# ITST J.F. KENNEDY PORDENONE

**Informatica**

Esercitazione:

Compito Realtà - Rimborsi auto
Documentazione

Team: Blu (Donadel Luca, Ermacora Massimo, Maccora Giuseppe)

Classe: 3B IA

Data: 27/4/2022

a.s: 2021-2022

## 1. Testo problema

# Rimborsi KM Auto

**1. Gestire i rimborsi Km Auto**
Inserite in una struct almeno i segeunti campi:

- IdAuto (un progressivo numerico)
- des_auto
- targa_auto
- costo_km_auto
- IdPersona (numerico)
- AnnoRimb
- MeseRimb
- km_rimborso

**2.** Gestire il caricamento da file per IdPersona in memoria dei rimborsi. Le informazioni relative ai rimborsi contenute nel file, corrispondono ai campi della struct.
Per generare i rimborsi auto usate Excel (esportare il foglio Excel dei rimborsi auto in formato .csv)

**3**. I rimborsi caricati in memoria relativi ad una persona deve essere sempre ordinata in modo crescente rispetto al campo targa_auto.

**4.** Gestire l'inserimento da programma c++ di nuovi rimborsi per persona (l'ordine deve essere mantenuto)

**5**. Gestione della ricerca per IdAuto e per IdPersona

**6**. Gestire la cancellazione di un rimborso (l'ordine deve essere mantenuto)

**7.** Visualizzazione del contenuto dei rimborsi auto

**8.** Gestione del salvataggio su file dei rimborsi auto.

**9.** Realizzazione di un programma che attraverso un menù richiami ognuna di queste gestioni

## 2. Analisi problema:

Questo programma fornisce diverse funzioni per modificare i dati di un azienda che si occupa di rimborsi auto salvati in un file .csv. Grazie al programma si potranno svolgere diverse azioni su questo file, ovvero:

- Caricare nuovi rimborsi (casi particolari previsti: l'utente inserisce un rimborso di un'auto che esiste già, identificabile dalla targa);
- Rimuovere rimborsi;
- Ordinare i rimborsi già presenti in base all'id della persona da rimborsare (casi particolari previsti: una persona ha più di un rimborso, quindi l'ordinamento va fatto in base alla targa);
- Stampare a video un determinato rimborso ricercato in base all'id dell'auto o della persona (nel secondo caso, viene stampato più di un rimborso nel caso in cui la persona cercata ne abbia più di uno);
- Mostrare a video una tabella contenente tutti i rimborsi attualmente caricati nel file;
- Salvare le modifiche sul file .csv quando lo si desidera;
- Uscire dal file.

Tutte queste azioni saranno eseguibili dall'utente grazie ad un menu.
Per la gestione dei dati sui rimborsi era possibile contenerli negli array, che sono di tipo statico (quindi non modificabili), ma sono stati utilizzati i vector, che sono di tipo dinamico (quindi modificabili senza essere dover inserire una dimensione). I vector potevanno essere suddivisi in tipo di dato astratto enum per la gestione dei dati, ma è stato preferito utilizzare una struct, come chiesto da consegna.

## 3. Strutture dati utilizzate:

I dati dei rimborsi vengono presi in input dal file "DATIRIMBORSO.csv" creato con Excel, formattato in maniera tale da avere diverse righe: ogni riga contiene, separati da un punto e virgola, i dati di ogni rimborso in ordine seguente:

IdPersona,IdAuto,targa_auto,costo_km_auto,AnnoRimb,MeseRimb,km_rimborso,des_auto.

Esempio:

367;2;AF456HJ;1.20;2004;9;28399;FIAT
560;3;WE456YU;1.89;2017;1;45678;HYUNDAI

Questi dati verranno presi da una struttura dati di tipo **fstream** impostata come "read", per poi essere caricati da una funzione apposita in diverse **struct** (una per rimborso), che a loro volta verranno caricate in un **vector** dalla stessa funzione.

La struct in cui si contiene il rimborso quindi sarà formata dai campi citati prima:

IdPersona;
IdAuto;
targa_auto;
costo_km_auto;
AnnoRimb;
MeseRimb;
km_rimborso;
des_auto;

### 5.Organizzazione programma

Il programma è stato organizzato secondo la seguente struttura:
Il file Rimborsi_Auto.cpp in cui ritroviamo la funzione principale main.
Il file funzioniRimborso.cpp in cui troviamo il codice di ogni funzione.
Il file funzioniRimborso.h in cui troviamo la dichiarazione di ogni funzione e della struct.

