

Report progetto “Car_Sharing”

i. Guida Utente – Sistema di Car Sharing

Questa sezione fornisce una panoramica delle funzionalità offerte dal sistema di car sharing e di come gli utenti e gli amministratori possono interagire con esso.

Avvio del Programma

1. Aprire il terminale nella cartella del progetto.
2. Compilare con il comando:

```
make
```

3. Eseguire con:


```
./car_sharing
```

Accesso e Registrazione

- Alla prima esecuzione viene generato automaticamente un utente amministratore:
 - **Username:** amministratore
 - **Password:** amministratore
 - Gli altri utenti possono registrarsi inserendo:
 - Nome completo
 - Nome utente (unico)
 - Password
 - L'accesso avviene tramite il menu di login iniziale.
-

Funzionalità per l'Utente

Dopo l'accesso come utente, sono disponibili le seguenti opzioni:

- **Prenotazione veicolo:** selezione del mezzo, giorno/ora di inizio e fine.
 *Nota: la prenotazione non aggiorna il calendario immediatamente, ma richiede conferma da parte dell'amministratore.*
 - **Visualizzazione prenotazioni:** elenco e dettagli delle prenotazioni effettuate.
 - **Disponibilità veicoli:** calendario settimanale con visualizzazione oraria.
 - **Consultazione tariffe:** elenco dei costi orari per ciascun tipo di veicolo.
 - **Promozioni attive:** sconti disponibili (es. 10% dopo 10 noleggi, 1 ora gratis ogni 5 ore).
-

Funzionalità per l'Amministratore

L'utente admin ha accesso a funzionalità estese:

- **Gestione veicoli:** aggiunta, rimozione e stampa dei veicoli.
 - **Gestione prenotazioni:** approvazione o rifiuto, aggiornamento del calendario settimanale.
 - **Gestione utenti:** visualizzazione e controllo dell'elenco utenti registrati.
 - **Avanzamento tempo di sistema:** simulazione del passaggio del tempo (per testare scadenze e priorità).
 - **Monitoraggio disponibilità e tariffe.**
-

Note sul Comportamento del Sistema

- Le prenotazioni vengono inserite in coda ma **non impattano la disponibilità** finché **non vengono accettate dall'amministratore**.
 - Il **calendario** viene aggiornato solo dopo l'approvazione.
 - Le tariffe vengono calcolate in base a:
 - Tipo veicolo
 - Durata in ore
 - Sconti attivi (fedeltà, pacchetti orari)
 - Tutti i dati vengono **salvati automaticamente** in file all'interno della cartella data/.
-

Persistenza dei Dati

- **Utenti:** data/utenti.txt
- **Prenotazioni:** data/prenotazioni.txt
- **Veicoli:** data/veicoli.txt

Alla chiusura del programma, il salvataggio avviene automaticamente.

Conclusione

La guida utente ha lo scopo di accompagnare l'utilizzatore nell'uso pratico del sistema. È pensata anche per il docente, al fine di facilitare la valutazione delle funzionalità interattive del progetto.

ii. Motivazione della scelta dell'ADT

Veicoli:

Per la gestione dei veicoli nel sistema di car sharing, è stato scelto un **ADT Lista concatenata** (**list**) come struttura dati di base. Questa scelta è stata motivata da:

- **Flessibilità nella dimensione:** il numero di veicoli può crescere o diminuire dinamicamente.
- **Efficienza nell'inserimento e rimozione:** operazioni che non richiedono spostamenti di elementi, come in un array.
- **Semplicità di implementazione:** la lista consente un'organizzazione sequenziale intuitiva dei veicoli.

L'ADT **veicoli** incapsula completamente la gestione di questa lista e fornisce un'interfaccia chiara per l'utente.

Prenotazioni:

Per la gestione delle prenotazioni si è scelto di utilizzare una **coda di priorità implementata tramite un heap binario**. Questo ADT è particolarmente adatto perché:

- Le prenotazioni devono essere processate in ordine di **priorità**, che dipende tipicamente dal tempo.
- L'heap binario permette di mantenere efficacemente la struttura ordinata, garantendo operazioni di inserimento (**aggiungi_prenotazione**) e rimozione della prenotazione con priorità più alta (**rimuovi_prenotazione**) con complessità logaritmica $O(\log_{10} n)$.
- L'heap è implementato tramite un array dinamico, il che rende semplice il ridimensionamento quando il numero di prenotazioni cresce, mantenendo un uso efficiente della memoria.
- Grazie alla struttura ad heap, è possibile aggiornare le priorità delle prenotazioni e mantenere la coda consistente con l'algoritmo di "bubble up" e "bubble down", assicurando che la prenotazione con priorità minima (cioè più urgente) sia sempre in cima.

Fasce_orarie:

Il sistema utilizza un **calendario settimanale** rappresentato da una matrice 2D **calendario[7][24]** (giorni × ore) contenente elementi di tipo **FasciaOraria**. Ogni cella rappresenta una specifica ora di un giorno della settimana per un veicolo.

Motivazioni principali:

- **Accesso diretto e costante:** La struttura a matrice consente l'accesso diretto alle informazioni su disponibilità in tempo costante $O(1)$.

- **Semplicità di visualizzazione:** La matrice si presta bene a essere visualizzata come una tabella leggibile.
 - **Facilità di aggiornamento e Verifica:** Il sistema può aggiornare lo stato di occupazione con semplici cicli for, rendendo intuitiva la gestione delle fasce orarie.
-

Tariffe:

Il sistema non fa uso di strutture dati complesse ma adotta:

- **Costanti macro** per definire le tariffe e i parametri di sconto.
- **Funzioni modulari** per il calcolo delle tariffe, sconti, e gestione delle ore.

Motivazioni principali:

- **Semplicità ed efficienza:** L'uso di valori costanti e semplici funzioni garantisce leggibilità e rapidità di esecuzione.
 - **Estendibilità:** È facile aggiungere nuovi tipi di veicolo o modificare le regole di sconto.
 - **Chiarezza:** Ogni funzione ha una responsabilità ben definita (calcolo ore, sconto fedeltà, stampa info ecc.).
-

Utenti:

Il sistema utilizza una **tabella hash** (`tabellaUtenti[TABLE_SIZE]`) per gestire gli utenti registrati.

Motivazioni principali:

- **Efficienza di accesso:** La ricerca, inserimento e accesso agli utenti avviene in tempo costante medio $O(1)$, grazie alla funzione di hash.
- **Gestione di collisioni:** La risoluzione avviene tramite **sondaggio lineare**, che evita sovrascritture.
- **Scalabilità:** La tabella può gestire fino a cento utenti (dimensione predefinita), sufficiente per contesti medi.

Hash:

Il sistema utilizza la funzione di hash **DJB2**, un algoritmo di hashing creato da Daniel J. Bernstein, noto per la sua **semplicità, efficienza e buona distribuzione**.

Motivazioni principali:

- **Efficiente:** Calcolo rapido grazie a operazioni bitwise e aritmetica semplice.
- **Buona distribuzione:** Minimizza le collisioni su insiemi di stringhe reali, rendendolo adatto a tabelle hash.
- **Diffusamente testato:** È uno degli algoritmi di hashing più utilizzati in contesti non crittografici (es. strutture dati).

Menu:

Questo modulo non utilizza una struttura dati complessa ma si basa su:

- **Funzioni modulari e interattive** per la gestione dell'interfaccia utente testuale.
- **Selezione tramite menu e switch-case** per la navigazione delle funzionalità.
- **Interazione strutturata** con altri moduli (prenotazioni, utenti, veicoli, tariffe).

Motivazioni principali:

- **Modularità:** Ogni funzionalità è incapsulata in una funzione chiara e indipendente.
- **Separazione delle responsabilità:** Il modulo gestisce esclusivamente l'interfaccia e la logica di navigazione.
- **Accessibilità:** L'interfaccia è pensata per un ambiente console, garantendo semplicità d'uso

Data_sistema:

Il sistema utilizza una struttura dati semplice, **DataSistema**, per rappresentare lo stato temporale del sistema (giorno e ora).

Motivazioni principali:

- **Semplicità:** Basta una struttura con due interi per rappresentare il tempo in modo settimanale.
- **Controllo centralizzato del tempo:** Le prenotazioni e altri moduli fanno riferimento a un'unica "data di sistema", utile per simulazioni e test.
- **Funzioni di utilità pronte all'uso:** Conversioni da e verso timestamp, gestione automatica dell'avanzamento del tempo e delle priorità temporali.

F_utili:

Il modulo non adotta un ADT formale ma fornisce un insieme di **funzioni d'utilità per:**

- L'interfaccia grafica a terminale (colori, separatori, bordi).
- La gestione cross-platform di input/output.
- La **pulizia e il salvataggio** del sistema prima della chiusura.

Motivazioni principali:

- **Riutilizzabilità:** Le funzioni possono essere invocate da qualsiasi altro modulo.
- **Portabilità:** Supporto sia per Windows che per Unix/Linux grazie a preprocessor directives.
- **Chiarezza visiva:** L'output è migliorato da bordi, separatori e colori.

Main:

Il file `main.c` rappresenta il **punto d'ingresso** del sistema, responsabile della **gestione del ciclo di vita dell'applicazione** di car sharing.

Motivazioni principali:

- **Modularità e orchestrazione:** Centralizza l'inizializzazione dei sottosistemi (utenti, veicoli, prenotazioni).

- **Gestione flessibile utenti/admin:** Permette accesso differenziato alle funzionalità tramite menu.
 - **Ciclo interattivo continuo:** Il programma rimane attivo fino a scelta esplicita di uscita da parte dell'utente.
 - **Simulazione realistica del tempo:** Include un sistema di avanzamento temporale che influenza la gestione delle prenotazioni.
-

iii. Come usare e interagire con il sistema

Veicoli:

L'interazione con il modulo `veicoli` avviene tramite le sue **funzioni**:

- `aggiungi_veicolo(...)`: permette di inserire un nuovo veicolo richiedendo dati da tastiera.
- `rimuovi_veicolo(...)`: consente la cancellazione di un veicolo esistente tramite ID.
- `stampa_veicolo(...)`: stampa le informazioni di un veicolo specifico.
- `carica_lista_veicoli()` / `salva_lista_veicoli()`: gestiscono il caricamento/salvataggio dei veicoli da/verso file `data/veicoli.txt`.
- `crea_veicolo()`: gestisce l'inserimento interattivo di un nuovo veicolo.
- `ottieni_lista_veicoli()` / `imposta_lista_veicoli()`: per accedere e aggiornare la lista globale di veicoli.

Queste funzioni costituiscono l'interfaccia pubblica dell'ADT e permettono di usare il modulo senza conoscere l'implementazione interna.

Prenotazioni:

L'interazione con il sistema si realizza attraverso funzioni definite nei file sorgente. Le principali funzionalità includono:

Creazione e Inserimento

- `inizializza_coda()`: inizializza la struttura dati.
- `crea_prenotazione(...)`: crea una nuova prenotazione.
- `aggiungi_prenotazione(...)`: inserisce una prenotazione nella coda secondo la sua priorità.

Modifica e Controllo

- `modifica_stato_prenotazione(...)`: modifica lo stato di una prenotazione.
- `valida_data_prenotazione(...)` e `verifica_sovrapposizioni(...)`: garantiscono la coerenza temporale e l'assenza di conflitti.

Ricerca

- `cerca_prenotazione(...)`: ricerca per ID della Prenotazione.

Gestione e Pulizia

- `aggiorna_priorita_prenotazioni(...)`: ricalcola le priorità secondo la data di sistema.
- `rimuovi_prenotazioni_scadute(...)`: marca automaticamente le prenotazioni scadute come completate.

Persistenza

- `salva_prenotazioni_su_file(...)`: salva prenotazioni sul file `data/prenotazioni.txt`.
- `carica_prenotazioni(...)`: carica da file.

Fasce_orarie:

Il modulo fornisce funzionalità per gestire la disponibilità oraria di ciascun veicolo.

Funzioni principali

- `inizializza_calendario(...)`
Inizializza il calendario settimanale del veicolo, marcando tutte le ore come libere.
 - `aggiorna_calendario(...)`
Popola il calendario a partire dalla coda delle prenotazioni. Tiene conto anche delle prenotazioni che attraversano più giorni.
 - `visualizza_calendario(...)`
Mostra in console la tabella con lo stato di ogni ora di ogni giorno per un determinato veicolo. Le ore occupate sono indicate con una "X".
 - `verifica_disponibilita(...)`
Verifica se un veicolo è disponibile per una data fascia oraria, anche su più giorni consecutivi.
-

Tariffe:

Il modulo è progettato per essere chiamato da altre componenti del programma (es. prenotazioni) per calcolare le tariffe.

Funzioni principali

- `ottieni_tariffa_oraria(...)`
Restituisce la tariffa oraria in base al tipo di veicolo.
 - `calcola_tariffa(...)`
Calcola la tariffa tenendo conto di pacchetti di ore gratuite.
 - `calcola_tariffa_prenotazione(...)`
Calcola la tariffa totale partendo da giorno e ora di inizio/fine.
 - `applica_sconto_fedelta(...)`
Applica uno sconto del 10% dopo 10 noleggi.
 - `stampa_info_sconti()`
Mostra all'utente le informazioni sulle promozioni attive.
-

Utenti:

- **Inizializzazione e Persistenza**

- `inizializza_tabella_utenti()`
Inizializza la tabella e, se necessario, crea automaticamente l'utente admin.
 - `salva_utenti_file()`
Salva tutti gli utenti correnti nel file `data/utenti.txt`.
 - `carica_utenti_file()`
Carica gli utenti dal file di testo all'avvio del sistema.
 - **Gestione Utenti**
 - `inserisci_utente(...)`
Inserisce un nuovo utente nella tabella hash, assegnando un ID univoco.
 - `cerca_utente(...)`
Ricerca utente tramite nome utente.
 - `cerca_utente_per_id(...)`
Ricerca utente tramite ID.
 - `stampa_utenti()`
Stampa tutti gli utenti presenti nella tabella, incluso l'admin.
 - **Sicurezza e Hashing**
 - Integrazione di un sistema di **hashing delle password** (`hash_password`), per maggiore sicurezza.
-

Hash:

Firma della funzione

```
unsigned int hash_djb2(const char* str);
```

- **Parametro:** una stringa (`char*`), tipicamente un nome utente.
- **Valore di ritorno:** un intero `unsigned int` che rappresenta l'hash della stringa.

