U.D. 8.2

Files

- Concetti generali - File sequenziali e a indice
- I FILE sequenziali in C
- FILE di testo e binari

Fonti:

Antola Dispense del Corso
A. Ciampolini – UNI_BO – Lucidi del corso

Gestione di file in C

I file hanno una struttura ***sequenziale:***

- i record logici sono organizzati in una sequenza
- per accedere ad un particolare record logico, e` necessario "scorrere" tutti quelli che lo precedono.

				X				…

Per accedere ad un file da un programma C, e` necessario predisporre una variabile che lo rappresenti (**puntatore a file)**

Puntatore a file:

e` una variabile che viene utilizzata per riferire un file nelle operazioni di accesso (lettura e scrittura.). Implicitamente essa indica:

- il file
- l'elemento corrente all'interno della sequenza

Ad esempio:

```
FILE *fp;
```

↣ il tipo FILE e` un tipo non primitivo dichiarato nel file **stdio.h.**

Gestione di file in C

Apertura di un file:

Prima di accedere ad un file e` necessario ***aprirlo:*** l'operazione di apertura compie le azioni preliminari necessarie affinche` il file possa essere acceduto (in lettura o in scrittura). L'operazione di apertura inizializza il puntatore al file.

Accesso ad un file:

Una volta aperto il file, e` possibile leggere/scrivere il file, riferendolo mediante il puntatore a file.

Chiusura di un file:

Alla fine di una sessione di accesso (lettura o scrittura) ad un file e` necessario chiudere il file per memorizzare permanentemente il suo contenuto in memoria di massa:

Apertura di un File

FILE *fopen(char *_name_, char *_mode_);

dove:

- ***name*** e` un array di caratteri che rappresenta il nome (assoluto o relativo) del file nel file system
- ***mode*** esprime la modalita` di accesso scelta.

"r",	in lettura (read)
"w",	in scrittura (write)
"a",	scrittura, aggiunta in fondo (append)
"b",	a fianco ad una delle precedenti, indica che il file e` binario
"t",	a fianco ad una delle precedenti, indica che il file e`di testo

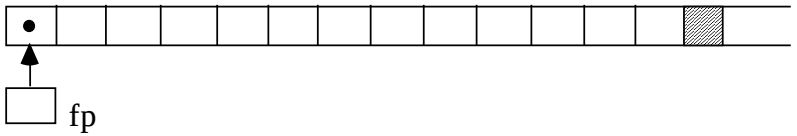
- **Se eseguita con** successo**, l'operazione di apertura ritorna come risultato un** *puntatore* **al file aperto**

- **Se, invece, l'apertura fallisce (ad esempio, perche` il file non esiste), fopen restituisce il valore NULL.**

Apertura in lettura

fp = fopen("filename", "r")

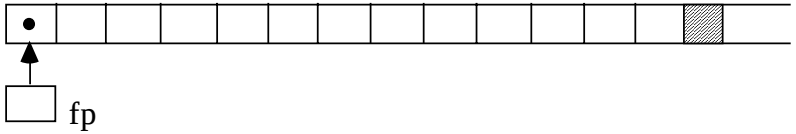
Se il file non e` vuoto:



Apertura in scrittura

fp = fopen("filename", "w")

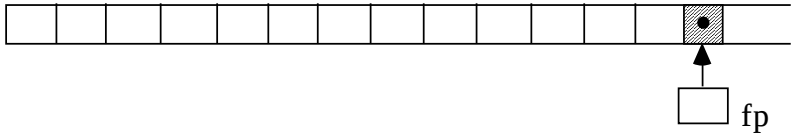
Anche se il file non e` vuoto:



- Se il file esisteva gia`, il suo contenuto viene **perso**.

Apertura in aggiunta (append)

fp = fopen("filename", "a")



Il puntatore al file si posiziona sull'elemento successivo all'ultimo significativo del file ➠ se il file esisteva gia`, il suo contenuto non viene perso.

Ad esempio:

```
File *fp;
fp=fopen("c:\anna\dati", "r");
<uso del file>
```

- fp rappresenta, dall'apertura in poi, il riferimento da utilizzare nelle operazioni di accesso a c:\anna\dati. Esso individua, in particolare:
  - il file
  - l'elemento corrente all'interno del file

Chiusura di un File

Al termine di una sessione di accesso al file, esso deve essere **chiuso**.

L'operazione di chiusura si realizza con la funzione **fclose**:

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

dove:

- fp rappresenta il puntatore al file da chiudere.

fclose ritorna come risultato un intero:

- Se l'operazione di chiusura e` eseguita correttamente restituisce il valore 0
- se la chiusura non e` andata a buon fine, ritorna la costante EOF.

Esempio:

```
#include <stdio.h>
main()
{
  FILE *fp;
  fp = fopen("prova.dat", "w")
  <scrittura di prova.dat>
  fclose(fp);
  return 0;
}
```


File standard di I/O

Esistono tre file testo che sono aperti automaticamente all'inizio di ogni esecuzione:

- ***stdin***, standard input (tastiera), aperto in lettura
- ***stdout***, standard output (video), aperto in scrittura
- ***stderr***, standard error (video), aperto in scrittura

↣ *stdin*, *stdout*, *stderr* sono variabili di tipo *puntatore a file* automaticamente (ed **implicitamente**) definite ➠ **non vanno definite**.

Funzione feof()

Durante la fase di accesso ad un file e` possibile verificare la presenza della marca di fine file con la funzione di libreria:

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

- feof(fp) controlla se e` stata raggiunta la fine del file fp nella operazione di lettura o scrittura **precedente**. Restituisce il valore 0 (falso logico) se non e` stata raggiunta la fine del file, altrimenti un valore diverso da zero (vero logico).

Lettura e Scrittura di file

Una volta aperto un file, su di esso si puo` accedere in lettura e/o scrittura, compatibilmente con quanto specificato in fase di apertura.

