Lezione 21

Laura Bozzelli

a.a. 2020/2021

Sommario - Lezione 21: Gestione file di testo e file binari

- Gestione dei file in C.
- Gestione dei flussi.
- Flussi di testo e flussi binari.
- Funzioni di apertura e chiusura di flussi.
- Lettura e scrittura di file di testo.
- Lettura e scrittura di file binari.
- Funzioni per l’accesso casuale.

Gestione dei file in C

- La memorizzazione dei dati nelle variabili di un programma è *temporanea*: tali dati vano *perduti* al termine dell'esecuzione del programma.

- I **file** sono "contenitori di dati" che vengono usati per la memorizzazione persistente dei dati. I file sono memorizzati su dispositivi di memoria secondaria, come dischi fissi, unità flash e DVD.

- Nel linguaggio C, la creazione, aggiornamento, ed elaborazione di file di dati avviene tramite le funzioni della libreria standard di Input/Output (I/O) i cui prototipi sono dichiarati nel file header **stdio.h**.

- I file sono gestiti dal Sistema Operativo. La realizzazione delle funzioni standard di I/O del linguaggio C tiene conto delle funzionalità del sistema operativo ospite.

Funzioni di libreria Input/Output (1/2)

Le funzioni di libreria I/O consentono di effettuare operazioni di lettura/scrittura su dispositivi di Input/Ouput in modo indipendente dalle caratteristiche proprie di tali dispositivi.

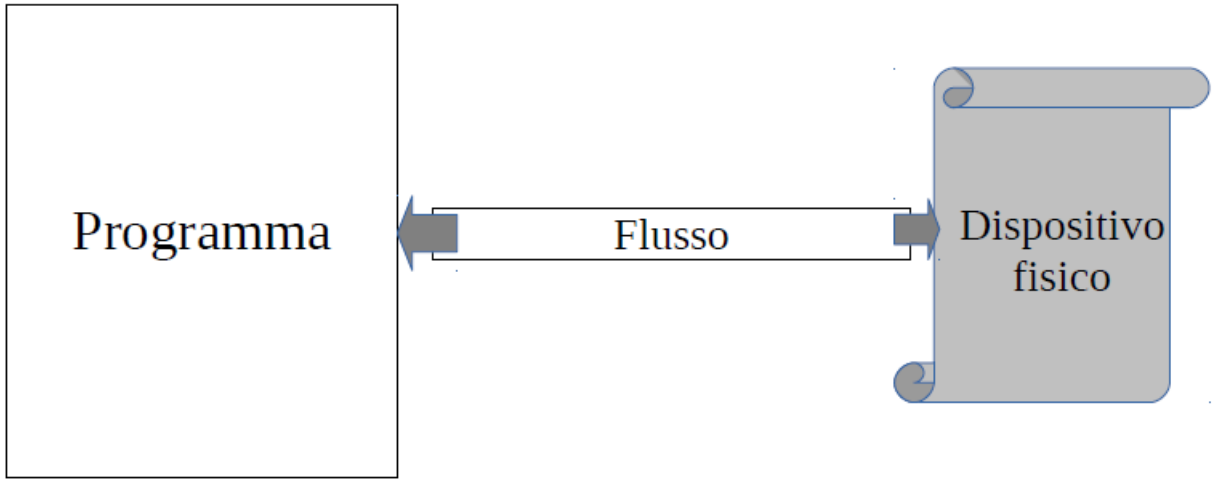
- Una stessa funzione per operazioni di input può essere utilizzata, ad esempio, sia per leggere un valore dalla tastiera sia per leggere un valore da un dispositivo di memoria di massa tramite un file in esso memorizzato.

- Similmente, una funzione per operazioni di output può essere utilizzata sia per la visualizzazione sullo schermo sia per scrivere su un disco o una stampante.

- Tale astrazione indipendente dal dispositivo effettivo è realizzata tramite un'*interfaccia*, chiamata **flusso** (o **stream**), che consente lo scambio di dati tra il programma ed il dispositivo.

Funzioni di libreria Input/Output (2/2)

Le funzioni di libreria I/O leggono/scrivono dati (sequenze di byte) a/da un dispositivo fisico attraverso i **flussi** (**stream**). Uno stream rappresenta il canale di comunicazione tra il programma ed il dispositivo.



