Linguaggio C: Files

Generalità

- Persistenza
  - Dati (primitivi o strutturati) prodotti e manipolati in un programma C possono essere persi quando il programma termina, a meno che questi non sono memorizzati in un dispositivo di persistenza, ad esempio, disponibile su file system.
- Input/Output
  - Spesso i dati da immettere e/o da presentare all'utente sono così numerosi che usare l'immissione da tastiera o la visualizzazione a console può risultare inefficiente ed inefficace. Spesso, si preferisce usare file per memorizzare dati (output) o file di dati precedentemente memorizzati (input)

Gerarchie di dati

- Nei file, i dati NON sono memorizzati in maniera casuale o non sistematica. Essi seguono la organizzazione indicata dai tipi di dati nel linguaggio C.
- In particolare, I file estendono la gerarchia esistente sui dati:

  - Bit – più piccolo dato
    - `0` / `1`
  - Byte – 8 bits
    - Usato per memorizzare un carattere, digits decimali, simboli speciali.
  - Variabile – gruppo di byte
    - usato per memorizzare un dato con un significato nel dominio del problema

Gerarchie di dati

- Record – gruppo di variabili correlate
  - Usato per memorizzare una struttura di variabili che hanno nel dominio del problema una relazione.
  - ad esempio, creati con struct
- Data file – insieme di record correlati
- Database- gruppi di data file correlati

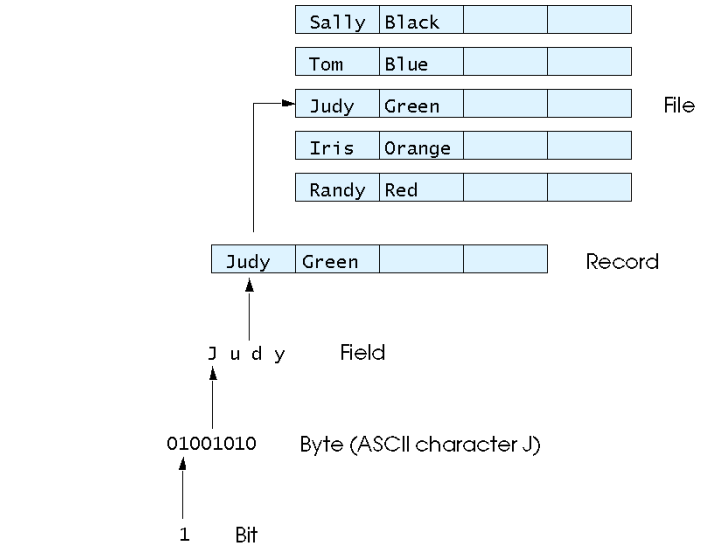


Fig. 11.1 The data hierarchy.

Più precisamente, il file supporta un livello di organizzazione di dati che estende il record (struct).

Gerarchie di dati

- Un file quindi può essere visto come un elenco di record in cui i rispettivi byte sono collocate in sequenza.
- In un file ogni record è descritto da un identificativo univoco (*key*) usato per ritrovare quel record. Così, ì record sono memorizzati seguendo un ordinamento della key.
- Un file termina con un marcatore di terminazione (*end-of-file*) o con uno specifico byte.

