



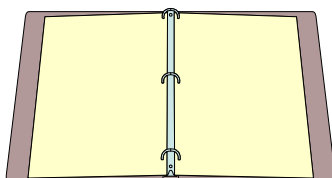
*U*niversità degli *S*tudi di *U*rbino

***Facolta' Di Scienze Matematiche Fisiche e Naturali  
C. d. L. in Informatica Applicata***

Corso di Ingegneria del Software

RELAZIONE PROGETTO DI LABORATORIO

*"Gestore Agenda e Rubrica"*



Studente  
Elvis Ciotti

Prof.  
Tommaso Pagnini

## **SPECIFICA DEL PROBLEMA**

Il progetto consiste in un'agenda e rubrica che gestisce i vari eventi e contatti. Eventi e contatti sono memorizzati su file, cioè letti all'apertura e salvati alla chiusura.

Gli eventi possono essere di tipo:

- Appuntamento (con oggetto, luogo, nome persona da incontrare, note varie)
- Scadenza (con oggetto, priorità da 1 a 5, note varie)
- Evento generico (solo un testo)

Tutti gli eventi sono relativi ad una certa data.

La rubrica contiene contatti, ognuno dei quali ha: nome, cognome, mail, indirizzo di casa, telefono di casa, indirizzo dell'ufficio, telefono dell'ufficio, telefono cellulare, note. Alcuni campi possono non essere specificati.

I contatti devono essere visualizzati in ordine alfabetico (ordinamento per cognome e nome) e non devono essere presenti contatti ripetuti.

## SPECIFICA DEI REQUISITI

Eventi dell'agenda e contatti della rubrica devono essere letti da file. I files possono essere modificati esternamente e deve essere prevista la gestione degli errori.

Ogni linea dell'agenda contiene le informazioni relative ad un Evento.

Ogni linea della rubrica contiene le informazioni relative ad un Contatto.

Ogni linea deve contenere campi separati da punto e virgola ';', in modo che sia importabile da altri programmi nel formato csv (comma separated values).

Se una linea dei files contiene un errore deve essere tralasciata e mostrato un messaggio di avvertimento con il numero di linea e il tipo di errore (data non valida, record non valido con numero di campi inferiore a quelli richiesti).

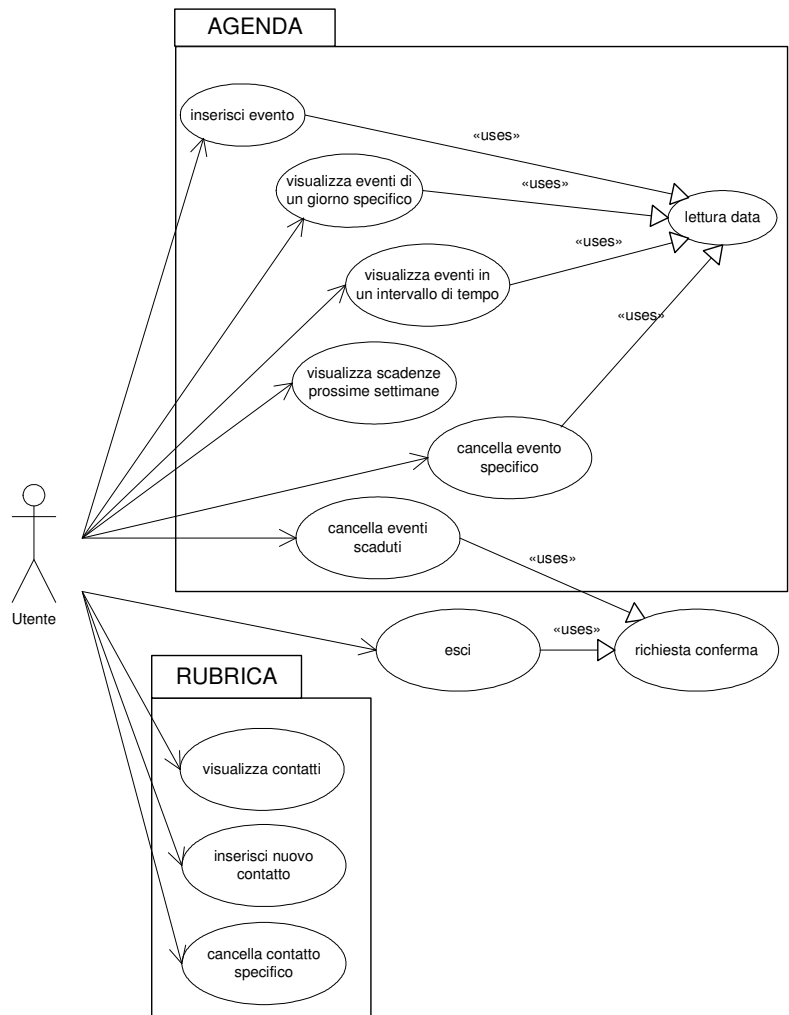
Se i files di agenda o rubrica non sono presenti deve essere mostrato un messaggio di avvertimento, e l'esecuzione deve continuare senza nessun dato caricato da essi. I files verranno ricreati alla chiusura con i nuovi dati immessi.

Il programma deve essere composto da un menu principale in cui compaiono l'elenco delle operazioni d'inserimento/cancellazione/visualizzazione su agenda e rubrica.

Oltre a uscire, deve essere permesso anche di uscire senza salvare le modifiche effettuate.

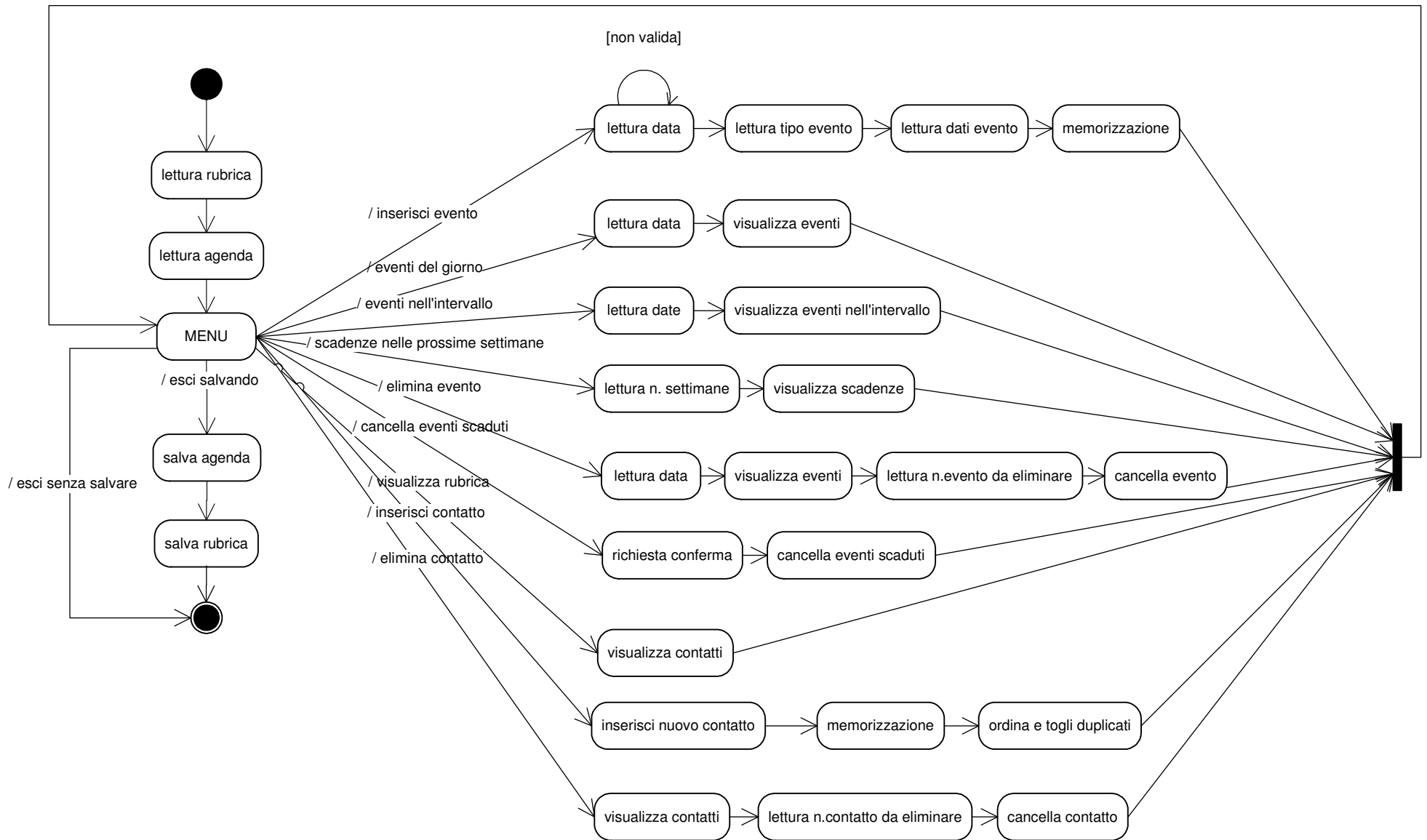
Operazioni effettuabili sugli eventi (vedi diagramma Use Case a destra):

- Inserimento evento: dopo aver inserito la data e scelto il tipo d'evento, s'inseriscono i vari campi richiesti a seconda che sia Appuntamento, Scadenza o Evento generico.
- Visualizzazione eventi d'oggi, domani o ad una certa data immessa: verranno stampati di tutti gli eventi relativi alla data scelta (ogni tipo di evento ha una sua visualizzazione che mostra di che tipo è e stampa in modo leggibile tutti i suoi campi).
- Visualizzazione eventi in un intervallo temporale specificato (intervallo di date o prossimi giorni): verranno stampati gli eventi per ogni data compresa nell'intervallo (cioè le date in ordine temporale con gli eventi relativi a ciascuna data).
- Visualizzazione delle scadenze delle prossime settimane: dopo aver inserito il n. di settimane vengono visualizzate in ordine temporale le scadenze, ognuna con data e numero di giorni rimasti a partire dalla data corrente.
- Cancellazione evento: dopo aver immesso la data vengono visualizzati gli eventi in tal data numerati in modo crescente. Poi viene richiesto il numero dell'evento da eliminare che verrà cancellato (cioè cancellato dalla memoria e quindi non salvato su file alla chiusura).
- Cancellazione eventi scaduti: Dopo un conferma da parte dell'utente, gli eventi che hanno una data inferiore a quella corrente vengono eliminati. Al termine viene visualizzato il numero di eventi cancellati.



Operazioni effettuabili sui contatti:

- Visualizzazione: vengono stampati i contatti in ordine alfabetico
- Inserimento: vengono richiesti tutti i campi del nuovo contatto da riempire
- Cancellazione: vengono visualizzati i contatti con un numero crescente. Poi viene richiesto il numero del contatto da eliminare che verrà cancellato (cioè cancellato dalla memoria e quindi non salvato su file alla chiusura).



## DIAGRAMMA UML DEGLI STATI

Digramma degli stati in cui si trova il programma. Dopo aver letto rubrica e agenda, si effettua l'operazione scelta, ognuna delle quali è composta dalla sequenza di altri stati al termine dei quali si ritorna al menu. L'uscita normale comprende salvataggio di agenda e rubrica. In caso di uscita senza modifiche si ha una transazione diretta all'uscita.

Per motivi di spazio non sono inserite le condizioni di guardia per la lettura date (la data deve essere valida) e per gli altri tipi di letture.

Il menu principale deve anche mostrare data e orario corrente del sistema (compreso giorno della settimana e mese in lettere), il numero di appuntamenti di oggi e domani.

## DATE

Le date inserite (giorno, mese, anno separati da '-') devono essere controllate e formattate allo stesso modo.

Il numero del mese deve essere compreso fra 1 e 12, quello del giorno in base al mese, l'anno compreso fra 1950 e 2400. L'anno può essere inserito nei formati:

- numero completo. Es :2005
- ultime due cifre . Es: 05
- ultima cifra. Es: 5

Gli eventuali zeri prima del campo (giorno, mese, anno ) vanno tralasciati. (01-02-2005 == 1-2-2005)

Esempi:

1-1-1 = 1-1-2001 =01-01-2001 =001-0001-0000001 = 1 gennaio 2001

30-2-05 NON VALIDA (febbraio ha 28 giorni)

32-13-2405 NON VALIDA (giorno, mese, anno fuori dall'intervallo valido).