- La proprietà fondamentale di tale gestione dell'I/O è che tutti i flussi si comportano nella stessa maniera e, quindi, possono essere gestiti nella stessa maniera indipendentemente dal tipo di dispositivo fisico (stampante, file presente su una memoria di massa, schermo, ecc).

Tipologie di flusso

Esistono due tipologie di flusso o stream:

- **Flussi di testo:** i dati vengono gestiti usando la loro rappresentazione in forma di sequenza di caratteri. Tali sequenze di caratteri sono organizzate in linee. Ogni linea termina con il carattere speciale di “newline” cioè la sequenza di escape ‘\n’.

- **Flussi binari:** i dati sono gestiti esattamente nel modo in cui sono rappresentati in memoria.

Ad esempio, in un flusso di testo, un dato di tipo **int** viene specificato tramite una sequenza di caratteri la cui lunghezza dipende dallo specifico dato (l’intero 125 viene rappresentato con un numero di caratteri diverso da 12). In un flusso binario, differenti dati di tipo **int** vengono rappresentati sempre con lo stesso numero di byte pari a **sizeof(int)**.

Gestione dei flussi

- **Apertura di un flusso**: per associare un flusso ad un dispositivo fisico (ad esempio, un file su disco fisso) è necessaria un'operazione di apertura (creazione dello stream). L'operazione di apertura compie le azioni preliminari necessarie affinchè il dispositivo possa essere acceduto (in lettura o in scrittura).

- **Accesso al flusso**: una volta associato un flusso ad un dispositivo fisico è possibile scambiare dati tra il dispositivo ed il programma.

- **Chiusura del flusso**: una volta terminate le operazioni di lettura/scrittura sul dispositivo, è necessaria un'operazione di chiusura per eliminare l'associazione tra il flusso ed il dispositivo.

Struttura FILE

Le funzioni di I/O utilizzano il tipo struct **FILE** (definito in **stdio.h**) per gestire internamente uno stream e consentire al programmatore il riferimento ad uno specifico stream.

- La funzione **fopen** della libreria di I/O utilizzata per associare un flusso ad un dispositivo fisico, alloca dinamicamente una struttura **FILE** e restituisce un puntatore a tale struttura.

- Una struttura **FILE** ha campi per tenere traccia, in particolare, delle seguenti informazioni:
  - Modalità di accesso al dispositivo (lettura o scrittura).
  - Posizione corrente sul dispositivo (indicante il prossimo byte o carattere da leggere o scrivere sul dispositivo).
  - Un indicatore di **end-of-file** che indica la posizione associata ad un byte utilizzato come marcatore finale dei dati associati al dispositivo.

Apertura di uno stream (1/2)

L'associazione di un flusso con un dispositivo fisico avviene tramite la funzione **fopen** (prototipo definito in **stdio.h**).
Prototipo:

FILE * fopen (char * name, const char * mode):

- **name**: stringa di caratteri indicante il nome del dispositivo fisico per il quale creare lo stream. Per file su disco, il parametro deve essere il nome relativo o il nome assoluto del file nel **file system** (ad esempio *"Desktop\\Prova.txt"*).
- **mode**: stringa di caratteri indicante la modalità di accesso.

Apertura di uno stream (2/2)

FILE * fopen (char * name, const char * mode):

- In caso di successo (è possibile creare l'associazione), la funzione alloca dinamicamente una struttura FILE e restituisce un puntatore a tale struttura.
- In caso di errore (non si hanno gli opportuni diritti di accesso per creare l'associazione o il dispositivo è incompatibile con la modalità di accesso indicata), la funzione restituisce il puntatore NULL.

NOTA: prima di accedere ad un dispositivo è necessario assicurarsi che la chiamata alla funzione **fopen** sia stata eseguita con successo, cioè che non abbia restituito NULL.