Per file di testo sono disponibili funzioni di:

- Lettura/scrittura **con formato**
- Lettura/scrittura di **caratteri**
- Lettura/scrittura di **stringhe di caratteri**

Per file binari si utilizzano funzioni di:

- Lettura/scrittura di **blocchi**

Accesso a file di testo: Lettura/scrittura con formato

Funzioni simili a **scanf** e **printf**, ma con un parametro aggiuntivo rappresentante il puntatore al file **di testo** sul quale si vuole leggere o scrivere:

Lettura con formato:

Si usa la funzione fscanf:

int fscanf **(FILE *fp,** ***stringa-controllo*****,** ***ind-elem*****);**

dove:

- fp e` il puntatore al file
- *stringa-controllo* indica il formato dei dati da leggere
- *ind-elem* e` la lista degli indirizzi delle variabili a cui assegnare i valori letti.

Esempio:

```
FILE *fp;
int A; char B; float C;
fp=fopen("dati.txt", "r");
fscanf(fp, "%d%c%f", &A, &B, &C);
...
fclose(fp);
```

Restituisce il numero di elementi letti, oppure un valore negativo in caso di errore.

Scrittura con formato:

Si usa la funzione fprintf::

int fprintf **(FILE *fp,** ***stringa-controllo*****,** ***elementi*****);**

dove:

- fp e` il puntatore al file
- *stringa-controllo* indica il formato dei dati da scrivere
- *elementi* e` la lista dei valori (espressioni) da scrivere.

Esempio:

```
FILE *fp;
float C=0.27;
fp=fopen("risultati.txt", "w");
fscanf(fp,"Risultato: %f", c*3.14);
...
fclose(fp);
```

Restituisce il numero di elementi scritti, oppure un valore negativo in caso di errore.

printf/scanf e fprintf/fscanf:

- Notiamo che:

printf **(stringa-controllo, elementi)**
scanf **(stringa-controllo, ind-elementi);**

equivalgono a:

fprintf **(stout, stringa-controllo, elementi);**
fscanf **(stdin, stringa-controllo, ind-elementi);**

Esempio:

Visualizzazione del contenuto di un file di testo:

```
#include <stdio.h>

main()
{
char buf[80]
FILE *fp;

fp=fopen("testo.txt", "r");
fscanf(fp,"%s",buf);
while(!feof(fp))
{   printf("%s", buf);
    fscanf(fp,"%s",buf);
}
fclose(fp);
}
```

oppure:

```
#include <stdio.h>

main()
{
char buf[80]
FILE *fp;

fp=fopen("testo.txt", "r");
while (fscanf(fp,"%s",buf)>0)
    printf("%s", buf);
fclose(fp);
}
```


Lettura/scrittura di caratteri:

Funzioni simili a **getchar** e **putchar**, ma con un parametro aggiuntivo rappresentante il puntatore al file (di testo) sul quale si vuole leggere o scrivere:

```
int    getc (FILE *fp);

int    putc (int c, FILE *fp);

int    fgetc (FILE *fp);

int    fputc (int c, FILE *fp);
```

- In caso di esecuzione corretta, restituiscono il carattere letto o scritto come intero, altrimenti EOF.

Esempio:

Programma che concatena i file dati come argomento in un unico file (*stdout*):

```
#include <stdio.h>

main(int argc, char **argv)
{
    FILE *fp;
    void filecopy(FILE *, FILE *);

    if (argc==1) filecopy(stdin, stdout);
    else
      while (--argc>0)
        if ((fp=fopen(*++argv, "r"))==NULL)
          {
           printf("\nImpossibile aprire il
                file %s\n", *argv);
           exit(1);
          }
        else
           {filecopy(fp, stdout);
            fclose(fp);
           }
    return 0;
}

void filecopy(FILE *inputFile,
                    FILE *outputFile)
{int c;

    while((c=getc(inputFile))!=EOF)
     putc(c, outputFile);
}
```


Note sull'esempio:

- se non ci sono argomenti (argc=1), il programma copia lo standard input nello standard output;
- la funzione **`filecopy`** effettua la copia del file carattere per carattere;
- se uno dei file indicati come argomento non esiste, la funzione **`fopen`** fallisce, restituendo il valore NULL. In questo caso il programma termina (**`exit`**) restituendo il valore 1 e stampando un messaggio di errore;
- sarebbe meglio scrivere i messaggi di errore sullo standard error (***ridirezione***).

```
printf(stderr,"\nImpossibile aprire
       il file %s\n", *argv);
```

Per non perdere dati, la funzione **`exit`** chiude automaticamente ogni file aperto.

- il ciclo:

```
while((c=getc(inputFile))!=EOF)
    putc(c, outputFile);
```

  poteva essere scritto anche come:

```
c=getc(inputFile);
while(!feof(inputFile))
 {  putc(c,outputFile);
    c=getc(inputFile);
 }
```

Lettura/scrittura di stringhe

Funzioni simili a **gets** e **puts**:

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

Trasferisce nella stringa s i caratteri letti dal file puntato da fp, fino a quando ha letto n-1 caratteri, oppure ha incontrato un newline, oppure la fine del file. La **fgets** mantiene il newline nella stringa s.

Restituisce la stringa letta in caso di corretta terminazione; '\0' in caso di errore o fine del file.

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

Trasferisce la stringa s (terminata da '\0') nel file puntato da fp. Non copia il carattere terminatore '\0' ne` aggiunge un newline finale.

Restituisce l'ultimo carattere scritto in caso di terminazione corretta; EOF altrimenti.

Accesso a file binari: Lettura/scrittura di blocchi

Si puo` leggere o scrivere da un file binario un intero blocco di dati (binari).

Un file binario memorizza dati di qualunque tipo, in particolare dati che non sono caratteri (interi, reali, vettori o strutture).

