la gestione dei caratteri *è necessario*:
`#include <ctype.h>`

Tutte le funzioni operano sui caratteri e su `EOF`. Però: `char` può essere `unsigned`, mentre `EOF` normalmente è –1:
Tutte le funzioni hanno parametri di tipo `int` (e non di tipo `char`).

Però possono prendere argomenti di tipo `char`, poichè `char` viene promosso ad `int`.

Funzioni di conversione maiuscolo – minuscolo:

`int tolower(int c);` se `c` è una lettera maiuscola, la converte in minuscolo altrimenti restituisce `c` invariata

`int toupper(int c);` se `c` è una lettera minuscola, la converte in maiuscolo altrimenti restituisce `c` invariata

Funzioni di libreria per la gestione dei caratteri

Funzioni che determinano il tipo di un carattere

- restituiscono vero (1) se il carattere è tra quelli indicati
- restituiscono falso (0) altrimenti

`int isdigit(int c);` cifra

`int isxdigit(int c);` cifra esadecimale

`int isalpha(int c);` lettera alfabetica (minuscola o maiuscola)

`int isalnum(int c);` cifra o lettera alfabetica

`int islower(int c);` lettera alfabetica minuscola

`int isupper(int c);` lettera alfabetica maiuscola

`int isspace(int c);` carattere di spazio bianco `' ','\n','\r','\t', '\v','\f'`

`int iscntrl(int c);` carattere di controllo

`int isprint(int c);` carattere stampabile (incluso lo spazio)

`int isgraph(int c);` carattere stampabile diverso dallo spazio

`int ispunct(int c);` carattere stampabile non alfanumerico o di spazio (ovvero un carattere di punteggiatura)

Funzioni di libreria per l'input/output di stringhe e caratteri

E' necessario: `#include <stdio.h>`

Input e output di caratteri

```
int getchar(void);
```

legge il prossimo carattere da standard input e lo restituisce

```
int putchar(int c);
```

- manda `c` in standard output
- restituisce `EOF` (`-1`) se si verifica un errore

Funzioni di libreria per l'input/output di stringhe e caratteri

Input e output di stringhe:
`char *gets(char *str);`

- legge i caratteri da standard input e li inserisce nel vettore `str` la lettura termina con un `'\n'` o un `EOF`, che *non* viene inserito nel vettore
- la stringa nel vettore viene terminata con `'\0'`
- Attenzione: non c'è alcun modo di limitare la lunghezza della sequenza immessa: vettore allocato potrebbe non bastare

`int puts(const char *str);`
manda la stringa str in standard output (inserisce un '\n' alla fine)

Funzioni di libreria per l'input/output di stringhe e caratteri

Input e output di stringhe da/su stringhe

```
int sprintf(char *str, const char *format, ...);
```

come `printf`, solo che invece di scrivere su standard output, scrive nel vettore `str`

```
int sscanf(char *str, const char *format, ...);
```

come scanf, solo che invece di leggere da standard input, legge dalla stringa str

Funzioni di libreria per la manipolazione di stringhe

E' necessario: `#include <string.h>`

Funzioni di copia:

```
char *strcpy(char *dest, const char *src);
```

- copia la stringa `src` nel vettore `dest`
- restituisce il valore di `dest`
- `dest` dovrebbe essere sufficientemente grande da contenere `src`
- se `src` e `dest` si sovrappongono, il comportamento di `strcpy` è indefinito

Esempio

```
char a[10], b[10];
char *x = "Ciao";
char *y = "mondo";
strcpy(a, x);   strcpy(b, y);
puts(a);        puts(b);
```

Stampa:

Ciao
mondo

Funzioni di libreria per la manipolazione di stringhe

`char *strncpy(char *dest, const char *src, size\_t n);`

- copia un massimo di n caratteri della stringa `src` nel vettore `dest`
`size_t` è il tipo del valore restituito da sizeof, (`unsigned long o unsigned int`)
- restituisce il valore di dest
- Attenzione: il carattere `'\0'` finale di src viene copiato *solo* se n è ≥ della lunghezza di `src+1`

Esempio

```
char a[10], b[10];
char *x = "Ciao";
char *y = "mondo";
strncpy(a, x, 5);
puts(a);            stampa: Ciao (b non è terminata da '\0‘)
strncpy(b, y, 5);
b[5] = ’\0';
puts(b);            stampa: mondo
```

