RIEPILOGO DI ALCUNE FUNZIONI DI LIBRERIA

#include <fstream.h>
istream& getline(char* buffer, streamsize num, char delim = '\n');
method is used with input streams, and reads characters into buffer until either:

num - 1 characters have been read,
a *newline* is encountered,
an *EOF* is encountered,
or, optionally, until the character *delim* is read.

The *delim* character is not put into buffer but is substituted by a '\0'.
The *delim* character is removed from the stream.

The returned value is an *istream* reference or *NULL* when the end-of-file is encountered.

#include <fstream.h>
istream& get(char &c);

la funzione menbro get con un argomento di tipo carattere effettua l'input del carattere successivo dallo stream di input considerato (anche se si tratta di un carattere di spaziatura) e lo memorizza nell'argomento.
Restituisce NULL (0) quando viene raggiunta la fine del file, altrimenti restituisce un riferimento all'oggetto istream per cui e' stata invocata.

#include <stdlib.h>
***float atof(char* buffer)*;**

The string argument to atof has the following form:
[whitespace] [sign] [digits] [.digits] [{d | D | e | E }[sign]digits]
A whitespace consists of space and/or tab characters, which are ignored; sign is either plus (+) or minus (–); and digits are one or more decimal digits.
If no digits appear before the decimal point, at least one must appear after the decimal point.
The decimal digits may be followed by an exponent, which consists of an introductory letter (d, D, e, or E) and an optionally signed decimal integer.

#include <stdlib.h>
int atoi(chat* buffer) and ***long int atol(char* buffer):***

atoi and atol do not recognize decimal points or exponents.
The string argument for these functions has the form:

[whitespace] [sign]digits

where whitespace, sign, and digits are exactly as described above for atof

#include <stdio.h>
int sprintf (char *buffer, const char *format [, argument] ...);

The ***sprintf*** (...) function formats and stores a series of characters and values in *buffer*. Each *argument* (if any) is converted and output according to the corresponding format specification in *format*. The format consists of ordinary characters and has the same form and function as the *format* argument for printf. A null character is appended after the last character written.

ESERCIZIO 1

Scrivere un programma in linguaggio C che memorizzi in un file “echo.txt” tutto quello che viene digitato da console sino a che non viene premuto il carattere ‘@’. Successivamente si chiuda il file e quindi lo si riapra per stampare a terminale l’intero contenuto del file.
(Nel file echo.txt dovrà essere accodato tutto quello che l’utente immette da console ad ogni invocazione del programma.)

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#define NOME_FILE "C:/echo.txt"

int main() {
    ifstream file_in;
    ofstream file_out;
    char ch;

    file_out.open(NOME_FILE,ios::app);
    if (file_out.fail()) {
      cout << "Problemi in apertura del file: " << NOME_FILE;
      return 1;
    }

    cin.get(ch); // stessa funzionalità di cin >> ch;
                 // ma la .get() permette di leggere anche
                 // i caratteri di spaziatura.
    while (ch !='@') {
         file_out.put(ch);
          cin.get(ch);
    }
    file_out.close();

    file_in.open(NOME_FILE);

    if (file_in.fail()) {
      cout << "Problemi in apertura del file: " << NOME_FILE;
      return 1;
    }
    cout << "\n\necho.....\n\n";
    while (!file_in.eof()) {
        file_in.get(ch);
        cout << ch;
    }
    // equivalentemente si sarebbe potuto strutturare il ciclo come:
    // while (file_in.get(ch))
    //     cout << ch;

    file_in.close();

    system("PAUSE");
    return 0;
}
```

ESERCIZIO 2:

Scrivere un programma in linguaggio C che risolva il seguente problema.

Leggere dal un file di testo "magazzino.txt" una sequenza di informazioni relative a pezzi meccanici.

Ogni pezzo è descritto dalle seguenti informazioni:

- Codice (alfanumerico con 5 caratteri)
- Quantità (int >= 0 composto al più da 4 cifre decimali senza segno)
- Prezzo (double >=0 -- con al più 5 cifre intere e 2 decimali)

I dati relativi ad un singolo pezzo sono registrati su un unico rigo del File.

I dati relativi ai vari pezzi sono separati nel File originale da un carattere '\n'.

Il file precedentemente descritto può contenere delle ripetizioni dello stesso pezzo.
Si chiede di progettare un sottoprogramma che crei un secondo File "nuovo_magazzino.txt" che contenga l'elenco dei pezzi senza ripetizioni in ordine alfabetico di codice e che per ogni pezzo indichi la quantità complessiva; rispettando le convenzioni di formattazione del File originale.
Esempio:

File: ”magazzino.txt”

```
AX354 1000 11340.67
BX355 1000 11340.67
AX354 1000 11340.67
BX355 1000 11340.67
ZX254 1000 11340.67
ZX254 1000 11340.67
AX354 1000 11340.67
BX355 1000 11340.67
```

File:”nuovo_magazzino.txt”

```
AX354 3000 11340.67
BX355 3000 11340.67
ZX254 2000 11340.67
```

```
#include <iostream.h>
#include <string.h>   // per poter usare strlen(...), strcmp(...),
                      //                 strcpy(...), strncpy(...)
#include <fstream.h>
#include <stdlib.h>   // per poter usare atoi(...),atof(...),system(...)
#include <stdio.h>    // per poter usare la funzione sprintf(...);

#define SEPARATORE ' '// Sul file uno <spazio> è usato per separare i
                      // campi di un record all'interno dello stesso rigo

#define NCHARSEP 1    // numero di caratteri SEPARATORE usati per
                      // separate i vari campi del record su un rigo del
                      // file.