Apertura in sola lettura

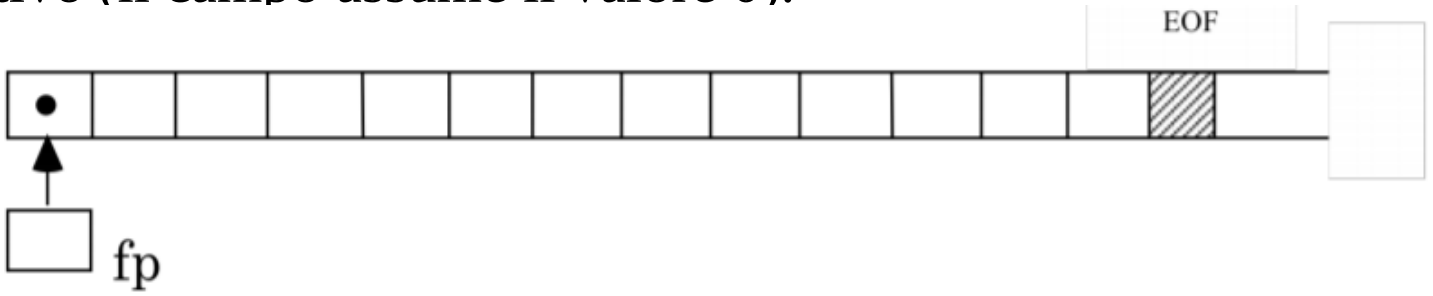
Chiamata per stream di testo:

FILE * fp = fopen (<nome dispositivo>, "r"):

Chiamata per stream binari:

FILE * fp = fopen (<nome dispositivo>, "rb"):

Il dispositivo viene acceduto solo per operazioni di lettura. La funzione restituisce NULL se il dispositivo (ad esempio, un file) non esiste. In caso di successo, per la struttura **FILE** creata, il campo indicatore di posizione all'interno del dispositivo indica l'inizio del dispositivo (il campo assume il valore 0).



Apertura in sola scrittura

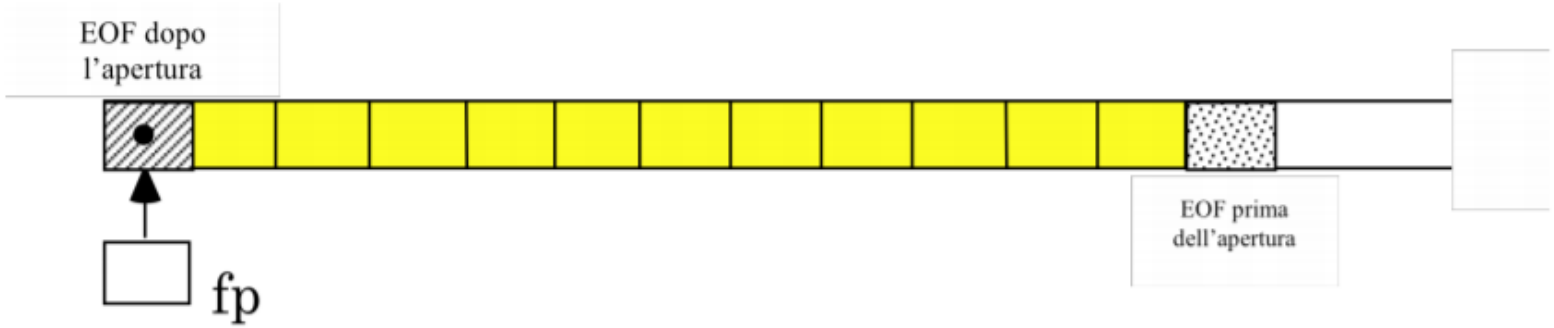
Chiamata per stream di testo:

FILE * fp = fopen (<nome dispositivo>, "w"):

Chiamata per stream binari:

FILE * fp = fopen (<nome dispositivo>, "wb"):

Il dispositivo viene acceduto solo per operazioni di scrittura. Per file di dati (in formato testo o binario), se il file già esiste, il contenuto corrente viene perso (sovrascritto).