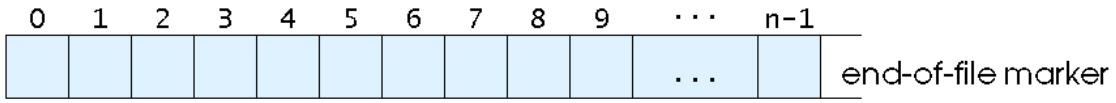


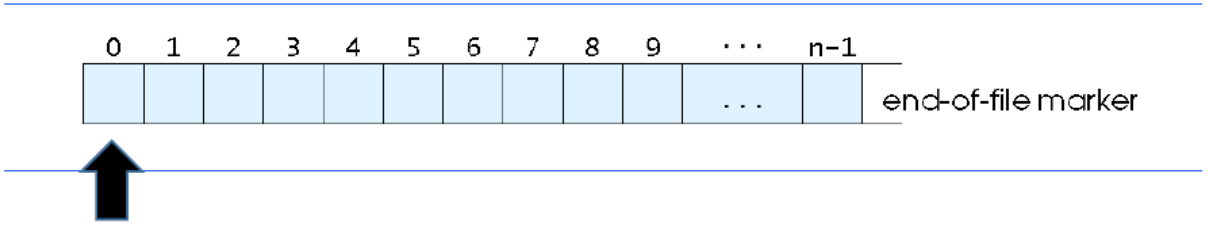
Fig. 11.2 C's view of a file of *n* bytes.

Stream

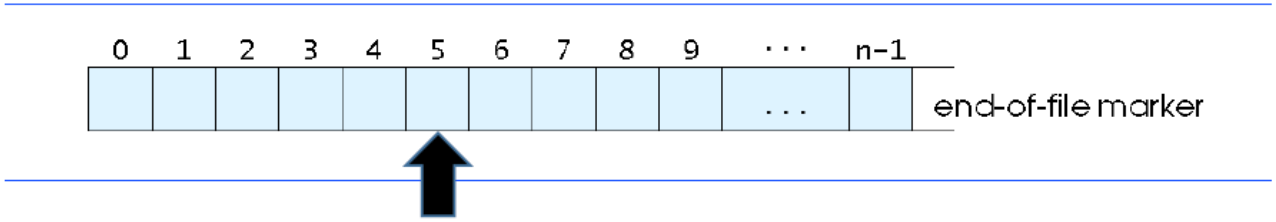
- Nel linguaggio C, un file non è gestito direttamente come entità di sistema operativo, ma attraverso una sua astrazione, *stream (flusso di byte)*.
- Uno stream denota un canale di comunicazione da/verso il file creato quando un file è "aperto" ed attraverso esso si può scrivere o leggere dati. Così, non si gestisce il file in quanto tale, ma attraverso un suo "surrogato" operativo a livello di programmazione.
- Uno stesso stream può essere usato per supportare entrambe le **direzioni** della comunicazione
  - Output: programma → file
  - Input: file → programma

Stream

- Gli stream possono essere distinti in base alla modalità di **accesso** al file
  - Sequenziale, lo stream è letto/scritto partendo dal primo byte fino ad arrivare all'ultimo



  - Casuale, lo stream può essere letto/scritto accedendo direttamente al byte(s)



Stream

- Gli stream possono essere distinti in base alla modalità di **accesso** al file
  - Sequenziale, lo stream è letto/scritto partendo dal primo byte fino ad arrivare all'ultimo.
    - detti anche file **testuali** perchè possono anche essere aperti con un normale editor di testo
  - Casuale, lo stream può essere letto/scritto accedendo direttamente al byte(s)
    - detti anche file **binari** perchè NON possono essere aperti con un normale editor di testo

Stream

- In C, usiamo gli stream attraverso puntatori:
  - **FILE *pointer**
  - il tipo di dato struct FILE è dichiarato in <stdio.h>
  - i campi della struct FILE sono informazioni da sistema operativo, tra cui
    - **Indice** del file nella tabella dei file aperti
    - **File Control Block** usato prime di accedere al file e contiene informazioni sui *permessi*, *data* creazione, *locazione* fisica dei blocchi

Stream

- In C, usiamo gli stream attraverso puntatori:
  - **FILE *pointer**
  - nell'accesso sequenziale, lo spazio occupato da un valore è **relato** al **numero** di **caratteri** per scrivere quel valore. Quindi, il puntatore scandisce i caratteri-elementi di un valore.
  - nell'accesso casuale, il puntatore accede **indipedentemente** dai digit usati per un valore, ma **dipendentemente** dal suo tipo di dato.