### 4. Descrizione programma

### int main()

Nel main del progetto si utilizza la funzione setlocale() per mostrare anche i caratteri speciali nella console. Si apre inoltre un file .csv (dove ci sono i dati dei rimborsi) in lettura, si controlla che non sia fallita la lettura con un if (utilizzando il metodo .fail() della libreria fstream), e con la funzione caricamentoDati vengono caricati i dati in un vector, dopodichè viene chiamata la funzione mostramenu() che permette all'utente di utilizzare il programma. Infine, quando l'utente uscirà dalla funzione mostramenu, lo si avvertirà del fatto che il programma si sta chiudendo con un cout e sarà chiuso il file .csv in lettura.

## Funzioni:

**1) void mostramenu(std::vector<datiRimborso> listaDati, std::fstream &fileDati)**

La funzione mostramenu è la funzione principale del programma. È chiamata dal main e contiene un ciclo while che chiede all'utente cosa vuole fare, aspetta l'input e chiama le funzioni necessarie. Ci sono un totale di 7 scelte (corrispondenti alle 7 azioni presenti nella sezione 2 della documentazione).

**2) void caricamentoDati(std::vector<datiRimborso>& listaDati, std::fstream& inputFile)**

La funzione caricamentoDati prende dal file della stream inputFile i dati necessari, poi li carica nel vector listaDati passato come riferimento. Alla fine della funzione viene chiamata la funzione di ordinamento ordinamentoDati(). Per fare ciò utilizza un ciclo while che si ripete fino alla fine del file e la funzione getline() per leggere i dati dalla variabile di tipo fstream.

**3) void ordinamentoDati(std::vector<datiRimborso>& listaDati)**

La funzione ordinamentoDati ordina in ordine crescente secondo il campo idPersona le struct contenute nel vector listaDati passato come riferimento. Se una persona possiede più macchine, le struct verranno ordinate nella memoria in base al campo "targa_auto". L'algoritmo utilizzato è il bubble sort.

**4) void inserimentoDati(std::vector<datiRimborso>& listaDati, bool &modDaSalvare)**

La funzione inserimentoRimborso chiede all'utente i dati sul rimborso che egli desidera aggiungere ad una persona già esistente oppure no, per poi caricarli in una struct che viene poi caricata nel vector. Tutti gli input vengono accettati solo se validi (ad es. nel campo "meseRimborso" vengono accettati solo valori da 1 a 12).

Se l'utente inserisce una targa che già esiste, questa funzione chiederà all'utente se egli desideri sovrascrivere il vecchio rimborso oppure non inserire quello nuovo. Nel caso in cui egli voglia sovrascrivere, alla fine dell'esecuzione della funzione la struct appena creata verrà salvata nella cella in cui si trovava precedentemente il rimborso da sovrascrivere, invece che venir inserita in coda al vector con push_back().

Quando la funzione termina la sua esecuzione (eccetto nel caso in cui l'utente abbia inserito una targa già presente e abbia deciso di interrompere l'inserimento) viene chiamata la funzione ordinamentoDati(). Il valore della variabile bool modDaSalvare passata byReference viene inoltre impostato a true. Questa variabile permetterà al programma di avvisare l'utente che ci sono delle modifiche non salvate nel file .csv.

**5) void convalida_targa(const std::string targa)**

La funzione convalida_targa viene chiamata durante l'inserimento di un nuovo rimborso per controllare che la targa inserita sia conforme alle targhe italiane (es. 00AAA00).

**6) int ricercaTarga(std::vector<datiRimborso> listaDati, std::string targaRicercata)**

La funzione ricercaTarga utilizza il metodo compare() della libreria string per ricercare la stringa passata come riferimento tra le targhe dei rimborsi presenti nel vector. Il metodo ritorna 0 se le stringhe comparate sono uguali, permettendoci di trovare la stringa cercata. Se non trova corrispondenze in nessun rimborso, questa funzione ritorna -1.

**7) int ricercaIdAuto(std::vector<datiRimborso> listaDati, const int idRicercato)**

La funzione ricercaIdAuto cerca nel vettore per idAuto il rimborso corrispondente al parametro idRicercato, poi ritorna la sua posizione nel vettore. Se esso non viene trovato, viene ritornato il valore -1. Poiché il vettore non è ordinato secondo il campo IdAuto, la ricerca è di tipo lineare.