Apertura in sola scrittura con aggiunta

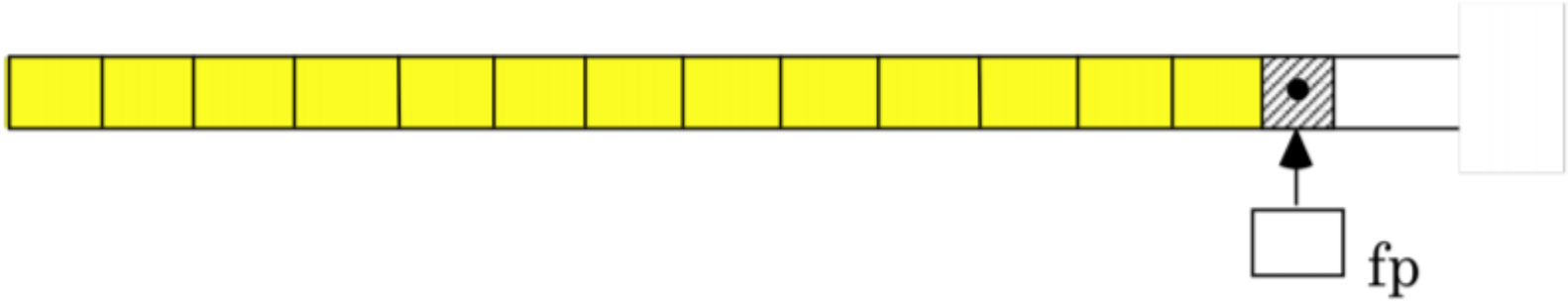
Chiamata per stream di testo:

FILE * fp = fopen (<nome dispositivo>, "a"):

Chiamata per stream binari:

FILE * fp = fopen (<nome dispositivo>, "ab"):

Il dispositivo viene acceduto solo per operazioni di scrittura. Per file di dati (in formato testo o binario), se il file già esiste, il contenuto corrente non viene perso, e l'indicatore di posizione si porta inizialmente alla fine del file.



Altre modalità di apertura per file

Per stream di testo:

- "**r+**": apre un file di testo già esistente per l'aggiornamento (lettura e scrittura).
- "**w+**": crea un file di testo per l'aggiornamento (lettura e scrittura). Se il file già esiste, il contenuto corrente viene perso.
- "**a+**": crea un file di testo già esistente per l'aggiornamento (lettura e scrittura). Se il file già esiste, il contenuto corrente non viene perso e l'indicatore di posizione si porta inizialmente alla fine del contenuto corrente.

Per stream binari:

- "**rb+**"
- "**wb+**"
- "**ab+**"

Tabella delle modalità di apertura per file

Modalità	Descrizione
`r`	Apre un file esistente per la lettura.
`w`	Crea un file per la scrittura. Se il file esiste già, *elimina* i contenuti correnti.
`a`	Apre o crea un file per scrivere alla fine del file – cioè, le operazioni di scrittura aggiungono dati al file.
`r+`	Apre un file esistente per l'aggiornamento (lettura e scrittura).
`w+`	Crea un file per l'aggiornamento. Se il file esiste già, *elimina* i contenuti correnti.
`a+`	Append: apre o crea un file per l'aggiornamento; tutta la scrittura è effettuata alla fine del file – cioè, le operazioni di scrittura aggiungono dati al file.
`rb`	Apre un file esistente per la lettura in forma binaria.
`wb`	Crea un file per la scrittura in forma binaria. Se il file esiste già, elimina i contenuti correnti.
`ab`	Append: apre o crea un file per la scrittura alla fine del file in forma binaria.
`rb+`	Apre un file esistente per l'aggiornamento (lettura e scrittura) in forma binaria.
`wb+`	Crea un file per l'aggiornamento in forma binaria. Se il file esiste già, elimina i contenuti correnti.
`ab+`	Append: apre o crea un file per l'aggiornamento in forma binaria; la scrittura è effettuata alla fine del file.

Flussi (stream) standard

Tre **stream di testo** vengono automaticamente aperti quando inizia l'esecuzione del programma:

- lo **standard input** (riceve input da tastiera): aperto in sola lettura.
- lo **standard output** (stampa output su schermo): aperto in sola scrittura.
- lo **standard error** (stampa messaggi di errore su schermo): aperto in sola scrittura.

Lo standard input, lo standard output, e lo standard error vengono manipolati usando le variabili globali puntatore di tipo FILE * **stdin**, **stdout**, e **stderr** definite nel file header **stdio.h**.

Chiusura di un flusso (1/2)

Al termine di una sessione di accesso ad un dispositivo, lo stream associato e creato tramite la funzione **fopen** deve essere chiuso tramite la funzione **fclose**.
Prototipo di fclose.

int fclose (FILE * fp);

- **fp** : deve puntare alla struttura **FILE** allocata e restituita dalla chiamata a **fopen** che ha creato lo stream che si vuole chiudere.
- Restituisce come risultato un intero:
  - Se l'operazione di chiusura viene eseguita con successo, il valore restituito è 0;
  - altrimenti, il valore restituito corrisponde a quello associato alla costante simbolica **EOF** definita nel file header **stdio.h**.

