ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

File di Testo in C

Persistenza e File

Per memorizzare informazioni in modo persistente:

- È necessario sfruttare la memoria di massa
- ...Attraverso il concetto di File

Un file:

- È una astrazione fornita dal sistema operativo (dal File System)
- È una sequenza di record uniformi. Distinguiamo:
  - File binari (sequenze di byte)
  - File di testo (sequenze di caratteri)

Noi ci limiteremo a considerare i file di testo

Persistenza e File

Per utilizzare un file si utilizzano le funzionalità del SO

In pratica, si possono distinguere sempre tre fasi:

- Apertura del file
  - Richiesta di accesso al file al SO
  - Predisposizione di strutture dati per la sua gestione
- Lettura/scrittura di informazioni
  - Tipicamente bufferizzata
- Chiusura del file
  - Svuotamento del buffer di scrittura
  - Notifica di rilascio di risorse al SO
  - Deallocazione delle strutture dati interne

Persistenza e File

All'apertura si specifica come si intende utilizzare il file

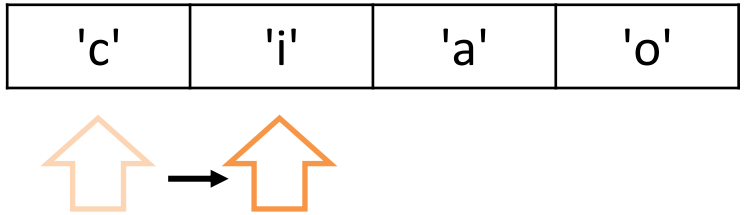
Ci sono tre modalità principali

- Lettura
  - È possibile (solo) leggere informazioni
  - Più processi possono accedere contemporaneamente
- Scrittura
  - Il contenuto corrente viene cancellato
  - È possibile (solo) scrivere informazioni
  - Un solo processo alla volta può accedere
- Aggiunta (append)
  - Come la scrittura, ma il contenuto corrente è preservato

Persistenza e File

In apertura, viene predisposto un cursore (astratto) sul file

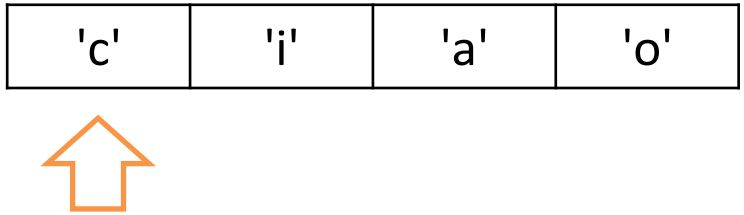
- Lettura e scrittura avvengono in corrispondenza del cursore
- Ad ogni operazione il cursore avanza di una posizione



Persistenza e File

In apertura, viene predisposto un cursore (astratto) sul file

- Lettura e scrittura avvengono in corrispondenza del cursore
- Ad ogni operazione il cursore avanza di una posizione



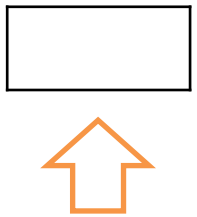
La posizione iniziale del cursore è determinata all'apertura

- In caso di apertura in lettura, il cursore è all'inizio

Persistenza e File

In apertura, viene predisposto un cursore (astratto) sul file

- Lettura e scrittura avvengono in corrispondenza del cursore
- Ad ogni operazione il cursore avanza di una posizione



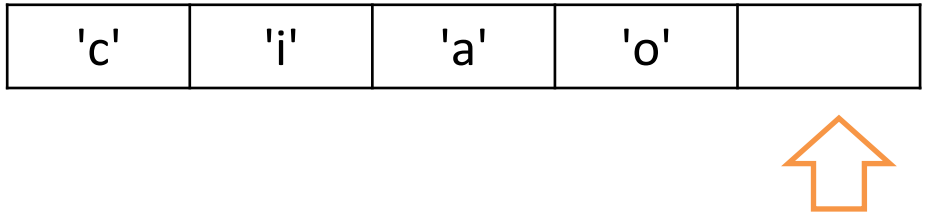
La posizione iniziale del cursore è determinata all'apertura