Funzioni di libreria per la manipolazione di stringhe

```
char *strcat(char *dest, const char *src);
```

- accoda la stringa `src` a quella nel vettore `dest` (il primo carattere di `src` si sostituisce al '\0' di `dest`) e termina dest con '\0'
- restituisce il valore di `dest`
- `dest` dovrebbe essere sufficientemente grande da contenere tutti i caratteri di dest, di `src`, ed il '\0' finale
- se `src` e `dest` si sovrappongono, il comportamento di `strcat` è indefinito

Esempio

```
char a[10], b[10];
char *x = "Ciao",  *y = "mondo";
strcpy(a, x);
strcat(a, y);
puts(a);
```

stampa:
Ciaomondo

Funzioni di libreria per la manipolazione di stringhe

`char *strncat(char *dest, const char *src, size\_t n);`
- accoda al più n caratteri di `src` alla stringa nel vettore `dest` (il primo carattere di `src` si sostituisce al '\0' di dest)
- termina in ogni caso dest '\0'

Funzioni di confronto
`int strcmp(const char *s1, const char *s2);`
- confronta le stringhe `s1` ed `s2`
  restituisce:
    0 se s1 = s2
    un valore < 0 se s1< s2
    un valore > 0 se s1 >s2
- il confronto è quello lessicografico: i caratteri vengono confrontati uno ad uno, ed il primo carattere diverso (o la fine di una delle due stringhe) determina il risultato
- per il confronto tra due caratteri viene usato il codice

Funzioni di libreria per la manipolazione di stringhe

Esempio

```
char *s1 = "abc";
char *s2 = "abx";
char *s3 = "abc altro";
```

Stampa:

```
printf("%d\n", strcmp(s1, s1));   0
printf("%d\n", strcmp(s1, s2));   -1
printf("%d\n", strcmp(s1, s3));   -1
printf("%d\n", strcmp(s2, s1));   1
```

Attenzione: per verificare l'uguaglianza bisogna confrontare il risultato con 0

Esempio

```
if (strcmp(s1, s2) == 0)
  printf("uguali\n");
else
  printf("diverse\n");
```

Funzioni di libreria per la manipolazione di stringhe

```
int strncmp(const char *s1, const char *s2, size\_t n);
```

- confronta al più n caratteri di `s1` ed `s2`

Esempio

```
char *s1 = "abc";
char *s3 = "abc altro";
printf("%d\n", strncmp(s1, s3, 3));
```

Stampa `0`, ovvero le due stringhe sono uguali.

Esercizio: Fornire un'implementazione delle funzioni di libreria su caratteri e stringhe.

Funzioni di libreria per la conversione di stringhe

E' necessario: `#include <stdlib.h>`

Convertono le strighe formate da cifre in valori interi ed in virgola mobile.

Funzioni atoi, atol, atof

```
int atoi(const char *str);
```

- converte la stringa puntata da `str` in un `int`
- la stringa deve contenere un numero intero valido
- in caso contrario il risultato è indefinito
- spazi bianchi iniziali vengono ignorati
- il numero può essere concluso da un qualsiasi carattere che non è valido in un numero intero (spazio, lettera, virgola, punto, ...)

`atoi("123")` restituisce l'intero 123
`atoi("123.45")` restituisce l'intero 123 (".45" viene ignorato)

Funzioni di libreria per la conversione di stringhe

`long atol(const char *str);`

analoga ad `atoi`, solo che restituisce un `long int`

`double atof(const char *str);`

analoga ad `atoi` ed `atol`, solo che restituisce un `double`

il numero può essere concluso da un qualsiasi carattere non valido in un numero in virgola mobile

`atof("123.45")` restituisce `123.45`

`atof("1.23xx")` restituisce `1.23`

`atof("1.23.45")` restituisce `1.23`

Esercizio

Scrivere una funzione che ricevendo come par. una stringa ed un numero m:
1. Calcola il numero di volte n in cui la somma di caratteri consecutivi è maggiore di m
2. Se n>0 si aggiorna l'n-esimo elemento con il successivo

```
int fun(char *st,int m){
 int i,n=0;
 for (i=0; st[i]!='\0';i++)
    if (st[i]+st[i+1]>m)
        n++;
 if(n>0) st[n-1]++;
 return n;
 }
```

```
#include <stdio.h>
#include <stdlib.h>
#include “funzione.c”
#define Lung 100
int main(){
  char str[Lung];
  int n;
  printf(" stringa:\n");
  scanf("%s",str);
  printf("digita un intero: \n");
  scanf("%d",&n);
  printf("la somma di elementi
consecutivi supera %d volte %d
\n",fun(str,n),n);
  printf("la nuova stringa è:
%s",str);
 return 0;
}
```