Contesto d'uso

- **Gestione utenti:** utilizzato per calcolare l'indice nella tabella hash degli utenti (`tabellaUtenti`).
- Permette inserimenti e ricerche rapide in strutture indicizzate.

Menu:

Questo modulo gestisce:

- **Accesso e registrazione**
- **Navigazione menu utente e admin**
- **Prenotazioni veicoli**
- **Visualizzazione e gestione prenotazioni**
- **Monitoraggio tariffe e disponibilità**

Principali funzionalità offerte

- `mostra_menu_login()`
Mostra il menu iniziale con opzioni di login/registrazione.
- `mostra_menu_cliente()`
Menu specifico per utenti normali: prenota, visualizza, tariffe, disponibilità, logout.
- `mostra_menu_admin()`
Pannello completo per l'amministrazione del sistema (veicoli, utenti, prenotazioni).
- `gestione_veicoli()`
Permette ad admin di aggiungere/rimuovere/visualizzare veicoli.
- `prenota_auto()`
Sistema guidato per inserire una prenotazione con verifica, calcolo costi e conferma.
- `gestione_prenotazioni_admin()`
Interfaccia per amministratori per filtrare e modificare prenotazioni in base a diversi criteri.
- `visualizza_disponibilita()`
Mostra disponibilità attuale dei veicoli tramite aggiornamento dinamico del calendario settimanale.
- `visualizza_tariffe()`
Mostra le tariffe correnti e gli sconti applicabili per tipo di veicolo.

Data_sistema:

Funzioni principali

- `inizializza_data_sistema()`
Imposta la data iniziale del sistema a Lunedì ore 08:00.
- `avanza_tempo(...)`
Avanza il tempo simulato di un certo numero di ore, con gestione automatica di ore e giorni.
- `ottieni_data_sistema()`
Restituisce la data corrente del sistema.
- `converti_data_in_timestamp(...)`
Converte giorno/ora in un unico intero ($\text{giorno} * 24 + \text{ora}$).
- `converti_timestamp_in_data(...)`
Operazione inversa: da intero a giorno/ora.
- `calcola_priorita_temporale(...)`
Restituisce la priorità di una prenotazione in base alla sua distanza temporale dalla data corrente.
- `ottieni_nome_giorno(...)`
Restituisce il nome del giorno della settimana come stringa.

F_utili:

Funzioni per l'UI console

- `imposta_color(...)`
Cambia il colore del testo in console (ANSI o Windows).
- `stampa_separatore()`
Stampa una riga di separazione tra sezioni.
- `stampa_bordo_superiore()` / `stampa_bordo_inferiore()`
Delimitano visivamente un blocco di output.

Funzioni di sistema

- `svuota_buffer()`
Pulisce il buffer `stdin` per evitare problemi con `scanf`.

- `pulisci_schermo()`
Cancella il contenuto del terminale (`cls` su Windows, `clear` su Unix).
 - `salvataggio()`
Esegue le operazioni finali:
 - Salva lista veicoli e prenotazioni.
 - Libera memoria.
-

Main:

Fasi principali del programma

1. Inizializzazione

- `inizializza_data_sistema()`
- `carica_lista_veicoli()`
- `inizializza_tabella_utenti()` e `carica_utenti_file()`
- `carica_prenotazioni()` + aggiornamento automatico delle prenotazioni scadute

2. Accesso

- `mostra_menu_login()` → Login o Registrazione
- `valida_nome_utente()`, `valida_nome_completo()`,
`cerca_utente()`, `inserisci_utente()`

3. Navigazione

- **Admin:** `mostra_menu_admin()`, `gestione_veicoli()`,
`gestione_prenotazioni_admin()`, `visualizza_disponibilita()`
- **Cliente:** `mostra_menu_cliente()`, `prenota_auto()`,
`visualizza_prenotazioni()`, `visualizza_tariffe()`

4. Gestione dello stato

- Il programma tiene traccia dell'utente corrente tramite `utente_corrente` e usa `stato` per distinguere login attivo/inattivo.

5. Chiusura

- `salvataggio()` salva tutti i dati prima di uscire.

iv. Progettazione

Veicoli:

- **Struttura `veicolo`**: contiene ID, tipo del veicolo (utilitaria, SUV, sportiva, moto), modello, targa, posizione e disponibilità.
- **Lista concatenata `list`**: gestita tramite nodi che contengono un `veicolo` e un puntatore al nodo successivo.
- **Persistenza**: i veicoli vengono salvati su file testuale (`veicoli.txt`) e possono essere caricati all'avvio.
- **Gestione ID**: ID veicolo generato automaticamente tramite `carica_ultimo_id()` che legge il massimo ID dal file.
- **Interfaccia colorata**: la stampa dei veicoli usa `imposta_color()` per colorare in base al tipo/stato.

Prenotazioni:

- **Struttura**
 - Contiene l'ID della prenotazione, dell'utente e del veicolo, il giorno, con annessa ora, di inizio e fine della prenotazione, lo stato della prenotazione, e la sua priorità.
- **Gerarchia dell'Heap**
 - Heap binario con ordinamento basato su `priorita`.
 - Nodo padre ha sempre priorità \leq dei figli.
- **Gestione del Tempo**
 - Gli orari sono convertiti in un singolo `timestamp` (giorno * 24 + ora).

- Le funzioni `converti_in_timestamp`, `estrai_giorno`, `estrai_ora` semplificano i calcoli temporali.
 - **Estendibilità**
 - Sistema predisposto per l'integrazione con moduli esterni (`data_sistema.h`) che forniscono data/ora corrente.
 - La funzione `calcola_priorita_temporale` può essere ridefinita per implementare logiche avanzate di gestione delle urgenze.
 - **Colori e UI Console**
 - La funzione `stampa_prenotazione` usa colori per evidenziare visivamente lo stato delle prenotazioni nella console.
-

Fasce_orarie:

- **Strutture:** Una struttura per rappresentare una fascia oraria che contiene lo stato della macchina (libera o occupata) e l'ID della prenotazione che occupa la fascia occupata.
Una struttura per rappresentare il calendario di un veicolo che contiene l'ID del veicolo in questione ed una matrice 2D `calendario[7][24]` (giorni × ore) contenente elementi di tipo `FasciaOraria`.
 - **Occupazione multigiorno:** Il codice gestisce prenotazioni che iniziano e terminano in giorni diversi, aggiornando coerentemente le celle coinvolte.
 - **Isolamento logico per veicolo:** Ogni `CalendarioVeicolo` è associato a un singolo `id_veicolo`, rendendo facile la gestione indipendente della disponibilità.
 - **Separazione delle responsabilità:** Il calendario non contiene direttamente le prenotazioni ma solo riferimenti indiretti (tramite `id_prenotazione`), garantendo basso accoppiamento con il modulo prenotazioni.
 - **UI Console integrata:** La funzione `visualizza_calendario` fornisce una rappresentazione leggibile e pronta per l'interfaccia utente da terminale.
-

Tariffe:

- **Definizioni e costanti:** Si definiscono 4 costanti (`TARIFFA_UTILITARIA`, `TARIFFA_SUV`, `TARIFFA_SPORTIVA`, `TARIFFA_MOTO`), una per ogni veicolo, per indicare il loro prezzo orario, ed altre 3 costanti (`SCONTO_FEDELTA`,

`NOLEGGI_PER_SCONTO, ORE_PER_PACCHETTO`) per l'implementazione degli sconti.

- **Struttura:** Struttura per memorizzare le informazioni sulla tariffa che contiene il tipo del veicolo e la tariffa oraria.
 - **Strategie di Calcolo**
 - **Calcolo ore totali:** Basato su differenza di giorni e ore.
 - **Sconto pacchetto ore:** 1 ora gratuita ogni 5.
 - **Sconto fedeltà:** Applicato solo se si superano dieci noleggi.
 - **Gestione di casi speciali**
 - La funzione `calcola_tariffa_prenotazione` gestisce correttamente le prenotazioni su più giorni.
 - In caso di tipo veicolo non valido, `ottieni_tariffa_oraria` ritorna `0.0`.
-

Utenti:

- **Struttura dati utente:** Contiene l'ID dell'utente, l'nome_utente, il nome completo, la password, ed un campo `isamministratore` per l'accesso nella modalità da admin.
- **Struttura di archiviazione:** `static Utente* tabellaUtenti[TABLE_SIZE];`
- **Hashing**
 - Utilizza una funzione `hash_djb2` (in `hash.h`) per calcolare l'indice in tabella.
- **Persistenza File**
 - Gli utenti sono memorizzati su disco in formato testuale.

- Ogni riga contiene: `id nome_utente nome_completo isamministratore`.
 - **Gestione Admin**
 - L'utente admin viene creato automaticamente se non esiste, con ID = 0 e privilegi `isamministratore = 1`.
-

Hash:

- **Algoritmo**

```
unsigned int hash_djb2(const char* str) {
    unsigned int hash = 5381;
    int c;

    while ((c = *str++))
        hash = ((hash << 5) + hash) + c; // hash * 33 + c

    return hash;
}
```
 - La costante iniziale `5381` è empiricamente scelta.
 - L'operazione `((hash << 5) + hash)` equivale a `hash * 33`, scelta per le sue buone proprietà matematiche.
 - L'algoritmo accumula ogni carattere nella stringa, producendo un hash finale.
-

Menu:

- **Modularità**

Le funzioni sono suddivise in blocchi tematici:

- Menu di sistema: `mostra_menu_login`, `mostra_menu_cliente`, `mostra_menu_admin`
- Funzioni operative: `prenota_auto`, `gestione_veicoli`, `gestione_prenotazioni_admin`
- Supporto: `visualizza_tariffe`, `visualizza_disponibilita`, `restituisce_auto`

- **Validazione input**

Sono presenti funzioni per controllare la correttezza:

```
int valida_nome_utente(const char* nome_utente);  
int valida_nome_completo(const char* nome);
```

- **Integrazione**

Il modulo fa ampio uso di funzioni importate:

- `stampa_prenotazione`, `salva_prenotazioni_su_file`, `ottieni_lista_veicoli`, ecc.
- L'accesso alle strutture `Utente`, `Prenotazione`, `Veicolo` è ben incapsulato.

Data_sistema:

- **Struttura dati:** contiene il giorno, e l'ora.
- **Timestamp**
 - Rappresentato come `giorno * 24 + ora`, consente calcoli temporali semplificati e confronti diretti.
- **Gestione del tempo**
 - L'avanzamento tiene conto dei cicli settimanali: dopo Domenica 23:00 si torna a Lunedì 00:00.
 - Usato per determinare la validità di una prenotazione e la sua **priorità**: più è vicina, più è urgente.
- **Contesto d'uso**
 - Il sistema funge da **orologio globale simulato**, utile per scenari testabili senza dipendere dall'orario reale del sistema operativo.

F_utili:

- **Portabilità**

```
#ifdef _WIN32  
    // include <windows.h>
```

```
#else
    // include <unistd.h>, usa ANSI escape sequences
#endif
```

Garantisce compatibilità tra sistemi operativi differenti.

- **Colori supportati**

Codice	Colore
7	Bianco
10	Verde
11	Ciano
12	Rosso
13	Magenta
14	Giallo

Modularità

Il modulo è completamente isolato e non contiene logica di business, rendendolo facilmente manutenibile e sostituibile.

Main:

Robustezza

- **Validazione input:** ogni input è validato prima dell'uso.
- **Fallback automatici:** se `utente_corrente` è NULL, il menu torna a login.
- **Pulizia risorse:** con `salvataggio()` alla chiusura.

Interfaccia utente

- Fortemente legata a console: colori, bordi, messaggi testuali.
 - Esperienza utente migliorata con `stampa_bordo_superiore()`, `imposta_colore()` e gestione pulita del terminale (`pulisci_schermo()`).
-

v. Specifica Sintattica e Semantica

Veicoli:

Specifica Sintattica

- `ottieni_lista_veicoli(void) → list`
- `imposta_lista_veicoli(list nuovaLista) → void`
- `salva_lista_veicoli(void) → void`
- `carica_lista_veicoli(void) → void`
- `pulisci_lista_veicoli(void) → void`

- `crea_veicolo(void) → veicolo`
- `aggiungi_veicolo(list) → list`
- `rimuovi_veicolo(list, int) → list`
- `stampa_veicolo(Veicolo) → void`
- `salva_veicolo_file(list) → void`
- `carica_veicolo_file(list) → list`
- `carica_ultimo_id(void) → int`

- `ottieni_id_veicolo(Veicolo) → int`
- `ottieni_tipo_veicolo(Veicolo) → int`
- `ottieni_modello_veicolo(Veicolo) → const char*`
- `ottieni_posizione_veicolo(Veicolo) → const char*`
- `ottieni_disponibilita_veicolo(Veicolo) → int`
- `ottieni_veicolo_senza_rimuovere(list) → Veicolo`
- `ottieni_successivo_nodo(list) → list`
- `ottieni_nome_tipo_veicolo(int) → const char*`

- `imposta_id_veicolo(Veicolo, int) → void`
- `imposta_tipo_veicolo(Veicolo v, int) → void`
- `imposta_modello_veicolo(Veicolo, const char*) → void`
- `imposta_targa_veicolo(Veicolo, const char*) → void`
- `imposta_posizione_veicolo(Veicolo, int) → void`
- `imposta_disponibilita_veicolo(Veicolo, int) → void`

- `cerca_veicolo(list, int) → Veicolo`

- `stampa_lista_veicoli(list) → void`

Specifica Semantica

- `ottieni_lista_veicoli(void) → listaVeicoli`

La funzione **restituisce** il valore corrente della variabile `listaVeicoli`.

Precondizioni:

- La variabile `listaVeicoli` deve essere inizializzata correttamente prima della chiamata alla funzione.

Postcondizioni:

- La funzione restituisce un riferimento alla lista dei veicoli attualmente memorizzata nel sistema.
- Nessuna modifica viene effettuata su `listaVeicoli` o su altri dati del programma.

Side Effect: Nessuno.

- `imposta_lista_veicoli(nuovaLista) -> void`

La funzione **aggiorna il contenuto della variabile** `listaVeicoli` assegnandole il valore passato tramite il parametro `nuovaLista`.

Precondizioni:

- Il parametro `nuovaLista` deve essere un valore valido per il tipo `list`.
- Se `listaVeicoli` conteneva già una lista precedente allocata dinamicamente, è responsabilità del **chiamante** liberare la memoria associata prima della sovrascrittura (se necessario).