Per la lettura/scrittura a blocchi e` necessario che il file sia stato aperto in modo ***binario*** (modo “b”)**.**

Lettura:

```
int   fread (void  *vet, int size, int n,  FILE *fp);
```

Legge (al piu`) *n* oggetti dal file puntato da *fp*, collocandoli nel vettore *vet*, ciascuno di dimensione *size*. Restituisce un intero che rappresenta il numero di oggetti effettivamente letti.

Scrittura

```
int   fwrite (void *vet, int size, int n, FILE *fp);
```

Scrive sul file puntato da *fp*, prelevandoli dal vettore *vet*, *n* oggetti, ciascuno di dimensione *size*. Restituisce un intero che rappresenta il numero di oggetti effettivamente scritti (inferiore ad *n* solo in caso di errore, o fine del file).

Esempio:

Programma che scrive una sequenza di record (dati da input) in un file binario:

```
#include<stdio.h>
typedef struct{ char nome[20];
                char cognome[20];
                int reddito;
              }persona;

main()
{ FILE *fp;
  persona p;
  int fine=0;

  fp=fopen("archivio.dat","wb");
  do
  { printf("Dati persona?");
    scanf("%s%s%d%d",&p.nome,
      &p.cognome,&p.reddito);
    fwrite(&p,sizeof(persona),1,fp);
    printf("Fine (si=1,no=0)?");
    scanf("%d", &fine);
  }while(!fine);
  fclose(fp);
}
```


Esempio:

Programma che legge e stampa il contenuto di un file binario:

```
#include<stdio.h>
typedef struct{ char nome[20];
                char cognome[20];
                int reddito;
              }persona;

main()
{ FILE *fp;
  persona p;

  fp=fopen("archivio.dat","rb");
  fread(&p, sizeof(persona),1, fp);
  while (!feof(fp))
  { printf("%s%s%d",p.nome,p.cognome,
           p.reddito);
    fread(&p,sizeof(persona),1,fp);
  }
  fclose(fp);
}
```

Esempio:

Programma che riceve come argomento il nome di un file e scrive in questo file un vettore di interi.

```
#include <stdio.h>

main(int argc, char **argv)
{FILE *file;
void stop(char *);
int i,n,tab[]={3, 6, -12, 5, -76, 3,
               32, 12, 65, 1, 0, -9};

if (argc==2)
 if ((file=fopen(argv[1],"wb"))==NULL)
     stop("Impossibile aprire file
           d'uscita\n");
else stop("Manca il nome del file di
          uscita\n");
n = sizeof(tab)/sizeof(tab[0]);
fwrite(tab, sizeof(tab[0]), n, file);
fclose(file);
exit(0);
}

void stop(char *msg)
{
    fprintf(stderr,msg);
    exit(1);
}
```


Esempio:

Programma che riceve come argomento il nome di un file di interi e memorizza il contenuto di questo file in un vettore di interi (di al piu` 40 elementi).

```
#include <stdio.h>
#define MAX 40

main(int argc, char **argv)
{FILE *file;
 void stop(char *);
 int i, n, tab[MAX];

 if (argc==2)
  {if ((file=fopen(argv[1], "rb"))==NULL)
       stop("Impossibile aprire file
            d'ingresso\n");
  }
 else stop("Manca il nome del file
           d'ingresso\n");

 n=fread(tab, sizeof(tab[0]), MAX, file);
 fclose(file);

 for(i=0;i<n;i++)
    printf("%d%c", tab[i],
                (i==n-1) ? '\n' : '\t');
 exit(0);
}

void stop(char *msg)
{
    fprintf(stderr,msg);
    exit(1);
}
```

Esempio:

File di record.

```
#include <stdio.h>
#include <ctype.h>
#define DIM 5
typedef struct  {  char nome[15],
                cognome[15],
                via[10];
                int eta;} Persona;

Persona P[DIM];

main(int argc, char **argv)
{
int crea_vettore(Persona V[], int dim);
int i, n;
FILE *file;

if (argc==2)
  {n=crea_vettore(P,DIM);

    if ((file=fopen(argv[1], "wb"))==NULL)
        printf("Impossibile aprire
           file\n");
    else
       {
        fwrite(P,sizeof(Persona),n,file);
        fclose(file);
         }
  }
else printf("Manca qualche parametro\n");

}
```



```
int crea_vettore(Persona P[], int dim)
{int i=0, n=0; char s[80];
while (!feof(stdin) && i<dim)
   { scanf("%s\n",P[i].nome);
     scanf("%s\n",P[i].cognome);
     scanf("%s\n",P[i].via);
     scanf("%d",&(P[i].eta));gets(s);
     i++; n++;
     printf("%s\n%s\n%s\n%d\n",P[i].nome,
     P[i].cognome,P[i].via,P[i].eta);
     }
}
```

File ad accesso diretto

Il C consente di gestire i file non solo come sequenziali, ma anche come file ad accesso diretto.

Posizionamento in un file:

La funzione ***fseek*** della Standard Library consente il posizionamento del puntatore al file su un qualunque byte.

int fseek (FILE *f, long *offset*, int *origin*)

si sposta di *offset* byte a partire dalla posizione *origin* (che vale 0, 1 o 2).

Restituisce:

- 0 se ha spostato la posizione sul file
- un valore diverso da 0, altrimenti.

Origine dello spostamento:

SEEK_SET	0	inizio file
SEEK_CUR	1	posizione attuale nel file
SEEK_END	2	fine file

Per posizionarsi all'inizio di un file gia` aperto e` possibile utilizzare anche la funzione ***rewind***:

void rewind (FILE *f)

```
file=fopen(argv[1], "r+");
...;
rewind(file);
```

equivale a:

```
fseek(f,0,SEEK_SET);
```

Posizione corrente nel file:

La funzione ***ftell*** restituisce la posizione del byte sul quale si e` posizionati nel file al momento della chiamata della funzione (restituisce -1 in caso di errore):

long ftell (FILE *f)

Il valore restituito da ***ftell*** puo` essere utilizzato in una chiamata ad ***fseek***.