Stream

- Una volta dichiarato un pointer ed a prescidere dalla modalità di accesso, possiamo usare lo stream per
  - Apertura file
  - Lettura/Scrittura dati
  - Chiusura File
  - Verifica end-of-file
  - Riavvolgimento dello stream
  - Accesso diretto

Apertura file

- **FILE* fopen(const char* filename, const char* mode);**
  - filename: nome del file da aprire
  - modalità di accesso, cioè input/output
  - se il file esiste (su file system), restituisce un puntatore valido, altrimenti dà NULL.
  - modalità ammissibili sono:

Mode	Description
r	Open a file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Append; open or create a file for writing at end of file.
r+	Open a file for update (reading and writing).
w+	Create a file for update. If the file already exists, discard the current contents.
a+	Append; open or create a file for update; writing is done at the end of the file.
rb	Open a file for reading in binary mode.
wb	Create a file for writing in binary mode. If the file already exists, discard the current contents.
ab	Append; open or create a file for writing at end of file in binary mode.
rb+	Open a file for update (reading and writing) in binary mode.
wb+	Create a file for update in binary mode. If the file already exists, discard the current contents.
ab+	Append; open or create a file for update in binary mode; writing is done at the end of the file.

Chiusura file

- **int fclose(const char* filename);**
  - filename: nome del file da chiudere
  - normalmente, un file ha un buffer di memoria associato per rendere efficienti le operazioni. Questo viene svuotato nello stream quando si chiude un file.

Chiusura file

- Esempio:

```c
#include <stdio.h>

int main() {
  FILE *file;

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Lettura da file

- **int fscanf(FILE* stream, const char* format, ...);**
  - stream: nome del file da cui leggere i dati
  - format: specificatore del formato dei dati
  - riprende la stessa sintassi di scanf(), considerato che scanf() legge da un particolare stream, cioè il dispositivo standard di input, *stdin*.

Lettura da file

- Esempio:

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Scrittura su file

- **int fprintf(FILE* stream, const char* format, ...);**
  - stream: nome del file su cui scrivere i dati
  - format: specificatore del formato dei dati
  - riprende la stessa sintassi di printf(), considerato che printf() scrive su un particolare stream, cioè il dispositivo standard di output, *stdout*.

Scrittura su file

- Esempio:

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");

}
```

Fine del file

- **int feof(FILE* stream);**
  - stream : nome dello stream da verificare
  - restituisce true se il file è terminato

Esempio

```
1  /* Fig. 11.3: fig11_03.c
2     Create a sequential file */
3  #include <stdio.h>
4
5  int main()
6  {
7     int account;     /* account number */
8     char name[ 30 ]; /* account name */
9     double balance;  /* account balance */
10
11    FILE *cfPtr;     /* cfPtr = clients.dat file pointer */
12
13    /* fopen opens file. Exit program if unable to create file  */
14    if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
15       printf( "File could not be opened\n" );
16    } /* end if */
17    else {
18       printf( "Enter the account, name, and balance.\n" );
19       printf( "Enter EOF to end input.\n" );
20       printf( "? " );
21       scanf( "%d%s%lf", &account, name, &balance );
22
```

Esempio

```
23        /* write account, name and balance into file with fprintf */
24        while ( !feof( stdin ) ) {
25           fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
26           printf( "? " );
27           scanf( "%d%s%lf", &account, name, &balance );
28        } /* end while */
29
30        fclose( cfPtr ); /* fclose closes file */
31     } /* end else */
32
33     return 0; /* indicates successful termination */
34
35  } /* end main */
```

Esempio

```
Enter the account, name, and balance.
Enter EOF to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Altro Esempio

```
1  /* Fig. 11.7: fig11_07.c
2     Reading and printing a sequential file */
3  #include <stdio.h>
4
5  int main()
6  {
7     int account;     /* account number */
8     char name[ 30 ]; /* account name */
9     double balance;  /* account balance */
10
11    FILE *cfPtr;     /* cfPtr = clients.dat file pointer */
12
13    /* fopen opens file; exits program if file cannot be opened */
14    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
15       printf( "File could not be opened\n" );
16    } /* end if */
17    else { /* read account, name and balance from file */
18       printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
19       fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
20
21       /* while not end of file */
22       while ( !feof( cfPtr ) ) {
23          printf( "%-10d%-13s%7.2f\n", account, name, balance );
24          fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
25       } /* end while */
26
```

Altro Esempio