Postcondizioni:

- La variabile `listaVeicoli` assume il valore `nuovaLista`.
- Eventuali riferimenti precedenti a `listaVeicoli` non sono più validi, a meno che non siano stati gestiti.

Side Effect: Modifica lo stato globale: aggiorna la variabile `listaVeicoli`.

- `salva_lista_veicoli(void) -> void`

La funzione **richiama** `salva_veicolo_file(listaVeicoli)` per salvare su file la lista dei veicoli mantenuta nella variabile globale `listaVeicoli`.

Precondizioni:

- `listaVeicoli` deve essere correttamente inizializzata. (Può essere anche `NULL`, se `salva_veicolo_file` lo gestisce come lista vuota.)
- La funzione `salva_veicolo_file(list)` deve essere implementata correttamente e sapere come serializzare la struttura `list`.

Postcondizioni:

- I dati contenuti in `listaVeicoli` vengono scritti su un file, in un formato stabilito da `salva_veicolo_file`.

Side effect: Sovrascrive il file `data/veicoli.txt`

- `carica_lista_veicoli(void) -> void`

La funzione carica la lista dei veicoli dalla memoria di massa

Precondizioni:

- La funzione `carica_veicolo_file(list)` deve essere correttamente implementata:
 - deve leggere i dati dal file previsto,
 - restituire una nuova lista coerente.
- Il file deve essere accessibile in lettura e contenere dati ben formati.
- Se `carica_veicolo_file()` non libera la memoria della vecchia lista, allora è compito del chiamante farlo prima dell'assegnazione.

Postcondizioni:

- La variabile globale `listaVeicoli` punta a una nuova lista allocata dinamicamente e popolata con i dati letti dal file.
- La lista precedente viene eventualmente persa se non gestita.

Side Effect:

- Sovrascrive la lista globale dei veicoli.

- `pulisci_lista_veicoli(void) -> void`

La funzione libera uno ad uno tutti i nodi della lista dei veicoli puntata dalla variabile globale `listaVeicoli`, evitando perdite di memoria. Alla fine, `listaVeicoli` sarà `NULL`.

Precondizioni:

- La variabile globale `listaVeicoli` deve essere inizializzata (può anche essere `NULL`).
- I nodi della lista sono stati allocati dinamicamente con `malloc` o funzioni equivalenti.
- La struttura dei nodi deve essere coerente (ogni nodo punta al successivo con il campo `next`).

Postcondizioni:

- Tutta la memoria dinamica associata a `listaVeicoli` viene liberata.
- La variabile `listaVeicoli` risulta vuota, ovvero `NULL`.

Side Effect: La lista globale viene azzerata e la memoria liberata.

- `crea_veicolo(void) -> v`

La funzione:

1. Genera un ID univoco per il veicolo.
2. Chiede all'utente di inserire tipo, modello e targa.
3. Restituisce la struttura `veicolo` compilata.

Precondizioni:

- La struttura `veicolo` è definita correttamente e contiene:
 - `int id, int tipo, char modello[30], char targa[8], int disponibile`.
- La funzione `carica_ultimo_id()` è disponibile e restituisce l'ultimo ID usato.
- Sono definiti i tipi di veicolo: `UTILITARIA`, `SUV`, `SPORTIVA`, `MOTO`.
- Sono definite le costanti `TARIFFA_UTILITARIA`, ecc.
- L'ambiente di esecuzione consente input/output via terminale (`scanf`, `printf`, `fgets`).

Postcondizioni:

- Restituisce un oggetto `veicolo` correttamente inizializzato.
- L'ID del veicolo è univoco e crescente. Il veicolo ha tipo, modello, targa e posizione correttamente assegnati.

Side Effect: Modifica di una variabile static `int id` interna, che mantiene stato tra chiamate successive.

- `aggiungi_veicolo(l) -> l`

La funzione:

1. Alloca un nuovo nodo.
2. Crea un nuovo veicolo tramite `crea_veicolo()`.
3. Inserisce il nuovo nodo in testa alla lista `l`.
4. Restituisce il puntatore alla nuova lista.

Precondizioni: Il parametro `l` può essere `NULL` o una lista valida di veicoli. La struttura `struct node` deve essere definita correttamente, con almeno:

- un campo `veicolo veicoli`;
- un campo `struct node* next`;

La funzione `crea_veicolo()` deve essere disponibile e funzionante.

Postcondizioni: Restituisce un nuovo puntatore alla lista, con il nuovo nodo in testa. Se `malloc` fallisce, la lista originale viene restituita invariata.

Side Effects: Nessuno.

- `rimuovi_veicolo(l, id) -> l`

La funzione:

1. Richiede all'utente l'ID di un veicolo da rimuovere.
2. Cerca il nodo corrispondente nella lista `l`.
3. Se lo trova, lo rimuove e libera la memoria.
4. Restituisce la nuova lista aggiornata.

Precondizioni:

- La lista `l` è valida (può essere anche `NULL`).
- Ogni nodo della lista contiene un campo `veicolo veicoli` con almeno `int id`.
- La lista è ben formata (non contiene cicli, puntatori invalidi, ecc.).
- Input da tastiera disponibile per leggere l'ID.

Postcondizioni:

- Se esiste un veicolo con l'ID fornito, viene rimosso e la memoria associata viene liberata.
- Se non esiste, la lista resta invariata e viene mostrato un messaggio di errore.
- La lista restituita è valida e coerente.

Side Effect:

- Libera memoria con `free()` se trova l'elemento.
- `stampa_veicolo(v) -> void`

La funzione stampa in output:

- L'ID del veicolo
- Il modello
- La targa
- Il tipo del veicolo (con colore associato)
- Lo stato di disponibilità (con colore associato)

Precondizioni: Il parametro `v` è un veicolo valido, ovvero:

`id` è un intero assegnato. `modello` e `targa` sono stringhe ben terminate.

`tipo` è uno dei valori previsti (`UTILITARIA`, `SUV`, `SPORTIVA`, `MOTO`).

La funzione `imposta_color(int)` è disponibile e funzionante.

Postcondizioni: Le informazioni del veicolo sono stampate a schermo in formato leggibile e con evidenziazione a colori. Lo stato del programma non viene modificato.

Side Effect: Output su console.

- `salva_veicolo_file(l) -> void`

La funzione salva i dati dei veicoli contenuti nella lista `l` nel file

`"data/veicoli.txt"` in modalità sovrascrittura ("`w`"). Ogni veicolo viene scritto su una riga con i seguenti campi:

id tipo modello targa posizione disponibile

Precondizioni: `l` può essere `NULL` (in tal caso, il file viene comunque aperto e poi chiuso vuoto).

La directory `data/` esiste e l'applicazione ha i permessi di scrittura al file `veicoli.txt`.

I campi `modello`, `targa`, `posizione` di ogni veicolo sono stringhe ben terminate.

Postcondizioni:

- I dati della lista `l` sono scritti nel file `data/veicoli.txt`, sovrascrivendo eventuali contenuti precedenti.
- Il file è chiuso correttamente al termine.
- Un messaggio di conferma viene stampato su console in caso di successo.

Side Effect:

- Scrittura su file: `data/veicoli.txt`
- Apertura e chiusura file;
- Output su console con `printf`

- `carica_veicolo_file(list l) → l`

La funzione apre il file `data/veicoli.txt`, legge ogni riga rappresentante un veicolo, ricostruisce i dati e li inserisce in testa alla lista `l`. La lista risultante viene restituita.

Precondizioni: Il file `data/veicoli.txt` **deve esistere** e contenere righe ben formate con il seguente ordine:

id tipo modello targa posizione disponibile

Il campo **modello** può contenere spazi, ma viene letto correttamente finché la **targa** ha esattamente sette caratteri e inizia con una lettera.

La lista `l` può essere `NULL` (in questo caso la funzione crea una nuova lista da zero).

Postcondizioni: Il contenuto del file viene caricato in memoria come lista di veicoli. I nuovi elementi vengono aggiunti in testa alla lista `l`. La lista risultante viene restituita.

Side Effect: Lettura da file: `data/veicoli.txt` Allocazione dinamica per ogni nodo veicolo (`malloc`).

- `int carica_ultimo_id(void)`

La funzione apre il file `data/veicoli.txt`, legge tutte le righe e restituisce il valore massimo del campo `id` trovato nel file. Se il file non esiste o è vuoto, restituisce `0`.

Precondizioni: Il file `data/veicoli.txt`, se esiste, deve contenere righe in cui l'`id` è il primo campo e rappresenta un intero valido.

Ogni riga deve iniziare con un intero (`%d`) che rappresenta l'ID del veicolo.

Postcondizioni: Restituisce l'`id` massimo tra quelli trovati nel file.

Se il file non esiste, viene restituito `0`.

Side Effect: Nessuno.

- `ottieni_id_veicolo(v) → id`

Restituisce l'identificativo del veicolo `v`.

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.

Postcondizioni:

- `id = v->id.`

Side Effect: Nessuno.

- `ottieni_tipo_veicolo(v) → tipo`

Restituisce il tipo del veicolo `v`, ad esempio utilitaria, suv, sportiva, moto. (codificato come intero).

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.

Postcondizioni:

- `tipo = v->tipo.`

Side Effect: Nessuno.

- `ottieni_modello_veicolo(v) → modello`

Restituisce una stringa costante contenente il modello del veicolo `v`.

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.

Postcondizioni:

- `modello = v->modello`.

Side effect: Nessuno.

- `ottieni_posizione_veicolo(v) → posizione`

Restituisce un intero predefinito della posizione attuale del veicolo `v`.

0 = "Deposito"

1 = "Posizione B"

2 = "Posizione C"

3 = "Posizione D"

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.

Postcondizioni:

- `posizione = v->posizione`.

Side Effect: Nessuno.

- `ottieni_disponibilita_veicolo(v) → disponibilita`

Restituisce un intero che indica se il veicolo è disponibile (1) o meno (0).

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.

Postcondizioni:

- `disponibilita = v->disponibilita`.

Side Effect: Nessuno.

- `ottieni_veicolo_senza_rimuovere(l) → v`

La funzione restituisce il veicolo (**Veicolo**) contenuto nel primo nodo della lista **l** **senza rimuoverlo** dalla lista stessa.

Precondizioni:

- La variabile **l** è una lista valida (può anche essere **NULL**).
- Ogni nodo della lista (se presente) contiene un campo **v** di tipo **Veicolo**.

Postcondizioni:

- Se la lista è vuota (**l == NULL**), restituisce **NULL**.
- Se la lista contiene almeno un elemento, restituisce il **Veicolo** del primo nodo senza modificare la lista.

Side Effect: Nessuno.

- `ottieni_successivo_nodo(list) → list`

Restituisce il nodo **successivo** nella lista collegata **l**.

Precondizioni

- Il parametro **l** è un puntatore a un nodo valido della lista, oppure **NULL**.
- Se **l** non è **NULL**, allora **l->next** è accessibile.

Postcondizioni

- Se **l == NULL**, la funzione restituisce **NULL**.
- Altrimenti, restituisce il puntatore al nodo successivo, cioè **l->next**.

Side Effect: Nessuno.

- `imposta_id_veicolo(v, id) → void`

Imposta l'ID del veicolo **v** al valore specificato.

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.
- `id` è un intero non negativo.

Postcondizioni:

- `v->id = id.`

Side Effect: Modifica diretta del campo `id` nella struttura.

- `imposta_tipo_veicolo(v, tipo) → void`

Imposta il tipo del veicolo `v` al valore `tipo` (utilitaria, suv, ecc.).

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.
- `tipo` è un intero valido secondo la codifica dei tipi veicolo.

Postcondizioni:

- `v->tipo = tipo.`

Side Effect: Modifica diretta del campo `tipo` nella struttura.

- `imposta_modello_veicolo(v, modello) → void`

Imposta il campo `modello` del veicolo `v` alla stringa passata come parametro.

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.
- `modello` è una stringa valida (non `NULL`).

Postcondizioni:

- Il campo `v->modello` conterrà una copia sicura di `modello`, troncata se necessario alla dimensione massima consentita.

Side Effect: Scrive su `v->modello`.

- `imposta_targa_veicolo(v, targa) → void`

Imposta il campo `targa` del veicolo `v` alla stringa fornita.

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.
- `targa` è una stringa valida.

Postcondizioni:

- Il campo `v->targa` conterrà una copia sicura di `targa`.

Side Effect: Scrive su `v->targa`.

- `imposta_posizione_veicolo(v, posizione) → void`

Imposta la posizione corrente del veicolo alla stringa specificata.

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.
- `posizione` è una stringa valida.

Postcondizioni:

- Il campo `v->posizione` conterrà una copia sicura di `posizione`, troncata se necessario.

Side Effect: Scrive su `v->posizione`.

- `imposta_disponibilita_veicolo(v, disponibilita) → void`

Imposta il campo `disponibilita` del veicolo `v`.

Precondizioni:

- `v` è un puntatore valido a una struttura `Veicolo`.
- `disponibilita` è un intero valido (es. 0 o 1).

Postcondizioni:

- `v->disponibilita = disponibilita`.

Side Effect: Modifica il campo `disponibilita` della struttura.

- `cerca_veicolo(l, id) → Veicolo`

Cerca nella lista `l` un veicolo con `id` uguale a quello passato come parametro.

Precondizioni:

- `l` è una lista di tipo `list`, eventualmente vuota.
- `id` è un intero valido.

Postcondizioni:

- Se esiste un nodo nella lista con `v->id == id`, viene restituito quel `Veicolo`.
- Altrimenti, viene restituito `NULL`.

Side Effect: Nessuno.

- `void stampa_lista_veicoli(l) → void`

Stampa a video tutti i veicoli contenuti nella lista `l`.

Precondizioni:

- `l` è una lista valida (può anche essere `NULL`).

Postcondizioni:

- Se la lista è vuota, stampa un messaggio corrispondente.
- Se contiene elementi, stampa ciascun veicolo con `stampa_veicolo`.

Side Effect:

- Output su standard output (console).

- `ottieni_nome_tipo_veicolo(tipo) → const char*`

Restituisce una stringa letterale che rappresenta il nome del tipo di veicolo corrispondente all'intero tipo.

Precondizioni:

- `tipo` è un intero.