Esempio:

Programma che sostituisce tutte le minuscole in maiuscole in un file testo dato come (unico) argomento.

```
#include <stdio.h>
#include <ctype.h>
main(int argc, char **argv)
{   FILE *file;
    void stop(char *);
    int ch;
if (argc==2)
 {if ((file=fopen(argv[1], "r+"))==NULL)
     stop("Impossibile aprire file
          d'ingresso\n");
 }
else stop("Manca qualche parametro\n");
while((ch=getc(file))!=EOF)
  if(islower(ch))
   {fseek(file, ftell(file)-1, SEEK_SET);
    putc(toupper(ch), file);
    fseek(file, 0, SEEK_CUR);
}
fclose(file);
exit(0);
}
void stop(char *msg)
{fprintf(stderr,msg);
 exit(1);
}
```


Note sull'esempio:

- Il file e` aperto con modalita` "r+" (aggiornamento, ma posizione all'inzio del file).
- Il programma legge ad uno ad uno i caratteri del file e, quando trova una lettera minuscola (funzione ***islower*** della libreria ***ctype***), retrocede con ***fseek*** di una posizione e la sostituisce con la corrsipondente maiuscola (funzione ***toupper***)
- L'utilizzo della funzione ***fseek*** e` utilizzata per riposizionarsi sul carattere appena letto, se questo e` una lettera minuscola:

  fseek(file, ftell(file)-1, SEEK_SET);

- E` inoltre ***obbligatoria*** per poter alternare scritture e letture su file:

  fseek(file, 0, SEEK_CUR);

- L'apertura di un file in modo di aggiornamento "+" (abbinato ad uno qualunque tra "r", "w", "a") richiede esplicitamente che, dopo una sequenza di letture, prima di iniziare qualunque scrittura venga usata una delle funzioni di posizionamento su file (e analogamente per scritture seguite da letture).

R/W a Blocchi

LETTURA/SCRITTURA DI BLOCCHI DI BYTE

```
int fread (void *punt, dim_blocco,num_blocchi, FILE *fp)
```

- legge dal file **fp** un numero di byte pari a **dim_blocco*num_blocchi** e li memorizza nell'area di memoria puntata da **punt**
- restituisce il numero di byte letti

```
int fwrite (void *punt, dim_blocco,num_blocchi, FILE *fp)
```

- scrive sul file **fp** un numero di byte pari a **dim_blocco*num_blocchi** e letti dell'area di memoria puntata da **punt**
- restituisce il numero di byte scritti

```
rewind (FILE *fp)
```

ACCESSO DIRETTO A BYTE SPECIFICO SU FILE

```
int fseek (FILE *fp, long offset, int refpoint)
```

- sposta l'indicatore di posizione del prossimo byte a cui accedere del valore di **offset** (positivo o negativo) a partire da **refpoint**
- restituisce 0 se l'operazione è possibile

```
long ftell(FILE *fp)
```

restituisce il valore dell'indicatore di posizione del prossimo byte a cui si può accedere

Block I/O

- previous examples to read/write set amount and/or datatype
  - read/write char, string up to max size, formatted data

- block i/o reads/writes variable number of “chunks” of datatype size
  - such as certain number of bytes
  - or a number of items in an array
  - or the items in a struct
  - or even an array of struct!

Block Input: fread()

- **block data input: fread()**
  - **fread(array, size, count, fptr);**
  - **array - address of variable or of 1st element of array**
  - **size - size of the element in bytes**
  - **count - number of elements to read**
  - **fptr - file pointer**
  - **returns number of elements read**
  - **0 indicates error (or EOF)**

Block Output: fwrite()

- **block data output: fwrite()**
  - **fwrite(array, size, count, fptr);**
  - **array - address of variable or of 1st element of array**
  - **size - size of the element in bytes**
  - **count - number of elements to write**
  - **fptr - file pointer**
  - **returns number of elements written**
  - **0 indicates error**

Example

- **Open files**

  infile_ptr = fopen(“input.dat”,”rb”);

  outfile_ptr = fopen(“output.dat”,”wb”);

- **read data into a struct**

  count = fread(&my_struct,sizeof(my_struct),1,infile_ptr);

- **write data from a struct**

  count = fwrite(&my_struct,sizeof(my_struct),1,outfile_ptr);

- **If successful, what is count in each case?**

R/W a caratteri

LETTURA E SCRITTURA DI FILE TESTUALI

1. LETTURA/SCRITTURA DI CARATTERI

`int getc(FILE *fp)`	macro
`int getchar (void)`	macro (=getc(stdin))
`int fgetc(FILE *fp)`	funzione

leggono dal file specificato come parametro (o da stdin) il prossimo carattere e lo restituiscono come intero. Restituiscono EOF in caso di errore

`int putc(int c, FILE *fp)`	macro
`int putchar (int c)`	macro (=putc(c, stdout))
`int fputc(int c, FILE *fp)`	funzione

scrivono sul file specificato come parametro (o su stdout) il carattere specificato come parametro e lo restituiscono come intero. Restituiscono EOF in caso di errore

2. LETTURA/SCRITTURA FORMATTATA

```
int  fscanf(FILE*fp,»stringa_di_controllo»,elenco_ind_elem)
int  scanf(»stringa_di_controllo»,elenco_ind_elem)
```

leggono dal file specificato come parametro (o da stdin) gli elementi specificati. Restituiscono il numero di elementi effettivamente letti o un numero negativo in caso di errore

```
int  fprintf(FILE*fp,»stringa_di_controllo»,elenco_elem)
int  printf(»stringa_di_controllo»,elenco_elem)
```

scrivono sul file specificato come parametro, che può essere **stdprn** (o su stdout) gli elementi specificati. Restituiscono il numero di elementi effettivamente scritti o un numero negativo in caso di errore.