Esercizio

Funzione che calcola se una stringa è sottostringa di un'altra, e, in tale caso, quante volte occorre.

```
int sottostringa(char * sotstr, char * str, int * n){
    int i = 0,j,continua;
    *n = 0;
    while (str[i] !='\0'){
       continua = 1; j=0;
       while (sotstr[j] !='\0' && str[j+i] !='\0' && continua){
           if (sotstr[j] != str[j+i]) continua = 0;
           j++;
       }
       if (continua && sotstr[j] =='\0') (*n)++;
       i++;
    }
    return *n > 0;
}
```

Programmi con argomenti passati da linea di comando

I programmi visti finora non prendevano argomenti. La funzione `main` non prendeva parametri:

```
int main(void) { ... }
```

Per poter usare gli argomenti con cui il programma è stato lanciato, la definizione di `main` deve diventare:

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

a. `argc`: numero di argomenti con cui il programma è stato lanciato più 1
b. `argv`: vettore di `argc` stringhe che compongono la riga di comando

- `argv[0]` è il nome del programma stesso
- `argv[1]`, …, `argv[argc-1]` sono gli argomenti (separati da spazio)

ogni `argv[i]` è una stringa (terminata da `'\0'`)

`argc` e `argv` vengono inizializzati dal sistema operativo (le stringhe di `argv` sono allocate dinamicamente)

Programmi con argomenti passati da linea di comando

Esempio Stampa del numero di argomenti ricevuti da linea di comando.

```
#include <stdio.h>
int main(int argc, char *argv[]){
  printf("Ho ricevuto %d argomenti\n", argc-1);
  return 0;
}
```

Compilando si ottiene argnum.exe, che può essere eseguito:

```
> argnum primo secondo terzo
> Ho ricevuto 3 argomenti
```

Esempio Stampa degli argomenti ricevuti da linea di comando

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  int i;
  printf("Ho ricevuto %d argomenti\n", argc-1);
  printf("Questi argomenti sono:\n");
  for (i = 1; i <= argc-1; i++) printf("%s\n", argv[i]);
  return 0;
}
```

Programmi con argomenti passati da linea di comando

Esempio Stampa della somma di due numeri interi passati come argomenti.

N.B. I due numeri sono passati come stringhe e non come numeri.

Vanno convertiti in numeri usando `atoi`.

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
  int a, b;
  if (argc-1 != 2) { /* verifica numero argomenti*/
    printf("Devi passare due argomenti\n");
    exit(1);
  }
  a = atoi(argv[1]); /* converte argomento in un intero*/
  b = atoi(argv[2]); /* converte argomento in un intero */
 printf("La somma dei %d e %d e`: %d\n", a, b, a+b);
 return 0;
}
```

Elaborazione dei file

Un file è una *sequenza di byte* (o caratteri) memorizzata su disco (memoria di massa), alla quale si accede tramite un *nome*.

I byte corrispondono a dei dati di tipo `char`, `int`, `double`, ecc. (esattamente come la sequenza di byte che immettiamo da input o leggiamo da tastiera).

I file vengono gestiti dal sistema operativo, e il programma deve invocare le funzioni del SO per accedere ai file: viene fatto dalla libreria standard di input/output.

Operazioni sui file

Principali operazioni sui file:

- lettura da file
- scrittura su file
- apertura di file: comunica al SO che il programma sta accedendo al file
- chiusura di file: comunica al SO che il programma rilascia il file

Per ogni programma vengono aperti automaticamente tre file:

`stdin` (sola lettura): lettura è da tastiera

`stdout` (sola scrittura): scrittura è su video

`stderr` (sola scrittura): per messaggi di errore (scritti su video)

Operazioni sui file

Apertura di un file: tramite la funzione `fopen`
Deve essere effettuata prima di poter operare su un qualsiasi file.

Esempio
```
FILE *fp; fp = fopen("pippo.dat", "r");
```

`FILE` è un tipo struttura definito in `stdio.h`.