#define NOME_FILE_ORIGINALE ".\\magazzino.txt"

#define NOME_FILE_NUOVO     ".\\nuovo_magazzino.txt"

struct piece {
       char   code[6];    // 5 char + '\0'
       int    quantity;   // rappresentabile con 4 char
       double price;      // rappresentabile con 5+1+2 char
};

struct node {
      piece data;
      node* next;
};

void  printList(node* h);
node* createNode(char* cod, int q, double price);
void  insertSorted(node* & h, node* x);
void  eraseList(node* &h);
void  eliminaDuplicatidaListaOrdinata(node* oldlist, node* & newlist);
```

```
// Su un rigo del File avrò:
// <5 char di codice><spazio><4 char quantita><spazio>
// <8 char di prezzo><'\n'>

// Il rigo e' composto da 20 char compreso '\n'.
// Quindi i parametri della .getline saranno .getline(buffer,21,'\n');
// che avra' l'effetto di leggere il 20-esimo carattere e riconoscerlo
// come terminatore '\n';
// dunque nell'array buffer (dichiarato come buffer[20])
// ci saranno i primi 19 caratteri del rigo del file seguiti da '\0'.

// Se riempissi il file coerentemente con questa regola, un file
// contenente N pezzi avrebbe N+1 righe;
// l'ultima sarebbe vuota per effetto dell'a-capo relativo al record
// dell'ultimo pezzo.
// Utilizzando la funzione getline(), quando andro' a leggere la riga N+1
// questa funzione leggerà una stringa vuota e soltanto dopo tale evento
// la fine del file (EOF) sarà stata raggiunta.
// Occorre dunque prestare attenzione alla modalità di lettura delle
// informazioni dal file e
// riconoscere il caso in cui sia stato letto un rigo vuoto (in
// particolare l'ultimo sarà sempre tale).
```

```
int main () {
   ifstream fin;
   ofstream fout;
   fin.open(NOME_FILE_ORIGINALE);
   if (fin.fail()) {
      cout << "Problemi in apertura del file magazzino.txt";
      system("PAUSE");
      return 1;
   }
   char buffer[20]; // 20 char, l'ultimo sarà un '\n' sostituito con '\0'
   char bcode[6];   // 5     char più il '\0'
   char bq[5];      // 4     char più il '\0'
   char bprice[9];  // 5+1+2 char più il '\0'

   int quantity;
   double price;
   node* firstlist = NULL;
   node* aux;
   int read;
   // Copio ogni riga del file in una lista dinamica ordinata per codice
   fin.getline(buffer,21,'\n');
   read = strlen(buffer);
     while (!fin.fail() && read > 0) {
       strncpy(bcode,buffer,5);bcode[5] = '\0';
       strncpy(bq,buffer+5+NCHARSEP,4); bq[4] = '\0';
       strncpy(bprice,(buffer+5+NCHARSEP)+4+NCHARSEP,8);bprice[8] = '\0';
       quantity = atoi(bq);
       price = atof(bprice);
       aux = createNode(bcode, quantity, price);
       insertSorted(firstlist,aux);

       fin.getline(buffer,21,'\n');
       read = strlen(buffer);
   } // end while
   fin.close();
   // printList(firstlist);

   node* secondlist = NULL;
   char auxbuff[20];
   // Elimino i duplicati sfruttando il fatto che la lista e' ordinata
   eliminaDuplicatidaListaOrdinata(firstlist, secondlist);


   // cout << "\nnewlist:....\n"; printList(secondlist);
```

```
// Maniera equivalente di impostare
// il ciclo di lettura del file:
fin.getline(buffer,21,'\n');
while (!fin.eof()) {
    if (strlen(buffer) > 0){
       ……
    }
    fin.getline(buffer,21,'\n');
}
```

```
   // Scrivo il contenuto della lista senza duplicati nel nuovo file che
   // sarà così già ordinato,
   // rispettando le convenzioni di formattazione del file originale.

   fout.open(NOME_FILE_NUOVO,ios::out);
   if (fout.fail()) {
      cout << "Problemi in apertura del file nuovo_magazzino.txt";
      system("PAUSE");
      return 1;
   }

   aux = secondlist;
   while (aux!=NULL) {

         // Formatto una riga del file
         fout << aux->data.code << SEPARATORE;

         sprintf(auxbuff,"%4d",aux->data.quantity);
         fout << auxbuff << SEPARATORE;

         sprintf(auxbuff,"%5.2lf",aux->data.price);
         fout << auxbuff << '\n';

         // passo al nodo successivo
         aux = aux->next;
   }
   fout.close();

   system("PAUSE");
   eraseList(firstlist);
   eraseList(secondlist);
   return 0;
} // end main
```