Sequential File Access

Character I/O

fgetc `c = fgetc(fptr);`

- returns next char read if successful
- returns EOF on error
- (if fptr is for a disk file, gets next char in file)
- (if fptr is stdin, gets next char from (typically) keyboard)

fputc `fputc(c, fptr);`

- returns char written if successful
- returns EOF on error
- (if fptr is for a disk file, writes char to file)
- (if fptr is stdout, writes char to (typically) screen

Reading and Writing from Standard Input

Recall C library procedures for reading/writing from standard input:

- **printf("control string",...)** writes to standard output
- **scanf("control string",...)** reads from standard input
- **int getchar(void)** read one character from standard input
- **putchar(int)** write one character to standard output

Example:

```
/* this proc writes what it reads */
void echo(void)
{
  int ch;        /* must be an integer since EOF is -1 */

  while ((ch = getchar()) != EOF)
  {
    putchar(ch);
  }
}
```

Reading from and writing to a file

There are versions of these routines that read from and write to files.

- **`fprintf(fd, “control string”...)`** writes to **`*fd`**
- **`fscanf(fd, “control string”...)`** reads from **`*fd`**
- **`int fgetc(fd)`** read character from **`*fd`**
- **`int fputc(int ch, FILE *fd)`** write character **`ch`** to **`*fd`**

```
void echo(void)  /* this proc writes what it reads */
{
  int ch;    /* must be an integer since EOF is -1 */
  FILE *fdin, *fdout;

  fdin = fopen(“inputfile”, “r”);
  fdout = fopen(“outputfile”, “w”);

  while ((ch = fgetc(fdin)) != EOF)
  {
    fputc(ch, fdout);
  }
  fclose(fdin); fclose(fdout);
}
```

LETTURA/SCRITTURA CARATTERI

```
#include <stdio.h>

void main(void)
{  FILE *f1, *f2;
   int carattere;

  if ((f1=fopen("prova1.txt","w"))!=NULL)
  {
   printf("Inserire i caratteri da memorizzare in prova1.txt\n");

   while ((carattere=getchar())!='\n')
    {
     putc(carattere,f1);
    }
    fclose(f1); /*f1 viene chiuso in scrittura*/

    if((f1=fopen("prova1.txt","r"))!=NULL)
    {
     if((f2=fopen("prova2.txt","w"))!=NULL)
     {
      printf("\nprova1.txt viene visualizzato sullo schermo e
copiato            in prova2\n");
     /*ciclo che copia f1 in f2 carattere per carattere*/
      while((carattere=getc(f1))!=EOF)
      {
       putchar(carattere);/*visualizza ogni carattere */
       putc(carattere,f2);
      } /* end while */

      fclose(f1);
      fclose(f2);
     } /* fine ramo apertura corretta di f1 e f2 */
     else /* apertura non corretta di f2 */
     {fclose(f1);
      printf("il file prova2.txt non puo' essere aperto in
                  scrittura\n");
     }
    } /* fine ramo apertura corretta di f1 in lettura */
    else /* f1 non puo' essere aperto in lettura */
    {
     printf("il file prova1.txt non puo'essere aperto in
lettura\n");
    }
  } /* fine ramo apertura corretta di f1 in scrittura */
  else
  {
    printf("il file prova1.txt non puo' essere aperto in
scrittura");
  }
}/*fine main */
```

R/W Formattata e a Stringhe

LETTURA E SCRITTURA DI FILE TESTUALI

1. LETTURA/SCRITTURA DI CARATTERI

`int getc(FILE *fp)`	macro
`int getchar (void)`	macro (=getc(stdin))
`int fgetc(FILE *fp)`	funzione

leggono dal file specificato come parametro (o da stdin) il prossimo carattere e lo restituiscono come intero. Restituiscono EOF in caso di errore

`int putc(int c, FILE *fp)`	macro
`int putchar (int c)`	macro (=putc(c, stdout))
`int fputc(int c, FILE *fp)`	funzione

scrivono sul file specificato come parametro (o su stdout) il carattere specificato come parametro e lo restituiscono come intero. Restituiscono EOF in caso di errore

2. LETTURA/SCRITTURA FORMATTATA

```
int  fscanf(FILE*fp,»stringa_di_controllo»,elenco_ind_elem)
int  scanf(»stringa_di_controllo»,elenco_ind_elem)
```

leggono dal file specificato come parametro (o da stdin) gli elementi specificati. Restituiscono il numero di elementi effettivamente letti o un numero negativo in caso di errore

```
int  fprintf(FILE*fp,»stringa_di_controllo»,elenco_elem)
int  printf(»stringa_di_controllo»,elenco_elem)
```

scrivono sul file specificato come parametro, che può essere **stdprn** (o su stdout) gli elementi specificati. Restituiscono il numero di elementi effettivamente scritti o un numero negativo in caso di errore.