0-0-1940 NON VALIDA (giorno, mese, anno fuori dall'intervallo valido).

Deve essere possibile anche:

- inserire la data del giorno corrente semplicemente digitando 'oggi'
- inserire la data del giorno seguente semplicemente digitando 'domani'

## ANALISI E PROGETTAZIONE

Il software è stato sviluppato in linguaggio C++, a norma delle regole riguardanti la OOP.

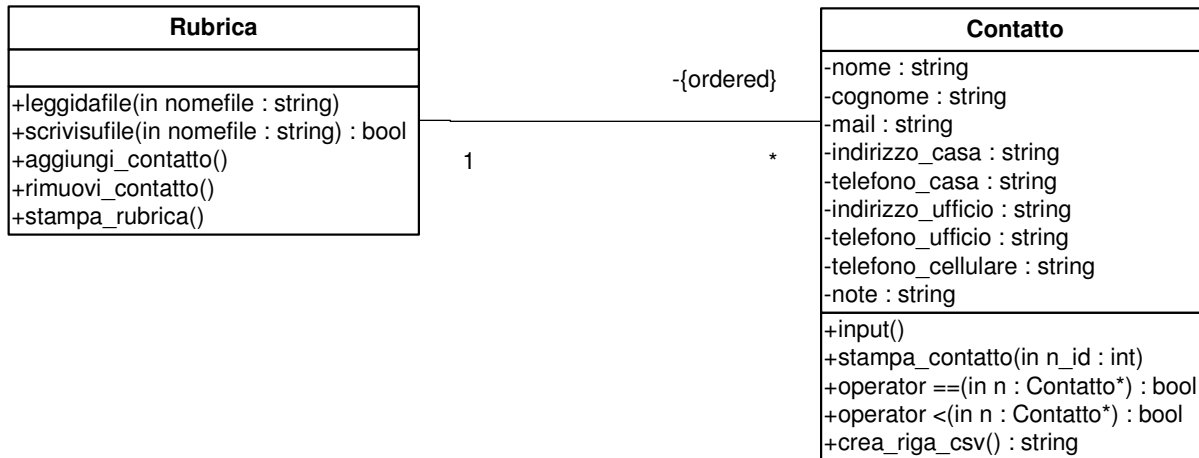
La progettazione è avvenuta seguendo i principi di modularità, indipendenza funzionale, coesione, information hiding, tranne qualche eccezione (es: funzioni friend nell'implementazione) per migliorare l'efficienza.

L'architettura è di tipo Call & Return, composta di varie classi rappresentate nei diagrammi che seguono.

Il programma di compone di due gruppi di classi principali:

- Classi relative alla rubrica di contatti (qui sotto e descrizione di seguito )
- Classi relative all'agenda di eventi (pagina successiva e descrizione seguente)

La descrizione delle scelte progettuali effettuate è inserita nelle descrizioni seguenti i diagrammi delle classi. Altri dettagli e descrizione di funzioni ausiliarie sono presenti nei commenti che precedono l'implementazione delle funzioni (sezione successiva).



### **Contatto:**

i diversi campi sono tutti di tipo stringa, compresi numeri di telefono ( possono comprendere infatti spazi e separatori ).

La funzione di `input` richiede in ordine tutti i campi da inserire. Il carattere punto e virgola ';' non è ammesso poiché è utilizzato come separatore di campi nei files. E'ammesso non specificare un valore per il campo immettendo 'nd'.

La funzione stampa\_contatto accetta un intero che stampa come riferimento per il contatto (necessario per indicarlo con precisione nel caso di cancellazione dalla rubrica).

Gli operatori `==` e `<` permettono un confronto fra contatti in modo da eliminare duplicati (stesso valore per tutti gli attributi) e ordinare (prima per cognome poi per nome) i contatti nella rubrica.

L'ultima funzione crea\_riga\_csv restituisce la stringa con tutti i campi del contatto separati dal delimitatore punto e virgola, necessaria per la memorizzazione su file.

### **Rubrica:**

Unico attributo è un insieme di Contatti. Nell'implementazione in C++ si preferisce una `<list>`, visto che l'operazione principale sui contatti è la scansione completa e inserimenti, ed è necessario eliminare i contatti ripetuti e ordinarli (rispettivamente `unique()` e `sort()` della std library C++).

leggidafile: Per ogni riga del file dei contatti (il cui nome è passato come parametro), si crea un nuovo Contatto (usando il costruttore di Contatto con tutti gli attributi, che in linea sono separati da `;') e si aggiunge alla lista di contatti. Se ci sono meno campi di quelli previsti viene mostrato un messaggio di errore e la riga viene tralasciata. In caso di file assente, nessun contatto viene caricato nella rubrica in memoria. Al termine vengono tolti i contatti duplicati e i rimanenti vengono ordinati.

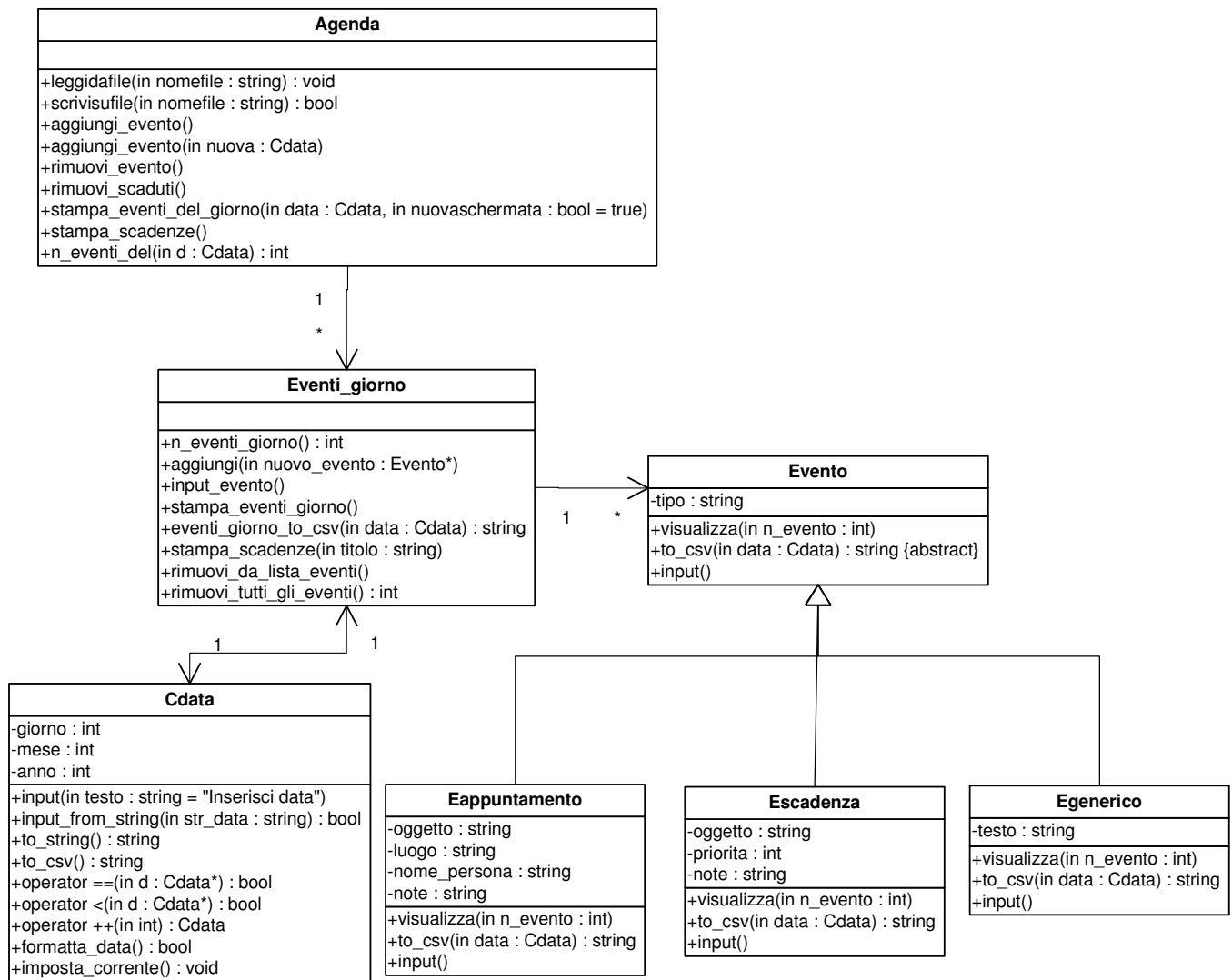
Con la scrivisufile tutti i contatti presenti nella lista in memoria vengono inseriti in righe successive del file indicato. Se il file non è scrivibile restituisce falso e permette di notificarlo all'esterno in modo che possa essere ritentata la scrittura.

aggiungi\_contatto istanzia un nuovo contatto e ci chiama la relativa input, lo aggiunge in coda alla lista di contatti. La rubrica viene nuovamente filtrata dai doppi e ordinata.

stampa\_rubrica : Stampa prima il numero di contatti presenti nella lista, poi la scorre e chiama la funzione stampa\_contatto su ogni elemento, passando ogni volta un numero intero a partire da 1. In questo modo ogni contatto della lista viene visualizzato con un numero univoco.

rimuovi\_contatto() chiama prima la funzione precedente in modo da scegliere il numero del contatto da eliminare. Dopo che l'utente ha inserito il numero N del contatto da eliminare, si scorre la lista fino all'N-esimo contatto e si elimina dalla lista.

---



Come si può vedere dal diagramma, l'agenda ha una lista di eventi per ogni giorno. Una lista di eventi è composta da Eventi (appuntamento, scadenze o eventi generici).

### **Cdata:**

Attributi: giorno, mese, anno.

formatta\_data converte se necessario l'anno nel formato a 4 cifre e restituisce vero/falso a seconda che la data sia valida o no (vedi specifica requisiti).

Giorno, mese e anno corrente sono impostati con la imposta\_corrente, in modo coerente con le altre date inserite dall'utente (cioè mesi da 1 a 12, giorni da 1 a 31...).

operator < permette di comparare due date, tenendo conto della differenza dei giorni contenuti nei vari mesi.

operator ++ aumenta la data di un giorno, considerando i casi dell'ultimo giorno del mese e ultimo giorno dell'anno. 30-4-2005=>1-5-2005 . 31-12-2005 => 1-1-2006

input\_from\_string imposta la data dalla stringa passata nel formato g-m-a. Se viene passata la stringa 'oggi' imposta la data corrente. Se viene passata la stringa 'domani' imposta la data corrente e la incrementa con operator++.

Se viene passata la stringa 'oggi' imposta la data corrente.

La input richiede la stringa contenente la data finché la input\_from\_string non restituisce vero. Restituisce (e quindi chiama) la formatta\_data.

to\_string: restituisce la data in formato giorno,mese,anno separati da spazi e con il mese in lettere. Es: 1 Luglio 2005"".

to\_csv restituisce giorno, mese,anno separati da '\-\'.

Per la classe Cdata è definita anche una funzione di overload dell'operatore << per le date che restituisce lo stream nel formato giorno, mese, anno separati da spazi e con il mese in lettere.



### **Evento:**

la stringa `'tipo'` memorizza in formato stampabile il tipo di Evento "Appuntamento", "Scadenza" o "Evento generico", che le relative istanze delle classi derivate impostano.

Le tre funzioni che costituiscono la classe sono tutte reimplementate nelle classi derivate.

Visualizza: stampa a video l'evento

to\_csv: restituisce la stringa csv per memorizzare l'evento su file. (virtuale pura: ogni classe derivata la implementa a modo suo).

Input: richiede i campi che costituiscono l'evento specifico

### **Eventi\_giorno:**

Contiene gli eventi di un particolare giorno.

Analogamente al caso dei contatti, si preferisce usare una `list<>` della std library di C++ poichè gli eventi solo scansiti dal primo all'ultimo e non è necessario un accesso indicizzato. Come risulta dal diagramma delle classe UML, una classe `'Eventi_giorno'` ha una corrispondente classe `'Cdata'`, ma nella implementazione in c++, questa associazione viene fatta nella classe `'Agenda'` tramite una `<map>` delle date sulle liste di eventi.

`Eventi_giorno` quindi è semplicemente una lista di Eventi relativi ad una certa data non nota all'interno della classe.

stampa\_eventi\_giorno scorre tutti gli eventi nella lista e chiama per ognuna la relativa funzione di visualizzazione (passando come numero identificativo un numero crescente, necessario per la identificazione univoca in caso di eliminazione).