- Può essere:
 - 0: "Utilitaria"
 - 1: "SUV"
 - 2: "Sportiva"
 - 3: "Moto"
 - qualsiasi altro valore produce "Tipo non valido"

Postcondizioni:

- Restituisce un puntatore a una stringa costante che rappresenta il tipo del veicolo.
- Non alloca memoria dinamica.

Side Effect: Nessuno.

- `ottieni_nome_posizione_veicolo(posizione_riconsegna) → const char*`

Restituisce una **stringa descrittiva** che rappresenta il nome di una posizione logica associata a un veicolo, in base al valore intero `posizione`.

Precondizioni

- Nessuna. Qualsiasi valore intero è accettato in input.

Postcondizioni

- Restituisce un puntatore a una stringa costante (`const char*`) che rappresenta:
 - "Deposito" se `posizione == 0`
 - "Posizione B" se `posizione == 1`
 - "Posizione C" se `posizione == 2`
 - "Posizione D" se `posizione == 3`
 - "Posizione sconosciuta" per qualsiasi altro valore

Side Effect: Nessuno.

Prenotazioni:

Specifica sintattica

- `converti_in_timestamp(int, int) -> int`
- `estrai_giorno(int) -> int`
- `estrai_ora(int) -> int`
-
- `ottieni_coda_prenotazioni(void) -> CodaPrenotazioni`
- `inizializza_coda(void) -> CodaPrenotazioni`
- `crea_prenotazione(int, int, int, int, int, int, int, int, int) -> Prenotazione`
- `verifica_fascia_oraria(int, int, int, int) -> int`
- `scambia_prenotazioni(struct Prenotazione*, struct Prenotazione*) -> void`
- `bubble_up(CodaPrenotazioni, int) -> void`
- `bubble_down(CodaPrenotazioni, int) -> void`
- `aggiungi_prenotazione(CodaPrenotazioni, Prenotazione) -> int`
- `cerca_prenotazione(CodaPrenotazioni, int) -> Prenotazione`
- `modifica_stato_prenotazione(CodaPrenotazioni, int, int) -> int`
- `stampa_prenotazione(Prenotazione) -> void`
- `salva_prenotazioni_su_file(CodaPrenotazioni) -> void`
- `carica_prenotazioni_da_file(CodaPrenotazioni) -> int`
- `carica_prenotazioni() -> void`
- `pulisci_coda(CodaPrenotazioni) -> void`
- `distruggi_coda(CodaPrenotazioni) -> void`
- `aggiorna_priorita_prenotazioni(CodaPrenotazioni) -> void`
- `rimuovi_prenotazioni_scadute(CodaPrenotazioni) -> void`
- `rimuovi_prenotazioni_utente(CodaPrenotazioni, int) -> void`
-
- `valida_data_prenotazione(int, int) -> int`
- `verifica_sovrapposizioni(CodaPrenotazioni, int, int, int) -> int`
- `conta_prenotazioni_completate(CodaPrenotazioni, int) -> int`
- `conta_prenotazioni_completate_prima_di(CodaPrenotazioni, int, int) -> int`
-
- `ottieni_id_prenotazione(Prenotazione) -> int`
- `ottieni_id_utente_prenotazione(Prenotazione) -> int`
- `ottieni_id_veicolo_prenotazione(Prenotazione) -> int`
- `ottieni_giorno_ora_inizio(Prenotazione) -> int`
- `ottieni_giorno_ora_fine(Prenotazione) -> int`
- `ottieni_stato_prenotazione(Prenotazione) -> int`
- `ottieni_priorita(Prenotazione) -> int`
-
- `ottieni_giorno_inizio(Prenotazione) -> int`
- `ottieni_ora_inizio(Prenotazione) -> int`
- `ottieni_giorno_fine(Prenotazione) -> int`
- `ottieni_ora_fine(Prenotazione) -> int`
-
- `imposta_id_prenotazione(int, Prenotazione) -> void`

- `imposta_stato_prenotazione(int, Prenotazione) → void`
- `ottieni_dimensione_coda(CodaPrenotazioni) → int`
- `ottieni_prenotazione_in_coda(CodaPrenotazioni coda, int i) → Prenotazione`

Specifica Semantica

- `converti_in_timestamp(giorno, ora) → giorno * 24 + ora`

La funzione restituisce un valore intero (timestamp) che rappresenta un momento temporale univoco nell'arco della settimana, calcolato a partire da un giorno e un'ora.

Precondizioni:

- I parametri `giorno` e `ora` devono essere numeri interi compresi rispettivamente tra 0 e 6 (inclusi) per il giorno, e tra 0 e 23 (inclusi) per l'ora.

Postcondizioni:

- Viene restituito un valore intero non negativo pari a `giorno * 24 + ora`.

Side effect: Nessuno.

- `estrai_giorno(timestamp) → timestamp / 24`

La funzione restituisce il giorno della settimana (0–6) a partire da un timestamp generato con `converti_in_timestamp`.

Precondizioni:

- Il parametro `timestamp` deve essere un intero non negativo, coerente con il formato usato in `converti_in_timestamp`.

Postcondizioni:

- Viene restituito un intero compreso tra 0 e 6, corrispondente al giorno della settimana contenuto nel timestamp.

Side effect: Nessuno.

- `estrai_ora(timestamp) → timestamp % 24`

La funzione restituisce un valore intero che rappresenta l'ora (da 0 a 23) estratta dal valore intero `timestamp`, interpretato come un momento temporale su scala oraria.

Precondizioni:

- Il parametro `timestamp` è un numero intero (può assumere qualsiasi valore intero).

Postcondizioni:

- Viene restituito un intero compreso tra 0 e 23, calcolato come `timestamp % 24`.

Side effect: Nessuno.

- `ottieni_coda_prenotazioni() → coda_globale`

Restituisce un riferimento alla **coda globale delle prenotazioni**, permettendo ad altre parti del programma di accedere alla struttura che contiene tutte le prenotazioni attualmente in memoria.

Precondizioni:

- La variabile `coda_globale` deve essere inizializzata correttamente prima della chiamata, altrimenti il valore restituito potrebbe essere `NULL` o non valido.

Postcondizioni:

- Restituisce un oggetto di tipo `CodaPrenotazioni`, corrispondente al valore attuale di `coda_globale`.

Side effect: Nessuno.

- `inizializza_coda() → CodaPrenotazioni`

La funzione crea e inizializza una nuova coda di prenotazioni allocando dinamicamente la memoria necessaria, impostando la capacità iniziale e la dimensione a zero.

Precondizioni:

- Deve essere disponibile memoria sufficiente sullo heap per allocare sia la struttura `CodaPrenotazioni` che l'array interno delle prenotazioni.
- La macro `INITIAL_CAPACITY` deve essere definita e rappresentare un intero positivo.

Postcondizioni:

- Se l'allocazione ha successo:
 - Restituisce un puntatore a una struttura `CodaPrenotazioni` correttamente inizializzata:
 - `capacita` = `INITIAL_CAPACITY`
 - `dimensione` = 0
 - `heap` = puntatore a un array di `Prenotazione`
- Se l'allocazione fallisce:
 - Restituisce `NULL`.

Side effect:

- Viene allocata memoria dinamica per la struttura `CodaPrenotazioni` e per l'array `heap`.
- `crea_prenotazione(id_utente, id_veicolo, giorno_inizio, ora_inizio, giorno_fine, ora_fine, priorit )` -> prenotazione

La funzione crea e restituisce una nuova struttura `Prenotazione` inizializzata con i dati forniti, calcolando il timestamp di inizio e fine e assegnando uno stato iniziale.

Precondizioni:

- `id_utente` e `id_veicolo` sono interi validi (positivi o zero a seconda del contesto).
- `giorno_inizio` e `giorno_fine` sono interi compresi tra 0 e 6.
- `ora_inizio` e `ora_fine` sono interi compresi tra 0 e 23.
- `priorita`   un intero; se negativo, la priorit  viene calcolata automaticamente.

Postcondizioni:

Viene restituita una struttura **Prenotazione** con:

- **id_prenotazione** univoco incrementato automaticamente;
- campi **id_utente**, **id_veicolo** impostati con i valori forniti;
- **giorno_ora_inizio** e **giorno_ora_fine** calcolati tramite **converti_in_timestamp**;
- **stato** impostato a **IN_ATTESA**;
- **priorita** impostata al valore passato o calcolata tramite **calcola_priorita_temporale** se priorità negativa.

Side effect: Modifica la variabile **id_counter** incrementandola di 1.

- **verifica_fascia_oraria**(giorno_inizio, ora_inizio, giorno_fine, ora_fine) -> int

La funzione verifica se l'intervallo temporale definito da giorno e ora di inizio e giorno e ora di fine è valido, cioè se rispetta i vincoli di intervalli corretti e coerenti.

Precondizioni:

- **giorno_inizio** e **giorno_fine** devono essere interi compresi tra 0 e 6 (inclusi).
- **ora_inizio** e **ora_fine** devono essere interi compresi tra 0 e 23 (inclusi).

Postcondizioni:

Viene restituito 1 se l'intervallo temporale è valido, cioè se:

- i giorni e le ore sono nei range corretti,
- il timestamp calcolato per inizio è strettamente minore di quello di fine.

Viene restituito 0 in caso contrario (parametri fuori range o intervallo non valido).

Side effect: Nessuno.

- **scambia_prenotazioni**(a, b) -> void

La funzione scambia i valori di due strutture **Prenotazione** passate tramite puntatore.

Precondizioni: I puntatori **a** e **b** devono essere validi e non NULL.

Postcondizioni: I contenuti delle strutture puntate da **a** e **b** risultano scambiati.

Side Effect: Modifica i dati delle strutture puntate da **a** e **b**.

- `bubble_up(coda, indice)`

La funzione ripristina la proprietà di heap minima nella coda delle prenotazioni risalendo dall'indice **indice** verso la radice, scambiando le prenotazioni per mantenere l'ordine di priorità.

Precondizioni:

- I parametri **coda** deve essere un puntatore valido a una struttura **CodaPrenotazioni** inizializzata.
- **indice** deve essere un intero compreso tra 0 e la dimensione attuale della coda meno uno.

Postcondizioni:

- La proprietà di heap minima è ristabilita dalla posizione **indice** fino alla radice della coda.

Side effect:

- Modifica l'array **heap** della coda, scambiando elementi tramite la funzione **scambia_prenotazioni**.

- `bubble_down(coda, indice)`

La funzione ripristina la proprietà di heap minima nella coda delle prenotazioni scendendo dall'indice **indice** verso i figli, scambiando le prenotazioni per mantenere l'ordine di priorità.

Precondizioni:

- `coda` deve essere un puntatore valido a una struttura `CodaPrenotazioni` inizializzata.
- `indice` deve essere un intero compreso tra 0 e la dimensione attuale della coda meno uno.

Postcondizioni:

- La proprietà di heap minima è ristabilita dalla posizione `indice` fino alle foglie della coda.

Side effect:

- Modifica l'array `heap` della coda, scambiando elementi tramite la funzione `scambia_prenotazioni`.
- `aggiungi_prenotazione(coda, prenotazione) -> int`

La funzione inserisce una nuova prenotazione nella coda delle prenotazioni, verificando la validità della fascia oraria, ridimensionando l'array se necessario e mantenendo la proprietà di heap minima.

Precondizioni:

- `coda` deve essere un puntatore valido a una struttura `CodaPrenotazioni`
- `prenotazione` deve contenere una fascia oraria valida, cioè l'intervallo tra inizio e fine deve essere corretto (verificato tramite `verifica_fascia_oraria`).

Postcondizioni:

- La prenotazione viene aggiunta alla coda e la struttura heap è riorganizzata per mantenere la proprietà di priorità minima.
- Se l'array era pieno, viene riallocato con capacità raddoppiata.
- In caso di errore (coda nulla, memoria insufficiente, fascia oraria non valida), la coda rimane invariata e viene restituito un codice di errore.

Side effect:

- Potenziale riallocazione dinamica dell'array `heap` della coda.
- Modifica dell'array `heap` con l'inserimento e la riorganizzazione della nuova prenotazione.
- `cerca_prenotazione(coda, id_prenotazione) -> Prenotazione`

La funzione cerca nella coda delle prenotazioni una prenotazione con l'id specificato e restituisce un puntatore ad essa.

Precondizioni: coda deve essere un puntatore valido a una struttura `CodaPrenotazioni` inizializzata.

Postcondizioni: viene restituito un puntatore alla prenotazione con `id_prenotazione` corrispondente se presente nella coda; se la prenotazione non è trovata o la coda è NULL, viene restituito NULL.

Side effect: Nessuno.

- `modifica_stato_prenotazione(coda, id_prenotazione, nuovo_stato) -> int`

La funzione modifica lo stato di una prenotazione specificata dall'id nella coda delle prenotazioni.

Precondizioni:

- coda deve essere un puntatore valido a una struttura `CodaPrenotazioni` inizializzata;
- `id_prenotazione` deve corrispondere a una prenotazione esistente nella coda; `nuovo_stato` deve essere un valore valido di tipo `StatoPrenotazione`.

Postcondizioni:

- lo stato della prenotazione con `id_prenotazione` viene aggiornato a `nuovo_stato`;
- se la prenotazione non viene trovata, la funzione restituisce -1 senza modifiche.

Side effect:

- modifica lo stato interno della prenotazione nella struttura `CodaPrenotazioni`.

- `stampa_prenotazione(p) -> void`

La funzione stampa su console i dettagli di una prenotazione, formattando le informazioni in modo leggibile e colorando lo stato in base al valore.

Precondizioni:

- p deve essere una struttura **Prenotazione** valida e inizializzata con dati coerenti.

Postcondizioni: vengono stampati su console i dati della prenotazione:

- ID prenotazione
- ID utente
- ID veicolo
- giorno e ora di inizio (con nome del giorno)
- giorno e ora di fine (con nome del giorno)
- stato della prenotazione, colorato a seconda del valore.

Side effect:

- stampa su console output formattato con colori modificando il colore della console tramite la funzione **imposta_color**.
- salva_prenotazioni_su_file(coda)-> void

La funzione salva tutte le prenotazioni contenute nella **CodaPrenotazioni** su un file di testo.

Precondizioni:

- **coda** deve essere un puntatore valido a una struttura **CodaPrenotazioni** inizializzata.
- Il file "**data/prenotazioni.txt**" deve essere accessibile per la scrittura (la cartella **data** deve esistere).

Postcondizioni:

- Tutte le prenotazioni presenti nella coda vengono scritte nel file, una per riga, nel seguente ordine:
id_prenotazione id_utente id_veicolo giorno_ora_inizio
giorno_ora_fine stato priorit.

