PROGRAMMAZIONE

Sequenze – File binari e file di testo

Algoritmi su file

Sequenze

Struttura dati SEQUENZA

- Insieme finito di dati omogenei tra loro in relazione d'ordine di numerosità non nota
  - Struttura lineare
    - Si sviluppa in una sola dimensione
  - Organizzazione di tipo sequenziale
    - L'unico accesso diretto è alla testa della sequenza
    - Per accedere ad un generico elemento occorre scandire la struttura dalla testa sino alla posizione dell'elemento
      - Se sono su un certo elemento, si può passare direttamente solo all'elemento successivo
  - Dimensione che può variare in esecuzione
    - Non si è obbligati a stabilire a priori la cardinalità
    - Possibilità di inserimento di elementi
    - Possibilità di eliminazione di elementi

Sequenze (*)

- Modalità di accesso
  - Individuazione della posizione in cui operare nella sequenza
    - Accesso per scansione
    - Accesso agli estremi
- Operazioni consentite
  - Possibilità di agire nella posizione individuata
    - Lettura del valore di un componente
    - Inserimento di un nuovo componente
    - Rimozione di un componente
    - Aggiornamento del valore di un componente

(*) trattazione limitata ai file sequenziali

Sequenze
Realizzazione

- Possono essere realizzate in memoria centrale
- Astrazione dei dati memorizzati su memorie secondarie e su supporti esterni
- Consentono di comunicare con l'ambiente operativo
  - File sequenziali (*archivi*)
    - Particolarmente importanti per la conservazione permanente di grandi quantità di dati
  - Supporti che implicano l'accesso sequenziale
    - Nastri e cassette magnetici
    - Nastri cartacei e schede
    - Stampanti (listati da)
    - Dischi e tamburi magnetici (relativamente ad ogni pista di memorizzazione)

Sequenze
Dichiarazione

- A livello di programma bisogna disporre di
  - costrutti per la definizione
    - Della struttura (identificatore FILE)
    - Del tipo degli elementi
  - un meccanismo per poter
    - accedere al e operare sul

    singolo elemento della struttura

Buffer Tampone

- "Speciale variabile" tramite cui è realizzato il meccanismo di accesso
  - Stesso tipo delle componenti della sequenza
  - Allocata e riservata all'atto della dichiarazione della struttura sequenziale
  - Contiene, di volta in volta, l'elemento da elaborare
- **Tutte** le operazioni applicabili alle sequenze (creazione, concatenazione, estrazione) **sono ricondotte** ad operazioni elementari attraverso il meccanismo del buffer tampone

Buffer Tampone

- Utilizzato per
  - Isolare l'elemento della sequenza su cui si opera
  - Prelevare o aggiungere componenti al file sequenziale
    - Il numero delle componenti (lunghezza della sequenza) può variare durante l'esecuzione del processo di calcolo
    - Le componenti vanno aggiunte o prelevate in ordine assolutamente sequenziale
- Solitamente indicato con
  - *nome_sequenza*↑ oppure ***nome_sequenza*** **(in C)**

  (puntatore all'area di memoria buffer associata al file)