La stampa\_scadenze si comporta allo stesso modo, considerando però solo gli oggetti `'Escadenza'`.

n\_eventi\_giorno restituisce il numero di eventi della lista eventi.

L'inserimento di eventi da parte dell'utente avviene con la input\_evento che chiede prima il tipo di evento inserire, poi viene istanziata una classe del tipo scelto e chiama la funzione di input su di essa.

Poi questa nuova classe istanziata viene aggiunta alla lista di eventi tramite la aggiungi.

La rimozione (rimuovi\_da\_lista\_eventi) avviene in modo simile ai contatti della rubrica, cioè dopo la stampa degli eventi del giorno, viene richiesto il numero dell' evento da eliminare che viene tolto dalla lista.

rimuovi\_tutti\_gli\_eventi rimuove tutti gli elementi dalla lista e restituisce il numero di eventi cancellati.

Utilizzata per rimuovere gli eventi scaduti.

### **Agenda:**

Unico attributo dell'agenda è, come già accennato, una mappa di date su liste di eventi, cioè una `<map>` di `'Cdata'` su `'Eventi_giorno'`.

Nell'implementazione questa mappa è chiamata `'map_agenda_eventi'`

In questo modo ad una data corrisponde una lista di eventi.

La lista di eventi del giorno X è accessibile con `map_agenda_eventi[X]`

Per caricare gli eventi da file, la leggidafile si comporta in modo simile alla omonima funzione della classe Rubrica. Ogni riga contiene un evento nella forma

`<tipo di evento>;<data evento>;<attr1>;...;<atccampo n>`

Se la data letta è valida, in base al tipo di evento (1=appuntamento, 2=scadenza, 3=evento generico) viene istanziata una classe e riempita con i campi da 1 a n. ogni tipo di evento ne ha un numero diverso e se ce ne fossero di meno viene mostrato un messaggio d'errore.

A questo punto l'evento viene aggiunto (nella `'map_agenda_eventi'`) alla lista di eventi corrispondente alla data letta.

In pseudocodice:

per ogni riga valida

```
{
    if ( valida(<data evento letta> )
    {
        leggi tipo di evento //es: 2
        if (tipo di evento==1) { ... }
        if (tipo di evento==3) {
            Escadenza <nuova scadenza>(<campo1>,...<campo n>) //istanzio nuova scadenza
                                e ne imposto gli attributi
            map_agenda_eventi[<data evento letta>].aggiungi(<nuova_scadenza>)
        }
        if (tipo di evento==3) { ... }
    }
    else errore ("data non valida")
}
```

nota:gli eventi chiamati su map\_agenda\_eventi[<data>] sono della classe 'Eventi\_giorno'

La scrittura su file (scrivisufile) è effettuata scorrendo tutta la 'map\_agenda\_eventi', chiamando su ogni elemento (cioè ogni classe 'Eventi\_giorno' contenente lista di eventi del giorno) la eventi\_giorno\_to\_csv e accodando al file la stringa risultante.

L'aggiunta di un evento (aggiungi\_evento) non fa altro che chiamare la sulla lista di eventi relativi alla data immessa (o passata direttamente nell'altra versione della funzione).

map\_agenda\_eventi[<data immessa o passata>].input\_evento();

Caso analogo per la rimuovi\_evento e stampa\_eventi\_del\_giorno che dopo aver immesso la data, chiamano rispettivamente rimuovi\_da\_lista\_eventi() e stampa\_eventi\_giorno() sugli eventi relativi alla data immessa.

Per rimuovere gli eventi scaduti (rimuovi\_scaduti) si rimuovono tutti gli eventi relativi a date inferiori a quella corrente, chiamando la rimuovi\_tutti\_gli\_eventi(). Alla fine restituisce il numero totale di eventi cancellati.

La stampa della scadenze (stampa\_scadenze) è implementata chiamando la stampa\_scadenze() sulle date a partire dalla corrente fino a quanto richiesto dall'utente (in settimane).

n\_eventi\_del: presa in input una data, restituisce la dimensione della lista degli eventi (cioè il numero di eventi nella data).

---

### **Corpo principale del software**

A lancio del programma, viene istanziata rubrica e agenda su cui vengono invocate le relative leggidafile().

Le varie operazioni sono effettuate con chiamate a metodi di queste due istanze (classi Agenda e Rubrica), dopo aver acquisito gli input necessari.

Nel caso di visualizzazione di eventi in intervalli temporali, vengono effettuate iterazioni sulle chiamate di funzione della classe Agenda.

L'operazione di uscita con salvataggio prevede l'invocazione dei metodi scrivisufile() su agenda e rubrica, (finché non sono effettuate correttamente) e poi l'uscita dal programma.

In caso di uscita senza salvataggio (con conferma) si esce direttamente senza salvare su file le modifiche.

### **Descrizione delle principali funzioni ausiliarie utilizzate (Funzioni.h e Funzioni.h):**

La funzione int\_input() permette di acquisire un intero e nel caso non fosse compreso nel range specificato, viene nuovamente richiesto.

L'acquisizione di una stringa spazi compresi è implementata nella string\_input().

L'acquisizione di una stringa che non contenga delimitatori csv (',';) e che possa essere vuota (se viene immesso 'nd') è affidata alla input\_e\_controlla().

richiedi\_conferma() stampa a video la richiesta di conferma (s/n) e restituisce vero/falso a seconda del valore immesso.

La funzione errore stampa solo a video la stringa dell'errore passata; una futura estensione potrebbe essere l'accodamento dell'errore in un file di log.

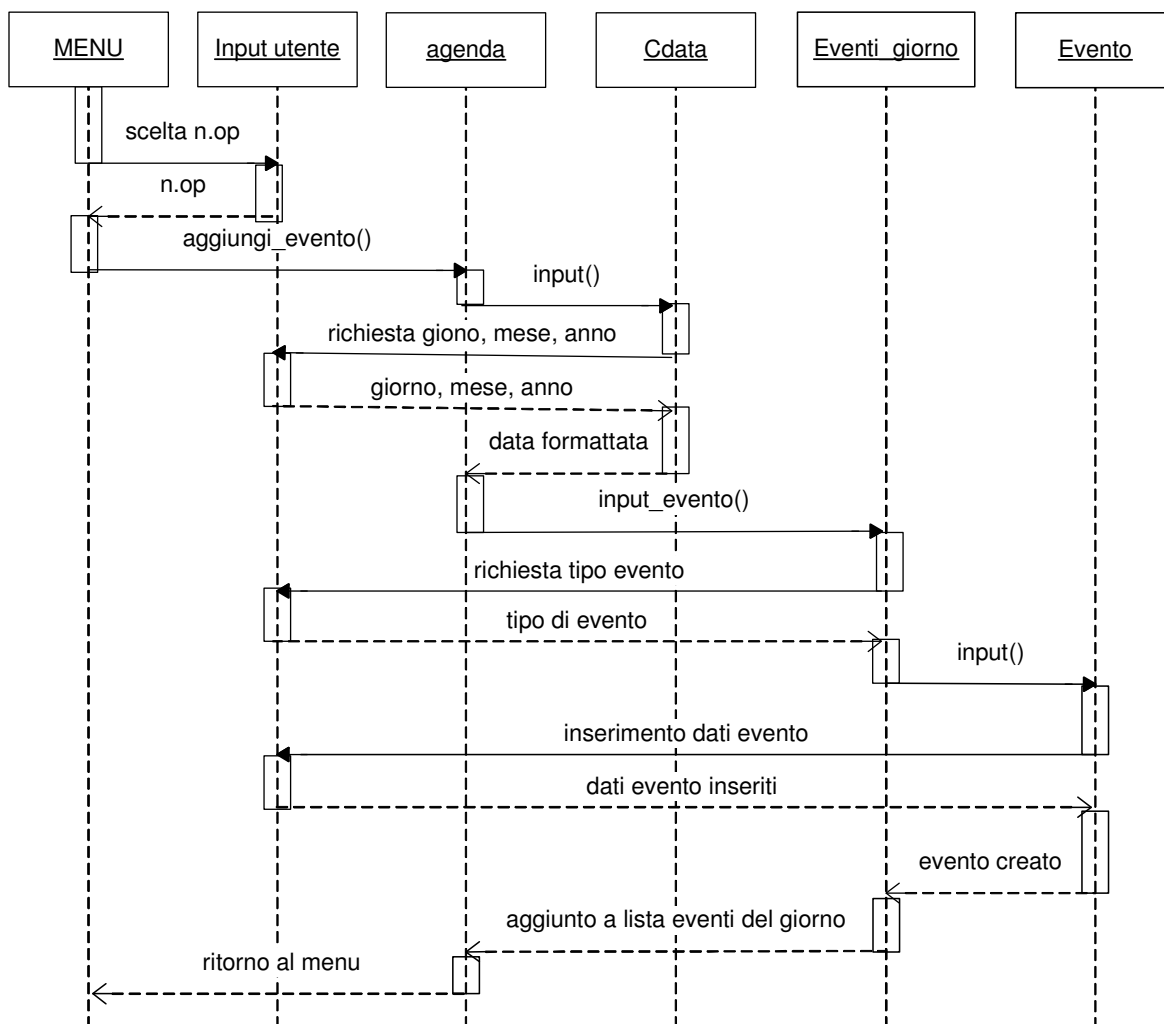
## DIAGRAMMI DI SEQUENZA

Le varie operazioni sono simili e inglobate in parte una nell'altra. Viene riportato di seguito il diagramma di sequenza per l'operazione più rappresentativa, cioè quella relativa all'inserimento di un nuovo evento, che chiama in causa tutti i componenti dell'agenda.

Il controllo è inizialmente del menu principale che aspetta il numero dell'operazione dall'utente.

Quando viene scelto di inserire un nuovo evento, viene chiamata in causa l'agenda, che passa il controllo alla Cdata per l'immissione e il controllo della data.

Il controllo torna poi all'agenda che chiama la `input_evento()` sugli 'Eventi\_giorno' relativi alla data immessa. Quest'ultimo richiede all'utente il tipo di evento da inserire e passa il controllo a `Evento` che richiederà in input i dati in base al tipo di evento scelto. Una volta creato l'evento, verrà aggiunto alla lista di eventi del giorno e il controllo ritorna a `Eventi_giorno`, che a sua volta risponde all'agenda. Essendo l'operazione effettuata, l'agenda ha adesso l'evento inserito in memoria e ritorna il controllo al menu principale.



## IMPLEMENTAZIONE

### Funzioni.h

```
#ifndef FUNZIONI_H
#define FUNZIONI_H

#include "Cdata.h"
#include <cstdlib> // per le system
#include <ctime>
#include <iostream>

const string delimitatore = "+=====\\n";

void pulisci_schermo();
void pausa();
int int_input( string testo, int min, int max = -1 );
string string_input();
void stampa_bordo_alto( string titolo = "AGENDA" );
void stampa_errore( string errore);
void input_e_controlla( string campo, string & valore );
bool richiedi_conferma( string testo );
void stampa_menu( int n_ev_oggi, int n_ev_domani );
string int_to_string( const int n );

#endif
```

### Funzioni.cpp

```
#include "Funzioni.h"

//esegue il comando di pulizia schermo della shell ms-dos
void pulisci_schermo()
{
    system( "cls" );
}

//stampa il delimitatore e esegue il comando di pausa ms-dos
void pausa()
{
    cout << delimitatore;
    system( "pause" );
}

//richiede da shell un intero finchè non è compreso fra min e max
int int_input( string testo, int min, int max /* = -1 */ )
```

```

{
    char buffer[15];
    int n;
    do
    {
        cout << testo;
        cin >> buffer;
        n = atoi( buffer );
    }
    while ( n < min || ( max != -1 && n > max ) ); //se supero limiti, richiedi ancora input
    return n;
}

//acquisisce una stringa compresi gli spazi
string string_input()
{
    char buffer[255];
    do
    {
        gets( buffer );
    }
    while ( buffer[0] == '\0' );
    return string( buffer );
}

//pulisce lo schermo e stampa l'header dell'agenda
void stampa_bordo_alto( string titolo /* = "AGENDA" */ )
{
    const string giorni[] =
    {
        "Domenica", "Lunedì", "Martedì", "Mercoledì", "Giovedì", "Venerdì", "Sabato"
    };

    Cdata oggi;

    oggi.imposta_corrente();
    time_t tmp;
    struct tm * tm_corrente;
    time( & tmp );
    tm_corrente = localtime( & tmp );

    pulisci_schermo();

    cout << delimitatore;