```
27        fclose( cfPtr ); /* fclose closes the file */
28     } /* end else */
29
30     return 0; /* indicates successful termination */
31
32  } /* end main */
```

Altro Esempio

```
Account    Name          Balance
100        Jones           24.98
200        Doe            345.67
300        White            0.00
400        Stone          -42.16
500        Rich           224.62
```

Riavvolgimento dello Stream

- **void rewind(FILE* stream);**
  - stream: nome dello stream da riavvolgere
  - usato per riportare il puntatore ad inizio del file

Riavvolgimento dello Stream su accesso sequenziale

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");
```

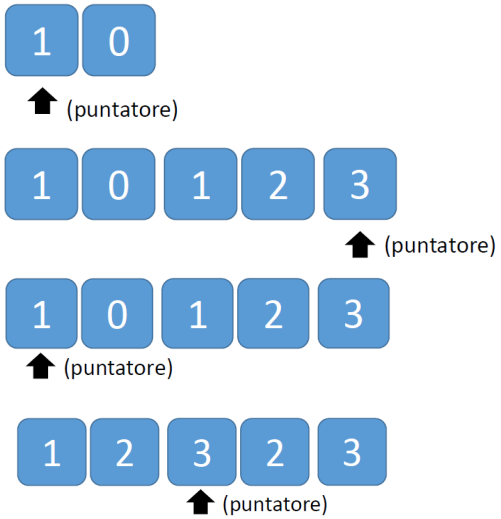
Riavvolgimento dello Stream su accesso sequenziale

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");
```

Dove viene scritto il valore 123 rispetto al valore letto in value?

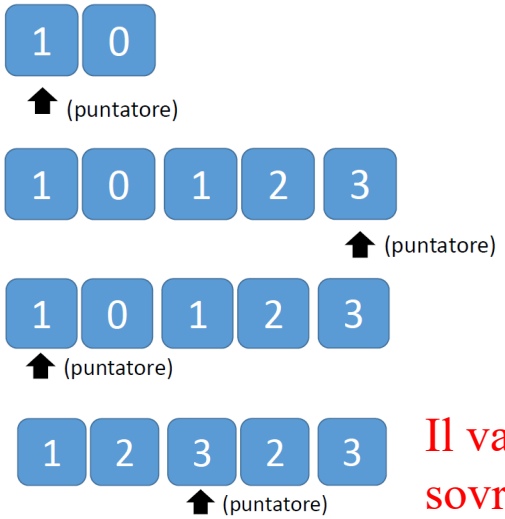
Riavvolgimento dello Stream su accesso sequenziale

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");
```



Riavvolgimento dello Stream su accesso sequenziale

```c
#include <stdio.h>
int value=0; // variabile

int main() {
  FILE *file; // puntatore a file

  if((file = fopen("test.txt","r+")) == NULL) {
    puts("Errore nell'apertura"); // errore in apertura
  }
  else {
    puts("File Aperto"); // apertura file

    fscanf(file, "%d", &value); // leggo valore da file (10)
    printf("Valore Letto: %d\n", value);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");

    rewind(file);

    fprintf(file, "%d", 123); // scrivo su file
    puts("Valore Scritto!");
  }

  if(!fclose(file)) // chiusura file
    puts("File Chiuso");
```



Il valore pre-esistente viene sovrascritto!