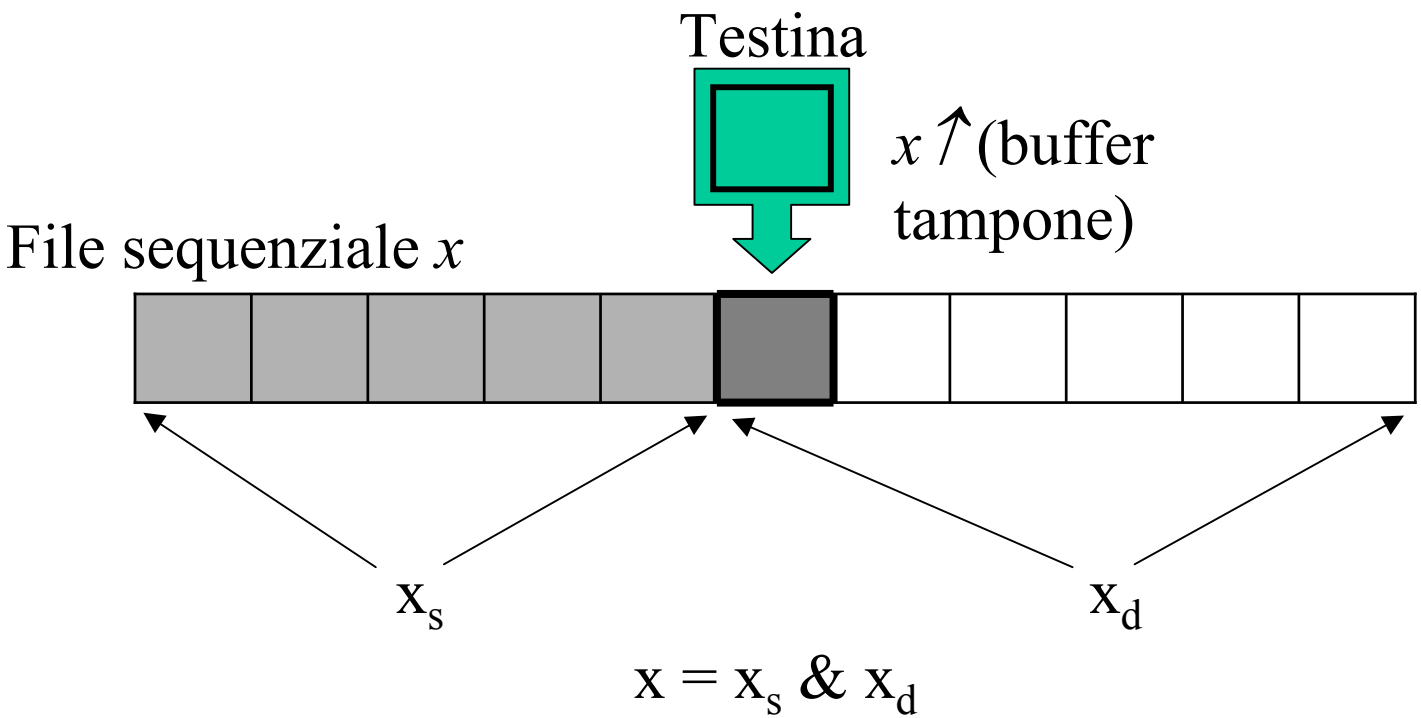
Esempi in C

```
#include  <stdio.h>
int main(void)
{
    int c;
    FILE *fp;
    fp = fopen ("mio_file", "r");
    …
    fclose (fp);
```

fp sta per file pointer
"r": file aperto in sola lettura

```
…
struct studente {char matricola[8],
                 char cognome[20],
                 char nome[20],
                 int voto_esonero;}
FILE *fp;
struct studente uno_studente;
fp = fopen ("corsoB.dat", "r");
…
n = fread (&uno_studente,
     sizeof(struct studente), 1, fp);
…
fclose (fp);
```

File Sequenziali
Organizzazione



File Sequenziali
Gestione

- Accesso sequenziale
  - È possibile esclusivamente
    - Posizionarsi all'inizio del file
    - Passare direttamente da un elemento *al successivo*
      - Posizione corrente modificabile attraverso opportuni operatori
- Due possibili stati
  - Stato di Creazione
    - È consentito solo inserire elementi in modo sequenziale
  - Stato di Ispezione
    - E' consentito solo leggere elementi in modo sequenziale

File Sequenziali
Gestione

- I procedimenti di costruzione e di scansione di un file sono *distinti,* cioè non possono essere mescolati
- In ogni istante, un file F si trova in uno dei due stati
  - Necessità di un costrutto per specificare lo stato di un file
- Per costruire un file F
  - F deve trovarsi nello stato di creazione
  - Le componenti vanno consecutivamente appese in coda a partire dalla sequenza vuota
- Per scandire un file F
  - F deve trovarsi nello stato di ispezione
  - Le componenti vanno lette sequenzialmente a partire dalla prima