```

```

cout << "|          --- " << titolo << " ---          \n";
cout << "| Oggi: " << giorni[tm_corrente->tm_wday] << " " << oggi << " - Ore " << tm_corrente->tm_hour << ":";
cout.fill( '0' ); cout.width( 2 );
cout << tm_corrente->tm_min << "\n";
cout << "|\n";
}

//gestisce errori lettura/scrittura su file, eventualmente uscendo dal programma
void stampa_errore( string errore )
{
    cout << "[ " << errore << " ]\n";
    pausa();
}

//chiede l'input e lo memorizza nel valore (escludendo punto e virgola)
void input_e_controlla( string campo, string & valore )
{
    string temp;
    bool corretto = false;
    while ( !corretto )
    {
        cout << campo ;
        temp = string_input();
        if ( temp == "nd" || temp == "ND" ) temp = "";
        if ( temp.find( ';' ) == -1 )
            corretto = true; //esce da ciclo
        else
        {
            corretto = false;
            cout << "Carattere ';' non ammesso !\n";
        }
    }
    valore = temp;
}

//restituisce true/false a seconda che l'utente immette s/n alla richiesta
bool richiedi_conferma( string testo )
{
    char sn;
    do
    {
        cout << testo;
        cin >> sn;
    }

```

```

while ( sn != 's' && sn != 'n' ); //esco solo se ho 's' oppure 'n'
return sn == 's';
}

//stampa menu principale del programma, con il numero di eventi di oggi e domani
void stampa_menu( int n_ev_oggi, int n_ev_domani )
{
    stampa_bordo_alto();
    cout << "|    1 - Inserisci Evento\n"; //
    cout << "|    2 - Eventi di OGGI [" << n_ev_oggi << "]\n"; //
    cout << "|    3 - Eventi di DOMANI [" << n_ev_domani << "]\n"; //
    cout << "|    4 - Eventi DEL ... \n"; //
    cout << "|    5 - Eventi dei prossimi ... giorni\n"; //
    cout << "|    6 - Eventi DAL ... AL ... \n"; //
    cout << "|    7 - Scadenze delle prossime ... settimane\n"; //
    cout << "|\n";
    cout << "|          --- RUBRICA ---\n";
    cout << "|    8 - VISUALIZZA rubrica\n"; //
    cout << "|    9 - Inserisci NUOVO contatto\n"; //
    cout << "|   10 - CANCELLA contatto...\n"; //
    cout << "|\n";
    cout << "|   11 - Cancella evento...\n";
    cout << "|   12 - Cancella eventi scaduti\n";
    cout << "|\n";
    cout << "|   13 - Salva ed esci\n"; //
    cout << "|   14 - Esci senza salvare\n"; //
    cout << delimitatore;
}

//converte intero in stringa
string int_to_string( const int n )
{
    string str_ret;
    char * buffer = new char[64]; // 100 ha spazio di c

    str_ret = itoa( n, buffer, 10 /*radix*/);
    delete[] buffer;
    return str_ret;
}

```

## Contatto.h

```

#ifndef CONTATTO_H
#define CONTATTO_H

#include <string>

```

```

using namespace std; // ?

//elemento della Rubrica
class Contatto
{
    /* int id; */
    string nome;
    string cognome;
    string mail;
    string indirizzo_casa;
    string telefono_casa;
    string indirizzo_ufficio;
    string telefono_ufficio;
    string telefono_cellulare;
    string note;

public:
    Contatto() //ctor
    {
    };

    // ctor con tutti i parametri
    Contatto( string n, string c, string m, string ic, string tc, string iu, string tu, string tcel, string no );
    void input();
    void stampa_contatto( int n_id ) const;
    bool operator == ( const Contatto & n ) const;
    bool operator < ( const Contatto & n ) const;
    string crea_riga_csv() const;
};

#endif

```

## Contatto.cpp

```

#include "Contatto.h"
#include "Funzioni.h"
#include <iostream>
#include <string>

using namespace std;

```

```

Contatto::Contatto( string n, string c, string m, string ic, string tc, string iu, string tu, string tcel, string no ) : nome( n ),

```



```

cognome( c ), mail( m ), indirizzo_casa( ic ), telefono_casa( tc ),
    indirizzo_ufficio( iu ), telefono_ufficio( tu ), telefono_cellulare( tcel ), note( no )
    {
    }

//chiede i dati del contatto da shell
void Contatto::input()
{
    cout << delimitatore;
    cout << "|      ---  INSERIMENTO CONTATTO ---      \n";
    cout << "|      \nnd\" per non specificare nulla\n";
    cout << "| \n";
    input_e_controlla( "| Nome: ", nome );
    input_e_controlla( "| Cognome: ", cognome );
    input_e_controlla( "| E-mail: ", mail );
    input_e_controlla( "| Indirizzo (casa): ", indirizzo_casa );
    input_e_controlla( "| Telefono (casa): ", telefono_casa );
    input_e_controlla( "| Indirizzo (ufficio): ", indirizzo_ufficio );
    input_e_controlla( "| Telefono (ufficio): ", telefono_ufficio );
    input_e_controlla( "| Telefono cellulare: ", telefono_cellulare );
    input_e_controlla( "| Note aggiuntive: ", note );
    cout << "|      [ CONTATTO CORRETTAMENTE INSERITO ] \n";
    cout << "| \n";
    cout << delimitatore;
    pausa();
}

//necessario per rimuovere i duplicati nella rubrica con la unique()
bool Contatto::operator == ( const Contatto & n ) const
{
    //sono uguali i contatti (non da eliminare) che hanno gli stessi identici valori
    return ( /* elimina==n.elimina==false && */ nome == n.nome && cognome == n.cognome && mail == n.mail
        && indirizzo_casa == n.indirizzo_casa && telefono_casa == n.telefono_casa && indirizzo_ufficio == n.indirizzo_ufficio
        && telefono_ufficio == n.telefono_ufficio && telefono_cellulare == n.telefono_cellulare && note == n.note );
}

//necessario per sort nella rubrica
bool Contatto::operator < ( const Contatto & n ) const
{
    //cognomi diversi => restituisci ordine cognomi.
    //cognomi uguali => restituisci ordine nomi
    return ( ( cognome != n.cognome ) ? ( cognome <= n.cognome ) : ( nome <= n.nome ) );
}

```

```

//stampa a video il contatto. Usato in iterazione nella rubrica
void Contatto::stampa_contatto(int n_id) const
{
    cout << "+- - - - -\n";
    cout << "| [ CONTATTO N. " << n_id << " ]\n";
    cout << "| " << cognome << " " << nome << " < " << mail << " >\n";
    cout << "| Casa: " << indirizzo_casa << " - Tel. " << telefono_casa << ".\n";
    cout << "| Ufficio: " << indirizzo_casa << " - Tel. " << telefono_casa << ".\n";
    cout << "| Cellulare: " << telefono_cellulare << "\n";
    cout << "| Note: " << note << "\n";
}

// crea la riga csv relativa al contatto
string Contatto::crea_riga_csv() const
{
    string temp = nome + ";" + cognome + ";" + mail + ";" + indirizzo_casa + ";" + telefono_casa + ";" + indirizzo_ufficio + ";"
        + telefono_ufficio + ";" + telefono_cellulare + ";" + note;
    return temp;
}

```

## Rubrica.h

```

#ifndef RUBRICA_H
#define RUBRICA_H

#include <list>
#include "Contatto.h"

class Rubrica
{
    list<Contatto> lista_contatti;
public:
    Rubrica() {}
    ~Rubrica() { lista_contatti.clear(); }
    void leggiadafile(string nomefile);
    bool scriviisufile(string nomefile) ;
    void aggiungi_contatto();
    void rimuovi_contatto();
    void stampa_rubrica() ;
};

#endif

```

## Rubrica.cpp

```
#include "Rubrica.h"
#include "Funzioni.h"
#include <fstream>
#include <string>

using namespace std;

//carica dati da file
void Rubrica::leggidafile( string nomefile )
{
    fstream RubricaFile;
    char str[4096]; //dim massima riga

    RubricaFile.open( nomefile.c_str(), ios::in | ios::out ); // sola lettura
    //in caso di errore esco dalla funzione, il file verrà ricreato all'uscita
    if ( RubricaFile.fail() )
    {
        stampa_errore( "Il file rubrica '" + nomefile + "' non esiste e verrà ricreato da zero!" );
        return;
    }

    //fino alla fine del file...
    while ( !RubricaFile.eof() )
    {
        int comma[9]; //array posizioni separatori csv
        int i, j, numero_linea = 0;

        str[0] = '\0';
        if ( RubricaFile.getline( str, 4096 ) ) //metto in str la linea corrente
        {
            if ( RubricaFile.fail() )
            {
                stampa_errore( "Errore in lettura dati in '" + nomefile + "'");
                return;
            }
            numero_linea++;
            //metto posizioni dei separatori in un altro array
            for ( i = 0, j = 0; str[i] != '\0' && j < 10; i++ )
            {
                if ( str[i] == ';' )
                {
                    comma[j++] = i;
                }
            }
            comma[j] = i; //salvo anche posizione ultimo carattere

            //controllo numero di campi nella riga
```

```

string s;
if ( j != 8 )
    stampa_errore( "Trovata riga csv non valida in '" + nomefile + "', linea " + int_to_string( numero_linea ) + " " );
else
{
    string riga_contatto( str ); // char -> string
    riga_contatto.append( ";" );

    string n( riga_contatto, 0, comma[0] - 0 );
    string c( riga_contatto, comma[0] + 1, comma[1] - comma[0] - 1 );
    string m( riga_contatto, comma[1] + 1, comma[2] - comma[1] - 1 );
    string ic( riga_contatto, comma[2] + 1, comma[3] - comma[2] - 1 );
    string tc( riga_contatto, comma[3] + 1, comma[4] - comma[3] - 1 );
    string iu( riga_contatto, comma[4] + 1, comma[5] - comma[4] - 1 );
    string tu( riga_contatto, comma[5] + 1, comma[6] - comma[5] - 1 );
    string tcel( riga_contatto, comma[6] + 1, comma[7] - comma[6] - 1 );
    string no( riga_contatto, comma[7] + 1, comma[8] - comma[7] - 1 );

    // aggiunge questo contatto alla rubrica
    Contatto cont_corr( n, c, m, ic, tc, iu, tu, tcel, no );
    //Contatto cont_corr( riga_contatto, "t", "t", "t", "t", "t", "t", "t", "t" );
    lista_contatti.push_back( cont_corr );
}
}

lista_contatti.unique(); //tolgo doppi
lista_contatti.sort(); //ordino
//cout << "Dopo unique e sort la lista ha: " << lista_contatti.size() << "elementi\n";
RubricaFile.close();
}

bool Rubrica::scrivisufil( string nomefile )
{
    ofstream RubricaFile;

    RubricaFile.open( nomefile.c_str(), ios::out ); // sola lettura
    if ( RubricaFile.fail() )
    {
        stampa_errore( "Errore apertura dati Rubrica!" );
        return false;
    }
    char str[4096]; //dim massima riga

    list < Contatto >::iterator it;

```

```

    for ( it = lista_contatti.begin(); it != lista_contatti.end(); it++ )
        RubricaFile << ( * it ).crea_riga_csv() << "\n";
    RubricaFile.close();
    return true;
}

void Rubrica::aggiungi_contatto()
{
    pulisci_schermo();
    Contatto nuovoc;
    nuovoc.input();
    lista_contatti.push_back( nuovoc ); //aggiungo nuovo contatto da shell
    lista_contatti.unique(); //tolgo doppioni
    lista_contatti.sort(); //ordino
}

void Rubrica::rimuovi_contatto()
{
    stampa_rubrica();
    int i, n_id;

    if ( lista_contatti.size() > 0 )
    {
        n_id = int_input( "Inserisci numero del contatto da eliminare(0 per annullare): ", 0, lista_contatti.size() );

        list < Contatto >::iterator it;

        // si ferma al n. di contatto giusto e lo flagga eliminato
        for ( it = lista_contatti.begin(), i = 0; it != lista_contatti.end(); it++ )
            if ( ++i == n_id )
            {
                lista_contatti.remove( * it );
                break;
            }
    }
}

void Rubrica::stampa_rubrica()
{
    int i;
    stampa_bordo_alto( "RUBRICA" );
    cout << "|          [" << lista_contatti.size() << " contatti presenti ]\n";
}