Side effect:

- scrittura su file "**data/prenotazioni.txt**" in modalità sovrascrittura.
- stampa di un messaggio di errore se il file non è accessibile.

- carica_prenotazioni_da_file(coda) -> int

La funzione carica tutte le prenotazioni da un file di testo e le inserisce nella struttura `CodaPrenotazioni`.

Precondizioni:

- coda deve essere un puntatore valido a una struttura `CodaPrenotazioni` inizializzata.
- Il file "data/prenotazioni.txt" deve esistere e deve essere accessibile in lettura.
- Il file deve contenere prenotazioni formattate correttamente, una per riga, nel seguente ordine:

```
id_prenotazione id_utente id_veicolo giorno_ora_inizio
giorno_ora_fine stato priorit .
```

Postcondizioni:

- Le prenotazioni lette correttamente vengono aggiunte alla coda.
- Il valore della variabile globale `id_counter` viene aggiornato se vengono trovati `id_prenotazione` maggiori di quello attuale.
- La funzione restituisce 0 se il caricamento   avvenuto con successo, -1 in caso di errore (file non aperto o coda non inizializzata).

Side effect:

- Lettura da file "`data/prenotazioni.txt`".
- Modifica del contenuto della struttura coda.
- Possibile modifica della variabile globale `id_counter`.
- Stampa di un messaggio di errore se il file non   accessibile o se la coda   NULL.
- Stampa di un messaggio di errore se si verifica un errore nella lettura di una prenotazione.

- void carica_prenotazioni()

La funzione carica le prenotazioni da file nella coda globale di prenotazioni, assicurandosi che la coda sia inizializzata correttamente e svuotata prima del caricamento.

Precondizioni:

- Nessuna precondizione esplicita richiesta da parte del chiamante. La funzione gestisce internamente l'inizializzazione o la pulizia della coda globale.

Postcondizioni:

- Se la coda globale `coda_globale` non esiste, viene creata.
- Se la coda esiste già, viene svuotata tramite `pulisci_coda`.
- Le prenotazioni salvate precedentemente su file vengono caricate nella coda.
- In caso di errore nell'inizializzazione della coda, viene stampato un messaggio e la funzione termina.

Side effect:

- Possibile stampa su standard output.
 - Modifica della variabile globale `coda_globale`, che può essere allocata o svuotata.
 - Lettura da file di prenotazioni.
-
- `pulisci_coda(coda) -> void`

La funzione resetta la coda di prenotazioni indicata, eliminando logicamente tutte le prenotazioni in essa contenute senza deallocare la memoria.

Precondizioni:

- Il puntatore `coda` deve puntare a una struttura `CodaPrenotazioni` inizializzata correttamente, oppure può essere `NULL`. In tal caso la funzione non esegue alcuna operazione.

Postcondizioni:

- La coda indicata avrà `dimensione` pari a 0. Le prenotazioni precedentemente contenute non sono più considerate valide dalla coda (anche se la memoria non viene liberata o sovrascritta).

Side effect:

- Modifica dello stato interno della struttura `coda`, in particolare del campo `dimensione`.
-
- `distruggi_coda(coda) → void`

La funzione libera tutta la memoria allocata dinamicamente associata alla coda di prenotazioni passata come parametro, deallocando sia l'array interno `heap` sia la struttura `CodaPrenotazioni`.

Precondizioni:

- Il puntatore `coda` può essere `NULL` oppure deve puntare a una struttura `CodaPrenotazioni` precedentemente allocata tramite `malloc` o simile.

Postcondizioni:

- Tutta la memoria associata alla coda viene liberata. Il puntatore passato non viene modificato (non è settato a `NULL` all'esterno della funzione).

Side effect:

- Deallocazione della memoria associata alla struttura `CodaPrenotazioni` e al suo campo `heap`. Se la funzione è chiamata su una coda valida, quella coda non è più utilizzabile dopo la chiamata.

- `aggiorna_priorita_prenotazioni(coda) → void`

La funzione `aggiorna` le priorità di tutte le prenotazioni attive (cioè non cancellate né completate) nella coda, in base alla data di sistema corrente. Dopo l'aggiornamento, riorganizza l'heap per mantenere la proprietà di struttura a priorità.

Precondizioni:

- `coda` deve essere un puntatore valido a una struttura `CodaPrenotazioni` inizializzata correttamente. La funzione non ha effetto se `coda` è `NULL`.

Postcondizioni:

- Tutte le prenotazioni non completate o cancellate avranno la loro priorità aggiornata coerentemente alla data di sistema. L'heap rispetterà nuovamente la struttura di max-heap (o min-heap, a seconda della definizione nel progetto).

Side effect:

- Modifica il contenuto della struttura `coda`, aggiornando i campi `priorita` delle prenotazioni e riorganizzando la struttura `heap`.

- `rimuovi_prenotazioni_scadute(coda) → void`

La funzione `aggiorna` lo stato delle prenotazioni nella coda che risultano scadute rispetto alla data di sistema: tutte le prenotazioni il cui timestamp di fine è precedente a quello corrente e che non sono ancora completate o cancellate vengono marcate come `COMPLETATA`.

Precondizioni:

- `coda` deve essere un puntatore valido a una struttura `CodaPrenotazioni` inizializzata. Se `coda` è `NULL`, la funzione non ha alcun effetto.

Postcondizioni:

- Ogni prenotazione scaduta nella coda, cioè con `giorno_ora_fine < timestamp_corrente` e stato diverso da `COMPLETATA` o `CANCELLATA`, verrà modificata ponendo `stato = COMPLETATA`.

Side effect:

- Modifica il campo `stato` di alcune prenotazioni presenti nella struttura `heap` di `coda`.
-
- `rimuovi_prenotazioni_utente(coda, id_utente) → void`

La funzione rimuove tutte le prenotazioni associate a un determinato utente dalla coda, mantenendo la proprietà dell'heap durante la rimozione.

Precondizione

- `coda` è un puntatore valido a una struttura `CodaPrenotazioni`
- `id_utente` è un identificatore valido

Postcondizione

- Tutte le prenotazioni dell'utente specificato vengono rimosse dalla coda
- La struttura dell'heap viene mantenuta valida dopo ogni rimozione

Side Effect

- Modifica la struttura della coda rimuovendo le prenotazioni dell'utente

- `valida_data_prenotazione(giorno_ora_inizio, giorno_ora_fine) → int`

Restituisce un intero che indica la validità di un intervallo di prenotazione.

La funzione verifica se l'intervallo temporale specificato da `giorno_ora_inizio` a `giorno_ora_fine` è valido rispetto alla data di sistema attuale.

- Se l'inizio è nel passato, restituisce `-1`.
- Se la fine è prima o uguale all'inizio, restituisce `-2`.

- Se entrambe le condizioni sono soddisfatte, restituisce 0.

Precondizioni:

- `giorno_ora_inizio` e `giorno_ora_fine` sono timestamp interi validi, ottenuti tramite `converti_in_timestamp`.

Postcondizioni:

- Il valore di ritorno indica lo stato di validità:
 - -1: data di inizio nel passato.
 - -2: data di fine precedente o uguale alla data di inizio.
 - 0: data valida.

Side effect: Nessuno.

- `verifica_sovrapposizioni(coda, id_veicolo, giorno_ora_inizio, giorno_ora_fine) → int`

Restituisce un intero che indica se esiste una sovrapposizione con altre prenotazioni per lo stesso veicolo.

La funzione verifica se l'intervallo temporale indicato (`giorno_ora_inizio` - `giorno_ora_fine`) si sovrappone a una prenotazione già presente nella coda per lo stesso veicolo (`id_veicolo`), ignorando quelle cancellate o completate.

Restituisce:

- 1 se c'è almeno una sovrapposizione;
- 0 se non ci sono sovrapposizioni;
- -1 se la coda è nulla.

Precondizioni:

- `coda` è una struttura valida di tipo `CodaPrenotazioni` o `NULL`.
- `id_veicolo` è un identificatore intero valido.

- `giorno_ora_inizio < giorno_ora_fine` e sono entrambi timestamp validi.

Postcondizioni:

- Viene restituito:
 - `-1` se la coda è nulla.
 - `1` se esiste una sovrapposizione con una prenotazione attiva dello stesso veicolo.
 - `0` altrimenti.

Side effect: Nessuno.

- `conta_prenotazioni_completate(coda, id_utente) → conteggio`

Restituisce il numero di prenotazioni completate effettuate dall'utente specificato.

La funzione conta quante prenotazioni nella coda appartengono all'utente con identificatore `id_utente` e hanno stato `COMPLETATA`.

Restituisce il conteggio totale di queste prenotazioni.

Precondizioni:

- `coda` è una struttura valida di tipo `CodaPrenotazioni` oppure `NULL`.
- `id_utente` è un intero identificativo valido di un utente.

Postcondizioni:

- Se la coda è nulla, viene restituito `0`.
- Altrimenti, viene restituito il numero di prenotazioni completate dall'utente specificato.

Side effect:

Nessuno. La coda non viene modificata.

- `conta_prenotazioni_completate_prima_di(coda, id_utente, id_prenotazione_corrente)`
→ conteggio

La funzione conta il numero di prenotazioni completate (stato = 2) da un utente specifico che hanno un ID inferiore a quello della prenotazione corrente. Questo è utile per calcolare lo sconto fedeltà applicabile a una prenotazione.

Precondizione

- `coda` è un puntatore valido a una struttura `CodaPrenotazioni`
- `id_utente` è un identificatore valido
- `id_prenotazione_corrente` è un identificatore valido

Postcondizione

- Viene restituito il numero di prenotazioni completate dall'utente con ID minore di `id_prenotazione_corrente`

Side Effect

- Nessuno

`ottieni_id_prenotazione(p) → id_prenotazione`

Restituisce l'identificatore della prenotazione `p`.

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `id_prenotazione = p->id_prenotazione`.
- Se `p` è `NULL`, `id_prenotazione = -1`.

Side Effect: Nessuno.

- `ottieni_id_utente_prenotazione(p) → id_utente`

Restituisce l'identificatore dell'utente associato alla prenotazione `p`.

Precondizioni:

- p è un puntatore valido a una struttura **Prenotazione**, oppure **NULL**.

Postcondizioni:

- Se p è valido, $id_utente = p \rightarrow id_utente$.
- Se p è **NULL**, $id_utente = -1$.

Side Effect: Nessuno.

- $ottieni_id_veicolo_prenotazione(p) \rightarrow id_veicolo$

Restituisce l'identificatore del veicolo associato alla prenotazione p .

Precondizioni:

- p è un puntatore valido a una struttura **Prenotazione**, oppure **NULL**.

Postcondizioni:

- Se p è valido, $id_veicolo = p \rightarrow id_veicolo$.
- Se p è **NULL**, $id_veicolo = -1$.

Side Effect: Nessuno.

- $ottieni_giorno_ora_inizio(Prenotazione\ p) \rightarrow giorno_ora_inizio$

Restituisce il valore che rappresenta l'istante di inizio della prenotazione p (giorno e ora codificati).

Precondizioni:

- p è un puntatore valido a una struttura **Prenotazione**, oppure **NULL**.

Postcondizioni:

- Se p è valido, $giorno_ora_inizio = p \rightarrow giorno_ora_inizio$.
- Se p è **NULL**, $giorno_ora_inizio = -1$.

Side Effect: Nessuno.

- `ottieni_giorno_ora_fine(p) → giorno_ora_fine`

Restituisce il valore che rappresenta l'istante di fine della prenotazione `p` (giorno e ora codificati).

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `giorno_ora_fine = p->giorno_ora_fine`.
- Se `p` è `NULL`, `giorno_ora_fine = -1`.

Side Effect: Nessuno.

- `ottieni_stato_prenotazione(Prenotazione p) → stato`

Restituisce lo stato della prenotazione `p`, ad esempio attiva, completata, annullata ecc. (codificato come intero).

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `stato = p->stato`.
- Se `p` è `NULL`, `stato = -1`.

Side Effect: Nessuno.

- `ottieni_priorita(p) → priorit `

Restituisce il valore della priorit  associata alla prenotazione `p`.

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `priorita = p->priorita`.
- Se `p` è `NULL`, `priorita = -1`.

Side effect: Nessuno.

- `ottieni_giorno_inizio(p) → giorno_inizio`

Restituisce il giorno di inizio della prenotazione `p`, estratto dal campo `giorno_ora_inizio`.

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `giorno_inizio = estrai_giorno(p->giorno_ora_inizio)`.
- Se `p` è `NULL`, `giorno_inizio = -1`.

Side effect: Nessuno.

- `ottieni_ora_inizio(p) → ora_inizio`

Restituisce l'ora di inizio della prenotazione `p`, estratta dal campo `giorno_ora_inizio`.

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `ora_inizio = estrai_ora(p->giorno_ora_inizio)`.
- Se `p` è `NULL`, `ora_inizio = -1`.

Side effect: Nessuno.

- `ottieni_giorno_fine(Prenotazione p) → giorno_fine`

Restituisce il giorno di fine della prenotazione `p`, estratto dal campo `giorno_ora_fine`.

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `giorno_fine = estrai_giorno(p->giorno_ora_fine)`.
- Se `p` è `NULL`, `giorno_fine = -1`.

Side effect: Nessuno.

- `ottieni_ora_fine(p) → ora_fine`

Restituisce l'ora di fine della prenotazione `p`, estratta dal campo `giorno_ora_fine`.

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `ora_fine = estrai_ora(p->giorno_ora_fine)`.
- Se `p` è `NULL`, `ora_fine = -1`.

Side effect: Nessuno.

- `imposta_id_prenotazione(id_prenotazione, p) → void`

Imposta il campo `id_prenotazione` della struttura `p` al valore fornito.

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `p->id_prenotazione = id_prenotazione`.
- Se `p` è `NULL`, non viene effettuata alcuna modifica.

Side effect:

- Modifica diretta del campo `id_prenotazione` della struttura `p`.

- `imposta_stato_prenotazione(stato, p) → void`

Imposta il valore `stato` nella prenotazione `p`, che rappresenta lo stato attuale della prenotazione (es. attiva, cancellata, etc.).

Precondizioni:

- `p` è un puntatore valido a una struttura `Prenotazione`, oppure `NULL`.

Postcondizioni:

- Se `p` è valido, `p->stato = stato`.
- Se `p` è `NULL`, non viene effettuata alcuna modifica.

