
Suggerimenti, note utili ed errori comuni

Fondamenti di informatica T

printf, scanf e stringhe di formato

- La lettura e scrittura di variabili in formato testo sono realizzate tramite `printf` e `scanf` sfruttando una **stringa di formato**.
- La stringa di formato contiene sequenze speciali di caratteri che specificano il tipo della variabile da leggere/scrivere.
- Un'utile tabella di riferimento:

Identificatore	Tipo	Descrizione
%c	char	Carattere
%d	int	Numero intero con segno
%u	unsigned int	Numero intero senza segno
%f	float	Numero reale (singola precisione)
%lf	double	Numero reale (doppia precisione)
%s	char*	Stringa

printf, scanf e stringhe di formato

- printf richiede, oltre alla stringa di formato, l'elenco delle variabili da stampare su video. Ad esempio:

```
int a;  
double b;  
char stringa[100];  
/* altro codice */  
printf("%d %lf %s\n", a, b, stringa);
```
- scanf richiede, oltre alla stringa di formato, l'indirizzo delle variabili in cui memorizzare i dati letti da tastiera. L'indirizzo di una variabile si ottiene tramite l'operatore indirizzo &. Ad esempio:

```
scanf("%d %lf %s", &a, &b, stringa);
```
- Attenzione! Le stringhe, essendo array di caratteri, sono già equivalenti ad indirizzi, per cui non è necessario riutilizzare l'operatore indirizzo!

Lettura da/scrittura su file

- A grandi linee, la sequenza di passi necessaria per operare su un file è sempre la stessa:
 1. Si apre il file
 2. Si effettuano una o più operazioni di lettura/scrittura sul file
 3. Si chiude il file
- Un file può essere aperto in tre modalità:
 - **r**: lettura. Il file viene aperto per leggerne il contenuto. L'apertura fallisce se il file non esiste.
 - **w**: scrittura. Il file viene aperto in modalità scrittura. Se il file non esiste viene creato, se esiste viene troncato a 0 byte (cioè svuotato del suo contenuto originario).
 - **a**: append. Il file viene aperto in modalità di scrittura, accodando i nuovi dati a quelli già contenuti. Se il file non esiste viene creato.

Aprire e chiudere un file

- La libreria standard implementa una funzione per aprire un file e una per chiuderlo. Il loro prototipo è:
`FILE *fopen(char *path, char *mode);`
`int fclose(FILE *fp);`
- `fopen` apre un file, il cui percorso è specificato da `path`, nella modalità `mode` (lettura, scrittura, append).
- `fopen` restituisce un puntatore a una struttura `FILE`, che può essere utilizzata per compiere ulteriori operazioni sul file. Restituisce la costante `NULL` se l'operazione è fallita (ad esempio apertura in lettura d un file inesistente).
 - N.B.: dal punto di vista di `fopen`, il nome del file da aprire deve essere una stringa, che questa sia una costante (ad es.: “prova.txt”) o che sia stata ottenuta chiedendola all'utente è assolutamente indifferente! (Rif.: Esercitazione 1 – primo esercizio)
- `fclose` chiude un file. Ricordarsi sempre di chiudere tutti i file aperti prima della terminazione del programma.

La modalità binaria

- Lo standard specifica che oltre alla modalità di accesso al file (r/w/a) si può specificare una “b”, che indica il fatto che il file deve essere acceduto in modo “binario” e non in modalità “testo”.
 - È un retaggio del passato, necessario su sistemi di alcuni decenni fa... e su Windows :)
- A cosa serve la modalità testuale? Windows rappresenta l’“a capo” di un file con due caratteri (byte), indicati simbolicamente da \r e \n. Quando si legge una stringa da un file di testo aperto in modalità testo, la libreria standard trasforma i due byte \r\n in un singolo \n.
- Dal punto di vista del programmatore è quasi tutto trasparente, bisogna “solo” ricordarsi di utilizzare il modo “b” quando si accede a file in cui si vogliono leggere/scrivere dati in formato binario.

Esempi aperture file

- Esempi di apertura di file, supponendo di aver dichiarato:
`FILE* fp;`
- Apertura di un file di testo:
 - in lettura:
`fp = fopen("prova.txt", "r");`
 - in scrittura:
`fp = fopen("prova.txt", "w");`
 - in append:
`fp = fopen("prova.txt", "a");`
- Apertura di un file binario:
 - in lettura:
`fp = fopen("binario.dat", "rb");`
 - in scrittura:
`fp = fopen("binario.dat", "wb");`
 - in append:
`fp = fopen("binario.dat", "ab");`

Scrittura s file (testo)

- Esistono diverse funzioni per la scrittura di testo su file (fprintf, fputs, fputc). Per i nostri scopi, fprintf offre tutta la flessibilità necessaria. La sua signature è:

```
int fprintf(FILE *stream, char *format, ...);
```

- Parametri:
 - stream: file su cui scrivere
 - format: stringa di formato
 - ...: variabili da formattare secondo la stringa di formato
- Restituisce:
 - Il numero di caratteri scritti

Scrittura su file (binario)

- Per scrivere dei dati in formato binario (cioè non trasformati in un formato testuale, leggibile anche da un essere umano) si utilizza la funzione `fwrite`:
`int fwrite(void *ptr, int size, int nmemb, FILE *stream);`
- Parametri:
 - `ptr`: indirizzo dell'elemento da scrivere, o del primo elemento di un array se si desidera scrivere un array
 - `size`: dimensione dell'elemento da scrivere (tipicamente ottenuto usando `sizeof`)
 - `nmemb`: numero di elementi da scrivere (1 per un singolo elemento)
 - `stream`: file su cui scrivere.
- Restituisce:
 - Il numero di elementi scritti con successo

Scrittura su file (binario) – esempio

```
typedef struct {
    int a;
    int b;
} pippo;

char c;
pippo p;
int v[100];
FILE *fp;

fp = fopen("test.bin", "wb");

fwrite(&c, sizeof(char), 1, fp);
fwrite(&p, sizeof(pippo), 1, fp);
/* NOTA: non serve l'operatore indirizzo per scrivere
   un vettore */
fwrite(v, sizeof(int), 100, fp);

fclose(fp);
```

Lettura da tastiera

- Le due funzioni più usate per leggere input da tastiera sono `scanf` e `gets`. Sono delle “scorciatoie” per `fscanf` e `fgets`, ma presentano alcune differenze, per cui le tratteremo separatamente.
`int scanf(char *format, ...);`
- Parametri:
 - `format`: la stringa di formato
 - `...`: le variabili in cui andare a memorizzare i dati letti da tastiera
- Restituisce:
 - Il numero di campi letti con successo, in caso di errore restituisce 0 o EOF.
- Note:
 - La lettura di stringhe (`%s`) legge solo singole parole, per leggere una linea intera (compresa di spazi) è necessario utilizzare `fgets`.
 - Una `scanf` può leggere (e assegnare), più valori contemporaneamente, sfruttate questa caratteristica! Ad esempio, per leggere tre interi distinti:
`scanf(“%d %d %d”, &a, &b, &c);`
 - Ricordatevi del problema delle letture miste di interi, stringhe e `char`! Se il vostro programma “salta” delle richieste di input, provate a far precedere alla `scanf` una chiamata a `fflush(stdin)`; (soluzione non standard)

Lettura da tastiera

- Per la lettura di una stringa immessa da tastiera, il cui inserimento è terminato dalla pressione del tasto invio:

```
char *gets(char *s);
```

- Parametri:
 - s: la stringa in cui salvare i dati letti da tastiera
- Restituisce:
 - s stesso se la lettura ha avuto successo, altrimenti NULL
- Note:
 - gets rimuove il carattere di “a capo” (‘\n’) al termine della stringa letta da tastiera.

Lettura da file (testuale)

- La `fscanf` funziona esattamente come la `scanf`, ma opera su file.

```
int fscanf(FILE *stream, const char *format,  
...);
```

- Parametri:
 - `format`: la stringa di formato
 - `...`: le variabili in cui andare a memorizzare i dati letti da tastiera
- Restituisce:
 - Il numero di campi letti con successo, in caso di errore restituisce 0 o EOF.