```

```

cout << "|\\n";
list < Contatto >::iterator it;
for ( it = lista_contatti.begin(), i = 0; it != lista_contatti.end(); it++ )
    ( * it ).stampa_contatto( ++i );
pausa();
}

```

## Cdata.h

```

#ifndef DATA_H
#define DATA_H
#include <iostream>
using namespace std;

/* 0-12 */
const string mesi[] =
{ "", /* [1] */ "Gennaio", "Febbraio", "Marzo", "Aprile", "Maggio", "Giugno", "Luglio", "Agosto", "Settembre",
  "Ottobre", "Novembre", /*[12]*/"Dicembre" };

const int duratamesi[] ={ 0, /* [1] */ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, /* [12] */ 31 };

//corrispondenza dei mesi con la somma dei giorni nei mesi precedenti
//es: febbraio->31
const int duratamesi_incrementale[]={0, /* [1] */ 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334 };

// Cdata
class Cdata
{
    int giorno; //1-31
    int mese; // 1-12
    int anno; //1950-2400
public:
    Cdata()
    { }

    //Costruttore g/m/a
    Cdata( int g, int m, int a ) : giorno( g ), mese( m ), anno( a )
    { }

    void input(string testo = "| Inserisci data "); //con testo dei default se non specificato
    bool input_from_string( string str_data );
    string to_string() const;
    string to_csv() const;
    bool operator == ( const Cdata & d ) const;
    bool operator < ( const Cdata & d ) const;

```

```

Cdata operator ++ (int); //postfisso
bool formatta_data() ;
void imposta_corrente();
friend ostream& operator<<(ostream& , Cdata& );
};

#endif

```

## Cdata.cpp

```

#include "Cdata.h"
#include <iostream>
#include <ctime>
#include <string>
#include "Funzioni.h"

using namespace std;

//legge la data da stringa:
//"oggi" -> imposta data corrente, "domani" -> imposta data di domani
//altra data in formato gg-mm-aaaa (04-02-2005).
//Accettata correttamente anche date del tipo "4-2-5", "004-000002-0000005" ...
//Restituisce true/false richiamando la formatta_data()
bool Cdata::input_from_string( string str_data )
{
    int d1 = str_data.find( '-' );
    int d2 = str_data.find_last_of( '-' );

    if ( str_data == "oggi" )
        imposta_corrente();
    else if ( str_data == "domani" )
    {
        imposta_corrente();
        ( * this ) ++;
    }
    else
    {
        giorno = atoi( str_data.substr( 0, d1 ).c_str() );
        mese = atoi( str_data.substr( d1 + 1, d2 - d1 ).c_str() );
        anno = atoi( str_data.substr( d2 + 1, str_data.length() ).c_str() );
    }
    return formatta_data();
}

```

```

//stampa il testo passato (se c'è) e richiede input della data da shell
void Cdata::input( string testo /* = */ )
{
    string buffer;
    do
    {
        buffer = string_input( testo + "(oggi | domani | gg-mm-aaaa): " );
    }
    while ( !input_from_string( buffer ) );
}

//restituisce data in formato gg mese testuale aaaa (es: 12 gennaio 2005)
string Cdata::to_string() const
{
    return string( int_to_string( giorno ) + " " + mesi[mese] + " " + int_to_string( anno ) );
}

//restituisce data in formato in gg-mm-aa per scrittura in csv
string Cdata::to_csv() const
{
    return string( int_to_string( giorno ) + "-" + int_to_string( mese ) + "-" + int_to_string( anno ) );
}

//confronta le date
bool Cdata::operator < ( const Cdata & d ) const
{
    //confronto riducendo al numero di giorni
    return ( ( anno * 365 + duratamesi_incrementale[mese] + giorno )
            < ( d.anno * 365 + duratamesi_incrementale[d.mese] + d.giorno ) );
}

//date uguali
bool Cdata::operator == ( const Cdata & d ) const
{
    return ( giorno == d.giorno && mese == d.mese && anno == d.anno );
}

//incrementa la data di un giorno
//se al 31-12-05 passa al 1-1-06
Cdata Cdata::operator ++( int )
{
    //istanzio nuova data con g,m,a della data corrente
    Cdata corr( /* this-> */ giorno, /* this-> */ mese, /* this-> */ anno );
}

```



```

    //incremento g,m,a della data this
    giorno++;
    if ( giorno > duratamesi[mese] )
    {
        mese++; giorno = 1;
    }
    if ( mese > 12 )
    {
        anno++; mese = 1;
    }
    //voglio op. postfisso => ritorno non this, ma la data corr istanziata all'inizio
    return corr;
}

//modifica eventualmente l'anno in migliaia e controlla che la data sia corretta
//es: 29-2-05, 15-13-2004 non valido
bool Cdata::formatta_data()
{
    bool val;
    if ( anno < 100 ) anno += 2000;
    val = ( giorno > 0 && giorno <= duratamesi[mese] && mese > 0 && mese <= 12 && anno >= 1950 && anno <= 2400 );
    if ( !val ) cout << "Data " << to_string() << " non valida !\n";

    return val;
}

//imposta la data corrente
void Cdata::imposta_corrente()
{
    time_t tmp;
    struct tm * tm_corrente;
    time( & tmp );
    tm_corrente = localtime( & tmp );
    giorno = tm_corrente->tm_mday;
    mese = tm_corrente->tm_mon + 1;
    anno = 1900 + tm_corrente->tm_year;
    formatta_data();
}

//overload ostream, friend di Cdata
ostream & operator << ( ostream & s, Cdata & d )
{
    return s << d.giorno << ' ' << mesi[d.mese] << ' ' << d.anno;
}

```

## Evento.h

```
#ifndef EVENTO_H
#define EVENTO_H

#include "Cdata.h"
#include "Funzioni.h"
#include <iostream>
#include <string>
using namespace std;

/** EVENTO ** */
class Evento //impostalo virtuale totalmente
{
    string tipo; //unico attributo in comune alle derivate = appuntamento/scadenza/GENERICO

public:
    Evento( string t );
    virtual void visualizza( int n_evento ) const;
    //crea la stringa csv per la memorizzazione su file
    virtual string to_csv( Cdata data ) const = 0; //virtuale pura

    void input();
};

/** APPUNTAMENTO ** */
class Eappuntamento : public Evento
{
    string oggetto;
    string luogo;
    string nome_persona;
    string note;

public:
    Eappuntamento();
    Eappuntamento( string ogg, string l, string np, string n );
    void visualizza( int n_evento ) const;
    string to_csv( Cdata data ) const;
    void input();
};
```

```

//per priorità scadenze
const string stelle[] =
{
    " ", " ", " **", " ****", " *****", " *******"
};

/** ESCADENZA ** */
class Escadenza : public Evento
{
    string oggetto;
    int priorit ;
    string note;

public:
    //ctor vuoto
    Escadenza() : Evento( "SCADENZA" )
    {
    }

    //ctor con tutti gli attributi
    Escadenza( string ogg, int prio, string n );
    void visualizza( int n_evento ) const;
    string to_csv( Cdata data ) const;
    void input();
};

/** GENERICO ** */
class Egenerico : public Evento
{
    string testo;

public:
    //ctor vuoto
    Egenerico() : Evento( "EVENTO GENERICO" )
    {
    }
}

```

```

    //ctor con tutti gli attributi
    Egenerico( string t );
    void visualizza( int n_evento ) const;
    string to_csv( Cdata data ) const;
    void input();
};

#endif

```

## Evento.cpp

```

#include "Evento.h"

Evento::Evento( string t ) : tipo( t )
{
}

// questa e la successiva sono richiamate anche dalle stesse funzioni delle classi derivate.
// si preferisce mantenere la parte comune nella funzione padre (piuttosto che ripetere la cout
// della funzione seguente in tutte le visualizza dei vari eventi) in modo che successive modifiche
// coinvolgano solo la funzione che segue
void Evento::visualizza( int n_evento ) const
{
    cout << "| --- [n. " << n_evento << "] -- " << tipo << " -----\n| |\n";
}

//crea la stringa csv per la memorizzazione su file
void Evento::input()
{
    cout << "|\n| -- INSERIMENTO " << tipo << " --\n";
}

//ctor vuoto
Eappuntamento::Eappuntamento() : Evento( "APPUNTAMENTO" )
{
}

//ctor con tutti gli attributi
Eappuntamento::Eappuntamento( string ogg, string l, string np, string n ) : Evento( "APPUNTAMENTO" ), oggetto( ogg ), luogo( l ),
nome_persona( np ), note( n )
{
}

```

```

// stampa gli attributi specifici degli appuntamenti
void Eappuntamento::visualizza( int n_evento ) const
{
    Evento::visualizza( n_evento ); //della classe padre
    cout << "| | " << oggetto << endl;
    cout << "| | Luogo:" << luogo << endl;
    cout << "| | Con:" << nome_persona << endl;
    cout << "| | " << note << "\n| +-----\n|\n";
}

// crea la riga per la memorizzazione su file
string Eappuntamento::to_csv( Cdata data ) const
{
    return string( "1;" + data.to_csv() + ";" + oggetto + ";" + luogo + ";" + nome_persona + ";" + note );
}

//input specifico per l'appuntamento
void Eappuntamento::input()
{
    Evento::input(); //stampa head inserimento
    input_e_controlla( "| [1/4] Oggetto: ", oggetto );
    input_e_controlla( "| [2/4] Luogo:", luogo);
    input_e_controlla( "| [3/4] Persona da incontrare: ", nome_persona);
    input_e_controlla( "| [4/4] Note:", note);
    cout << "|\\t[fine acquisizione]\\n";
}

Escadenza::Escadenza( string ogg, int prio, string n ) : Evento( "SCADENZA" ), oggetto( ogg ), priorita( prio ), note( n )
{
}

//stampa gli attributi specifici delle scadenze
void Escadenza::visualizza( int n_evento ) const
{
    Evento::visualizza( n_evento );
    cout << "| | " << oggetto << " (" << stelle[priorita] << ")\\n";
    cout << "| | " << note << "\\n| +-----\\n|\\n";
}

// crea la riga per la memorizzazione su file
string Escadenza::to_csv( Cdata data ) const
{
    return string( "2;" + data.to_csv() + ";" + oggetto + ";" + int_to_string( priorita ) + ";" + note );
}

```

```

//input specifico per scadenze
void Escadenza::input()
{
    Evento::input();
    input_e_controlla( "| [1/3] Oggetto: ", oggetto );
    priorit   = int_input( "| [2/3] Priorita [1-5]:", 1, 5 );
    input_e_controlla( "| [3/3] Note:" ,note);
    cout << "\t[fine acquisizione]\n";
}

Egenerico::Egenerico( string t ) : Evento( "EVENTO GENERICO" ), testo( t )
{
}

//stampa gli attributi specifici per eventi generici
void Egenerico::visualizza( int n_evento ) const
{
    Evento::visualizza( n_evento );
    cout << "| | " << testo << "\n| +-----\n";
}

// crea la riga per la memorizzazione su file
string Egenerico::to_csv( Cdata data ) const
{
    return string( "3;" + data.to_csv() + ";" + testo );
}

//input specifico per eventi generici
void Egenerico::input()
{
    Evento::input(); //stampa head inserimento
    input_e_controlla( "| [1/1] Testo: " ,testo);
    cout << "\t[fine acquisizione]\n";
}

```

## Eventi\_giorno.h

```

#ifndef EVENTI_GIORNO_H
#define EVENTI_GIORNO_H

#include "Evento.h"
#include <list>
// #include <string>

```

```

using namespace std;

/** secondo pair del map dell'agenda */
class Eventi_giorno
{
    // lista di puntatori a eventi
    list < Evento * > lista_eventi;

public:
    Eventi_giorno()
    {
    };

    ~Eventi_giorno();

    int n_eventi_giorno() const;
    void aggiungi( Evento * nuovo_evento );
    void input_evento();
    void stampa_eventi_giorno() ;
    string eventi_giorno_to_csv( Cdata data ) ;
    void stampa_scadenze( string titolo ) ;
    void rimuovi_da_lista_eventi();
    int rimuovi_tutti_gli_eventi();
};

#endif