Side effect:

- Modifica diretta del campo `stato` nella struttura `p`.

- `ottieni_dimensione_coda(coda) → dimensione`

Restituisce il numero attuale di prenotazioni presenti nella coda `coda`.

Precondizioni:

- `coda` è un puntatore valido a una struttura `CodaPrenotazioni`, oppure `NULL`.

Postcondizioni:

- Se `coda` è valida, `dimensione = coda->dimensione`.
- Se `coda` è `NULL`, `dimensione = -1`.

Side effect: Nessuno.

- `ottieni_prenotazione_in_coda(coda, i) → prenotazione`

La funzione restituisce il puntatore alla prenotazione presente nella posizione specificata all'interno della coda delle prenotazioni.

Precondizione

- `coda` è un puntatore valido a una struttura `Codaprenotazioni`
- `i` è un indice valido ($0 \leq i < \text{dimensione_coda}$)

Postcondizione

- Se i parametri sono validi, viene restituito il puntatore alla prenotazione richiesta
- Se i parametri non sono validi, viene restituito `NULL`

Side Effect

- Nessuno

Fasce_orarie

Specifica Sintattica:

- `inizializza_calendario(int) → CalendarioVeicolo`
- `aggiorna_calendario(CalendarioVeicolo, CodaPrenotazioni) → CalendarioVeicolo`
- `visualizza_calendario(CalendarioVeicolo) → void`
- `verifica_disponibilita(CalendarioVeicolo, int, int, int, int) → int`

Specifica Semantica:

- `inizializza_calendario(calendario, id_veicolo) → void`

Inizializza il calendario di un veicolo specifico, marcando tutte le fasce orarie settimanali come libere.

La funzione inizializza la struttura `CalendarioVeicolo` per il veicolo identificato da `id_veicolo`, impostando tutte le 168 fasce orarie (24 ore × 7 giorni) come disponibili. Ogni fascia oraria sarà marcata come non occupata e senza alcuna prenotazione associata.

Precondizioni:

- `calendario` è un puntatore valido a una struttura `CalendarioVeicolo`.
- `id_veicolo` è un intero che rappresenta l'identificatore del veicolo.

Postcondizioni:

- Il campo `id_veicolo` della struttura viene impostato al valore passato.
- Tutti gli slot del calendario settimanale risultano inizializzati con `occupato = 0` e `id_prenotazione = 0`.

Side effect:

- La memoria puntata da `calendario` viene modificata.

- `aggiorna_calendario(calendario, coda) → void`

Aggiorna il calendario di un veicolo in base alle prenotazioni attive presenti nella coda.

La funzione aggiorna il campo `calendario` di un veicolo specifico (`id_veicolo`) marcando come occupate le fasce orarie corrispondenti alle prenotazioni attive nella struttura `coda`, ignorando quelle cancellate. Le prenotazioni su più giorni vengono gestite dividendo l'intervallo su ciascun giorno coinvolto.

Precondizioni:

- `calendario` è un puntatore valido a una struttura `CalendarioVeicolo`.
- `coda` è un puntatore valido a una struttura `CodaPrenotazioni`.

Postcondizioni:

- Il calendario viene inizializzato a tutte fasce libere (`occupato = 0`, `id_prenotazione = 0`).
- Per ogni prenotazione attiva relativa al veicolo (`id_veicolo`), le fasce orarie comprese tra inizio e fine vengono marcate come occupate.
- Le prenotazioni multi-giorno vengono suddivise correttamente su più giorni nel calendario.

Side Effect:

- La memoria puntata da `calendario` viene modificata.
 - Nessun effetto collaterale esterno alla struttura `calendario`.
-
- `visualizza_calendario(calendario) → void`

Stampa a video il calendario settimanale di un veicolo, indicando le fasce orarie occupate o libere per ciascun giorno della settimana.

La funzione stampa una tabella con righe per ogni giorno (da Lunedì a Domenica) e colonne per ogni ora (0–23). Ogni cella della tabella indica se una fascia oraria è **occupata** (X) oppure **libera** (spazio vuoto), secondo i dati memorizzati nella struttura `calendario`.

Precondizioni:

- `calendario` è un puntatore valido a una struttura `CalendarioVeicolo`.

Postcondizioni:

- Nessuna modifica allo stato delle variabili globali o alla memoria puntata.
- L'output sul terminale rappresenta graficamente lo stato delle fasce orarie del veicolo specificato tramite `calendario`.

Side effect:

- Viene stampato un output formattato sul terminale.
- `verifica_disponibilita(calendario, giorno_inizio, ora_inizio, giorno_fine, ora_fine) → int`

Verifica se una fascia oraria è completamente disponibile (cioè non occupata) nel calendario settimanale di un veicolo.

Precondizioni:

- `calendario` è un puntatore valido a una struttura `CalendarioVeicolo`.
- `giorno_inizio` e `giorno_fine` $\in [0, 6]$ (dove 0 = Lunedì, 6 = Domenica).
- `ora_inizio` e `ora_fine` $\in [0, 23]$.
- `giorno_inizio` \leq `giorno_fine`.
- Se `giorno_inizio == giorno_fine`, allora `ora_inizio < ora_fine`.

Postcondizioni:

- Nessuna modifica allo stato del `calendario`.
- Restituisce:
 - 1 se tutte le ore comprese tra `[giorno_inizio, ora_inizio]` e `[giorno_fine, ora_fine)` sono libere (cioè `occupato == 0`).
 - 0 se anche solo una fascia oraria nel range specificato è occupata, oppure se i parametri sono fuori dai limiti accettabili.

Side Effect: Nessuno.

Tariffe

Specifica Sintattica:

- `calcola_tariffa(int, int) → double`
- `calcola_tariffa_prenotazione(CodaPrenotazioni, int, int, int, int) → double`
- `ottieni_tariffa_oraria(int) → double`
- `applica_sconto_fedelta(double, int) → double`
- `applica_sconto_pacchetto_ore(double, int) → double`
- `calcola_ore_gratuite(int) → int`
- `stampa_info_sconti(void) → void`

Specifica Semantica:

- `calcola_tariffa(tipo, ore_totali) → tariffa`

Calcola la tariffa totale in base al tipo di veicolo e al numero di ore prenotate, applicando uno sconto promozionale (1 ora gratuita ogni 5 ore).

Precondizioni:

- `tipo` rappresenta un codice valido per un veicolo esistente (tra 0 e 3)
- `ore_totali` ≥ 0

Postcondizioni:

- restituisce il costo complessivo della prenotazione, calcolato moltiplicando la tariffa oraria del tipo specificato per il numero di ore, con eventuali sconti applicati
- la tariffa oraria viene ottenuta tramite la funzione `ottieni_tariffa_oraria`
- lo sconto viene applicato tramite la funzione `applica_sconto_pacchetto_ore`

Side Effect: Nessuno.

- `calcola_tariffa_prenotazione(coda, tipo, giorno_ora_inizio, giorno_ora_fine, id_utente) → tariffa`

Calcola la tariffa totale per una prenotazione considerando il tipo di veicolo e l'intervallo di tempo tra inizio e fine prenotazione.

Precondizioni:

- `tipo` rappresenta un codice valido per un veicolo esistente (tra 0 e 3)
- `giorno_ora_inizio` e `giorno_ora_fine` sono interi]

- `giorno_ora_fine` rappresenta un momento successivo a `giorno_ora_inizio` (anche su giorni diversi)

Postcondizioni:

- calcola il numero totale di ore tra giorno/ora iniziale e finale
- richiama `calcola_tariffa(tipo, ore_totali)` per ottenere la tariffa complessiva con eventuali sconti
- restituisce il costo della prenotazione per l'intervallo specificato

Side Effect: Nessuno.

- `ottieni_tariffa_oraria(tipo) → tariffa_oraria`

La funzione restituisce la tariffa oraria associata al tipo di veicolo specificato.

Precondizioni:

- `tipo` è un intero che rappresenta un codice valido per un veicolo (0 = Utilitaria, 1 = SUV, 2 = Sportiva, 3 = Moto)

Postcondizioni:

- restituisce la tariffa oraria costante associata al tipo di veicolo
- se `tipo` non è valido, restituisce 0.0

Side Effect: nessuno

- `applica_sconto_fedelta(tariffa_base, noleggi_completati) → tariffa_scontata`

La funzione applica uno sconto fedeltà alla tariffa base se il numero di noleggi supera una soglia predefinita.

Precondizioni:

- `tariffa_base` ≥ 0
- `noleggi_completati` ≥ 0

- `NOLEGGI_PER_SCONTO` e `SCONTO_FEDELTA` sono costanti definite nel sistema

Postcondizioni:

- se `numero_noleggi` \geq `NOLEGGI_PER_SCONTO`, restituisce `tariffa_base` scontata del valore percentuale definito da `SCONTO_FEDELTA`
- altrimenti restituisce la `tariffa_base` senza sconto

Side Effect: Nessuno.

- `applica_sconto_pacchetto_ore(tariffa_base, ore_totali) → tariffa_scontata`

La funzione calcola il costo totale applicando una promozione che prevede 1 ora gratuita ogni certo numero di ore prenotate (ad esempio, 1 ora gratis ogni 5 ore).

Precondizioni:

- `tariffa_base` ≥ 0
- `ore_totali` ≥ 0
- la funzione `calcola_ore_gratuite` è definita e restituisce un numero intero corretto di ore gratuite

Postcondizioni:

- calcola quante ore sono gratuite tramite `calcola_ore_gratuite(ore_totali)`
- restituisce il costo pari a `tariffa_base` moltiplicata per il numero di ore effettivamente da pagare (`ore_totali - ore_gratuite`)

Side Effect: Nessuno.

- `calcola_ore_gratuite(ore_totali) → ore_gratuite`

La funzione determina quante ore gratuite spettano in base a una promozione che regala 1 ora ogni pacchetto di `ORE_PER_PACCHETTO` + 1 ore totali.

Precondizioni:

- `ore_totali` ≥ 0

- `ORE_PER_PACCHETTO` è una costante intera positiva definita nel sistema

Postcondizioni:

- restituisce il numero intero di ore gratuite spettanti secondo la regola: 1 ora gratis ogni $(\text{ORE_PER_PACCHETTO} + 1)$ ore
- il valore restituito è $\text{ore_totali} / (\text{ORE_PER_PACCHETTO} + 1)$

Side Effect: Nessuno.

- `stampa_info_sconti()` → void

La funzione stampa a video informazioni descrittive sugli sconti disponibili nel sistema di noleggio, inclusi lo sconto fedeltà e le offerte sui pacchetti orari.

Precondizioni: Nessuna

Postcondizioni:

- vengono visualizzate sullo standard output le descrizioni degli sconti con valori aggiornati delle costanti `SCONTO_FEDELTA`, `NOLEGGI_PER_SCONTO` e `ORE_PER_PACCHETTO`
- non restituisce valore

Side Effect: scrittura su standard output (console).

Utenti

Specifica Sintattica:

- `carica_ultimo_id_utente()` → int
- `inizializza_tabella_utenti()` → void
- `salva_utenti_file()` → void

- carica_utenti_file() → int
- inserisci_utente(const char*, const char*) → int
- cerca_utente(const char*) → Utente*
- cerca_utente_per_id(int) → Utente*
- stampa_utenti() → void
- rimuovi_utente(int) → int
- ottieni_id_utente(Utente) → int
- ottieni_nome_completo_utente(Utente) → const char*
- ottieni_isamministratore_utente(Utente) → int
- imposta_password_utente(const char*, Utente) → void
- valida_nome_utente(const char*) → int
- valida_nome_completo(const char*) → int
- verifica_password(const char*, Utente) → int
- hash_password(const char*, char*) → void

Specifica Semantica:

- carica_ultimo_id_utente() → max_id

La funzione restituisce l'ID utente più alto presente nel file `utenti.txt`.

Precondizioni:

- Il file "`data/utenti.txt`" dovrebbe contenere righe nel formato:
`id nome_utente password nome_completo isamministratore`
dove `id` è un intero e `nome_utente` è una stringa senza spazi.
- Il file se non esiste, viene inizializzato con utente Amministratore con id 0

Postcondizioni:

- Se il file non esiste o non è leggibile, restituisce 0.
- Altrimenti, restituisce il massimo `id` numerico trovato nel file, **escludendo** eventuali righe dove il nome utente è "`Amministratore`".

Side Effect: Nessuno

- inizializza_tabella_utenti() → void

Inizializza la tabella hash degli utenti e garantisce la presenza dell'utente amministratore (`Admin`) con ID 0.

Precondizioni:

- La variabile globale `tabellaUtenti` deve essere un array di puntatori a `Utente`, di dimensione `TABLE_SIZE`.
- Deve esistere la funzione `hash_djb2` per calcolare l'indice della tabella.
- La struttura `Utente` deve contenere almeno i campi: `id`, `nome_utente`, `nome_completo`, `isamministratore`.

Postcondizioni:

- Tutti gli elementi della tabella hash sono inizializzati a `NULL`.
- Se l'utente `"Admin"` non è già presente nel file `data/utenti.txt`, viene creato e inserito nella tabella con:
 - `id = 0`
 - `nome_utente = "amministratore"`
 - `nome_completo = "Amministratore"`
 - `isamministratore = 1`
- Se il file `data/utenti.txt` non esiste, viene creato.
- L'utente admin viene scritto nel file, in modalità `append` o `write`, a seconda dei casi.

Side Effect:

- Apre e legge il file `data/utenti.txt`.
 - Può scrivere nel file `data/utenti.txt`.
 - Alloca dinamicamente memoria per l'utente `Admin` e lo inserisce nella tabella.
 - Stampa un messaggio se l'admin viene creato.
-
- `salva_utenti_file() → void`

La funzione salva su file "data/utenti.txt" tutte le informazioni degli utenti presenti nella tabella `tabellaUtenti`.

Precondizioni:

- `tabellaUtenti` deve essere inizializzata
- ogni utente puntato contiene campi `id`, `nome_utente`, `nome_completo`, `password` e `isamministratore`

Postcondizioni:

- il file "data/utenti.txt" viene sovrascritto con le informazioni di tutti gli utenti presenti nella tabella
- ogni riga del file contiene: `id nome_utente password nome_completo isamministratore`
- se il file non può essere aperto, la funzione stampa un messaggio di errore e termina senza salvare
- in caso di successo, stampa messaggio di conferma

Side Effect:

- apertura e scrittura su file "data/utenti.txt"
 - scrittura su standard output (printf)
-
- `carica_utenti_file()` → successo

La funzione legge il contenuto del file `data/utenti.txt` e inserisce gli utenti nella tabella hash in memoria. Ogni riga del file rappresenta un utente, e il formato atteso è:

```
<id> <nome_utente> <nome_completo> <password>  
<isamministratore>
```

Precondizioni:

- La funzione `inserisci_utente` deve essere disponibile e correttamente implementata, e deve:
 - Inserire un utente nella tabella globale `tabellaUtenti`.