File Sequenziali
Operatori basici

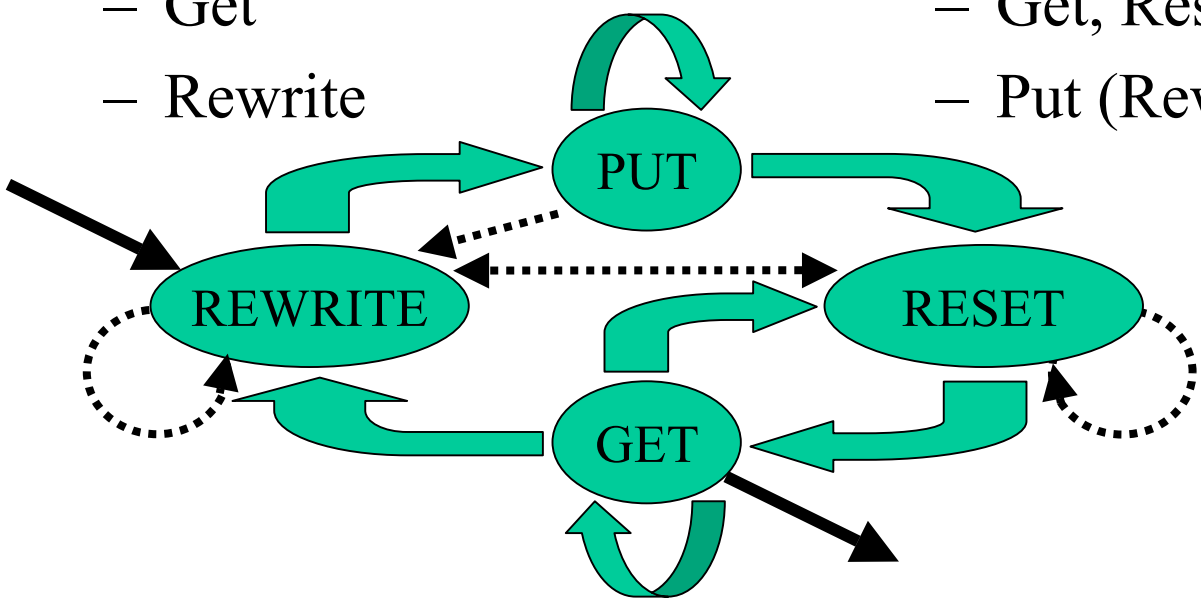
- Sono definibili su una sequenza *x*
  - Rewrite(x) Inizializzazione in scrittura (specifica lo stato di creazione)
    - x = < >
  - Reset(x) Inizializzazione in lettura (specifica lo stato di ispezione)
    - x_s = < > x_d = x *x = first(x_d)

File Sequenziali
Operatori basici

- E gli operatori basici
  - Put(x) Inserimento di un elemento
    - Appende il contenuto del buffer alla fine del file
    - x = x & <*x>
  - Get(x) Passaggio all'elemento successivo
    - Trasferisce nel buffer il contenuto dell'elemento successivo
    - $x_s = x_s$ & <first(x_d)> x_d = rest(x_d) *x = first(x_d)

File Sequenziali
Operatori

- Operazione
  - Put
  - Reset
  - Get
  - Rewrite
- Operazione successiva
  - Put, Reset (Rewrite)
  - Get (Reset, Rewrite)
  - Get, Reset, Rewrite
  - Put (Rewrite, Reset)



File Sequenziali
Ciclo di Vita

- All'inizio il file F è solo dichiarato
  - Si entra nel grafo
- La prima cosa da fare è impostarlo in scrittura
  - Dopo di ciò è possibile fare un insieme di inserimenti
- In seguito, per leggere l'informazione inserita, occorre cambiare lo stato di F e ripartire dall'inizio in lettura
  - Dopo alcune letture, si potrà
    - terminare la manipolazione
    - ricominciare ad inserire elementi (att.ne: si parte da file vuoto)
    - annullare il file costruito

Sequenze
Fine Sequenza (end-of-file)

- Dimensione delle strutture sequenziali ignota
  - Necessità di un segnalatore che indichi che sono stati esaminati tutti gli elementi
- Funzione booleana **fine_sequenza(·)** **(In C, feof)**
  - Applicata ad una sequenza, restituisce
    - Il valore *vero* se si è alla fine della sequenza
    - Il valore *falso* altrimenti
  - fine_sequenza(x) = *vero* sse $x_d = <>$
    - Dopo ogni operazione di scrittura
    - Dopo aver esaminato tutti gli elementi del file x

File vs. Array

- Elementi omogenei organizzati secondo una relazione d'ordine
  - Accesso agli elementi tramite scansione
  - Numero di elementi non limitato a priori
  - Residente in memoria secondaria
    - Permanente
- Elementi omogenei organizzati secondo una relazione d'ordine
  - Accesso diretto a qualunque elemento
  - Numero di elementi fissato a priori
  - Residente in memoria principale
    - Volatile

File
Strutturazione

- I file non possono essere usati come componenti di tipi strutturati
- I file possono essere composti da elementi di qualunque tipo
  - Anche strutturati
  - Non file

File Esterni

- I file elaborati da un programma vengono memorizzati su memoria secondaria
  - File logico
    - File visto dalla prospettiva del programma
  - File fisico
    - File reale in memoria secondaria
- In C, la diversità dei dispositivi è resa trasparente al programmatore tramite la gestione bufferizzata dell'I/O dei file che consente una trattazione unificata attraverso il concetto di *stream* o *flusso di comunicazione*