- In caso di apertura in lettura, il cursore è all'inizio
- In caso di apertura in scrittura, il cursore è all'inizio
  - Ricordate: il contenuto è cancellato

Persistenza e File

In apertura, viene predisposto un cursore (astratto) sul file

- Lettura e scrittura avvengono in corrispondenza del cursore
- Ad ogni operazione il cursore avanza di una posizione



La posizione iniziale del cursore è determinata all'apertura

- In caso di apertura in lettura, il cursore è all'inizio
- In caso di apertura in scrittura, il cursore è all'inizio
  - Ricordate: il contenuto è cancellato
- In caso di apertura in aggiunta, il cursore è alla fine

File di Testo in C: Apertura

Per aprire un file (di testo) in C si usa la funzione:

```
FILE* fopen(char *fname, char* mode);
```

- È definita in "stdio.h"
- Apre il file il cui percorso è specificato da "fname"
- "mode" è una stringa, che specifica la modalità di apertura
  - "r": apertura in lettura
  - "w": apertura in scrittura
  - "a": apertura in aggiunta "append"
- In caso di problemi, la funzione restituisce NULL
- ...Altrimenti restituisce un puntatore a FILE

File di Testo in C: Struttura FILE

FILE è una struttura (struct), definita in "stdio.h"

Contiene le informazioni necessarie all'utilizzo del file

- Il suo contenuto dipende dal compilatore
- È utilizzata in modo diretto soltanto dalle funzioni in "stdio.h"
- Per questa ragione, i suoi dettagli non ci interessano

L'utente definisce ed usa solo puntatori a FILE

- Il puntatore ottenuto in fase di apertura...
- ...Viene usato in tutte le operazioni su tale file

File di Testo in C: Controllo di Apertura

Quando si lavora con i file molte cose possono andare storte:

- Il file (da leggere) può non esistere
- Il file (da scrivere) può essere non accessibile (permessi)
- …

Per questa ragione è bene controllare l'esito di ogni apertura

Un esempio di apertura:

```
FILE *fp = fopen(<nome file>, <modo>);
if(fp == NULL) {
   /* Stampa messaggio di errore ed esci */
}
```

File di Testo in C: Controllo di Apertura

Quando si lavora con i file molte cose possono andare storte:

- Il file (da leggere) può non esistere
- Il file (da scrivere) può essere non accessibile (permessi)
- …

Per questa ragione è bene controllare l'esito di ogni apertura

Spesso le prime due operazioni vengono "compresse":

```
FILE *fp;
if((fp = fopen(<nome file>, <modo>)) == NULL) {
   /* Stampa messaggio di errore ed esci */
}
```

- Si sfrutta il fatto che "=" denoti il valore assegnato

File di Testo in C: Chiusura

Per chiudere un file in C si usa la funzione:

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

- Il puntatore a "FILE" specifica quale sia il file da chiudere
- Se l'operazione ha successo, la funzione restituisce 0
- In caso di problemi, restituisce EOF
  - EOF è una costante simbolica definita in "stdio.h"
- Prima della chiusura, tutti i buffer vengono svuotati
  - Le scritture pendenti sono trasferite in memoria di massa

File di Testo in C: Scrittura

Per scrivere su un file si può usare:

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

- Il puntatore a "FILE" specifica quale sia il file su cui scrivere
- Per il resto funziona esattamente come la printf!
  - "format" è la stringa di formato
  - Seguono le espressioni da cui prendere eventuali valori
  - Restituisce il numero di caratteri scritti

File di Testo in C: Lettura

Per leggere su un file si può usare:

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

- Il puntatore a "FILE" specifica quale sia il file su cui scrivere
- Per il resto funziona esattamente come la scanf!
  - "format" è la stringa di formato
  - Seguono gli indirizzi su cui scrivere i valori
  - Restituisce il numero di conversioni effettuate