- Ritornare `1` in caso di successo, `0` in caso di errore.
- Il file `data/utenti.txt` deve essere nel formato previsto:
una riga per ogni utente con almeno quattro elementi (ID, nome_utente, nome completo, password, isamministratore), separati da spazi.
- La tabella hash `tabellaUtenti` deve essere inizializzata.

Postcondizioni:

- Tutti gli utenti validi trovati nel file vengono inseriti nella tabella hash tramite `inserisci_utente`.
- Se tutti gli utenti vengono inseriti con successo, la funzione ritorna `1`.
- Se almeno un inserimento fallisce, la funzione ritorna `0`.

Side Effect:

- Apre e chiude il file `data/utenti.txt` in modalità lettura ("`r`").
 - Alloca e modifica strutture dati in memoria (tramite `inserisci_utente`).
 - Stampa messaggi di errore se l'apertura del file fallisce o se un inserimento non va a buon fine.
 - Modifica la tabella globale `tabellaUtenti`.
-
- `inserisci_utente(nome_utente, nome_completo, password) → successo`

La funzione inserisce un nuovo utente nella tabella hash globale `tabellaUtenti`. L'ID dell'utente è assegnato automaticamente, incrementale, e unico, a meno che l'utente non sia l'admin (che ha sempre ID 0). La funzione imposta anche il flag `isamministratore` per l'utente.

Precondizioni:

- `nome_utente` e `nome_completo` e `password` devono essere stringhe valide non nulle.

- La tabella hash `tabellaUtenti` deve essere già allocata e inizializzata a `NULL`.
- La funzione `carica_ultimo_id_utente()` deve essere correttamente implementata per restituire il massimo ID utente presente nel file.

Postcondizioni:

- Viene inserito un nuovo utente nella tabella hash, in una posizione libera secondo probing lineare.
- L'utente avrà:
 - un ID progressivo (a partire da 1 se non ne esistono),
 - ID 0 se l'utente è "Admin",
 - `isamministratore = 1` se `nome_utente == "Admin"`, altrimenti 0.
- Viene allocato dinamicamente un oggetto `Utente` con i valori assegnati.
- Il valore di `id_counter` interno viene aggiornato coerentemente.

Side Effect:

- Legge il file `data/utenti.txt` tramite `carica_ultimo_id_utente()` la prima volta che viene chiamata per inizializzare `id_counter`.
 - Modifica la tabella globale `tabellaUtenti`.
 - Alloca dinamicamente memoria per la struttura `Utente`.
 - Scrive nei campi della struttura (`id`, `nome_utente`, `nome_completo`, `isamministratore`).
 - L'ID viene incrementato automaticamente dopo ogni inserimento, tranne per "Admin".
-
- `cerca_utente(nome_utente) → utente`

La funzione cerca un utente nella tabella hash `tabellaUtenti` in base al campo `nome_utente`. Utilizza probing lineare per risolvere eventuali collisioni.

Precondizioni:

- `nome_utente` è una stringa valida (non `NULL`).
- La tabella hash globale `tabellaUtenti` è già inizializzata (ogni slot è `NULL` o punta a una struttura `Utente` valida).
- È già stato utilizzato lo stesso algoritmo di hashing (`hash_djb2`) durante l'inserimento.

Postcondizioni:

- Se l'utente con il nome specificato è presente nella tabella:
 - Viene restituito un puntatore alla relativa struttura `Utente`.
- Se l'utente non è presente:
 - Viene restituito `NULL`.

Side Effect: Nessuno.

- `cerca_utente_per_id(id) → utente`

La funzione cerca e restituisce un puntatore all'utente corrispondente all'id fornito nella tabella degli utenti, se presente.

Precondizioni:

- `id` è un intero valido come identificatore utente
- la tabella `tabellaUtenti` è un array di puntatori a `Utente` di dimensione `TABLE_SIZE` già inizializzato

Postcondizioni:

- restituisce un puntatore all'utente con campo `id` uguale al valore passato, se presente nella tabella
- restituisce `NULL` se nessun utente con quell'id è presente

Side Effect: Nessuno.

- `stampa_utenti()` → void

La funzione stampa a video la lista di tutti gli utenti presenti nella tabella hash `tabellaUtenti`, mostrando ID, nome_utente, nome completo e se sono admin o meno.

Precondizioni:

- `tabellaUtenti` è un array di dimensione `TABLE_SIZE` contenente puntatori a strutture `Utente` o NULL
- ogni struttura `Utente` ha i campi `id`, `nome_utente`, `nome_completo`, e `isamministratore` correttamente valorizzati

Postcondizioni:

- nessuna modifica allo stato interno (funzione di sola lettura)
- vengono stampate a video le informazioni di ogni utente presente nella tabella

Side Effect:

- stampa su standard output (console) le informazioni degli utenti

- `verifica_password(password, u)` → int

Verifica se la password fornita corrisponde alla password memorizzata per l'utente u passato in input.

Precondizioni:

- `password` è un puntatore valido a una stringa terminata da `\0`.

Postcondizioni:

- Se l'utente esiste e la password corrisponde (dopo hashing), la funzione restituisce 1.
- Se l'utente non esiste o la password non corrisponde, la funzione restituisce 0.

Side effect:

- Alloca dinamicamente memoria per un puntatore Utente ma questa viene sovrascritta senza essere liberata (potenziale perdita di memoria).
 - Chiama hash_password per ottenere l'hash della password in input.
- hash_password(input, output) → void

Calcola l'hash della stringa input usando la funzione hash_djb2 e scrive il risultato (in forma numerica) nella stringa output.

Precondizioni:

- input è un puntatore valido a una stringa terminata da \0.
- output è un buffer sufficientemente grande per contenere la rappresentazione in stringa dell'hash (almeno MAX_PASSWORD_LENGTH caratteri).

Postcondizioni:

- output contiene la rappresentazione in stringa dell'hash calcolato da hash_djb2(input).

Side effect:

- Nessun effetto collaterale esterno; modifica diretta del buffer output.

- ottieni_id_utente(nome_utente) → id_utente

Restituisce l'ID (id_utente) dell'utente fornito in input.

Precondizioni:

- nome_utente è un puntatore valido a una stringa terminata da \0.

Postcondizioni:

- Se l'utente esiste, restituisce utente->id come id_utente.
- Se l'utente non esiste, restituisce -1 come id_utente.

Side Effect: Nessuno.

- ottieni_nome_completo_utente(u) → nome_completo

Restituisce il nome completo di un utente

Precondizioni:

- u è un puntatore valido a una stringa terminata da \0.

Postcondizioni:

- Se l'utente esiste, restituisce utente->nome_completo come nome_completo
- Se l'utente non esiste, restituisce NULL come nome_utente_utente.

Side Effect: Nessuno.

- ottieni_isamministratore_utente(u) → isamministratore_utente

Restituisce il valore intero isamministratore associato all'utente.

Precondizioni:

- u è un puntatore

- **Postcondizioni:**

- Se l'utente esiste, restituisce utente->isamministratore come isamministratore_utente.
- Se l'utente non esiste, restituisce -1 come isamministratore_utente.

Side Effect:

- Alloca memoria con malloc ma la perde (memory leak).
- Chiama cerca_utente(nome_utente).

- imposta_nome_utente_utente(new_nome_utente) → void

Imposta il campo nome_utente dell'utente corrente con `new_nome_utente`, se questo non è già usato da un altro utente.

Precondizioni:

- `new_nome_utente` è un puntatore valido a una stringa terminata da \0.
- L'utente corrente è correttamente definito (non gestito nel codice).

Postcondizioni:

- Se `new_nome_utente` non è già in uso, il campo nome_utente dell'utente corrente è aggiornato.
- Se `new_nome_utente` è già in uso, stampa un messaggio di errore e non modifica nulla.

Side effect:

- Allocazione inutile di memoria e perdita di memoria (memory leak).
 - Stampa messaggi su stdout.
 - Modifica diretta del campo nome_utente nella struttura utente.
- valida_nome_utente(nome_utente) → int

La funzione verifica se la stringa `nome_utente` rispetta i criteri di validità definiti per i nomi utente del sistema. In particolare, controlla:

- che la lunghezza sia compresa tra 3 e 29 caratteri inclusi;
- che tutti i caratteri siano lettere (A-Z, a-z), cifre (0-9) o il carattere underscore `_`.

Se tutti i controlli sono superati, la funzione restituisce 1 (valido), altrimenti 0 (non valido).

Precondizioni:

- `nome_utente` deve essere una stringa terminata da carattere nullo (`'\0'`).
- La stringa `nome_utente` non deve essere `NULL`.

Postcondizioni:

- Non modifica il contenuto di `nome_utente` o altre variabili esterne.
- Restituisce 1 se `nome_utente` rispetta le regole di validità.
- Restituisce 0 in caso contrario.

Side Effect: Nessuno.

- valida_nome_completo(nome) → int

La funzione controlla se la stringa `nome` rispetta i criteri di validità per un nome completo nel sistema. In particolare:

- la lunghezza deve essere compresa tra 3 e 49 caratteri inclusi;
- ogni carattere deve essere una lettera (A-Z, a-z), uno spazio `' '`, un apostrofo `'` o un trattino `-`.

Se tutti i caratteri sono validi e la lunghezza è corretta, restituisce 1 (valido); altrimenti restituisce 0 (non valido).

Precondizioni

- `nome` deve essere una stringa terminata da carattere nullo (`'\0'`);
- `nome` non deve essere `NULL`.

Postcondizioni

- Non modifica la stringa `nome` né altre variabili esterne;
- Restituisce 1 se il nome è valido;
- Restituisce 0 se non è valido.

Side Effect: Nessuno.

Hash

Specifica Sintattica:

- `hash_djb2(const char*) → unsigned int`.

Specifica Semantica:

- `hash_djb2(str) → hash`

La funzione calcola il valore hash di una stringa secondo l'algoritmo DJB2. È usata per mappare stringhe (come `nome_utente`) in indici di una tabella hash.

Precondizioni

- `str` deve essere una stringa valida e terminata da `\0`.

Postcondizioni

- Viene restituito un valore hash.
- La stringa di input non viene modificata.

Side Effect: Nessuno.

Menu

Specifica Sintattica:

- `visualizza_tariffe(Utente) → void`
- `gestione_veicoli(void) → void`
- `prenota_auto(Utente) → void`
- `visualizza_prenotazioni(Utente) → void`
- `restituisce_auto(void) → void`
- `visualizza_disponibilita(void) → void`
- `mostra_menu_cliente(Utente) → void`
- `mostra_menu_amministratore(Utente) → void`
- `gestione_prenotazioni_amministratore(void) → void`
- `monstra_menu_iniziale(void) → void`
- `mostra_logo(void) → void`
- `mostra_menu_login(void) → void`
- `gestione_utenti_amministratore(void) → void`

Specifica Semantica:

- `visualizza_tariffe(utente_corrente) → void`

La funzione mostra all'utente una schermata a video con le informazioni sulle tariffe orarie dei diversi tipi di veicoli disponibili, gli sconti applicabili, la data corrente e, se l'utente è loggato (cioè `utente_corrente` non è NULL), mostra anche il numero di noleggi completati da quell'utente. Infine, aspetta che l'utente prema INVIO per continuare.

Precondizioni

- `utente_corrente` è un puntatore valido a una struttura `Utente` oppure NULL se non c'è utente loggato.
- Le costanti di tariffa (`TARIFFA_UTILITARIA`, `TARIFFA_SUV`, `TARIFFA_SPORTIVA`, `TARIFFA_MOTO`) devono essere definite.
- Le funzioni di supporto chiamate all'interno (`pulisci_schermo`, `stampa_bordo_superiore/inferiore`, `stampa_separatore`, `imposta_colore`, ecc.) devono essere correttamente implementate.

Postcondizioni

- Viene visualizzata a schermo una schermata ben formattata contenente:
 - Intestazioni colorate e bordi per migliorare l'estetica.
 - La data di sistema corrente.
 - Le tariffe orarie per ogni tipo di veicolo.
 - Le informazioni sugli sconti disponibili.
 - Il numero di noleggi completati dall'utente corrente, se presente.
- Il programma rimane in pausa fino a quando l'utente non preme INVIO.
- Il buffer di input viene svuotato per evitare input residui indesiderati.

Side Effect:

- Modifica la schermata del terminale (colore testo, pulizia schermo).
 - Stampa informazioni a video.
 - Legge il numero di noleggi completati tramite funzioni esterne.
 - Effettua una pausa in attesa di input utente.
 - Svuota il buffer di input.
-
- `gestione_veicoli() → void`

La funzione implementa un menu interattivo per la gestione dei veicoli in un sistema di noleggio. Consente all'utente di eseguire operazioni quali aggiungere un veicolo, rimuovere un veicolo, visualizzare tutti i veicoli disponibili o tornare al menu principale.

Il menu viene ripetuto finché l'utente non sceglie di uscire (scelta 0).

Precondizioni

- Le funzioni di supporto chiamate (`pulisci_schermo`, `stampa_bordo_superiore/inferiore`, `imposta_colore`, `svuota_buffer`, ecc.) devono essere correttamente implementate e funzionanti.
- Le funzioni per la manipolazione della lista dei veicoli (`ottieni_lista_veicoli`, `aggiungi_veicolo`, `rimuovi_veicolo`,

`salva_lista_veicoli`, `ottieni_veicolo_senza_rimuovere`, ecc.) devono essere implementate e devono gestire correttamente la lista.

- La struttura dati della lista di veicoli deve essere inizializzata e corretta.
- L'input da tastiera deve essere valido o gestito adeguatamente.