```

## Eventi\_giorno.cpp

```

#include "Eventi_giorno.h"
#include <typeinfo>

Eventi_giorno::~~Eventi_giorno()
{
    rimuovi_tutti_gli_eventi();
}

int Eventi_giorno::n_eventi_giorno() const
{
    return lista_eventi.size();
}

```

```

//passo oggetto evento e lo mette nella giusta lista
void Eventi_giorno::aggiungi( Evento * nuovo_evento )
{
    //upcasting del puntatore. lista_eventi contiene oggetti 'Evento'
    lista_eventi.push_back( ( Evento * ) nuovo_evento );
}

// richiede tipo di app da inserire, chiede l'input relativo e lo aggiunge alla lista giusta tramite la aggiungi()
void Eventi_giorno::input_evento()
{
    int tipo_ev;
    tipo_ev = int_input( "| Inserisci tipo evento: [1=APPUNTAMENTO, 2=SCADENZA, 3=GENERICO ]: ", 1, 3 );
    if ( tipo_ev == 1 ) //dynamic cast ???
    {
        Eappuntamento * nuovo_app = new Eappuntamento();
        nuovo_app->input();
        aggiungi( nuovo_app ); //passo il puntatore
    }
    else if ( tipo_ev == 2 )
    {
        Escadenza * nuova_scad = new Escadenza();
        nuova_scad->input();
        aggiungi( nuova_scad );
    }
    else // 3
    {
        Egenerico * nuovo_ae = new Egenerico();
        nuovo_ae->input();
        aggiungi( nuovo_ae );
    }
}

//stampa tutti gli eventi dalla lista con numero crescente
void Eventi_giorno::stampa_eventi_giorno()
{
    int i = 1;

    if ( lista_eventi.size() == 0 )
        cout << "|\\t[ Nessun evento da visualizzare ]\\n\\n";
    else
    {
        list < Evento * >::iterator it;
        for ( it = lista_eventi.begin(); it != lista_eventi.end(); it++ )

```



```

        ( * it )->visualizza( i++ ); //polimorfismo sul tipo di evento
    }
    cout << "|\n";
}

//creo le righe csv concatenate degli eventi del giorno
string Eventi_giorno::eventi_giorno_to_csv( Cdata data )
{
    string temp( "" );
    list < Evento * >::iterator it;
    for ( it = lista_eventi.begin(); it != lista_eventi.end(); it++ )
        temp += ( ( * it )->to_csv( data ) ) + "\n"; //polimorfismo
    return temp;
}

void Eventi_giorno::stampa_scadenze( string titolo )
{
    int i = 1;
    list < Evento * >::iterator it;
    for ( it = lista_eventi.begin(); it != lista_eventi.end(); it++ )
    {
        if ( typeid( * ( * it ) ) == typeid( Escadenza ) ) //RTTI || match su tipo
        {
            cout << titolo;
            ( * it )->visualizza( i++ ); //polimorfismo
        }
    }
}

//rimuove l'elemento
void Eventi_giorno::rimuovi_da_lista_eventi()
{
    int numero_ev, i;

    stampa_eventi_giorno();
    numero_ev = int_input( "| Inserisci numero evento da eliminare (0 per annullare): ", 0 );

    list < Evento * >::iterator it;
    for ( it = lista_eventi.begin(), i = 0; it != lista_eventi.end(); it++ )
        if ( ++i == numero_ev ) //parte da 1
        {
            lista_eventi.remove( * it );
            break;
        }
}

```

```

int Eventi_giorno::rimuovi_tutti_gli_eventi()
{
    int n_ev = lista_eventi.size();
    list < Evento * >::iterator it;
    for ( it = lista_eventi.begin(); it != lista_eventi.end(); it++ )
        delete( * it );
    lista_eventi.clear(); //altrimenti la map su questo giorno dà errore
    return n_ev;
}

```

## Agenda.h

```

#ifndef AGENDA_H
#define AGENDA_H

#include <iostream>
#include <map>
#include "Funzioni.h"
#include "Cdata.h"
#include "Evento.h"
#include "Eventi_giorno.h"
#include <fstream>

using namespace std;

/** AGENDA ***** */
class Agenda
{
    //ogni data avrà 3 liste di eventi
    map < Cdata, Eventi_giorno > map_agenda_eventi; // mi chiama distruttori

public:
    Agenda()
    {
    }; //ctor

    ~Agenda()
    {
        map_agenda_eventi.clear();
    }
    void leggidafile( string nomefile );

```

```

bool scrivi_su_file( string nomefile ) ;
void aggiungi_evento();
void aggiungi_evento( Cdata nuova );
void rimuovi_evento();
void rimuovi_scaduti();
void stampa_eventi_del_giorno();
void stampa_eventi_del_giorno( Cdata data, bool nuova_schermata = true );
void stampa_scadenze();
int n_eventi_del( Cdata d );

};

#endif

```

## Agenda.cpp

```

#include "Agenda.h"
using namespace std;

void Agenda::leggi_da_file( string nomefile )
{
    fstream AgendaFile;
    int comma[99]; //array posizioni separatore
    int i, j, numero_linea = 0, tipo_ev;
    char str[4096]; //dim massima riga

    AgendaFile.open( nomefile.c_str(), ios::in ); // sola lettura
    if ( AgendaFile.fail() )
    {
        stampa_errore( "Il file agenda '" + nomefile + "' non esiste e verrà ricreato da zero!" );
        return; //il file non esiste e verrà ricreato da zero all'uscita dal programma
    }

    while ( !AgendaFile.eof() )
    {
        str[0] = '\0'; //azzerò buffer linea
        if ( AgendaFile.getline( str, 4096 ) )
        {
            numero_linea++; // per messaggi di errore eventuali
            if ( AgendaFile.fail() )
            {
                stampa_errore( "Errore in lettura dati '" + nomefile + "', riga " + int_to_string( numero_linea ) + " !" ); //esce
                return;
            }
            //metto nell'array comma[] le posizioni dei separatori trovati nella riga (che contiene attributi dell'evento)

```

```

for ( i = 0, j = 0; str[i] != '\0'; i++ )
    if ( str[i] == ';' )
        comma[j++] = i;
comma[j] = i; //salvo anche posizione ultimo carattere
//es: "2;21-6-2005;bollo auto;4;pagare" ha 5 campi e j=4

string riga_agenda( str ); // char -> string
riga_agenda.append( ";" ); // aggiungo altro delimitatore csv(;) alla fine per semplicità parsing

//tipo dell'evento : 1=app, 2=scad, 3=gener
tipo_ev = atoi( riga_agenda.substr( 0, 1 ).c_str() ); // 1 | 2 | 3

//ci devono essere almeno 3 campi.
if ( j >= 2 )
{
    Cdata data_evento;
    //leggo la data
    if ( data_evento.input_from_string( riga_agenda.substr( comma[0] + 1, comma[1] - comma[0] - 1 ) ) )
    {
        if ( tipo_ev == 1 ) //se si tratta di un appuntamento
        {
            if ( j >= 4 ) //se ha almeno i campi previsti per un appuntamento
            {
                Eappuntamento * e = new
                    Eappuntamento( riga_agenda.substr( comma[1] + 1, comma[2] - comma[1] - 1 ),
                        riga_agenda.substr( comma[2] + 1, comma[3] - comma[2] - 1 ), riga_agenda.substr( comma[3] + 1, comma[4] -
comma[3] - 1 ),
                        riga_agenda.substr( comma[4] + 1, comma[5] - comma[4] - 1 ) );
                map_agenda_eventi[data_evento].aggiungi( e );
            }
            else
                stampa_errore( "Appuntamento non valido in '" + nomefile + "', riga " + int_to_string( numero_linea ) + " !" );
        }
        else if ( tipo_ev == 2 ) //scadenza
        {
            if ( j >= 4 )
            {
                Escadenza * e = new Escadenza( riga_agenda.substr( comma[1] + 1, comma[2] - comma[1] - 1 ),
                    atoi( riga_agenda.substr( comma[2] + 1, comma[3] - comma[2] - 1 ).c_str() ),
                    riga_agenda.substr( comma[3] + 1, comma[4] - comma[3] - 1 ) );
                map_agenda_eventi[data_evento].aggiungi( e );
            }
            else
                stampa_errore( "Scadenza non valida in '" + nomefile + "', riga " + int_to_string( numero_linea ) + " !" );
        }
    }
}

```

```

    }
    else if ( tipo_ev == 3 ) //evento generico. non serve controllare j>=2
    {
        Egenerico * e = new Egenerico( riga_agenda.substr( comma[1] + 1, comma[2] - comma[1] - 1 ) );
        map_agenda_eventi[data_evento].aggiungi( e );
    }
    else //tipo di evento con numero diverso => errore
        stampa_errore( "Tipo di evento non valido in '" + nomefile + "', riga " + int_to_string( numero_linea ) + " !" );
    }
    else
        stampa_errore( "Data non valida in '" + nomefile + "', riga " + int_to_string( numero_linea ) + " !" );
    }
    else
        stampa_errore( "Linea non valida in '" + nomefile + "', riga " + int_to_string( numero_linea ) + " !" ); //fine linea
valida

    } //fine if getline
} //fine while !eof
//
AgendaFile.close();
}

//scrive su file tutti gli eventi presenti in memoria
bool Agenda::scrivisufil( string nomefile )
{
    fstream AgendaFile;
    AgendaFile.open( nomefile.c_str(), ios::out ); // sola lettura
    //in caso di errore esco dalla funzione senza leggere nulla
    if ( AgendaFile.fail() )
    {
        stampa_errore( "Errore apertura file agenda '" + nomefile + "!' );
        return false;
    }

    //scrivo le righe csv su file relative agli eventi in memoria
    map < Cdata, Eventi_giorno >::iterator it;
    for ( it = map_agenda_eventi.begin(); it != map_agenda_eventi.end(); it++ )
    {
        // stampo su file le scadenze utilizzando il metodo della classe Eventi_giorno e passandogli la data.
        // Avendo implementato l'operatore < per la classe Cdata(primo elemento del pair della map), gli eventi sono memorizzati in
ordine temporale
        AgendaFile << ( * it ).second.eventi_giorno_to_csv( ( * it ).first );
    }
}

```

```

    AgendaFile.close();
    return true;
}

//dopo aver richiesto la data, chiama l'inserimento sugli eventi del giorno corrispondenti
void Agenda::aggiungi_evento()
{
    Cdata nuova;
    stampa_bordo_alto( "INSERIMENTO EVENTO" );
    nuova.input(); //input data gg/mm/aaaa
    map_agenda_eventi[nuova].input_evento();
}

//come sopra ma con data già passata
void Agenda::aggiungi_evento( Cdata nuova )
{
    map_agenda_eventi[nuova].input_evento();
}

//stampo eventi nella immessa e chiamo la funzione di eliminazione alla data immessa
void Agenda::rimuovi_evento()
{
    int numero_ev;
    Cdata nuova;

    stampa_bordo_alto( "RIMOZIONE EVENTO" );
    nuova.input(); //input data gg/mm/aaaa
    map_agenda_eventi[nuova].rimuovi_da_lista_eventi();
    pausa();
}

//rimuove eventi scaduti se la data è minore di quella corrente
void Agenda::rimuovi_scaduti()
{
    int n_rimossi = 0;
    Cdata oggi;
    Cdata data_corr;
    oggi.imposta_corrente();

    map < Cdata, Eventi_giorno >::iterator it;
    for ( it = map_agenda_eventi.begin(); it != map_agenda_eventi.end(); it++ )

```

```

{
    if ( ( * it ).first < oggi )
        n_rimossi += ( * it ).second.rimuovi_tutti_gli_eventi();
}
if ( n_rimossi > 0 )
    cout << "| " << n_rimossi << " Eventi rimossi correttamente\n";
else
    cout << "| Nessun evento da rimuovere !\n";
pausa();
}

//richiedo data e stampo gli eventi in tal data in una nuova schermata
void Agenda::stampa_eventi_del_giorno()
{
    Cdata data;
    data.input();
    stampa_bordo_alto( "VISUALIZZAZIONE EVENTI DEL" + data.to_string() );
    map_agenda_eventi[data].stampa_eventi_giorno(); //f. della classe Eventi_giorno
    pausa();
}

//come sopra ma ho già la data che non richiedo.
//se passo 'false 'come secondo parametro posso usarla in visualizzazioni batch del tipo "eventi dal .. al ..."
void Agenda::stampa_eventi_del_giorno( Cdata data, bool nuovaschermata /* = true */ ) //passare la data
{
    string titolo = "VISUALIZZAZIONE EVENTI DEL " + data.to_string();
    if ( nuovaschermata )
    {
        pulisci_schermo();
        stampa_bordo_alto( titolo );
    }
    else
        cout << "+----- " << titolo << " -----\\n\\n";
    //stampo tutti gli eventi del giorno
    map_agenda_eventi[data].stampa_eventi_giorno();
    if ( nuovaschermata )
        pausa();
}

//stampa scadenze nelle prossime x settimana
void Agenda::stampa_scadenze()
{