```
void eliminaDuplicatidaListaOrdinata(node* oldlist, node* & newlist) {

     char prec[6];
      node *nl;

     if (oldlist == NULL) { newlist = NULL; return; }

     newlist = createNode((oldlist->data).code,
                          (oldlist->data).quantity,
                          (oldlist->data).price);
     nl = newlist;
      strcpy(prec,(oldlist->data).code);
      oldlist = oldlist->next;
      while (oldlist != NULL) {

            if (strcmp((oldlist->data).code, prec) == 0) {
                nl->data.quantity += oldlist->data.quantity;
            }
            else {
                nl->next = createNode((oldlist->data).code,
                                      (oldlist->data).quantity,
                                      (oldlist->data).price);
                nl = nl->next;
                strcpy(prec, (oldlist->data).code);
            }
            oldlist = oldlist->next;
     }
} // end eliminaDuplicatidaListaOrdinata



node *createNode(char *code, int q, double price) {

     node *p = new node;      // Creazione nuovo nodo
     strcpy((p->data).code,code);
     (p->data).quantity = q;
     (p->data).price = price;
      p->next = NULL;
       return p;
}
```

```
void insertSorted(node* & h, node *x) {
// Si suppone la lista in ordine alfabetico di codice
// si inserisce un nuovo nodo rispettando tale ordinamento
// h è passato per reference, per scandire
// la lista devo quindi usare un puntatore ausiliario.

    node *curr;
    node *pred = NULL;

    if (h == NULL) { h = x;
                     return;
    }
    curr = h ;
    while ((curr!=NULL)&&(strcmp((curr->data).code,(x->data).code)<= 0)){
         pred = curr;
         curr = curr->next ;
    }

    if (pred != NULL) {
       pred->next = x;
      x->next = curr;
    } else {            // inserimento in testa
        x->next = h;
        h = x;
    }
} // end insertOrd

void printList(node* h) {
  cout << "\n";
  if (h != NULL) {
    while (h!=NULL) {
        cout << "\n";
        cout << (h->data).code << " | ";
        cout << (h->data).quantity << " | ";
        cout << (h->data).price << " ; ";
           cout.flush();
        h = h->next;
    }
    cout << "\n";
  }
  else cout << "Empty!";
} // end printList

void eraseList(node* &h) {
    node * aux;
    while (h!= NULL) {
      aux = h;
      h = h->next;
      delete aux;
    }
} // end eraseList
```

ESERCIZIO 3 (Bozza di Soluzione)

Si realizzi un programma che una casa discografica può utilizzare per gestire gli album musicali degli artisti.

- Il programma deve gestire i dati degli artisti, quindi consente di inserire i dati degli album, dove ogni album è un insieme di canzoni.
- Per ogni album si vuole avere l'anno di uscita e il prezzo.
- Il programma deve consentire di consultare l'elenco degli album in base a diversi criteri di ricerca (per artista, per anno, per titolo, per canzone contenuta).

Si realizzi il programma in modo che possa gestire un numero arbitrariamente ampio di artisti e di album.

Interpretazione della traccia -- Ipotesi di lavoro semplificate:

1. Gli artisti sono associati solo agli Album.
   Ciascun Album può avere un solo Artista autore dello stesso.
2. Le canzoni sono associate solo agli Album.
   In pratica un Artista può essere presente in più Album ma le canzoni sono specifiche di ciascun Album.

Gestiamo il problema con una coppia di archivi separati: quello degli Artisti e quello degli Album, perché sono archivi che possono essere interessati separatamente (ma anche congiuntamente) da operazioni di aggiornamento o più in generale di manipolazione delle informazioni in esso contenute.

File Artisti:

Contiene i record relativi ad ogni artista le cui canzoni sono commercializzate dalla casa discografica.

Ogni artista è identificato da un suo codice univoco.
Il singolo record contiene:

- codice_artista
- Nome artista
- Compenso Artista

File Album:
Contiene i record relativi ad ogni album prodotto dalla casa discografica.
Il singolo record contiene:

- Codice Album
- Nome dell'album
- Codice Artista
- Anno di uscita
- Prezzo
- Numero di canzoni contenute
- Titoli delle canzoni contenute nell'album.

Il programma prevederà tre moduli:

Artisti.h, Artisti.cpp
per implementare tutti i sottoprogrammi utili alla gestione degli Artisti e le routine per la creazione di una lista dinamica contenente in ogni nodo i successivi record nel file.

Album.h, Album.cpp
per implementare tutti i sottoprogrammi utili alla gestione degli Artisti e le routine per la creazione di una lista dinamica contenente in ogni nodo i successivi record nel file.

Const_and_Utility.h, Const_and_Utility.cpp
Per avere un unico punto all'interno del progetto software di definizione delle costanti e delle funzioni di utilità.
Per implementare i menù interattivi di interfaccia con l'utente.

CasaDiscografica.cpp
Programma principale (con il main…) per implementare i sottoprogrammi che usano i moduli precedenti per rispondere alle specifiche del problema.

Vedi i file allegati.