Postcondizioni

- Se l'utente sceglie 1, un nuovo veicolo viene aggiunto alla lista e la lista aggiornata viene salvata su file.
- Se l'utente sceglie 2, un veicolo viene rimosso dalla lista e la lista aggiornata viene salvata su file.
- Se l'utente sceglie 3, viene mostrato a video l'elenco dei veicoli attualmente disponibili (quelli con disponibilità zero), con informazioni dettagliate e tariffe orarie.
- Se l'utente sceglie 0, il menu termina e si ritorna al menu principale.
- Per scelte non valide, viene mostrato un messaggio d'errore e si attende che l'utente prema INVIO per riprovare.
- La lista dei veicoli rimane aggiornata in memoria e su file dopo operazioni di aggiunta o rimozione.
- Durante la visualizzazione dei veicoli, la funzione mostra una schermata pulita e formattata.

Side Effect

- Modifica la visualizzazione del terminale (pulizia schermo, colori, stampa a video).
- Legge input da tastiera e gestisce il buffer di input.
- Modifica la lista dei veicoli in memoria tramite chiamate a funzioni di aggiunta/rimozione.
- Salva i cambiamenti della lista dei veicoli su file.
- Può allocare e liberare memoria temporanea per la visualizzazione dei veicoli.
- Blocca l'esecuzione in attesa di input utente dopo la visualizzazione.

- `prenota_auto(utente_corrente) → void`

Gestisce l'interfaccia testuale per la gestione delle prenotazioni auto per l'utente `utente_corrente`. Permette di creare, visualizzare, cancellare e modificare prenotazioni, visualizzare tariffe e avanzare il tempo di sistema.

Precondizioni

- `utente_corrente` deve essere un puntatore valido a una struttura `Utente` già inizializzata e autenticata.
- Il sistema deve aver inizializzato correttamente la coda delle prenotazioni (`CodaPrenotazioni`) e la lista dei veicoli, accessibili tramite `ottieni_coda_prenotazioni()` e `ottieni_lista_veicoli()`.
- Devono essere presenti funzioni e strutture dati utilizzate dalla funzione (ad es. funzioni per la gestione della coda, prenotazioni, veicoli, utenti, colori, e input/output).

Postcondizioni

- Se `coda_prenotazioni` è NULL, la funzione termina senza modificare alcun dato.
- In caso di creazione di una nuova prenotazione, se tutti i dati sono validi e la prenotazione viene confermata dall'utente, la nuova prenotazione viene aggiunta a `coda_prenotazioni` e salvata su file.
- Se viene visualizzata la lista delle prenotazioni, nessun dato viene modificato.
- Se viene cancellata una prenotazione e l'utente ha i permessi, lo stato della prenotazione viene modificato (ad esempio, impostato a "cancellato") e le modifiche sono salvate su file.
- Se uno stato prenotazione viene modificato (solo da amministratore), la modifica è salvata su file.
- In ogni altra opzione, la funzione esegue le operazioni indicate senza modificare lo stato delle prenotazioni se non esplicitamente richiesto.
- Alla fine di ogni operazione, il sistema attende la pressione di INVIO per tornare al menu delle prenotazioni o al menu principale.

Side Effect:

- Modifica la struttura dati globale o condivisa `coda_prenotazioni` con nuove prenotazioni, cancellazioni o modifiche.
 - Scrive su file i dati aggiornati della coda delle prenotazioni (`salva_prenotazioni_su_file`).
 - Modifica l'output del terminale, inclusa la colorazione del testo tramite `imposta_colore`.
 - Richiede input utente tramite `scanf` e attende la conferma tramite pressione di INVIO.
 - Pulisce lo schermo e stampa menù, messaggi di errore e informazioni tramite funzioni dedicate (`pulisci_schermo`, `stampa_bordo_superiore`, etc.).
-
- `visualizza_prenotazioni()` → void

La funzione mostra a video tutte le prenotazioni presenti nella coda delle prenotazioni. Visualizza un'intestazione con titolo e data di sistema, poi, se ci sono prenotazioni, elenca ciascuna mostrando ID, dettagli e costo stimato calcolato in base al tipo di veicolo e alla durata della prenotazione. Se non ci sono prenotazioni, stampa un messaggio di avviso.

Precondizioni:

- La coda delle prenotazioni è stata inizializzata e accessibile tramite `ottieni_coda_prenotazioni()`.
- La lista dei veicoli è disponibile e accessibile tramite `ottieni_lista_veicoli()`.
- Le funzioni di stampa e calcolo (`stampa_prenotazione()`, `calcola_tariffa_prenotazione()`, `stampa_data_sistema()`) sono correttamente implementate.
- È possibile usare `imposta_colore()` per cambiare il colore del testo nel terminale.

Postcondizioni:

- Sullo schermo viene stampato un elenco delle prenotazioni presenti, con relativi dettagli e costo stimato.
- Se la coda è vuota, viene stampato un messaggio di “Nessuna prenotazione presente”.
- La funzione non modifica strutture dati interne o esterne.

Side Effect:

- Pulizia dello schermo e stampa su console.
- Cambio colore del testo sul terminale.
- Eventuale rilascio di memoria di veicoli temporanei (tramite `free(v)`).
- `visualizza_disponibilita() → void`

La funzione mostra a schermo una lista dei veicoli con il loro stato di disponibilità attuale, aggiornato in base alle prenotazioni correnti. Dopo aver elencato i veicoli, chiede all'utente di inserire l'ID di un veicolo e mostra il calendario delle prenotazioni per quel veicolo, evidenziando le fasce occupate e libere.

Precondizioni:

- Sono disponibili le funzioni per la gestione dello schermo, stampa, colori e input (es. `pulisci_schermo()`, `stampa_bordo_superiore()`, `imposta_color()`, `scanf()`, ecc.).
- Le funzioni di gestione prenotazioni e veicoli sono implementate correttamente (es. `inizializza_coda()`, `carica_prenotazioni_da_file()`, `ottieni_lista_veicoli()`, ecc.).
- Le strutture dati `CalendarioVeicolo` e `CodaPrenotazioni` sono definite e supportano le operazioni di inizializzazione, aggiornamento e visualizzazione.
- Il sistema è in grado di fornire la data e ora correnti (es. `ottieni_data_sistema()`).

Postcondizioni:

- A video viene mostrata una lista aggiornata dei veicoli con il loro stato di disponibilità attuale.

- Viene chiesto all'utente di inserire l'ID di un veicolo e successivamente viene mostrato il calendario dettagliato delle prenotazioni per quel veicolo.
- Lo stato di disponibilità del veicolo è temporaneamente modificato durante la visualizzazione ma viene sempre ripristinato allo stato originale.

Side Effect:

- Accesso e lettura di dati da file (prenotazioni).
 - Modifica temporanea dello stato di disponibilità nei dati veicoli (solo in memoria durante la funzione).
 - Stampa su console e cambio colore.
- `mostra_menu_cliente(utente_corrente) → void`

Stampa a video il menu principale riservato all'utente cliente, suddiviso in sezioni (Prenotazioni, Informazioni, Account), con le relative opzioni numerate e il nome dell'utente visualizzato in cima.

Precondizioni:

- `utente_corrente` è un puntatore valido a una struttura `Utente` con i dati dell'utente attualmente connesso.
- Funzione `ottieni_nome_complet_utente(Utente)` restituisce una stringa valida con il nome dell'utente.
- Sono disponibili funzioni per la gestione della grafica testuale e colori (`stampa_bordo_superiore()`, `stampa_separatore()`, `stampa_bordo_inferiore()`, `imposta_colore()`).

Postcondizioni:

- Viene stampato a video il menu cliente con intestazioni colorate, opzioni numerate, e il nome dell'utente.
- Nessuna modifica di dati o stato interno.

Side Effect:

- Output testuale sul terminale con utilizzo di colori.

- `mostra_menu_admin(utente_corrente) → void`

Stampa a video il menu principale riservato all'amministratore del sistema di car sharing, con varie sezioni per la gestione del sistema, il monitoraggio e le operazioni di sistema, includendo il nome dell'amministratore connesso.

Precondizioni:

- `utente_corrente` è un puntatore valido a una struttura `Utente` contenente i dati dell'utente amministratore connesso.
- La funzione `ottieni_nome_completo_utente(Utente)` restituisce una stringa valida col nome dell'utente.
- Sono disponibili funzioni per la gestione della grafica testuale e colori (`stampa_bordo_superiore()`, `stampa_separatore()`, `stampa_bordo_inferiore()`, `imposta_colore()`).

Postcondizioni:

- Viene mostrato a video un menu strutturato in sezioni, con intestazioni colorate e opzioni numerate.
- Non modifica dati o stato del programma.

Side Effect:

- Output testuale sul terminale con colori.

- `gestione_prenotazioni_amministratore() → void`

Gestisce il sottomenu amministrativo per le prenotazioni, permettendo di visualizzare, filtrare, ordinare e modificare lo stato delle prenotazioni registrate nel sistema.

Precondizioni:

- Le funzioni ausiliarie (`ottieni_coda_prenotazioni()`, `stampa_prenotazione()`, `modifica_stato_prenotazione()`, `salva_prenotazioni_su_file()`, ecc.) sono implementate correttamente e restituiscono dati coerenti.
- La coda di prenotazioni contiene dati validi.

- Le funzioni di gestione memoria e I/O (`malloc()`, `free()`, `scanf()`, `printf()`, `svuota_buffer()`) funzionano correttamente.
- L'utente ha privilegi amministrativi (non verificato direttamente in questa funzione, deve essere garantito da chiamante).

Postcondizioni:

- Eventuali modifiche allo stato delle prenotazioni vengono salvate su file.
- L'output video mostra liste e filtri di prenotazioni a seconda della scelta.
- La funzione si ripete fino alla scelta di uscita (`0`).

Side Effect:

- Input e output testuali tramite terminale.
 - Allocazione e liberazione di memoria dinamica temporanea.
 - Modifica della struttura dati delle prenotazioni se si cambia stato.
- `mostra_logo() → void`

La funzione stampa a video un logo in ASCII art che rappresenta il nome e l'identità grafica del sistema di car sharing. Il logo viene mostrato in colore giallo per renderlo visivamente evidente. Alla fine della stampa, il colore del testo viene riportato a bianco.

Precondizioni:

- La console o terminale deve supportare la stampa a video e i codici colore.
- La funzione `imposta_colore(int)` deve essere definita e funzionante per modificare il colore del testo in output.

Postcondizioni:

- Sullo schermo è visibile il logo del sistema car sharing stampato in giallo.
- Il colore del testo viene resettato a bianco (`imposta_colore(7)`).
- Nessuna variabile o struttura dati viene modificata.

Side Effect:

- Cambia temporaneamente il colore del testo in console.
 - Produce output testuale a schermo tramite `printf`.
 - Non modifica dati globali o parametri esterni.
-
- `mostra_menu_login() → void`

La funzione visualizza a schermo il menu iniziale di login del sistema di car sharing, con le opzioni per accedere, registrarsi o uscire dal programma. Il menu è formattato con bordi e separatori, utilizza colori diversi per evidenziare sezioni e testo, e include il logo del sistema.

Precondizioni:

- La console o terminale deve supportare l'output testuale e i codici colore ANSI o equivalenti.
- Le funzioni `stampa_bordo_superiore()`, `stampa_bordo_inferiore()`, `stampa_separatore()`, `imposta_color(int)` e `mostra_logo()` devono essere definite e funzionanti.

Postcondizioni:

- Sullo schermo è visibile un menu strutturato e colorato con il logo, il titolo "BENVENUTO", e le opzioni di accesso, registrazione e uscita.
- Nessun dato di programma viene modificato.

Side Effect:

- Produce output testuale e grafico (bordo, testo colorato) a schermo tramite `printf` e altre funzioni di stampa.
- Modifica temporaneamente il colore del testo con `imposta_colore`.
- Non modifica variabili o strutture dati globali o passate per riferimento.

- gestione_utenti_amministratore() → void

Gestisce il sottomenu amministrativo valido per la gestione degli utenti, permettendo di visualizzare, filtrare, aggiungere, modificare e rimuovere utenti registrati nel sistema.

Precondizioni:

- Le funzioni ausiliarie (`ottieni_lista_utenti()`, `stampa_utente()`, `aggiungi_utente()`, `rimuovi_utente()`, `salva_utenti_su_file()`, ecc.) sono implementate correttamente e restituiscono dati coerenti.
- La lista degli utenti contiene dati validi.
- Le funzioni di gestione memoria e I/O funzionano correttamente.
- L'utente ha privilegi amministrativi (non verificato direttamente da questa funzione, deve essere garantito dal chiamante).

Postcondizioni:

- Eventuali modifiche agli utenti (aggiunta, modifica, rimozione) vengono salvate su file.
- L'output video mostra liste, filtri e dettagli degli utenti a seconda della scelta.
- La funzione si ripete fino alla scelta di uscita (`0`).

Side Effect:

- Input e output testuali tramite terminale
- Allocazione e liberazione di memoria dinamica temporanea.
- Modifica della struttura dati degli Utenti se si aggiunge, modifica o rimuove un utente.

Data_sistema

Specifica sintattica:

- `inizializza_data_sistema(void) → void`
- `avanza_tempo(int) → void`
- `ottieni_data_sistema(void) → DataSistema`
- `converti_data_in_timestamp(DataSistema) → int`
- `calcola_priorita_temporale(int) → int`
- `ottieni_nome_giorno(int) → const char*`
- `ottieni_giorno_sistema(DataSistema) → int`
- `ottieni_ora_sistema(DataSistema) → int`
- `ottieni_ora_corrente(void) → int`

- `ottieni_giorno_corrente(void) → int`

Specifica Semantica:

- `inizializza_data_sistema() → void`

Inizializza la data e l'ora del sistema impostandole a Lunedì alle 8:00.

Precondizioni:

- `data_corrente` è un puntatore valido a una struttura `DataSistema`.

Postcondizioni:

- `data_corrente->giorno` è impostato a 0 (Lunedì).
- `data_corrente->ora` è impostato a 8 (8:00).

Side Effect:

- Modifica diretta dei campi giorno e ora nella struttura puntata da `data_corrente`.

- `avanza_tempo(ore) → void`

Avanza l'orario del sistema di un numero di ore specificato, aggiornando giorno e ora con rollover appropriati.

Precondizioni:

- `ore` è un intero maggiore o uguale a 0.
- `data_corrente` è un puntatore valido a una struttura `DataSistema`.

Postcondizioni:

- `data_corrente->ora` è incrementato di ore, con rollover a 0 dopo 23.
- Se l'ora supera le 23, il giorno viene incrementato di conseguenza, con rollover a 0 dopo il sesto giorno (domenica).

Side Effect:

- