POLITECNICO DI MILANO
Applicazione SchedaIncarico

Scheda Riassuntiva

Anno Accademico	2005/06
Facoltà	Facolta' di Ingegneria Industriale
Tipo Insegnamento	MONODISCIPLINARE
Codice Identificativo	060065
Denominazione Insegnamento	INFORMATICA C
Docente	MARTUCCI RENATO
CFU	5.0

Corsi di Studio cui l'insegnamento è offerto

Nome Corso di Laurea	Corso Unione	Indirizzo	DA	A
Ing.IV(1 liv.) - BV (100) INGEGNERIA AEROSPAZIALE	-	*	N	ZZZZ

- M8 parte b
  - Files
    - Files binari - cenni
    - Esercizi

http://www.elet.polimi.it/upload/martucci/index.html

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

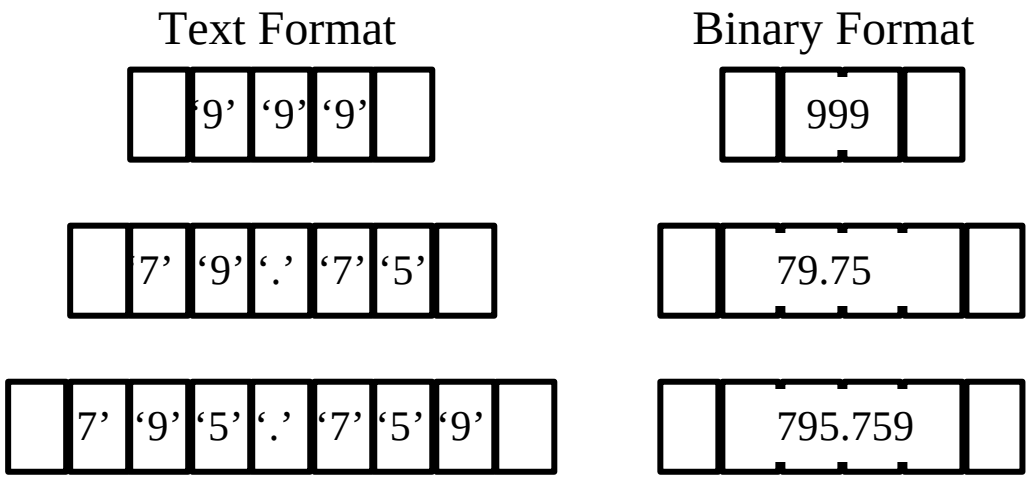
Writing a Text File

- **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

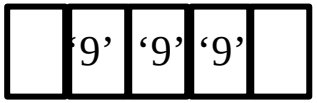
Text vs Binary Format



- 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

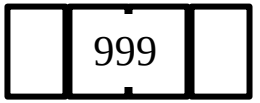
Text Format



Bit Pattern:
00111001 00111001 00111001

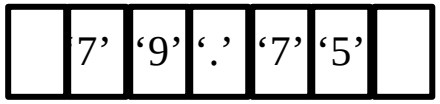
Decimal Value:
3,750,201

Binary Format



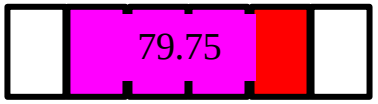
Bit Pattern:
00000011 11100111

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!

Esercizi

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 */
```

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);
```