Lettura da file (testuale)

```
char *fgets(char *s, int size, FILE *stream);
```

- Parametri:
 - s: la stringa in cui salvare i dati letti da tastiera
 - size: il numero massimo di caratteri + 1 da leggere
 - stream: il file da cui leggere
- Restituisce:
 - s stesso se la lettura ha avuto successo, altrimenti NULL
- Note:
 - fgets non rimuove il carattere di “a capo” (‘\n’) al termine della stringa letta da tastiera (al contrario di gets)

Lettura da file (testuale) – esempi

- Supponendo di avere un file che contiene un numero sconosciuto di linee di testo, ognuna contenente tre interi con segno, il metodo per leggere l'intero file è:

```
FILE* fp;
fp = fopen("nomefile.txt", "r");
/* ripetiamo il ciclo finché riusciamo a leggere con successo i 3 interi */
while (fscanf(fp, "%d %d %d", &a, &b, &c) == 3)
{
    /* elaborazione dei dati letti */
}
fclose(fp);
```

- Per leggere lo stesso file una linea alla volta:

```
FILE* fp;
char str[200];
fp = fopen("nomefile.txt", "r");
/* ripetiamo il ciclo finché fgets non riesce a leggere alcuna stringa */
while (fgets(str, 200, fp) != NULL)
{
    /* elaborazione dei dati letti */
}
fclose(fp);
```

Lettura da file (binario)

- Per leggere dei dati in formato binario si utilizza la funzione fread:

```
int fread(void *ptr, int size, int nmemb, FILE *stream);
```

- Parametri:
 - ptr: indirizzo a partire dal quale scrivere gli elementi
 - size: dimensione dell'elemento da leggere (tipicamente ottenuto usando sizeof)
 - nmemb: numero di elementi da leggere (1 per un singolo elemento) stream: file su cui scrivere.
- Restituisce:
 - Il numero di elementi scritti con successo

Lettura da file (binario) – esempio

- Supponendo di avere un file che contiene un numero sconosciuto di interi, il metodo per leggere l'intero file è:

```
FILE *fp;
int tmp;
fp = fopen("interi.bin", "rb");
while (fread(&tmp, sizeof(int), 1, fp) == 1)
{
    /* elaborazione dell'intero letto */
}
fclose(fp);
```

- Se, ad esempio, si sa che il file contiene esattamente 20 interi e si desidera memorizzarli tutti in un array:

```
FILE *fp;
int tmp[20];
fp = fopen("interi.bin", "rb");
fread(tmp, sizeof(int), 20, fp);
fclose(fp);
```

Operazioni su stringhe

- La libreria standard C mette a disposizione diverse funzioni per manipolare stringhe di caratteri. Per poterle utilizzare bisogna aggiungere all'inizio del proprio sorgente:

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

- Due tra le più utili sono:
 - `int strlen(char *s);`
restituisce la lunghezza di una stringa
 - `int strcmp(char *s1, char *s2)`
effettua un confronto tra `s1` e `s2` e restituisce:
 - 0 se le due stringhe sono uguali
 - Un numero negativo se `s1` è precedente, in ordine alfanumerico, a `s2`
 - Un numero positivo nel caso opposto

Miscellanea – gli errori comuni

- Non aggredite la traccia come se fosse un unico, enorme problema compatto, ma scomponetelo in sottoproblemi.
- Almeno all'inizio, non pensate a come scriverete il codice necessario, ma pensate a livello logico quali sono le operazioni da effettuare per rispondere ai requisiti della traccia. Una volta preparato il piano di battaglia, passate al codice.

Miscellanea – gli errori comuni

- Le operazioni matematiche tra tipi omogenei dà come risultato un tipo ad essi omogenei. Ad esempio:
`3 / 5 == 0`
- Se volete davvero ottenere un numero reale operando tra interi, dovete usare il cast:
`float f;`
`f = (float)3 / 5;`
- Gli array (e quindi le stringhe, che sono array di char) valgono già come indirizzi, per cui su di essi non va applicato l'operatore indirizzo & quando li si legge o scrive (fa eccezione `fwrite`, vedi esempi nelle slide precedenti)
- Gli assegnamenti possono essere effettuati solo tra tipi omogenei. Mentre questa regola è facile da applicare con le operazioni aritmetiche, grazie alle regole di conversione implicite, trae qualcuno in inganno con le stringhe. Il seguente codice non è valido:
`char s[50];`
`gets(s);`
`s[0] = "inizio"; // NO!`
questo perché `s[0]` è un singolo char, mentre "inizio" è un array di caratteri, che è un tipo differente.
- I confronti con `==`, `!=`, `<`, `<=`, `>`, `>=` possono essere effettuati solo sui tipi primitivi (char, short, int, float, ecc.). Per i confronti tra stringhe è necessario utilizzare `strcmp`.
- Ricordatevi che potete usare le funzioni di libreria, in particolare `strlen` e `strcmp`.