```

```

int n_giorni;
Cdata data;

//richiedo n. settimane
n_giorni = 7 * int_input( "Numero di settimane : ", 1 );
pulisci_schermo();
//nuova schermata
stampa_bordo_alto( "SCADENZE NELLE PROSSIME " + int_to_string( n_giorni / 7 ) + " SETTIMANE" );
data.imposta_corrente();
//per tutti i giorni compresi nelle prossime x settimane...

for ( int i = 0; i < n_giorni; i++, data++ )
    map_agenda_eventi[data].stampa_scadenze( "|--- fra " + int_to_string( i ) + " giorni (" + data.to_string() + ")--\n" );

cout << "|-- Nessun altro evento prima del " << data.to_string() << "\n";
pausa();
}

//restituisce il numero degli eventi in una data
int Agenda::n_eventi_del( Cdata d )
{
    return map_agenda_eventi[d].n_eventi_giorno();
}

```

## main.cpp

```

#include "Rubrica.h"
#include "Funzioni.h"
#include "Agenda.h"
#include <iostream>
#include <ctime>
#include <string>
#include <fstream>
using namespace std;

const string file_rubrica = "rubrica.csv";
const string file_agenda = "agenda.csv";

int main()
{
    int n_op; //n. op corrente dello switch principale
    Rubrica r_principale;
    Agenda a_principale;
    int temp1, temp2; //temporanee per lo switch

```



```

Cdata data1, data2;

data1.imposta_corrente();
data2.imposta_corrente(); data2++;

//apro rubrica e agenda
r_principale.leggidafile( file_rubrica );
a_principale.leggidafile( file_agenda );

do
{
    //stampa menu
    stampa_menu( a_principale.n_eventi_del( data1 ), a_principale.n_eventi_del( data2 ) );
    //richiedi operazione da effettuare
    n_op = int_input( " Operazione da effettuare: ", 1, 14 ); //
    //switch principale delle operazioni
    switch ( n_op )
    {
        case 1: //Inserisci Evento
            a_principale.aggiungi_evento();
            pausa();
            break;
        case 2: //Eventi di oggi
            data1.imposta_corrente();
            a_principale.stampa_eventi_del_giorno( data1 );
            break;
        case 3: //Eventi di domani
            data1.imposta_corrente();
            data1++;
            a_principale.stampa_eventi_del_giorno( data1 );
            break;
        case 4: //Eventi del ...
            data1.input(); //richiedo data
            a_principale.stampa_eventi_del_giorno( data1 );
            break;
        case 5: //Eventi dei prossimi ... giorni
            temp1 = int_input( "Numero di giorni (1-30): ", 1, 30 );
            data1.imposta_corrente();
            stampa_bordo_alto( "EVENTI DEI PROSSIMI " + int_to_string( temp1 ) + " GIORNI" );
            for ( int i = 1; i <= temp1; i++ )
                a_principale.stampa_eventi_del_giorno( data1++, false );
            pausa();
            break;
        case 6: //Eventi dal ... al ...
            data1.input( " Dal " );

```

```

data2.input( " A1 " );
stampa_bordo_alto( "EVENTI DAL " + data1.to_string() + " AL " + data2++.to_string() );
while ( data1 < data2 )
    a_principale.stampa_eventi_del_giorno( data1++, false );
pausa();
break;

case 7: //Scadenze delle prossime ... settimane
    a_principale.stampa_scadenze();
break;
case 8: //Visualizza rubrica
    r_principale.stampa_rubrica();
break;
case 9: //Inserisci nuovo contatto
    r_principale.aggiungi_contatto();
break;
case 10: //Cancella contatto...
    r_principale.rimuovi_contatto(); //rimuovo da vettore
break;
case 11: //Cancella evento...
    a_principale.rimuovi_evento();
break;
case 12: //Cancella eventi scaduti
    if ( richiedi_conferma( " Elimino eventi scaduti (s/n) ?" ) )
        a_principale.rimuovi_scaduti();
break;
case 14: //Esci senza salvare
    if ( richiedi_conferma( " Vuoi uscire senza salvare (s/n) ?" ) )
        exit( 1 );
break;
}
}
while ( n_op != 13 );

//finchè non salvo correttamente entrambi i files...
while ( !r_principale.scrivisufilere( file_rubrica ) || !a_principale.scrivisufilere( file_agenda ) )
{
    cout << "\t[Errore scrittura file]\n";
    pausa();
}
cout << "\t[ Modifiche correttamente registrate su file ]\n";

pausa();
return 0;
}

```

--

# TESTING

Numerosissimi test e successive correzioni sono stati effettuati in corso d'opera per tutte le funzioni e moduli scritti (singolarmente e nel complesso), utilizzando:

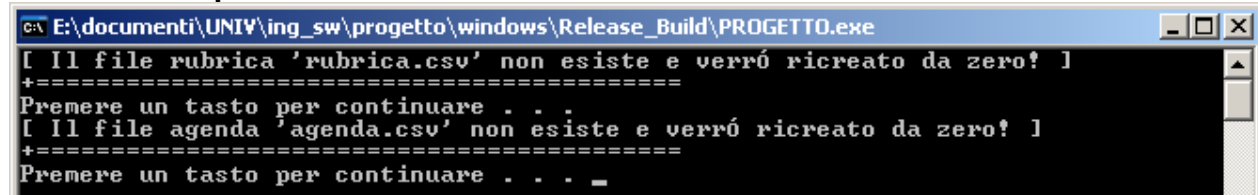
- codice appositamente scritto per testarle
- strumento di debug di Borland C++BuilderX.

Sono stati effettuati molti test Black-box e in alcuni casi sono stati corretti errori scaturiti dall'interazione fra i vari moduli.

Si riporta di seguito una serie di test Black-box rappresentativa delle principali operazioni.

(gli input immessi sono indicati con un bordo esterno)

## Caso files non presenti:



Visualizzazione corretta.

## OP-1 Inserimento eventi

Sono inseriti i seguenti eventi:

oggi -> 1 -> Oggetto = "oggetto test", Luogo="nd", Persona="x", Note="y"

oggi -> 3 -> Testo="testing programma"

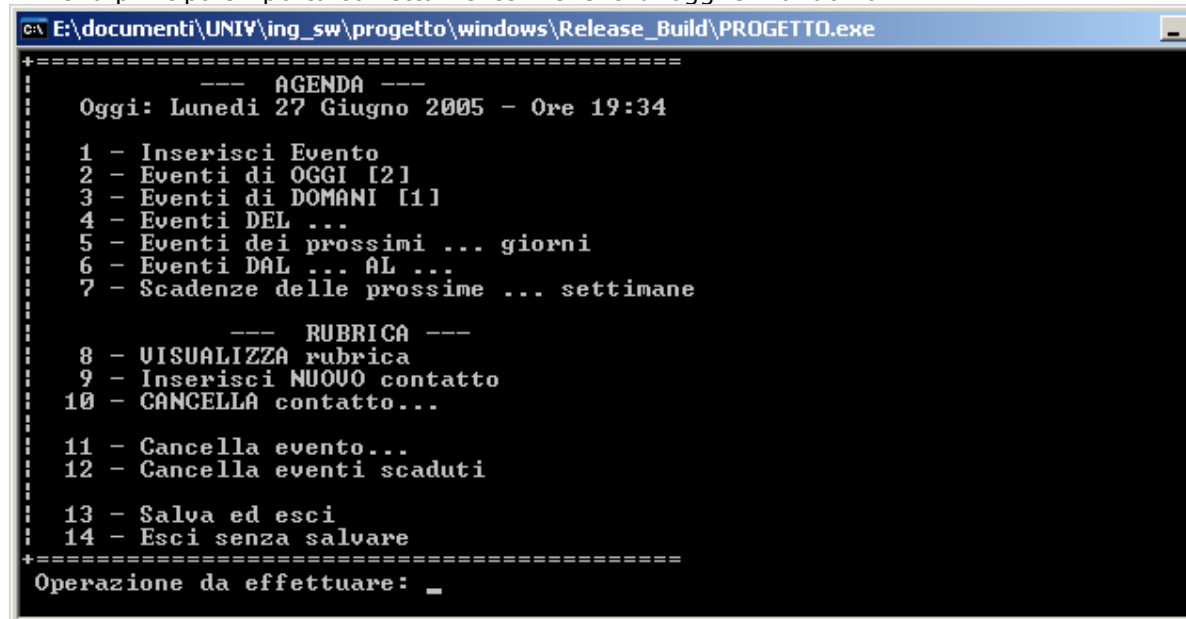
domani -> 2 -> Oggetto="ab.internet", Priorit =3, Note="cambia profilo"

2-7-5 -> 2 -> Oggetto="esame ing. Sw.", Priorit =5, Note="porta libretto esami e penna"

26-6-5 -> 3 -> Testo="evento di ieri"

Gli inserimento sono stati effettuati correttamente.

Il menu principale riporta correttamente 2 eventi di oggi e 1 di domani



**OP 2-3** La visualizzazione degli eventi di oggi e di domani è corretta:

```
C:\E:\documenti\UNIV\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
+=====
|      ---  VISUALIZZAZIONE EVENTI DEL 27 Giugno 2005  ---
|      Oggi: Lunedì 27 Giugno 2005 - Ore 19:35
|
|      +--- [n. 1] -- EVENTO GENERICO -----
|      |      testing programma
|      +-----
|      +--- [n. 2] -- APPUNTAMENTO -----
|      |      oggetto test
|      |      Luogo:nd
|      |      Con:x
|      |      y
|      +-----
|
|      +=====
|      Premere un tasto per continuare . . . _
```

```
C:\E:\documenti\UNIV\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
+=====
|      ---  VISUALIZZAZIONE EVENTI DEL 28 Giugno 2005  ---
|      Oggi: Lunedì 27 Giugno 2005 - Ore 19:37
|
|      +--- [n. 1] -- SCADENZA -----
|      |      abbonamento internet (***)
|      |      cambia profilo
|      +-----
|
|      +=====
|      Premere un tasto per continuare . . .
```

**OP.4** Eventi del...

2-7-5 -> Visualizzazione corretta

```
C:\E:\documenti\UNIV\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
+=====
|      ---  VISUALIZZAZIONE EVENTI DEL 2 Luglio 2005  ---
|      Oggi: Lunedì 27 Giugno 2005 - Ore 19:40
|
|      +--- [n. 1] -- SCADENZA -----
|      |      esame ing. Sw (*****)
|      |      porta libretto esami e penna
|      +-----
|
|      +=====
|      Premere un tasto per continuare . . . _
```

6->Visualizzazione corretta

```

+-----EVENTI DEI PROSSIMI 6 GIORNI-----
Oggi: Lunedì 27 Giugno 2005 - Ore 19:42

+----- VISUALIZZAZIONE EVENTI DEL 27 Giugno 2005 -----
+-- [n. 1] -- EVENTO GENERICO -----
|
| testing programma
|-----
+-- [n. 2] -- APPUNTAMENTO -----
|
| oggetto test
| Luogo:nd
| Con:x
| y
|-----
+----- VISUALIZZAZIONE EVENTI DEL 28 Giugno 2005 -----
+-- [n. 1] -- SCADENZA -----
|
| abbonamento internet (**)
| cambia profilo
|-----
+----- VISUALIZZAZIONE EVENTI DEL 29 Giugno 2005 -----
[ Nessun evento da visualizzare ]