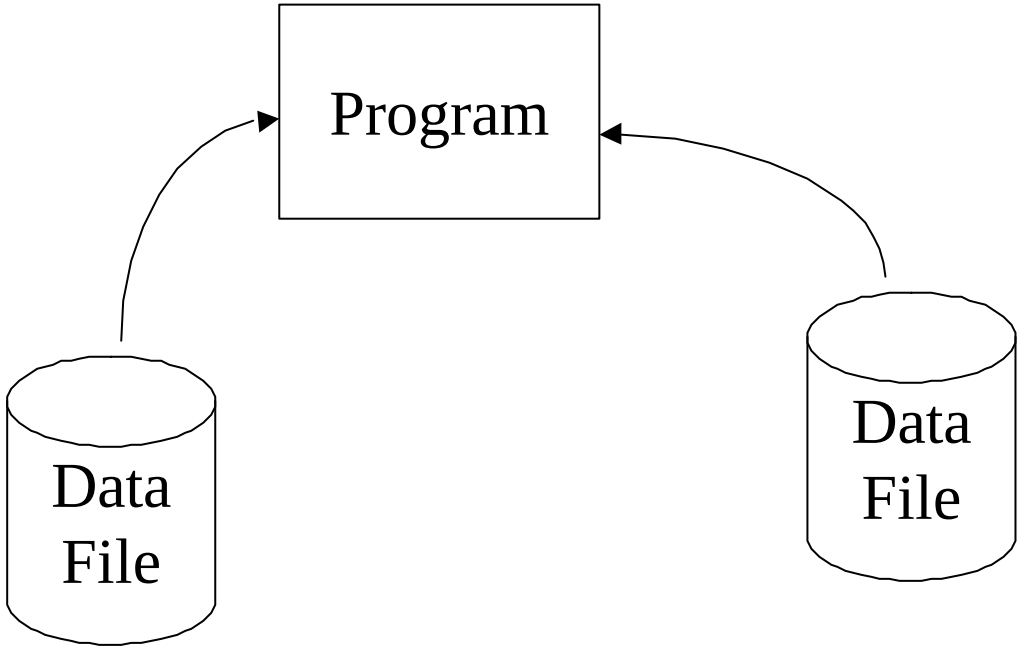
  poteva essere scritto anche come:

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

LETTURA/SCRITTURA 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 */
```

Writing Sequential Output Files



Writing Sequential Output Files

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

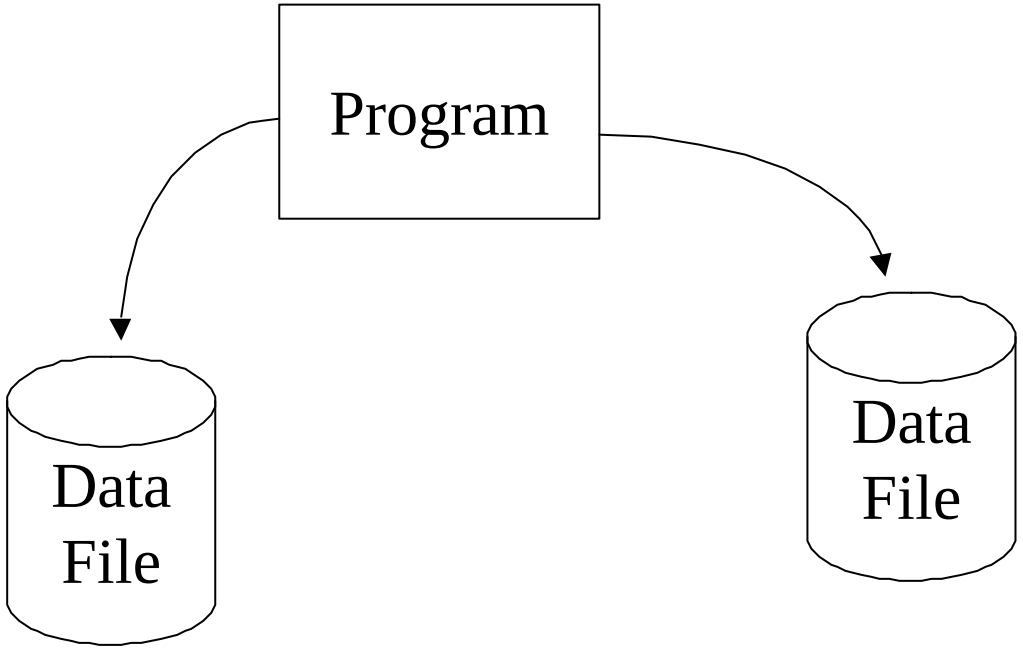
Open the files write mode and write the data sequentially from an initialized array of structs. The file contains binary data.

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

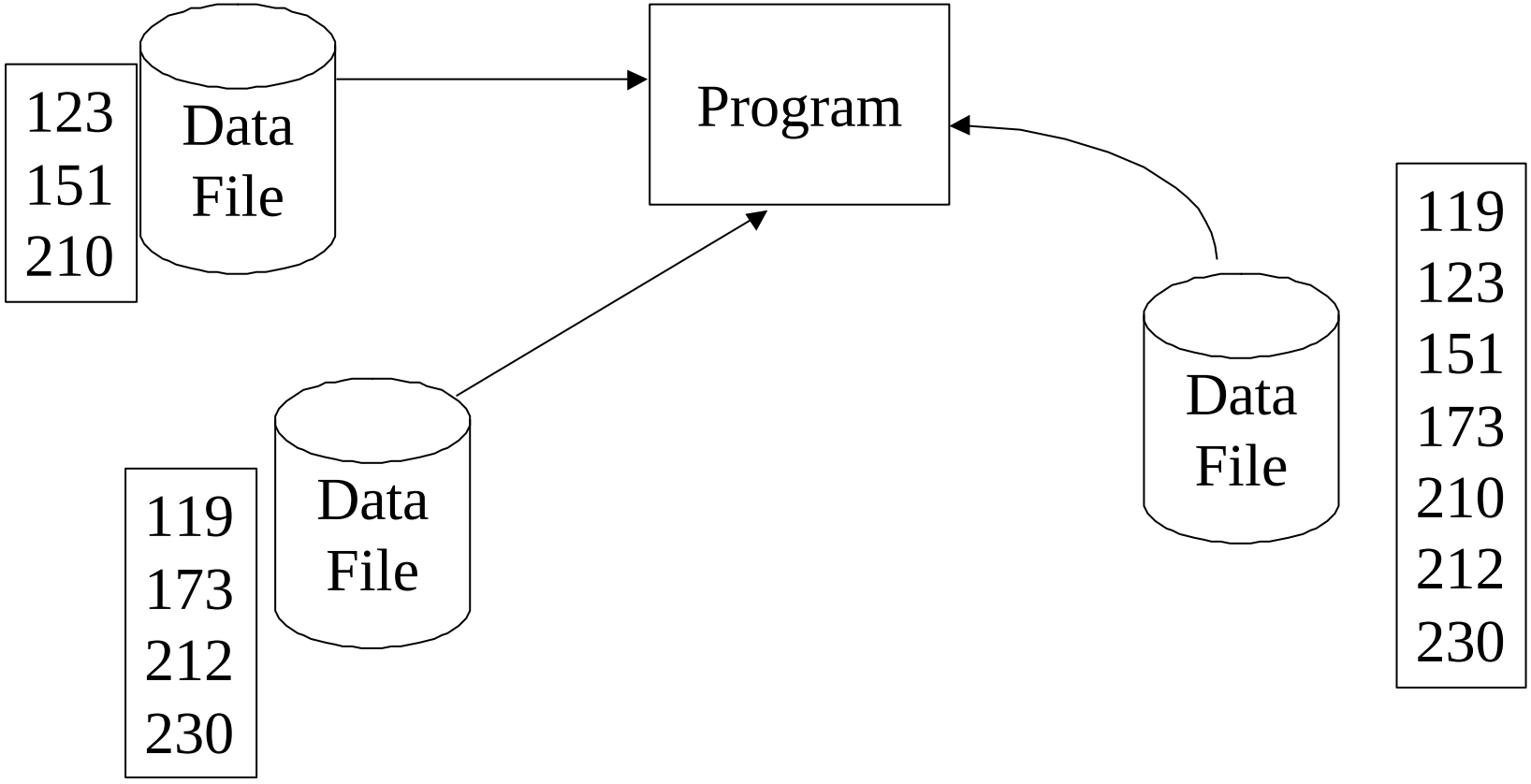
Open the files read-only and read in the data sequentially into an instance of a struct. Display the data. The file contain binary data.

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!