Dichiarazione di `fopen`:
```
FILE * fopen(char *nomefile, char *modo);
```

Operazioni sui file

Parametri di `fopen`:
- il nome del file
- la modalit\`a di apertura:
  - `"r"` (read) … sola lettura
  - `"w"`(write)… crea un nuovo file per la scrittura; se il file esiste già ne elimina il contenuto corrente
  - `"a"` (append) … crea un nuovo file in scrittura, o accoda ad uno esistente per la scrittura
  - `"r+”` … apre un file per l'aggiornamento (lettura e scrittura), a partire dall'inizio del file
  - `"w+"` … crea un nuovo file in lettura e scrittura; se il file esiste già ne elimina il contenuto corrente
  - `"a+"` ... se il file non esiste allora viene creato ed aperto in lettura e scrittura. Le operazioni di lettura avvengono a partire dall'inizio del file stesso mentre la scrittura avviene in append mode.

Operazioni sui file

`fopen` restituisce un puntatore ad una struttura di tipo `FILE`

- la struttura mantiene le informazioni necessarie alla gestione del file
- il puntatore (detto *file pointer*) viene utilizzato per accedere al file
- se si verifica un errore in fase di apertura, `fopen` restituisce `NULL`

Esempio

```
FILE *fp; if ((fp = fopen("pippo.dat", "r")) == NULL) {
  printf("Errore!\n");
  exit(1);
} else ...
```

Operazioni sui file

Chiusura di un file: `fclose`, passando il file pointer

Esempio

```
fclose(fp);
```

Dichiarazione di `fclose`:

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

Se si è verificato un errore, `fclose` restituisce `EOF` (-1).

La chiusura va `sempre` effettuata, appena sono terminate le operazioni di lettura/scrittura da effettuare sul file.

Scrittura su e lettura da file

Consideriamo solo file ad *accesso sequenziale*: si legge e si scrive in sequenza, senza poter accedere agli elementi precedenti o successivi.

Scrittura su file: tramite `fprintf`
Dichiarazione di `fprintf`:

```
int fprintf(FILE *fp, char *formato, ...);
```

- come `printf`, tranne che per `fp` (`printf(...);` è equivalente a `fprintf(stdout, ...);`)
- scrive sul file identificato da `fp` a partire dalla posizione corrente
- il file deve essere stato aperto in scrittura, append, o aggiornamento
- in caso di errore restituisce `EOF`, altrimenti il numero di byte scritti

Esempio:

```
int ris1, ris2; FILE *fp;
if ((fp = fopen("risultati.dat", "w"))!= NULL) {
  ris1 = ...;   ris2 = ...;
  fprintf(fp, "%d %d\n", ris1, ris2);
  fclose(fp);}
```

Scrittura su e lettura da file

Lettura da file: tramite `fscanf`

Dichiarazione di `fscanf`:

```
int fscanf(FILE *fp, char *formato, ...);
```

- come `scanf`, tranne che per `fp` (`scanf(...);` è equivalente a `fscanf(stdin, ...);`)
- legge dal file identificato da `fp` a partire dalla posizione corrente
- il file deve essere stato aperto in lettura o aggiornamento
- restituisce il numero di assegnamenti fatti agli argomenti specificati nell'attivazione dopo la stringa di formato
- se il file termina o si ha un errore prima del primo assegnamento, restituisce `EOF`

Scrittura su e lettura da file

Verifica di fine file: si può usare `feof`

Dichiarazione di `feof`: `int feof(FILE *fp);`

Restituisce vero (1) se per il file è stato impostato l'*indicatore di end of file*, ovvero:

- quando si tenta di leggere oltre la fine del file
- per lo standard input, quando viene digitata la combinazione di tasti
  - ctrl-z (per Windows)
  - return ctrl-d (per Unix)

Esempio

Esempio: Calcolare la somma degli interi in un file.

```
int somma = 0, n; FILE *fp;

if ((fp = fopen("pippo.txt", "r")) != NULL) {
  while (fscanf(fp, "%d", &n) == 1)
    somma += n;
  fclose(fp);
}
```

Esempio

Esempio: Conteggio del numero di caratteri e di linee in un file

```
FILE *fp;  int caratteri = 0;  int linee = 0;  char ch;

if ((fp = fopen(“pippo.txt”, "r")) == NULL) {
    printf("Errore in apertura in lettur file\n”);
    exit(1);  }
  fscanf(fp, "%c", &ch);
  while (!feof(fp)) {
    caratteri++;
    if (ch == '\n')
      linee++;
    fscanf(fp, "%c", &ch);
  }
  fclose(fp);

  printf("Il numero di caratteri e` %d.\n", caratteri);
  printf("Il numero di linee e` %d.\n", linee);
```