File di Testo in C: Canali Standard

La somiglianza delle funzioni non è un caso

Quando un programma è avviato, vengono aperti tre file canonici:

- stdout: in scrittura, di norma agganciato al terminale
- stdin: in lettura, di norma agganciato al terminale
- stderr: in scrittura (per errori), di norma agganciato al terminale

Si chiamano anche canali standard di I/O:

- `scanf(...)` è equivalente a `fscanf(stdin, ...)`
- `printf(...)` è equivalente a `fprintf(stdout, ...)`

File di Testo in C: Alcune funzioni di I/O

Tutte le funzioni di I/O viste finora...

...Sono casi particolari di funzioni che operano su file di testo:

Funzione	Funzione Generale
`int getchar();`	`int fgetc(FILE* f);`
`int putchar();`	`int putc(FILE* f);`
`int gets(char *s);`	`int fgets(FILE* f, char* s);`
`int put(char* s);`	`int fputs(FILE* f, char *s);`
`int printf(char *fmt, ...);`	`int fprintf(FILE* f, chr *fmt, ...);`
`int scanf(char *fmt, ...);`	`int fscanf(FILE *f, char *fmt, ...);`

File di Testo in C: Funzioni Specifiche

Ci sono poi alcuni funzioni specifiche dei file:

```
void rewind(FILE *f);
```

- Riporta il cursore all'inizio del file
- La useremo molto (ma non in questa lezione)

```
int feof(FILE *f);
```

- Restituisce "vero" se si è raggiunta la fine del file
- La useremo di rado (ci sono alternative preferibili)

File di Testo in C: Funzioni Specifiche

Ci sono poi alcuni funzioni specifiche dei file:

```
void perror(char *s);
```

- Stampa un messaggio di errore su stderr
- Di solito preferiremo `fprintf(stderr, ...)`

```
fseek(...) / ftell(...)
```

- Permettono di ottenere e modificare la posizione del cursore
- Non le useremo

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Esempio: Lettura Carattere per Carattere

Esempio: Lettura Carattere per Carattere

Per leggere un file carattere per carattere, possiamo usare:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char c, nome[] = "testo.txt";
    if((fp = fopen(nome, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }

    while(fscanf(fp, "%c", &c) == 1) {
        printf("%c", c);
    }
    fclose(fp);
}
```

Esempio: Lettura Carattere per Carattere

Per leggere un file carattere per carattere, possiamo usare:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char c, nome[] = "testo.txt";
    if((fp = fopen(nome, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }

    while(fscanf(fp, "%c", &c) == 1) {
        printf("%c", c);
    }
    fclose(fp);
}
```

percorso relativo
(in questo caso si suppone che il file sia nella stessa cartella dell'eseguibile)

Esempio: Lettura Carattere per Carattere

Per leggere un file carattere per carattere, possiamo usare:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char c, nome[] = "testo.txt";
    if((fp = fopen(nome, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }

    while(fscanf(fp, "%c", &c) == 1) {
        printf("%c", c);
    }
    fclose(fp);
}
```

Apertura e controllo su una singola riga

Esempio: Lettura Carattere per Carattere

Per leggere un file carattere per carattere, possiamo usare:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char c, nome[] = "testo.txt";
    if((fp = fopen(nome, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }

    while(fscanf(fp, "%c", &c) == 1) {
        printf("%c", c);
    }
    fclose(fp);
}
```

Stampa su stderr in caso di errore

Esempio: Lettura Carattere per Carattere

Per leggere un file carattere per carattere, possiamo usare:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char c, nome[] = "testo.txt";
    if((fp = fopen(nome, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }

    while(fscanf(fp, "%c", &c) == 1) {
        printf("%c", c);
    }
    fclose(fp);
}
```

exit (da stdlib.h) per terminare

Esempio: Lettura Carattere per Carattere

Per leggere un file carattere per carattere, possiamo usare:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char c, nome[] = "testo.txt";
    if((fp = fopen(nome, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }

    while(fscanf(fp, "%c", &c) == 1) {
        printf("%c", c);
    }
    fclose(fp);
}
```

procedo finché la lettura ha successo
(scanf restituisce il numero di conversioni effettuate)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Esempio: Lettura di Interi