Sequential File Access

Formatted I/O

fprintf() **fprintf(fptr, “ctrl-str”, arg,...);**

- **just like printf() except you specify file pointer**

fscanf() **fscanf(fptr, “ctrl-str”, arg,...);**

- **just like scanf() except you specify file pointer**

LETTURA/SCRITTURA FORMATTATA (ARRAY DI STRUTTURE)

```
#include <stdio.h>
#define Lmax 10
#define N 5
typedef struct {
          char nome[Lmax];
          int somma;
          float media;
          }ELE;
FILE *f1;

void main(void)
{
  ELE  elenco[N];
  int i;
  void Scrivi_su_file(const char *nome_file, ELE dati[]);

  printf("inserire gli elementi dell'elenco\n");
  for (i=0;i<N;i++)
   {
   printf("valori relativi all'elemento %d\n",i);
   scanf("%s%d%f",elenco[i].nome, &elenco[i].somma,
&elenco[i].media);
   }

  printf("\nvisualizzazione degli elementi nell'elenco\n");
  for (i=0;i<N;i++)
   {
    printf("valori relativi all'elemento %d\n",i);
    printf("%s %d %.2f\n",elenco[i].nome,elenco[i].somma,
                  elenco[i].media);
   }
  Scrivi_su_file("anna.txt",elenco);
}/* fine main */

void Scrivi_su_file(const char *nome_file, ELE dati[])
{ int i;

  if ((f1=fopen(nome_file,"w"))!=NULL)
   {
     for (i=0;i<N;i++)
     { printf("valori relativi all'elemento %d\n",i);
       fprintf(f1,"%s %d %.3f\n",dati[i].nome,dati[i].somma,
              dati[i].media);
     }
   }
   else
    printf("il file non puo' essere aperto\n");
} /* fine procedura */
```

3. LETTURA/SCRITTURA DI STRINGHE (LINEE)

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

- legge dal file specificato come parametro al più **n - 1** caratteri e li memorizza nell'array **s**, aggiungendo l'indicatore di fine stringa \0
- la lettura si interrompe (prima di n-1) se incontra il carattere di **newline** (\n) (che viene inserito) o la fine del file. In ogni caso inserisce \0.
- resituisce **s** se ha letto almeno un carattere, **NULL** se incontra la fine del file a inizio lettura

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

- scrive sul file specificato come parametro i caratteri contenuti nell'array **s**, eliminando l'indicatore di fine stringa \0
- resituisce 0 se l'operazione è andata a buon fine, un valore diverso da 0 se non è andata a buon fine.

`char *gets(char *s)` funzione

- legge da standard input (da video), i caratteri fino al carattere newline, li memorizza nel vettore **s**, aggiungendo l'indicatore di fine stringa \0 (non inserisce il carattere newline)
- resituisce **s** se ha letto almeno un carattere, **NULL** se non ha letto nessun carattere

`int puts(char *s)` funzione

- scrive su standard output (video) i caratteri contenuti nel vettore **s**, inserendo il carattere di newline e eliminando l'indicatore di fine stringa \0
- restituisce 0 se l'operazione è andata a buon fine, un valore diverso da 0 se non è andata a buon fine.

Sequential File Access

String I/O

fgets fgets(array, maxsize, fptr);

- reads maxsize - 1 chars; appends null char
- newline or EOF terminates read
- newline is retained
- returns pointer to char (array) if successful
- returns null if error

fputs fputs(array, fptr);

- writes array to file without terminal null
- returns nonnegative (int) if successful
- returns EOF on error

File Text vs File Binari

File Types

- **Text files**
  - for portability; all systems support these
  - a text stream is a sequence of characters divided into lines
  - each line is 0 or more characters followed by a newline character
  - C not concerned with how it is physically stored
  - (DOS inserts <cr><lf> in place of newline)
  - File ends with EOF condition set (defined in stdio.h)
- **Binary files**
  - not supported by all systems; might get treated as a text file
  - use when speed or storage conditions require
  - does not insert or change characters

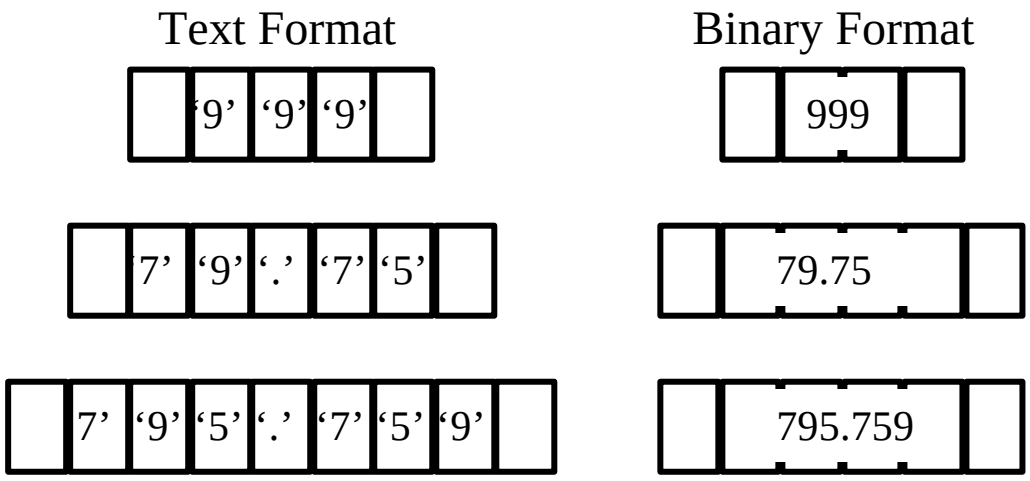
Writing a Text File

- **Typically for a report to disk**
  - just like a printed report, except data goes to disk, not printer
  - several choices for output; simplest to use is fprintf()
  - file is opened in text mode
- **Different data format for characters & “numbers”**
  - text output is made up of ASCII characters
  - numbers (used for math calculations) are binary values that must be converted to text before display/output
  - printf()/fprintf() does the conversion for you!

File Open Mode vs Data Format

- **Text Mode**
  - converts ‘\n’ to CR/LF for DOS
  - adds EOF (1Ah)
- **Binary Mode**
  - no conversion
- **Data: Text vs Binary Format**
  - how numbers are stored
  - If need binary format data, use binary mode

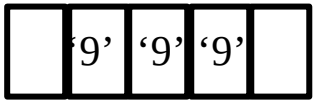
Text vs Binary Format



- text format usually requires more bytes for disk storage
- in text format, numbers are stored as ASCII characters
- using binary number storage for text files causes problems!
- need to use binary format in memory for math operations
- requires conversion!