Chiusura di un flusso (2/2)

- Se uno stream creato con **fopen** non viene esplicitamente chiuso tramite una chiamata alla funzione **fclose**, il sistema operativo chiuderà automaticamente lo stream al termine dell'esecuzione del programma.

- Comunque la chiusura di uno stream può liberare risorse richieste da altri utenti o programmi. Pertanto, è importante chiudere esplicitamente uno stream quando non è più necessario accedere al relativo dispositivo, invece di aspettare che sia il sistema operativo a chiuderlo al termine dell'esecuzione del programma.

Controllare l'indicatore di end-of-file

In operazioni di lettura da uno stream, per verificare se è stato raggiunto il marcatore finale **end-of-file** dei dati associati al dispositivo, si utilizza la funzione **feof**.
Prototipo di feof.

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

- Restituisce un valore maggiore di zero (valore Booleano true) nel caso in cui l'indicatore di posizione si è portato in corrispondenza del marcatore finale **end-of-file**, e 0 altrimenti (valore Booleano false).

Funzioni di scrittura su stream di testo (1/3)

Funzione fprintf

int fprintf (FILE * fp, char * stringa_di_controllo,…..);

consente di scrivere testo formattato su uno stream di testo aperto in modalità di scrittura o aggiornamento. Rappresenta una generalizzazione della funzione **printf** per stream di testo associati ad arbitrari dispositivi d'output (in particolare, file di testo). In caso di successo, la funzione restituisce il numero di caratteri scritti nello stream e l'indicatore di posizione si porta in corrispondenza del carattere successivo all'ultimo carattere scritto. Altrimenti, la funzione restituisce **EOF**.

Esempio:

```
int num = 3;
printf("Il numero e\' %d\n",num);
//L'istruzione precedente è equivalente a
fprintf(stdout,"Il numero e\' %d\n",num);
```

Funzioni di scrittura su stream di testo (2/3)

Funzione fputc

int fputc (char c, FILE * fp);

consente di scrivere il carattere **c** (convertito ad **unsigned char**) su uno stream di testo aperto in modalità di scrittura o aggiornamento. In caso di successo, la funzione restituisce il carattere scritto nello stream e l'indicatore di posizione si porta in corrispondenza del carattere successivo. Altrimenti, la funzione restituisce **EOF**.

Esempio:

```
char c = 'A';
//Scrive il carattere c sullo standard output
fputc(c,stdout);
```

Funzioni di scrittura su stream di testo (3/3)

Funzione fputs

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

consente di scrivere una stringa di caratteri su uno stream di testo aperto in modalità di scrittura o aggiornamento. In caso di successo, la funzione restituisce un valore positivo e l'indicatore di posizione si porta in corrispondenza del carattere successivo all'ultimo carattere scritto. Altrimenti, la funzione restituisce **EOF**.

Esempio:

```
char s[] = "Ciao mondo!";
//Scrive la stringa s sullo standard output
fputs(s,stdout);
```

Funzioni di lettura da stream di testo (1/3)

Funzione fscanf

int fscanf (FILE * fp, char * stringa_di_controllo,…..);

consente di leggere testo in modo formattato da uno stream di testo aperto in modalità di lettura o aggiornamento. Rappresenta una generalizzazione della funzione **scanf** per stream di testo associati ad arbitrari dispositivi d'input (in particolare, file di testo). In caso di successo, la funzione restituisce il numero di caratteri letti dallo stream e l'indicatore di posizione si porta in corrispondenza del carattere successivo all'ultimo carattere letto. Altrimenti, la funzione restituisce **EOF**.

Esempio:

```
int val;
scanf("%d",&val);
//L'istruzione precedente è equivalente a
fscanf(stdin,"%d",&val);
```

Funzioni di lettura da stream di testo (2/3)

Funzione fgetc

int fgetc (FILE * fp);

consente di leggere il carattere correntemente puntato dall'indicatore di posizione da uno stream di testo aperto in modalità di lettura o aggiornamento. In caso di successo (non si è raggiunto il marcatore **end-of-file** di fine dati), la funzione restituisce il carattere letto e l'indicatore di posizione si porta in corrispondenza del carattere successivo. Altrimenti, la funzione restituisce **EOF**.