```

```
Premere un tasto per continuare . . .
```

**OP.6** Eventi dal...al...(da) domani -> (al) 3-7-5 -> Visualizzazione corretta

```

C:\E:\documenti\UNIV\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
+-----+
+-----  EVENTI DAL 28 Giugno 2005 AL 3 Luglio 2005  -----+
Oggi: Lunedì 27 Giugno 2005 - Ore 19:49
+-----  VISUALIZZAZIONE EVENTI DEL 28 Giugno 2005  -----+
+-- [n. 1] -- SCADENZA -----+
+-- | abbonamento internet (***) |
+-- | cambia profilo |
+-----+
+-----  VISUALIZZAZIONE EVENTI DEL 29 Giugno 2005  -----+
+-- | [ Nessun evento da visualizzare ] |
+-----+
+-----  VISUALIZZAZIONE EVENTI DEL 30 Giugno 2005  -----+
+-- | [ Nessun evento da visualizzare ] |
+-----+
+-----  VISUALIZZAZIONE EVENTI DEL 1 Luglio 2005  -----+
+-- | [ Nessun evento da visualizzare ] |
+-----+
+-----  VISUALIZZAZIONE EVENTI DEL 2 Luglio 2005  -----+
+-- [n. 1] -- SCADENZA -----+
+-- | esame ing. Sw (*****) |
+-- | porta libretto esami e penna |
+-----+
+-----  VISUALIZZAZIONE EVENTI DEL 3 Luglio 2005  -----+
+-- | [ Nessun evento da visualizzare ] |
+-----+
+=====+
Premere un tasto per continuare . . . _

```

**OP.7** Scadenze nelle prossime .. settimane3 -> Visualizzazione corretta. dopo 21 giorni il 27 giu 2005 è infatti il 18 luglio e sono presenti fino allora solo due scadenze.

```

C:\E:\documenti\UNIV\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
+-----+
+-----  SCADENZE NELLE PROSSIME 3 SETTIMANE  -----+
Oggi: Lunedì 27 Giugno 2005 - Ore 19:46
+-----+
+-- fra 1 giorni (28 Giugno 2005)--+
+-- [n. 1] -- SCADENZA -----+
+-- | abbonamento internet (***) |
+-- | cambia profilo |
+-----+
+-- fra 5 giorni (2 Luglio 2005)--+
+-- [n. 1] -- SCADENZA -----+
+-- | esame ing. Sw (*****) |
+-- | porta libretto esami e penna |
+-----+
+-- Nessun altro evento prima del 18 Luglio 2005
+=====+
Premere un tasto per continuare . . . _

```

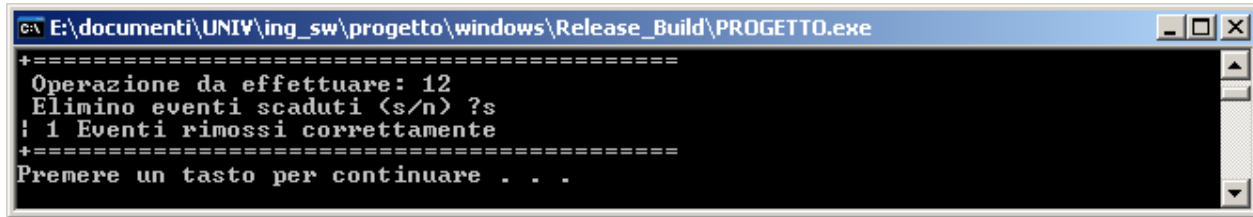
**OP.11** Cancella evento

28-6-5-> viene visualizzata correttamente l'unica scadenza presente.

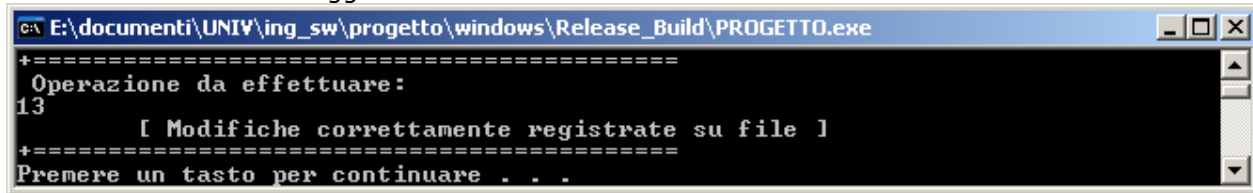
1 -> Visualizzazione corretta dell'eliminazione

Nel menu principale non sono più indicati eventi per domani ([0]) e l'op n.3 conferma che non ci sono eventi per domani.

**OP.12** Cancella eventi scaduti. Solo un evento da rimuovere -> Visualizzazione corretta



**OP.13** Uscita con salvataggio -> Visualizzazione corretta



Contenuto del file **agenda.csv** -> Visualizzazione corretta

3;27-6-2005;testing programma 1;27-6-2005;oggetto test;nd;x;y 2;2-7-2005;esame ing. Sw;5;porta libretto esami e penna
---

Alla rilancio del programma tutte le operazioni di visualizzazione (2-3-4-5-6-7) danno gli stessi risultati visti. Il caricamento dei dati è quindi corretto.

**OP.14** uscita senza salvare

L'uscita senza salvare è corretta. Gli eventi cancellati risultano essere ancora presenti su file.



### OP-1 Inserimento data e campi errati

Le date oggi, oggi, 30-2-2005, 0-0-1940 sono correttamente riconosciute come non valide.

La data 1-1-1 è riconosciuta correttamente.

I tipi di evento 0 e 4 non sono ammessi.

Il tipo 2 è correttamente ammesso.

Nei campi Oggetto e Note non sono ammessi campi contenenti punto e virgola.

Come priorità non sono ammessi 0 e 6. E' ammesso invece correttamente il numero 3.

```

E:\documenti\univ\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
+=====+
      --- INSERIMENTO EUENTO ---
      Oggi: Mercoledì 29 Giugno 2005 - Ore 0:26

      ! Inserisci data <oggi | domani | gg-mm-aaaa>: oggi
Data 0 2000 non valida !
      ! Inserisci data <oggi | domani | gg-mm-aaaa>: oggi;
Carattere ';' non ammesso !
      ! Inserisci data <oggi | domani | gg-mm-aaaa>: 30-2-2005
Data 30 Febbraio 2005 non valida !
      ! Inserisci data <oggi | domani | gg-mm-aaaa>: 0-0-1940
Data 0 1940 non valida !
      ! Inserisci data <oggi | domani | gg-mm-aaaa>: 1-1-1
      ! Inserisci tipo evento: [1=APPUNTAMENTO, 2=SCADENZA, 3=GENERICO ]: 0
      ! Inserisci tipo evento: [1=APPUNTAMENTO, 2=SCADENZA, 3=GENERICO ]: 4
      ! Inserisci tipo evento: [1=APPUNTAMENTO, 2=SCADENZA, 3=GENERICO ]: 2

      -- INSERIMENTO SCADENZA --
      [1/3] Oggetto: bolletta;
Carattere ';' non ammesso !
      [1/3] Oggetto: bolletta
      [2/3] Priorita [1-5]:0
      [2/3] Priorita [1-5]:6
      [2/3] Priorita [1-5]:3
      [3/3] Note:note;
Carattere ';' non ammesso !
      [3/3] Note:note
      [fine acquisizione]
+=====+
Premere un tasto per continuare . . .

```

### OP.9 inserisci contatto

Sono inseriti i contatti seguenti senza rilevare errori. I caratteri punto e virgola non sono stati correttamente ammessi

nome	cognome	mail	Ndirizzo _casa	Telefono _casa	Indirizzo _ufficio	Telefono _ufficio	Telefono _cellulare	note
paolo	rossi	aolorossi@ mail.com	via roma n.43	0541923456	via prato		3334565443	telefonare ore pasti
Luigi	Bianchi	luigib@ libero.it	via mozart n.45. Rimini	0541 123456				
paolo	rossi	aolorossi@ mail.com	via roma n.43	0541923456	via prato		3334565443	telefonare ore pasti

## OP.8 Visualizza rubrica

I contatti sono correttamente ordinati e sono rimossi i duplicati.

```

C:\E:\documenti\UNIV\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
+=====
+--- RUBRICA ---
+Oggi: Lunedì 27 Giugno 2005 - Ore 20:28
+
+ [2 contatti presenti ]
+
+-----
+ [ CONTATTO N. 1 ]
+Bianchi Luigi < luigib@libero.it >
+ Casa: via mozart n.45. Rimini - Tel. 0541 123456.
+ Ufficio: via mozart n.45. Rimini - Tel. 0541 123456.
+ Cellulare:
+ Note:
+
+-----
+ [ CONTATTO N. 2 ]
+rossi paolo < paolorossi@mail.com >
+ Casa: via roma n.43 - Tel. 0541923456.
+ Ufficio: via roma n.43 - Tel. 0541923456.
+ Cellulare: 3334565443
+ Note: telefonare ore pasti
+=====
+Premere un tasto per continuare . . .

```

## OP.10 Cancella contatto

2-> La successiva visualizzazione conferma la avvenuta cancellazione di "Bianchi Luigi"

```
C:\E:\documenti\UNIV\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
```

```
+=====+  
      --- RUBRICA ---  
Oggi: Lunedì 27 Giugno 2005 - Ore 20:35  
  
[1 contatti presenti ]  
  
+-----+  
[ CONTATTO N. 1 ]  
rossi paolo < paolorossi@mail.com >  
Casa: via roma n.43 - Tel. 0541923456.  
Ufficio: via roma n.43 - Tel. 0541923456.  
Cellulare: 3334565443  
Note: telefonare ore pasti  
+=====+  
Premere un tasto per continuare . . .
```

contenuto di **rubrica.csv** -> Corretto

paolo;rossi;paolorossi@mail.com;via roma n.43;0541923456;via prato;;3334565443;telefonare ore pasti

Al rilancio del programma l'operazione n.8 restituisce la stessa schermata sopra -> Visualizzazione corretta.

## Caricamento da file agenda con dati errati

1	3;27-6-2005;testing programma	OK
2	3;27-6-2005	Troppi pochi campi
3		nessun carattere nella linea
4	1;27-6-2005;oggetto test;nd;x;y	OK
5	1;30-2-2005;ogg;nd;x;y	Data 30 febbraio non valida
6	2;2-7-2005;esame ing. Sw;5;porta ... penna	OK
7	4;27-6-2005;oggetto test;nd;x;y	EVENTO 4 no valido
8	linea;non;valida	rilevato errore data (prima di rilevazione errore tipo di evento)

```

C:\E:\documenti\UNIV\ing_sw\progetto\windows\Release_Build\PROGETTO.exe
[ Linea non valida in 'agenda.csv', riga 2 ! ]
+=====
Premere un tasto per continuare . . .
[ Linea non valida in 'agenda.csv', riga 3 ! ]
+=====
Premere un tasto per continuare . . .
Data 30 Febbraio 2005 non valida !
[ Data non valida in 'agenda.csv', riga 5 ! ]
+=====
Premere un tasto per continuare . . .
[ Tipo di evento valido in 'agenda.csv', riga 7 ! ]
+=====
Premere un tasto per continuare . . .
Data 0 2000 non valida !
[ Data non valida in 'agenda.csv', riga 8 ! ]
+=====
Premere un tasto per continuare . . .

```

Visualizzazione corretta.

All'uscita con salvataggio rimangono :

```

3;27-6-2005;testing programma
1;27-6-2005;oggetto test;nd;x;y
2;2-7-2005;esame ing. Sw;5;porta libretto esami e penna

```

Ci solo le righe 1,4 e 6 -> Visualizzazione corretta

## Riepilogo

Verifica: Il software è conforme alle specifiche

Testing: I test effettuati hanno avuto esito positivo.

## FILE ALLEGATI E NOTE SULL'UTILIZZO DEL SW

Il programma cerca agenda.csv e rubrica.csv nella stessa cartella dell'eseguibile, così come impostato con le variabili globali `file_rubrica` e `file_agenda`.

Sono già presenti i file:

agenda.csv

```

3;21-6-2005;Compleanno Marco
3;29-6-2005;fine test progetto ing sw
2;30-6-2005;consegna progetto ing sw;5;in pdf al prof
3;1-7-2005;domani esame ing sw ore 9:30 sti
1;2-7-2005;Esame Ing SW;STI;Pagnini;porta libretto esame e penna
3;25-12-2005;BUON NATALE !!!
3;1-1-2006;BUON ANNO !!

```

rubrica.csv

```

Andrea;Albini;albinia@libero.it;Via trasversa n.14 Secchiano Novafeltria(PU);0541912095;;;3331234567;Andrea
Alessandro;Balchesini;balche@libero.it;;;3345676345;Alessandro
Gianluigi;Brizzi;gbrizzi@libero.it;Via Poggio Ancisa 4, Casteldelci (PU);541786723;Via ravennate 8
Forli(FO);543675674;334567643;Gianluigi
Elvis;Ciotti;mia@posta.it;via;ca;bicci;0541;333;note

```