File Esterni

- Necessità di associare a ciascun file logico un corrispondente file fisico
  - Caso 1: file già presente
  - Caso 2: file creato dal programma
- Dipendono dall'implementazione
  - Primitiva di associazione (*fopen* in C, *assign* in Pascal,…)
  - Necessità di chiusura del file (*fclose* in C, *close* in TurboPascal,…)

File
Errori Comuni nel trattamento file

- Errata modalità di apertura
  - Caso 1: apertura in modalità "*w*" per un file preesistente, ma il programmatore intendeva conservarne il contenuto
  - Caso 2: apertura in modalità "*r*" per un file inesistente
- Tentativo di operare sul file senza (o prima di) averlo aperto
- Scansione non governata da un test di *feof*
  - Tentativo di lettura oltre la fine dei dati
- Usare un puntatore sbagliato per far riferimento ad un file
- *...la lista non è esaustiva*

Il trattamento dei file in C

- Il programmatore inoltra al Sistema Operativo le richieste di manipolazione dei file
- E' il SO che gestisce la tabella dei file aperti dai programmi in esecuzione
  - Nome file, modalità di apertura, indicatore posizione corrente, indicatore di EOF, indicatore di errore
- Nel linguaggio C
  - La *standard library* contiene funzioni per la gestione dei file implementate ad hoc per il SO: apertura, chiusura, cancellazione, ridenominazione,…

Il trattamento dei file in C

Il C non impone una struttura ai file

- Spetta al programmatore imporre una struttura di record ad un file per le esigenze applicative

1. Dichiarare una variabile di tipo puntatore
2. Aprire uno *stream* o *flusso di comunicazione* (si associa lo *stream* al file)
   - Binario o Di testo
     - Sequenza di byte oppure sequenza di caratteri ASCII
   - Preesistente o da creare
   - Specifica della modalità di utilizzo
3. Elaborare il file secondo le esigenze
4. Chiudere lo *stream* (si elimina la associazione tra file e *stream*)

Il trattamento dei file in C

- fopen:
  - Scopo: Apertura di uno *stream*
    - File esistente
    - File da creare
  - Effetto: Assegnamento dell'indirizzo del descrittore del file alla variabile puntatore associata al file; ciò consente di riferire il file nelle operazioni successive
  - Nota: il SO inserirà un elemento nella tabella dei file aperti
- ***FILE** *fopen (nomefile, modalità);*
  - varie modalità di apertura (sia per file binari che per file di testo): *r* sta per read, *w* sta per write, *a* sta per append, *b* specifica un file binary, + sta per indicare modalità aggiornamento, ovvero sia lettura che scrittura

Il trattamento dei file in C

- fopen:
  - Quando si apre un file in *sola lettura ("r")*, se il file non esiste, la funzione di apertura non ha successo, mentre se il file esiste; l'indicatore di posizione è impostato all'inizio del file
  - Quando un file viene aperto in scrittura *("w"),* se il file non esiste, viene creato, mentre se il file esiste già, i dati precedenti vengono "persi" e i nuovi dati sono sovrascritti a partire dall'inizio del file; in entrambi i casi l'indicatore di posizione è impostato all'inizio del file
  - Quando si apre un file in modalità *append ("a")*, se il file non esiste, verrà creato, mentre se il file esiste già, i dati inseriti verranno scritti di seguito ai dati preesistenti; l'indicatore di posizione è impostato alla fine del file

Il trattamento dei file in C

- fclose:
  - Scopo: chiusura di un flusso di comunicazione
  - Effetto
    - Assegnamento del valore NULL alla variabile puntatore associata al file e rilascio del descrittore del file
    - Restituzione del valore 0 se l'operazione viene eseguita correttamente, altrimenti restituzione del valore EOF (EOF è una costante definita in *stdio.h*)
  - Nota: il SO eliminerà la riga corrispondente dalla tabella dei file aperti
- ***int*** *fclose (FILE *fp);*

Il trattamento dei file in C

- fclose:
  - La funzione svuota nel file i dati eventualmente rimasti nel buffer del disco e, dopo, esegue una richiesta al Sistema Operativo di chiusura dello stream associato al file.
  - fclose() entrerà in errore se il file è già stato chiuso, quando un disco viene prematuramente estratto dal drive o quando non vi è più spazio sul disco.
  - Si può utilizzare la funzione standard *ferror*()

Il trattamento dei file in C

- Tutti i file vengono chiusi automaticamente, se il programma termina normalmente, da main() che ritorna al sistema operativo o da una chiamata a *exit*().
- I file non vengono invece chiusi se un programma termina in modo anormale, ad esempio quando blocca il sistema o quando viene eseguita una chiamata ad *abort*().
- Ogni stream a cui è associato un file ha una propria struttura di controllo di tipo FILE. Si faccia attenzione a non modificare mai questo blocco di controllo del file.