Esempio:

```
//Legge il carattere corrente dallo standard d'input
char c = fgetc(stdin);
printf("%c",c);
```

Funzioni di lettura da stream di testo (3/3)

Funzione fgets

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

consente di leggere dalla posizione corrente di uno stream di testo (aperto in modalità di lettura o aggiornamento) una sequenza di al più **n-1** caratteri e memorizzarla nell'array di caratteri puntato da **s** insieme al carattere '\0' di terminazione stringa. In caso di successo, la funzione restituisce **s**: il numero **k** di caratteri letti può essere inferiore a **n-1** se il k-esimo carattere è un newline. In caso di insuccesso (errore o la funzione incontra il marcatore di fine dati), la funzione restituisce NULL.

Esempio:

```
char str[30];
//Inizializza la string str con i primi 30
//caratteri letti dallo standard d'input
fgets(str,30,stdin);
printf("%s",str);
```

Esempi: stampa di un file di testo

Stampa del contenuto di un file di testo.

```
//Apre in lettura un file di testo
//avente il dato nome e lo stampa a video
void StampaFileTesto(char * nomeFile)
{
    FILE * fp = fopen(nomeFile,"r");
    if(fp != NULL)
    {
        char c;
        while((c = fgetc(fp)) != EOF)
            printf("%c",c);

        fclose(fp);
    }
    else
      printf("Errore nell'apertura del file"
       " di testo %s", nomeFile);
}
```

Esempi: copia di un file di testo

```c
//Apre in lettura il file di testo 'sorgente' e lo copia nel file di testo
// 'target'. Se il file di testo 'target' già esiste il suo contenuto viene sovrascritto
void CopiaFileTesto(char * sorgente, char * target)
{
    FILE * fs = fopen(sorgente,"r");
    FILE * ft = fopen(target,"w");
    if(fs == NULL)
    {
        printf("Errore nell'apertura del file"
        " di testo %s", sorgente);
        return;
    }

    if(ft == NULL)
    {
        printf("Errore nell'apertura del file"
        " di testo %s", target);
        return;
    }

    char c;
    while((c = fgetc(fs)) != EOF)
        fprintf(ft,"%c",c);

    fclose(fs);
    fclose(ft);
}
```

Esempi: scrittura e lettura di una matrice (1/3)

Il seguente esempio illustra come scrivere una matrice di interi in un file di testo e come estrarre in lettura la matrice dal file. La struttura del file di testo è come segue:

- La prima linea contiene due interi M e N rappresentanti il numero di righe ed il numero di colonne della matrice, rispettivamente.
- Nelle successive M linee del file di testo sono riportate le M righe della matrice ordinate per valori crescenti dell'indice di riga: sulla seconda linea del file è riportata la prima riga della matrice, sulla terza linea la seconda riga della matrice, e così via.

Ad esempio:

```
3  4
14    5     67    -21
23    -5    11    10
-1    3     12    18
```

Esempi: scrittura e lettura di una matrice (2/3)

```c
void ScriviMatrice(char * nomeFile, int ** mat,
                   size_t M, size_t N)
{
    FILE * fs = fopen(nomeFile,"w");
    if(fs != NULL)
    {
        fprintf(fs, "%d %d\n",M,N);
        int i,j;
        for(i=0;i<M;i++)
        {
            for(j=0;j<N;j++)
                fprintf(fs,"%d ",mat[i][j]);

            fprintf(fs,"\n");/* a capo dopo una riga*/
        }

        fclose(fs);
    }
    else
        printf("Errore nell'apertura del file"
        " di testo %s", nomeFile);
}
```

```c
void main()
{
    size_t M = 10, N=10;
    int A[M][N];
    int i,j;
    for(i=0;i<M;i++)
    {
        for(j=0;j<N;j++)
            A[i][j] = rand() % 201 - 100;
    }

    int * B[M];
    for(i=0;i<M;i++)
        B[i] = A[i];

    ScriviMatrice("Prova.txt", B, M, N);
}
```

Esempi: scrittura e lettura di una matrice (3/3)

```c
int ** LeggiMatrice(char * nomeFile, size_t * pM, size_t * pN)
{
    FILE * fs = fopen(nomeFile,"r");

    if(fs != NULL)
    {
       fscanf(fs, "%d %d",pM, pN);

       int ** mat = (int **) malloc((*pM)*sizeof(int *));
       int i,j;
       for(i=0;i<(*pM);i++)
           mat[i] = (int *) malloc((*pN)*sizeof(int));

       for(i=0;i<(*pM);i++)
       {
           for(j=0;j<(*pN);j++)
               fscanf(fs,"%d ",&mat[i][j]);
       }

       fclose(fs);
       return mat;
    }
    else
    {
        printf("Errore nell'apertura del file"
               " di testo %s", nomeFile);
        return NULL;
    }
}
```

Inizializzazione di stringhe tramite fscanf (1/2)

- Similmente alla funzione **scanf**, è possibile utilizzare lo specificatore di conversione %s per inizializzare tramite la funzione **fscanf** un array di caratteri ad una stringa di caratteri dello stream di testo.
- Quando **fscanf** incontra lo specificatore %s, legge i caratteri e li memorizza nell'array di caratteri associato finchè non incontra uno spazio, una tabulazione, un newline o un indicatore di fine file. A questo punto, la funzione inizializza l'elemento corrente dell'array di input con '\0'.

Inizializzazione di stringhe tramite fscanf (2/2)

- È importante assicurarsi che il numero di caratteri processati + il carattere nullo di terminazione non superi la lunghezza del vettore di caratteri (altrimenti si genera un **overflow del buffer**). Ciò può essere garantito utilizzando lo specificatore di conversione per stringhe nel formato %Ns dove N è una costante intera non negativa che indica il numero massimo di caratteri che possono essere letti ed inseriti nell'array di input.

File binari (1/2)

- I file di testo non si prestano bene ad operazioni di modifica che comportano un accesso casuale al file (cambio programmatico dell'indicatore di posizione). Questo perché dati dello stesso tipo sono rappresentati come sequenze di caratteri aventi lunghezze diverse (**dati a lunghezza variabile**).
- Ad esempio, in un file di testo strutturato come una sequenza di *record* contenenti informazioni della stessa tipologia, per modificare un certo record (dal momento che i record sono a lunghezza variabile), è necessario aggiornare tutto il contenuto del file a partire dal record da modificare.

File binari (2/2)

- Nei file binari, dati dello stesso tipo sono rappresentati con lo stesso numero di byte.
- In file binari strutturati come una sequenza di record della stessa tipologia, i record sono a **lunghezza fissa**. È, dunque, possibile modificare un record senza dovere aggiornare l'intero file.
- Inoltre, per record indicizzati da una chiave di ricerca, la posizione esatta di un record all'interno del file può essere calcolata come funzione della chiave del record. Ciò consente l'accesso immediato (**accesso casuale**) a record specifici anche in file di grandi dimensioni.
- Un inconveniente dei file binari è che il numero di byte per rappresentare un tipo primitivo (ad esempio, un intero **int**) dipende dalla macchina soggiacente. Dunque un file binario scritto su una macchina può non essere leggibile su un'altra macchina.

Funzione di scrittura su stream binari

Funzione fwrite

size_t fwrite (void *buffer, size_t n_byte, size_t num, FILE *pf);

- **buffer** rappresenta l'indirizzo iniziale della regione di memoria che contiene i dati da scrivere sullo stream binario riferito da **pf**.
- Il valore di **num** determina il numero di oggetti di ampiezza **n_byte** byte da scrivere sullo stream. Dunque, la regione di memoria allocata puntata da **buffer** deve avere un'ampiezza di almeno **n_byte * num** byte.
- In caso di successo, il valore di ritorno indica il numero di oggetti effettivamente inviati allo stream. Altrimenti, il valore restituito è un numero negativo.

Funzione di lettura da stream binari

Funzione fread

size_t fread (void *buffer, size_t n_byte, size_t num, FILE *pf);

- **buffer** rappresenta l'indirizzo iniziale della regione di memoria su cui scrivere i dati letti dallo stream binario riferito da **pf**.
- Il valore di **num** determina il numero di oggetti di ampiezza **n_byte** byte da leggere dallo stream. Dunque, la regione di memoria allocata puntata da **buffer** deve avere un'ampiezza di almeno **n_byte * num** byte.
- In caso di successo, il valore di ritorno indica il numero di oggetti effettivamente letti dallo stream. Altrimenti, il valore restituito è un numero negativo.

Esempi: scrittura e lettura di file binari (1/3)

```c
typedef struct
{
    unsigned int matricola;
    unsigned int N_esami;
} Studente;

void AggiungiStudente(FILE * pf, Studente st)
{
    fwrite(&st,sizeof(Studente),1,pf);
}


void AggiungiStudenti(FILE * pf, Studente st[],
                      size_t N_studenti)
{
    fwrite(st,sizeof(Studente),N_studenti,pf);
}
```

Esempi: scrittura e lettura di file binari (2/3)

```
void main()
{
    FILE * pf = fopen("studenti.dat","ab");
    if(pf== NULL)
    {
        printf("Il file non puo\' essere aperto \n");
        return;
    }

    int continua;
    do
    {
        printf("Inserisci dati studente!\n");
        Studente st;
        scanf("%u%u",&st.matricola, &st.N_esami);
        AggiungiStudente(pf,st);
        printf("Vuoi continuare (si=1,no=0)?\n");
        scanf("%d", &continua);
    }while(continua);

    close(pf);
    StampaStudenti("studenti.dat");
}
```

Esempi: scrittura e lettura di file binari (3/3)

```
void StampaStudenti(char * nomeFile)
{
    FILE * fs = fopen(nomeFile,"rb");
    if(fs != NULL)
    {
        int result,i=1;
        Studente st;
        while((result = fread(&st,sizeof(Studente),1,fs)) == 1)
        {
            printf(" Studente %d: matricola = %u, numero esami = %u\n",i,
                        st.matricola,st.N_esami);
            i++;
        }
        fclose(fs);
    }
    else
        printf("Errore nell'apertura del file"
        " di testo %s", nomeFile);
}
```

Funzioni per l'accesso casuale (1/2)

Funzione fseek

int fseek(FILE *pf, long offset, int origine);

- Imposta l'indicatore di posizione dello stream associato a **pf** (in termini di numero di byte dall'inizio dello stream). La prossima operazione di I/O sullo stream verrà eseguita dalla nuova posizione impostata.
- La posizione è calcolata aggiungendo **offset** (che può assumere anche valori negativi) a **origine**. Il parametro **origine** può assumere i seguenti valori:
  - **SEEK_SET**: indica l'inizio del file.
  - **SEEK_CUR**: indica la posizione corrente.
  - **SEEK_END**: indica la fine del file.
- In caso di successo, restituisce 0 e **viene cancellato l'indicatore di fine file**. Altrimenti, restituisce -1.

Funzioni per l’accesso casuale (2/2)

Funzione ftell

int ftell(FILE *pf);

Restituisce il valore corrente dell'indicatore di posizione dello stream associato a **pf** (posizione corrente rispetto all’inizio del file, espressa come numero di byte).

Esempio accesso casuale (1/2)

```
void main()
{
    FILE * pf = fopen("studenti.dat","rb+");
    if(pf== NULL)
    {
        pf= fopen("studenti.dat","wb+");
        if(pf == NULL)
        {
            printf("Risultato inatteso! \n");
            return;
        }
    }
    int continua;
    do
    {
        printf("Inserisci dati studente!\n");
        Studente st;
        scanf("%u%u",&st.matricola, &st.N_esami);
        fseek(pf,0,SEEK_SET);
        AggiungiStudenteInOrdine(pf,st);
        printf("Vuoi continuare (si=1,no=0)?\n");
        scanf("%d", &continua);
    }while(continua);

    close(pf);
    StampaStudenti("studenti.dat");
}
```

Esempio accesso casuale (2/2)

Inserimento degli studenti in un file binario in ordine crescente di matricola.

```
void AggiungiStudenteInOrdine(FILE * pf, Studente st)
{
    Studente currSt;
    long currPos = ftell(pf);
    fseek(pf,0,SEEK_END);
    if(currPos == ftell(pf))
        fwrite(&st,sizeof(Studente),1,pf);
    else
    {
        fseek(pf,currPos,SEEK_SET);
        fread(&currSt,sizeof(Studente),1,pf);
        if(currSt.matricola<=st.matricola)
            AggiungiStudenteInOrdine(pf,st);
        else
        {
            long currPos = ftell(pf);
            fseek(pf,currPos-sizeof(Studente),SEEK_SET);
            fwrite(&st,sizeof(Studente),1,pf);
            AggiungiStudenteInOrdine(pf,currSt);
        }
    }
}
```