A Little Bit of Storage

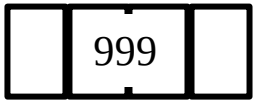
Text Format



Bit Pattern:
00111001 00111001 00111001

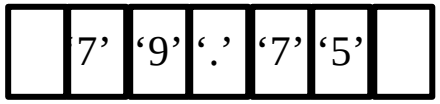
Decimal Value:
3,750,201

Binary Format



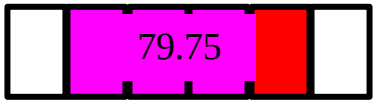
Bit Pattern:
00000011 11100111

Decimal Value:
999



Bit Pattern:
00110111 00111001 00101110 00110111 00110101

Decimal Value:
959,330,101



Decimal Value:
79.75

Bit Pattern:
00000000 00011111 00100111 00000001

mantissa exponent

Easy Text Output

Use printf() / fprintf()

to output characters, use %c and %s - not an issue

to output numbers, use %d, %f, etc. - still not an issue!

the binary-stored number 999

(999 in decimal, 00000011 11100111 in binary)

becomes the ASCII character string 999

(3,750,201 in decimal, 00111001 00111001 00111001 in binary)

because printf() / fprintf() does the conversion!

Oh, Here's the Rub!

- **Text Input is another matter!**
  - **could use fgetc()**
    - » but must concatenate strings, convert numbers
  - **could use fscanf()**
    - » OK for some text, but not all
    - » how to read in strings embedded in quotes and/or with embedded blanks? (e.g., a comma-delimited text data file)
  - **could use fgets()**
    - » reads entire line (record) as a string
    - » still must parse string

Parsing

- **“to break down into component parts”**

Given a typical “record” from a comma-delimited file:

“Mary Jones”, 3, 12, 1960, M, 254.59

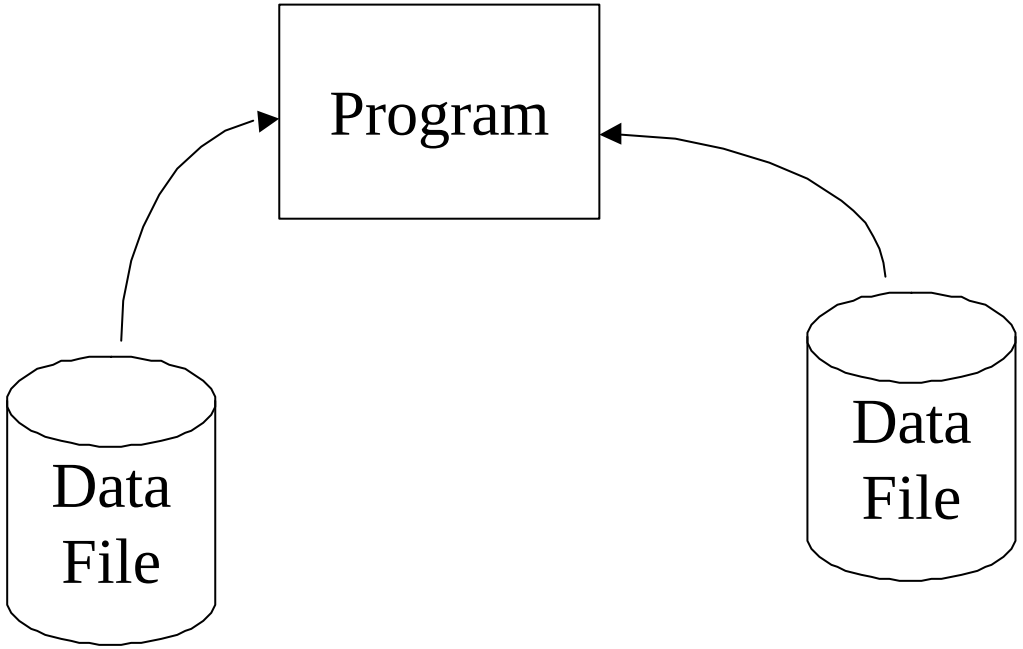
Parse:

```
char name[20];
int month;
int day;
int year;
char marital_sts;
float acct_bal;
```

How? Many ways, but in all of them, by using several tools together!

Esercizi

Writing Sequential Output Files



Writing Sequential Output Files

Create 2 output files containing data that is in order by key.
file01.dat file02.dat

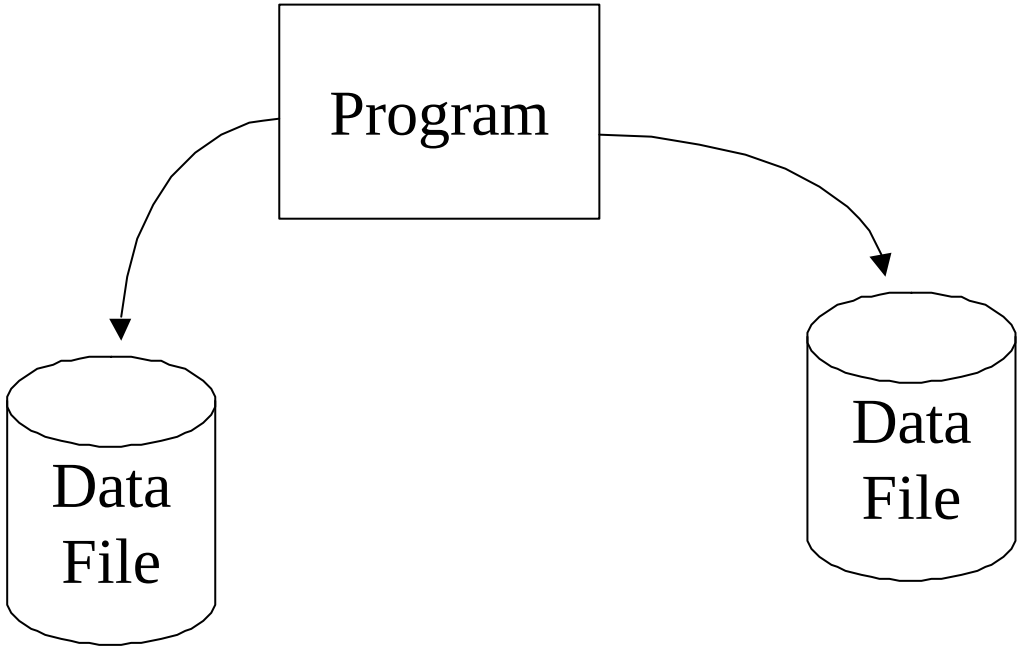
Open the files write mode and write the data sequentially from an initialized array of structs. The file contains binary data.