Esempio

Esempio: Creazione di una copia di un file

```
char ch; FILE * fpr, *fpw;

if (((fpr = fopen("in.txt", "r")) != NULL) &&
    ((fpw = fopen("out.txt", "w")) != NULL))  {
  while (fscanf(fpr, "%c", &ch) == 1)
    fprintf(fpw, "%c", ch);
  fclose(fpr);
  fclose(fpw);
}
```

Esercizio

Programma che prende in input una stringa str ed un file e conta:
- il numero di stringhe nel file di cui str è sottostringa
- il numero di occorrenze di str nel file.

```
void occorre(FILE * pf,char * sotstr,int *n,int *nstr){
   char str[LungMax+1];
   int ms;
   *nstr=0; *n = 0;
   while (fscanf(pf, "%s",str)==1){
        if(sottostringa(sotstr,str,&ms)) {
           (*nstr)++; (*n)+=ms;
           }
   }
}
```

Esercizio

```
#include <stdio.h>
#include <stdlib.h>
#define LungMax = 10
#include “miofile.c”
int main() {
    char sottostr[LungMax+1], str[LungMax+1];
     FILE * pf;
     int nstr,n,m;
     if((pf =fopen("prova.txt","r"))==NULL){
        printf("errore apertura file \n");
        exit(1);
     }
     printf("digita la stringa \n");
     scanf("%s",sottostr);
     occorre(pf,sottostr,&n,&nstr);
    printf("%s occorre in %d stringhe e %d volte nel file
\n",sottostr,nstr,n);
      system("PAUSE");
      return 0;
}
```

Esercizio

In un file seq. sono memorizzati i risultati delle partite di calcio di un campionato ad N squadre. Un esempio del contenuto del file è:
Roma Como 3 0 Juventus Lazio 1 1 Chievo Bologna 2 0 Milan Inter 1 1 Piacenza
Juventus Roma 1 2 Lazio Chievo 0 2 Bologna Milan 1 1 Inter Piacenza 3 0 Como
Roma Milan 2 1 Inter Lazio 1 0 Piacenza Bologna 2 3 Chievo Como 5 2 Juventus

Se N è dispari, una squadra riposa per ogni turno, questa viene scritta dopo i risultati della giornata.

Programma in C che, letta la giornata, stampi i risultati su file nel modo seguente:
Juventus Roma 1-2
Lazio Chievo 0-2
Bologna Milan 1-1
Inter Piacenza 3-0
riposa Como

Esercizio

```
void avanzaGiornate(int n,FILE * pf, int squadre){
   int i,j;
   char nomeSq1[20], nomeSq2[20];

   int goal1, goal2;
   for(i=0; i < n; i++){
      for (j=0; j < squadre/2;j++)
        fscanf(pf,"%s %s %d %d",nomeSq1,nomeSq2,&goal1,&goal2);
      if (squadre%2) fscanf(pf,"%s",nomeSq1);
   }
}
```

Esercizio

```
void stampaRisultati(int n, FILE *pf1,FILE *pf2, int squadre) {
   char nomeSq1[20], nomeSq2[20];
   int goal1, goal2,j;
   avanzaGiornate(n-1,pf1,squadre);
   for (j=0; j < squadre/2;j++){
       fscanf(pf1,"%s %s %d %d",nomeSq1,nomeSq2,&goal1,&goal2);
       fprintf(pf2,"%s %s %d-%d \n",nomeSq1,nomeSq2,goal1,goal2);
   }
   if (squadre%2){
      fscanf(pf1,"%s",nomeSq1);
      fprintf(pf2,"riposa %s \n",nomeSq1);
   }}
```

Esercizio

```
#include <stdio.h>
#include <stdlib.h>
#define Squadre 9
#include “funzioni.c”
void stampaRisultati(int n, FILE *pf1, FILE *pf2, int sq);
void avanzaGiornate(int n,FILE * pf, int sq);
int main(){
      FILE *pf1,*pf2;
      if((pf1= fopen("partite.dat","r"))==NULL){
         printf("errore apertura file \n");
         exit(1);
      }
      if((pf2= fopen("risultati.dat","w"))==NULL){
         printf("errore apertura file \n");
         exit(1);
      }
      stampaRisultati(2,pf1,pf2,Squadre);
      system("PAUSE");
      return 0;
}
```