Altro Esempio

```
1  /* Fig. 11.8: fig11_08.c
2     Credit inquiry program */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int request;     /* request number */
9     int account;     /* account number */
10    double balance;  /* account balance */
11    char name[ 30 ]; /* account name */
12    FILE *cfPtr;     /* clients.dat file pointer */
13
14    /* fopen opens the file; exits program if file cannot be opened */
15    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
16       printf( "File could not be opened\n" );
17    } /* end if */
18    else {
19
20       /* display request options */
21       printf( "Enter request\n"
22               " 1 - List accounts with zero balances\n"
23               " 2 - List accounts with credit balances\n"
24               " 3 - List accounts with debit balances\n"
25               " 4 - End of run\n? " );
```

Altro Esempio

```
26       scanf( "%d", &request );
27
28       /* process user's request */
29       while ( request != 4 ) {
30
31          /* read account, name and balance from file */
32          fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
33
34          switch ( request ) {
35
36             case 1:
37                printf( "\nAccounts with zero balances:\n" );
38
39                /* read file contents (until eof) */
40                while ( !feof( cfPtr ) ) {
41
42                   if ( balance == 0 ) {
43                      printf( "%-10d%-13s%7.2f\n",
44                              account, name, balance );
45                   } /* end if */
46
47                   /* read account, name and balance from file */
48                   fscanf( cfPtr, "%d%s%lf",
49                           &account, name, &balance );
50                } /* end while */
51
```

Altro Esempio

```
52                break;
53
54             case 2:
55                printf( "\nAccounts with credit balances:\n" );
56
57                /* read file contents (until eof) */
58                while ( !feof( cfPtr ) ) {
59
60                   if ( balance < 0 ) {
61                      printf( "%-10d%-13s%7.2f\n",
62                              account, name, balance );
63                   } /* end if */
64
65                   /* read account, name and balance from file */
66                   fscanf( cfPtr, "%d%s%lf",
67                           &account, name, &balance );
68                } /* end while */
69
70                break;
71
72             case 3:
73                printf( "\nAccounts with debit balances:\n" );
74
```

Altro Esempio

```
75                /* read file contents (until eof) */
76                while ( !feof( cfPtr ) ) {
77
78                   if ( balance > 0 ) {
79                      printf( "%-10d%-13s%7.2f\n",
80                              account, name, balance );
81                   } /* end if */
82
83                   /* read account, name and balance from file */
84                   fscanf( cfPtr, "%d%s%lf",
85                           &account, name, &balance );
86                } /* end while */
87
88                break;
89
90             } /* end switch */
91
92             rewind( cfPtr ); /* return cfPtr to beginning of file */
93
94             printf( "\n? " );
95             scanf( "%d", &request );
96          } /* end while */
97
```

Altro Esempio

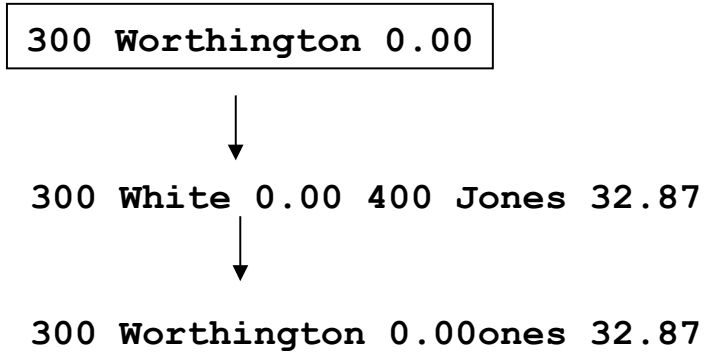
```
98          printf( "End of run.\n" );
99          fclose( cfPtr ); /* fclose closes the file */
100      } /* end else */
101
102      return 0; /* indicates successful termination */
103
104 } /* end main */
```

Altro Esempio

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1

Accounts with zero balances:
300       White           0.00

? 2

Accounts with credit balances:
400       Stone         -42.16

? 3

Accounts with debit balances:
100       Jones          24.98
200       Doe           345.67
500       Rich          224.62