Each “record” of data is represented by the following struct:

```
struct my-rec
{
    int             id;         /* ordered key */
    char            last_name[20];
    char            first_name[20];
    char            acct_type;
    double          acct_bal;
};
```

Write the program!

Reading Files Sequentially



Reading Files Sequentially

Assume input files containing data that is in order by key.

file01.dat file02.dat

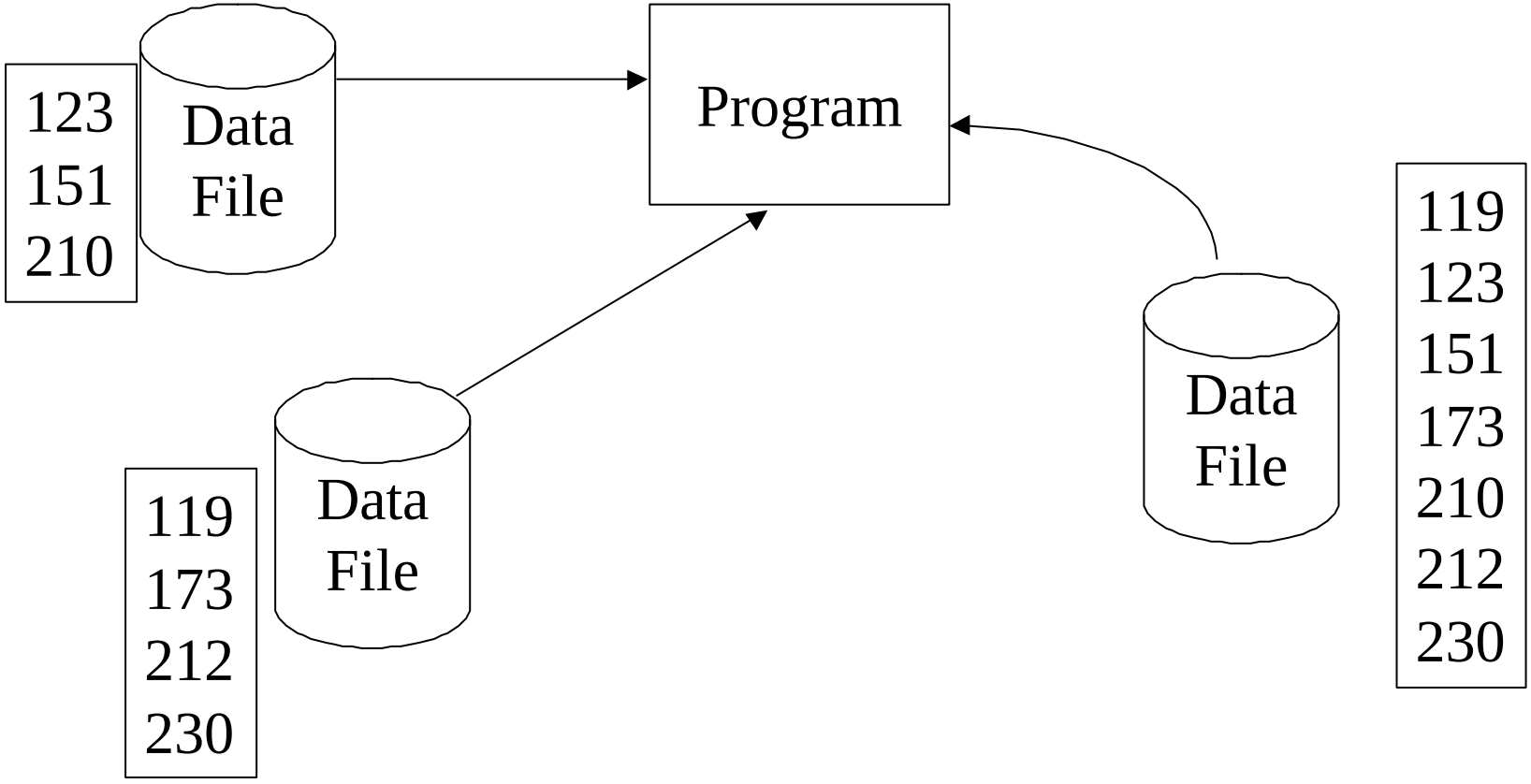
Open the files read-only and read in the data sequentially into an instance of a struct. Display the data. The file contain binary data.

Each “record” of data is represented by the following struct:

```
struct my-rec
{
        int     id;     /* ordered key */
        char    last_name[20];
        char    first_name[20];
        char    acct_type;
        double  acct_bal;
};
```

Write the program!

Merging Ordered Data Files



Merging Ordered Data Files

Assume two input files containing data that is in order by key.

file01.dat file02.dat

Open both files read-only. Read in data and write out to a single, ordered results file (result.dat). The files contain binary data.

Each “record” of data is represented by the following struct:

```
struct my-rec
{
        int     id;     /* ordered key */
        char    last_name[20];
        char    first_name[20];
        char    acct_type;
        double  acct_bal;
};
```

Write the program!