Esempio: Lettura di Interi

Possiamo usare lo stesso approccio:

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

int main() {
    FILE* fp;
    int n;
    char nome[] = "numeri.txt";
    if((fp = fopen(nome, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }
    while(fscanf(fp, "%d", &n) == 1) {
        printf("%d\n", n);
    }
    fclose(fp);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Esempio: Scrittura di Stringhe

Esempio: Scrittura di Stringhe

Scrittura di stringhe (una per linea), fino alla parola "fine":

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char s[256], nome[] = "out.txt";
    if((fp = fopen(nome, "w")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }
    do {
        scanf("%s", s);
        fprintf(fp, "%s\n", s);
    } while(strcmp(s, "fine") != 0);
    fclose(fp);
}
```

Esempio: Scrittura di Stringhe

Scrittura di stringhe (una per linea), fino alla parola "fine":

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char s[256], nome[] = "out.txt";
    if((fp = fopen(nome, "w")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }
    do {
        scanf("%s", s);
        fprintf(fp, "%s\n", s);
    } while(strcmp(s, "fine") != 0);
    fclose(fp);
}
```

apertura in scrittura
(cancella il file, nel caso esista)

Esempio: Scrittura di Stringhe

Scrittura di stringhe (una per linea), fino alla parola "fine":

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    FILE* fp;
    char s[256], nome[] = "out.txt";
    if((fp = fopen(nome, "w")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nome);
        exit(1);
    }
    do {
        scanf("%s", s);
        fprintf(fp, "%s\n", s);
    } while(strcmp(s, "fine") != 0);
    fclose(fp);
}
```

quando la parola letta è "fine", interrompiamo il ciclo

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Esempio: Lista di Account

Esempio: Lista di Account

Si vogliono gestire gli account di un servizio di chat

Occorre perciò leggerne le informazioni dal file "partecipanti.txt"

I dati sono formattati nel modo seguente:

```
email tipo n_accessi
```

- "email" è una stringa senza spazi di al più 255 caratteri
- "tipo" è un carattere e precisamente:
  - 'A' per gli amministratori
  - 'S' per gli studenti
- "n_accessi" è un intero

Esempio: Lista di Account

Si definisca una struttura "partecipante" atta a contenere i dati

Si definisca una funzione:

```
int leggi_partecipanti(char *nomefile, partecipante *lista)
```

Che legga dal file "nomefile" i dati su una lista di partecipanti

- I dati letti vanno scritti nell'array "lista"
- La funzione deve restituire il numero di elementi letti

Si scriva un main di prova

- Si assuma che il max numero di elementi nel file sia 300

Soluzione

Dipendenze e definizione del tipo di dato

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

#define NEMAIL 256
#define NMAX 300

typedef struct {
    char email[NEMAIL];
    char tipo;
    int n_accessi;
} partecipante;
```