? 4
End of run.
```

Accesso sequenziale: discussione

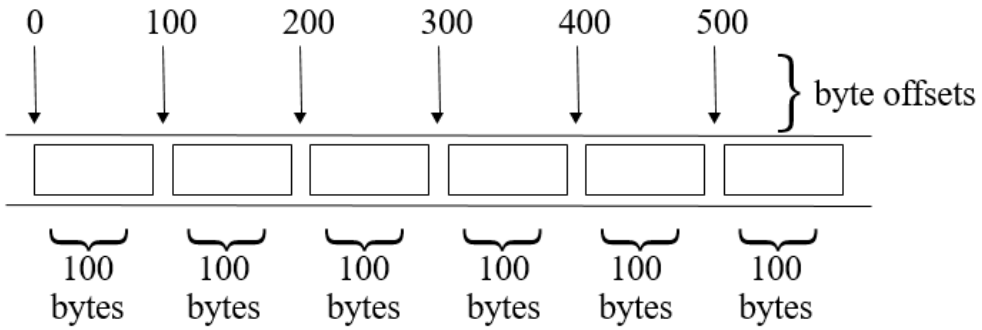
- Il contenuto viene letto elemento per elemento, il che porta a rischi di sovrascrittura di contenuti precedentemente memorizzati nel file
- Dati possono occupare differenti spazi:
  - 1, 34, -890 sono interi ma occupano differente spazio su disco
- Esempio: sostituiamo Worthington con White



- Questo NON accade nell'accesso casuale

Accesso casuale

- Gli elementi non sono selezionati, dopo aver scandito quelli precedenti, ma accedendo direttamente alla locazione.
- Flessibile nell' aggiornamento e modifica di file precedentemente scritti, senza rischio di sovrascrittura.
- Usato per gestire file binari
- I dati dello stesso tipo occupano gli stessi spazi e i valori sono memorizzati riportando la loro codifica binaria



Stream

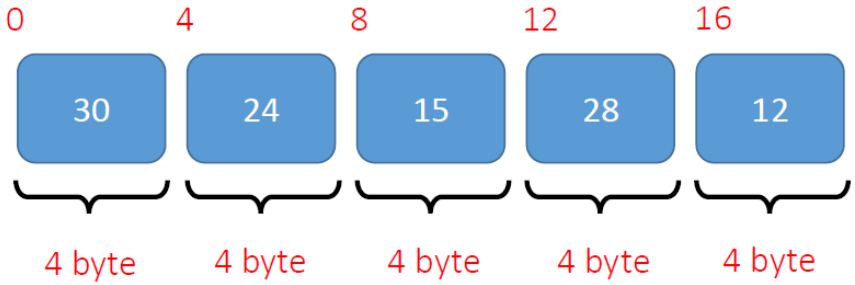
- Anche per i file binari, possiamo usare le medesime funzioni
  - Apertura file
  - Lettura/Scrittura dati
  - Chiusura File
  - Verifica end-of-file
  - Riavvolgimento dello stream
  - Accesso diretto
- Nei file binari non si parla più di elementi o caratteri ma di blocchi di dati

Scrittura su file

- **size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);**
  - scrive su *ptr, nmemb* blocchi di dati, aventi ciascun una dimesione *size*
  - ptr: puntatore al dato da «copiare» nel blocco dati
  - size: dimensione del blocco dati da scrivere
  - nmemb: numero dei blocchi di memoria da scrivere
  - stream: puntatore al file su cui scrivere i dati
  - size_t: (sotto)tipo di dato di int a 2 byte, usato per dimensioni di variabili o contatori

Scrittura su file

- **size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);**
  - nmemb: numero dei blocchi di memoria da scrivere
    - tipicamente, nmemb=1
  - size: dimensione del blocco dati da scrivere
    - necessario sapere a-priori la dimensione dei dati, quindi useremo *sizeof()*
    - fwrite(&voto, sizeof(voto), 1, file)



Esempio