**8) void ricercaIdPersona(std::vector<datiRimborso> listaDati, const int idRicercato, std::vector<int> &IdPersonaTrovati);**

La funzione ricercaIdPersona cerca nel vettore per idPersona il rimborso corrispondente al parametro idRicercato mediante ricerca binaria, poi controlla rispettivamente a destra e a sinistra dell'ID trovato finchè non trova un IdPersona diverso da quello cercato (la ricerca binaria trova l'indice di una cella contenente un rimborso con IdPersona = IdRicercato, ma se ne ho più di uno con lo stesso Id non ho garanzie riguardo a quale di essi venga trovato). Durante questi passaggi, carica gli indici delle celle contenenti rimborsi con IdPersona uguale a IdRicercato in un vettore passato per riferimento IdPersonaTrovati. Si può sapere se la funzione ha trovato corrispondenze controllando la grandezza di questo vettore.
Questo vettore viene poi utilizzato dalla funzione mostra_per_IDAuto() per stampare a video tutti i rimborsi con l'ID appena cercato dall'utente.

**9) void intBubbleSort(std::vector<int>& vect)**

La funzione intBubbleSort ordina un vector di interi secondo l'algoritmo di bubble sort. Questa funzione è necessaria dopo aver chiamato ricercaIdPersona() per ordinare il vettore contenente gli indici trovati, in maniera da mostrare la tabella ordinata secondo l'ordinamento originale quando viene chiamato mostra_per_IDAuto()

**10) void cancellaRimborso(std::vector<datiRimborso>& listaDati, const int idDaEliminare, bool &modDaSalvare)**

la funzione cancellaRimborso parte dalla cella di indice idDaEliminare nel vettore listaDati e compatta il vettore di 1, eliminando quella cella. Infine, per diminuire di 1 la grandezza del vettore, chiama la funzione pop_back() che rimuove l'ultima cella (alla fine del compattamento, le ultime due celle del vettore saranno uguali, per questo possiamo eliminare l'ultima). Il valore della variabile bool modDaSalvare passata byReference viene inoltre impostato a true. Questa variabile permetterà al programma di avvisare l'utente che ci sono delle modifiche non salvate nel file .csv.

**11) void mostraDati(std::vector<datiRimborso> listaDati)**

La funzione mostraRimborsi stampa a video, in una tabella, tutti i dati contenuti nel vettore dei rimborsi (mostrando quindi a video tutti i rimborsi ordinati).

**12) void salvaDati(std::vector<datiRimborso> &listaDati, std::fstream &fileDati)**

La funzione salvaDati sovrascrive il file puntato dalla fstream fileDati (che è il file .csv contenente i dati dei rimborsi) con i dati del vettore listaDati. Con questa funzione l'utente può salvare le modifiche apportate durante la sessione nel file database. L'opzione di effettuare il salvataggio nel menu è affiancata dalla scritta [RILEVATE MODIFICHE NON SALVATE] nel caso in cui siano state effettuate modifiche nella sessione attuale.

**13) void mostra_per_IDAuto(std::vector<datiRimborso>& listaDati,const int pos_ID)**

La funzione mostra_per_IDAuto mostra a video una colonna a una riga contenente le informazioni relative al rimborso di indice pos_ID. Questo indice viene trovato dalla funzione ricercaIdAuto() che viene sempre chiamata prima di queta funzione.

**14) void mostra_per_IDPersona(std::vector<datiRimborso>& listaDati, std::vector<int> &IdPersonaTrovati)**

La funzione mostra_per_IDPersona mostra a video tanti rimborsi quanti sono gli indici contenuti nel vettore IdPersonaTrovati. L'ordinamento con cui li stampa è lo stesso del vettore originale.

Tutte le funzioni sono dichiarate in un file .cpp e in un file .h (chiamati rispettivamente "funzioniRimborso.cpp e funzioniRimborso.h"

**15) int stringToInt(std::string)**

la funzione stringToInt permette di trasformare una stringa in un intero, e viene utilizzata nel caricamento dei dati dal file .csv

**16) double stringToDouble(std::string)**

la funzione stringToInt permette di trasformare una stringa in un valore double (con virgola), e viene utilizzata nel caricamento dei dati dal file .csv