Soluzione

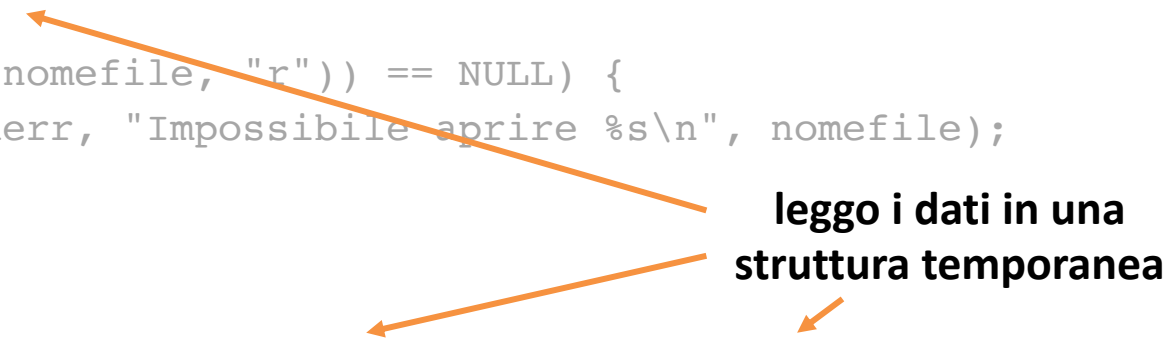
Codice della funzione

```
int leggi_partecipanti(char *nomefile, partecipante *lista) {
    FILE *fp;
    char s_tipo[2];
    int i;
    partecipante p;
    if((fp = fopen(nomefile, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nomefile);
        exit(1);
    }
    i = 0;
    while(fscanf(fp, "%s%s%d", p.email, s_tipo, &p.n_accessi) == 3) {
        p.tipo = s_tipo[0];
        lista[i++] = p;
    }
    fclose(fp);
    return i;
}
```

Soluzione

Codice della funzione

```
int leggi_partecipanti(char *nomefile, partecipante *lista) {
    FILE *fp;
    char s_tipo[2];
    int i;
    partecipante p;
    if((fp = fopen(nomefile, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nomefile);
        exit(1);
    }
    i = 0;
    while(fscanf(fp, "%s%s%d", p.email, s_tipo, &p.n_accessi) == 3) {
        p.tipo = s_tipo[0];
        lista[i++] = p;
    }
    fclose(fp);
    return i;
}
```



Soluzione

Codice della funzione

```
int leggi_partecipanti(char *nomefile, partecipante *lista) {
    FILE *fp;
    char s_tipo[2];
    int i;
    partecipante p;
    if((fp = fopen(nomefile, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nomefile);
        exit(1);
    }
    i = 0;
    while(fscanf(fp, "%s%s%d", p.email, s_tipo, &p.n_accessi) == 3) {
        p.tipo = s_tipo[0];
        lista[i++] = p;
    }
    fclose(fp);
    return i;
}
```

La lettura di caratteri singoli con scanf è sempre problematica. Per evitarli, uso una stringa

Soluzione

Codice della funzione

```
int leggi_partecipanti(char *nomefile, partecipante *lista) {
    FILE *fp;
    char s_tipo[2];
    int i;
    partecipante p;
    if((fp = fopen(nomefile, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nomefile);
        exit(1);
    }
    i = 0;
    while(fscanf(fp, "%s%s%d", p.email, s_tipo, &p.n_accessi) == 3) {
        p.tipo = s_tipo[0];
        lista[i++] = p;
    }
    fclose(fp);
    return i;
}
```

Il carattere con il tipo sarà il primo nella stringa

Soluzione

Codice della funzione

```
int leggi_partecipanti(char *nomefile, partecipante *lista) {
    FILE *fp;
    char s_tipo[2];
    int i;
    partecipante p;
    if((fp = fopen(nomefile, "r")) == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nomefile);
        exit(1);
    }
    i = 0;
    while(fscanf(fp, "%s%s%d", p.email, s_tipo, &p.n_accessi) == 3) {
        p.tipo = s_tipo[0];
        lista[i++] = p;
    }
    fclose(fp);
    return i;
}
```

A questo punto posso copiare la struttura temporanea nell'array

Soluzione

Codice del main

```
int main() {
    partecipante lista[NMAX];
    int i, n;

    n = leggi_partecipanti("partecipanti.txt", lista);

    printf("LISTA:\n");
    for(i = 0; i < n; ++i) {
        printf("%s %c %d\n", lista[i].email,
                             lista[i].tipo,
                             lista[i].n_accessi);
    }
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Andrea Acquaviva < andrea.acquaviva@unibo.it>
Michele Lombardi <michele.lombardi2@unibo.it
Andrea Borghesi <andrea.borghesi3@unibo.it>
Giuseppe Tagliavini <giuseppe.tagliavini@unibo.it>
Allegra De Filippo < allegra.defilippo@unibo.it>