```
1  /* Fig. 11.11: fig11_11.c
2     Creating a randomly accessed file sequentially */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum;           /* account number */
8     char lastName[ 15 ];  /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance;        /* account balance */
11 }; /* end structure clientData */
12
13 int main()
14 {
15    int i; /* counter */
16
17    /* create clientData with no information */
18    struct clientData blankClient = { 0, "", "", 0.0 };
19
20    FILE *cfPtr; /* credit.dat file pointer */
21
22    /* fopen opens the file; exits if file cannot be opened */
23    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
24       printf( "File could not be opened.\n" );
25    } /* end if */
```

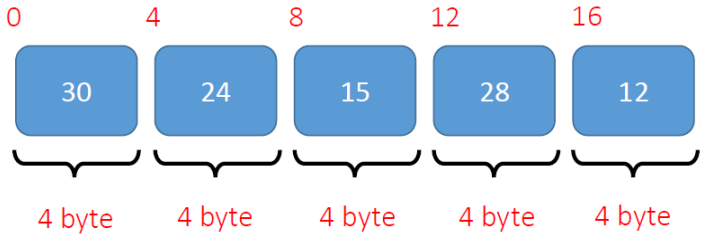
Esempio

```
26    else {
27
28       /* output 100 blank records to file */
29       for ( i = 1; i <= 100; i++ ) {
30          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
31       } /* end for */
32
33       fclose ( cfPtr ); /* fclose closes the file */
34    } /* end else */
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

Lettura da file

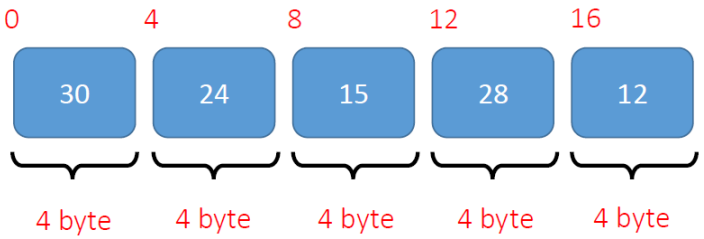
- **size_t fread(const void* ptr, size_t size, size_t nmemb, FILE* stream);**
  - ptr: puntatore al dato da «copiare» nel blocco dati
  - size: dimensione del blocco dati da leggere
  - nmemb: numero dei blocchi di memoria da leggere
  - stream: puntatore al file da cui leggere i dati
  - lavora in maniera duale a fwrite()
  - fread(&voto, sizeof(voto), 1, file),

•leggerà 30. E per leggere il quarto?



Accesso diretto

- **int fseek(FILE* stream, int offset, int reference);**
  - posizione il puntatore dopo *offset* byte rispetto a *whence*
  - stream: puntatore al file su cui scrivere i dati
  - offset: spostamento all'interno del file (0, prima locazione)
  - reference:posizione iniziale del puntatore con valori:
    - SEEK_SET: vai dall'inizio del file
    - SEEK_END: vai alla fine del file
    - SEEK_CUR:vai dalla posizione corrente
  - fseek(FILE* stream, 12, SEEK_SET), leggerà il quarto dato
    - 12= 3* sizeof()



Esempio

```c
#include <stdio.h>

int main() {
    FILE* file;

    if((file = fopen("test.dat","rb+")) == NULL) {
        puts("Errore nell'apertura"); // errore in apertura
    }
    else {
      for(int i=1; i<=5; i++) {
        int value = i*10; //assegno valore

        fwrite(&value, sizeof(i), 1, file); //scrivo
        printf("Write:%d\n",value);
        }

      puts(""); //formattazione
```

Esempio

```
    for(int i=0; i<5; i++) {
      int value = 0;

      fseek(file, i*sizeof(i), SEEK_SET); //posiziono puntatore
      fread(&value, sizeof(value), 1, file); //leggo valore

      printf("Read:%d\n",value); //stampo valore letto
    }
  }
}
```

Esercizio

- Implementare un programma che acquisisca da tastiera nome(o matricola) e voto d'esame per cinque individui.
- I valori acquisiti devono essere memorizzati su file (una coppia di valori per ogni riga).
- Il programma deve poi leggere il file dall'inizio e stampare a schermo i nomi degli studenti che hanno superato l'esame.