Schemi algoritmici

- Ispezione di un file X

```
FILE *fp; /*dichiarativa puntatore*/
fp = fopen("C:/x.txt", "r");
          /* apertura in lettura*/
…/* controlli */
if (fp== NULL)
    printf ("Errore di apertura
    file");
else  while non-fine-file do
    /* ciclo di scansione */
        {  fread (….., fp) ;
           azioni-varie;}
fclose(fp);
```

- Creazione di un file X

```
/* inserire le dichiarative
   necessarie */
FILE *fp;
fp = fopen("C:/x.txt", "w");
/* apertura in scrittura*/
…/* controlli */
while predicato_per
   _continuare do
       {azioni-varie;
          fwrite (….., fp) ;
         }
fclose(fp);
```

Algoritmi su File

ALGORITMI BASE (copia, confronto, aggiornamento,…)

RICERCA

FUSIONE

Algoritmi su FILE

- Algoritmi basilari di creazione e aggiornamento, ed inoltre
  - Fusione
  - Ordinamento
- Supponiamo di operare su file di record
  - Identificazione tramite campo chiave numerico intero
    - Struttura del record: | *chiave* | …altri dati… |
- Operazione dominante rispetto alla quale valutare la complessità
  - Lettura/scrittura di un elemento di un file

Algoritmi su FILE

- Non sono disponibili operatori globali sui file (a meno di voler considerare tali gli operatori di apertura e chiusura)
  - Creazione di un file
    - Apertura in scrittura, ciclo di scrittura nel file
  - Copia del file *f* nel file *g*
    - Assegnamento fra file
  - Stampa il contenuto di un file
    - Apertura in lettura, ciclo di scansione sino a EOF: leggi e scrivi su Standard Output
      - Copia del file sul file Standard Output
  - Confronto di uguaglianza fra 2 file
    - Apertura in lettura, Ciclo di scansione: verifica di uguaglianza elemento per elemento (controllo di fine-file, devono terminare insieme)

Algoritmi su FILE

- Aggiornamento
  - Da includere tra gli algoritmi basilari
- Ricerca
- Fusione
- Ordinamento

Algoritmo di copia

Procedura

- Trasferisce il valore di un elemento alla volta dal file che appare come primo parametro al file che appare come secondo parametro
  - Molto semplice
    - Controllo di fine file
  - **void** copia(**FILE** *fp, *gp);

{
apri il file F in lettura;
apri il file G in scrittura;
while *non è finito il file F* ***do***
    leggi un elemento da F;
    scrivi in G l'elemento appena letto;
}

Aggiornamento File
Aggiunta

- Aggiunta di elementi ad un file esistente
  - Non vanno cancellati gli elementi esistenti
    - Dopo averli scorsi in lettura non si può scrivere
  - Necessità di un file di appoggio

Aprire in lettura il file originale ed in scrittura il file di appoggio
Finché non si è raggiunta la posizione di inserimento ed il file non è finito
    copiare un elemento dal file originale a quello di appoggio
Inserire l'elemento da aggiungere nel file di appoggio
Finché non si è raggiunta la fine del file
    copiare un elemento dal file originale a quello di appoggio
Riversare il file di appoggio nel file originale (copia)

- Il C mette a disposizione la modalità di apertura in aggiornamento con scrittura alla fine del file: *append*

Aggiornamento File
Modifica di un record

- Modifica di un elemento
  - Bisogna individuarne la posizione
    - Dopo avere scorso il file in lettura non vi si può scrivere
  - Necessità di un file di appoggio

Aprire in lettura il file originale ed in scrittura il file di appoggio
Finché non si è raggiunto l'elemento da modificare
    copiare un elemento dal file originale a quello di appoggio
Inserire l'elemento modificato nel file di appoggio
Finché non si è raggiunta la fine del file
    copiare un elemento dal file originale a quello di appoggio
Riversare il file di appoggio nel file originale (copia)

- Il C consente l'accesso diretto tramite la *fseek*

Algoritmi su File

- E' il programmatore C che impone la struttura agli elementi di un file

```
int tipo_chiave;              /* per ordinamento, ricerca e fusione */
struct qualche_tipo {
                        /* …dati…*/
                        };
 struct tipo_rec {tipo_chiave chiave;
                        qualche_tipo resto_record;
                        } ; /* fine della struttura */
FILE *fp;
struct tipo_rec elemento;
```

Uso di *fread* e di *fwrite*

Algoritmi su File

- Indicheremo con < la relazione d'ordine definita sui record del file
  - Basata sul campo chiave
  - Implementabile come funzione booleana

    function minore(x, y: tipo_rec): **boolean**; **true** sse x < y
- Per generalità definiamo un tipo *tipo_chiave* per il campo chiave
  - Può essere non strutturato o una stringa o comunque un tipo sui cui valori cui poter usare l'operatore <
    - Nel caso più semplice coinciderà con l'elemento del file
      - Es. chiave int e file costituito da elementi interi

Fusione di File

- Dati due file di record, entrambi ordinati in modo crescente sul campo chiave, produrre un unico file ottenuto dalla fusione dei primi due
  - Il risultato è ancora un file di record ordinato in modo crescente sul campo chiave
  - La fase di fusione vera e propria termina quando finisce uno dei file
    - Uso di una variabile booleana

```
void file_merge(FILE *in1_p, *in2_p, *out_p);     /* prototipo C */
   int    fine_in1_in2;
```

Fusione di File

- Dimensioni dei file non note a priori
  - Algoritmo ottimizzato per la fusione di array ordinati è inadatto
  - L'algoritmo di fusione "ingenuo" è più appropriato
  - Sostituzione delle condizioni
    - $i \leq m$ (fine primo array)
    - $j \leq n$ (fine secondo array)

    con dei test EOF per riconoscere la terminazione degli insiemi di dati

Fusione di File
codice Pascal

```
begin
reset(in1); reset(in2); rewrite(out); fine_in1_in2 := eof(in1) or eof(in2);
while not fine_in1_in2 do
    begin
    if in1^.chiave < in2^.chiave then
            begin    out^ := in1^; get(in1); fine_in1_in2 := eof(in1)          end
    else
            begin    out^ := in2^; get(in2); fine_in1_in2 := eof(in2)          end;
    put(out);
    end;
while not eof(in2) do       begin    out^ := in2^; put(out); get(in2)       end;
while not eof(in1) do       begin    out^ := in1^; put(out); get(in1)       end
end;
```

Ordinamento NaturalMerge

- Ordinare un file di record in modo tale che i record risultino sistemati in base all'ordine ascendente del campo chiave
  - Possibilità di riusare le idee relative all'ordinamento di array attraverso fusioni successive
    - *Natural merging*

Lo farete in ASD

NMS
Anno accademico 2005-2006

- Impostazione concettuale del NMS
  - Rif. MergeSort per array
- Algoritmo principale
  - Inglese strutturato o pseudo-codice
- La codifica in C è opzionale

File Testo

- Particolari file i cui componenti sono caratteri
  - Utili particolarmente per astrarre i diversi dispositivi periferici e consentire al programmatore un trattamento unico indipendente dal particolare dispositivo
  - In alcuni linguaggi esiste un apposito tipo file (text)

- Testo tipicamente suddiviso in linee
  - Ripercussioni sulla tipica struttura dei programmi che elaborano file testo
    - Struttura ciclica a 2 livelli

Mentre non è finito il file
    Mentre non è finita la linea
        Elabora i caratteri che compongono la linea

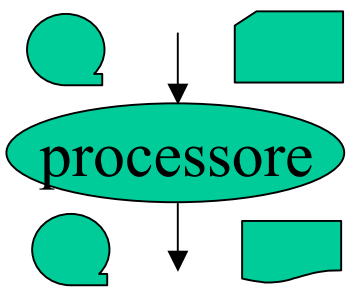
File Testo
Suddivisione in Linee

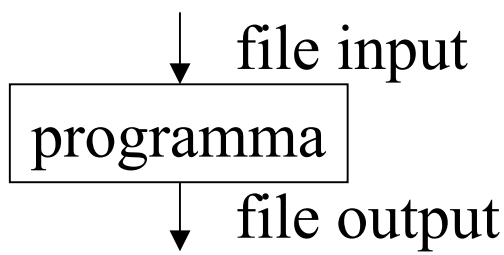
- Diverse possibili rappresentazioni
  - Separazione di due linee consecutive indicata tramite caratteri di controllo
    - *CR* (ritorno carrello)
    - *LF* (interlinea)
  - Uso di operatori speciali che gestiscono le linee
    - Validi solo per i file testo
    - Basati sull'esistenza di una funzione booleana *fine_linea*(*x*)
      - Indica se è stata raggiunta la fine della linea corrente nel file di testo *x – eol(x)*
    - ***newline*** in C

File Testo
Ingresso/Uscita

- Le operazioni di ingresso/uscita possono essere viste come operazioni di lettura e scrittura su particolari file testo
  - Un processo di calcolo che utilizza supporti di dati
    - Nastri, dischi, schede, listati

  è un processo di trasformazione di un file testo di ingresso (*input*) in un file testo di uscita (*output*)





File di I/O standard

- Tutti i linguaggi di programmazione trattano, in maniera più o meno esplicita, almeno 2 tipi di file sequenziali (*)
  - File di Ingresso
  - File di Uscita
- Automaticamente aperti all'atto dell'esecuzione
  - Non c'è bisogno di inizializzarli prima dell'uso

(*) In C esistono tre flussi standard: *stdin, stdout* e *stderr (*generalmente associati alla tastiera (il primo) e al video (gli altri due))

Operatori di Ingresso/Uscita

- Realizzano operazioni sui file implicitamente definiti nel programma
- Consentono di leggere/scrivere oggetti di tipi diversi dal testo
  - Necessità di conversioni
    - Effettuate automaticamente
    - Trasparenti all'utente

Operatori di Ingresso/Uscita

scanf (stringa-di-controllo_formato, altri_parametri);

- Opera sul file *stdin*
- Equivale a:

fscanf (*stdin*, stringa-di-controllo_formato, altri_parametri)

printf (specifiche-di-formato, altri_parametri);

- Opera sul file *stdout*
- Equivale a:

fprintf (*stdout*, specifiche-di- formato, altri_parametri)

Operatori di Ingresso/Uscita
Bufferizzazione

- Il testo in ingresso viene
  - Temporaneamente salvato in un'area di memoria interna
    - Possibilità di apportare correzioni
      - Tasto <*backspace*> (←)
  - Convertito a fine linea
  - Trasferito al programma
- Il testo in uscita viene
  - Convertito
  - Temporaneamente salvato in un'area di memoria interna
    - Possibilità di aggiunte
  - Trasferito all'esterno a fine linea

Operazioni di lettura e scrittura su file in C

- Sono possibili quattro differenti modi
  - Operazioni formattate
    - Specifica della formattazione con cui i dati devono essere letti/scritti
      - scanf, printf (sui file *stdio* e *stdout*)
      - fscanf, fprintf (coinvolgono il file precisato dall'utente tramite il puntatore fp)
    - Restituiscono il numero di elementi letti/scritti oppure un numero negativo in caso di errore
  - Accesso ai dati per blocco
  - Accesso ai dati per carattere
  - Accesso ai dati per linea

Accesso per blocchi (in C)

- Lettura/scrittura di un intero blocco di dati testuali o binari: *fread* e *fwrite*
  - ***int*** *fread(**void** *ptr, dimelem, numelem, FILE *fp);*

    Legge un blocco di dati dal file cui fa riferimento *fp* e li memorizza nella zona di memoria puntata da *ptr*, il blocco è composto da *numelem* elementi ciascuno di dimensione *dimelem* in byte
  - ***int*** *fwrite(**void** *ptr, dimelem, numelem, FILE *fp);*

    Scrive un blocco di dati dal file cui fa riferimento *fp* prelevandoli dalla zona di memoria puntata da *ptr*, il blocco è composto da *numelem* elementi ciascuno di dimensione *dimelem* in byte
- Utili per file che abbiano elementi di tipo *struct*
  - La dimensione può essere calcolata tramite la *sizeof*
- Utilizzati per leggere e scrivere file binari senza operare alcuna conversione

Accesso per carattere e per stringa (in C)

- Lettura e scrittura di caratteri
  - **int** fgetc(**FILE** *fp)
    - Quando viene raggiunta la fine del file, *fgetc* restituisce EOF
  - **int** fputc(**FILE** *fp, int c)
    - *c* è il carattere che deve essere scritto sul file, per motivi storici il tipo è *int*, ma *c* è contenuto nel byte di ordine inferiore, se si verifica un errore è segnalato con un valore non nullo nel byte di ordine superiore

- Lettura e scrittura di stringhe

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

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

Lettura

- *fgetc(stdin)* è equivalente a *getchar( )*
  - getchar( ) è definita in *stdio.h*
- *fgetc(FILE*)*
- primitiva di estrazione

  fgetc( ) (c'è anche *fgets*)
  - Estrae il prossimo carattere dal file
  - Occorre consultare la funzione *feof(x)*
    - Ritorna 1 se ci si trova a fine file, altrimenti 0

Scrittura

- *fputc('a', stdout)* è equivalente a
  putchar('a')
- *fputc(char, FILE*)*
- operazione di accodamento
  fputc( ) (c'è anche *fputs*)
  - Accoda al file un carattere

Lettura di un file testo
Esempio in C

```
do
{
ch = fgetc(fp);
}
while(ch!=EOF);
```

Il marcatore di fine-file corrisponde in Windows alla combinazione

<Ctrl> z

```
while (!feof(fp))
{
ch= fgetc(fp);
}
```

Funzione di libreria per testare il raggiungimento della fine del file

*int feof (FILE *fp)*

Ritorna 0 se non eof, valore diverso da 0 altrimenti

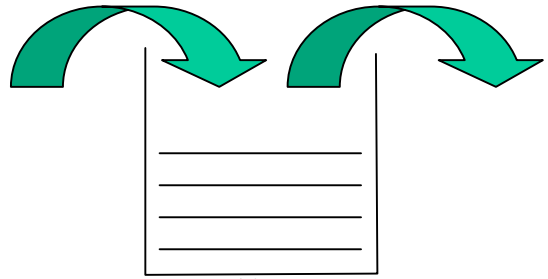
In C

- *#include stdio.h*
  - per importare la definizione del tipo FILE (che dipende dal sistema) e di alcune costanti (usate nelle funzioni di libreria che gestiscono file)
- Il file header *stdio.h* contiene i prototipi delle funzioni di I/O, definisce tre tipi (size-t, fpos-t e FILE) e contiene numerose macro (tra cui NULL, EOF, SEEK-CUR,…)

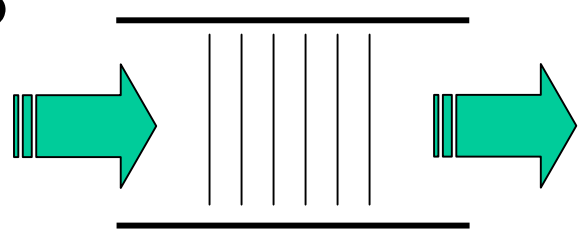
Accesso diretto

- La standard library comprende anche funzioni per l'accesso diretto ad un file
  - *fseek*: consente il posizionamento su un particolare byte
  - *ftell*: comunica il numero di byte sul quale si è posizionati al momento della chiamata della funzione
  - *rewind*: altera il valore della posizione corrente nel file assegnandogli il valore di inizio file
    - "Riavvolge" il file
    - Si può ottenere lo stesso effetto con una *fseek* al byte 0
      - fseek (f, 0, SEEK_SET)
  - SEEK_SET: inizio file; SEEK_CUR: posizione corrente; SEEK_END: fine file

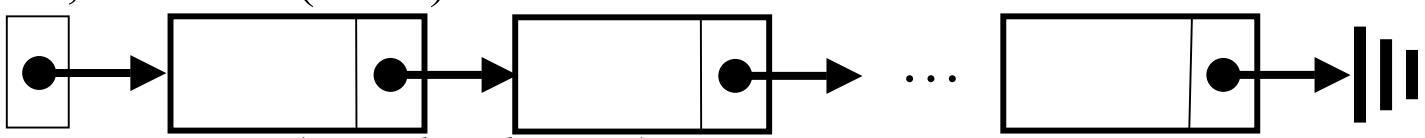
Sequenze
in Memoria Centrale (cenni)



- Pile (*Stack*)
  - Sono possibili solo operazioni di accodamento e di estrazione dell'ultimo elemento inserito
    - Last In, First Out (LIFO)



- Code (*Queue*)
  - Comportamento tale che il primo elemento inserito è il primo ad essere elaborato
    - First In, First Out (FIFO)



- Liste concatenate (*Linked List*)
  - Ciascun elemento conosce la posizione del successivo
    - Generico elemento raggiungibile tramite scansione

Allocazione Dinamica della Memoria (cenni)

- Strategia di allocazione della memoria opposta alla allocazione statica
  - indispensabile per le strutture dati dinamiche
- Consiste nell'allocare memoria durante l'esecuzione invece che all'atto della traduzione
  - Il compilatore alloca una quantità fissata di memoria per contenere l'indirizzo della componente allocata dinamicamente invece che la componente stessa
    - Si supera il problema della contiguità delle locazioni relativa ad una medesima struttura o programma
    - La memoria, così, è allocata automaticamente non appena la componente è referenziata

Allocazione Dinamica della Memoria (cenni)

- Tecnica basata sull'uso di un *puntatore*
  - Variabile il cui valore è il nome di un'altra variabile creata dinamicamente
- Operazioni disponibili sul puntatore *p*
  - Accesso alla locazione il cui indirizzo è in *p*
  - Richiesta di nuova locazione e memorizzazione di tale indirizzo in *p*
  - Rilascio della locazione il cui indirizzo è in *p*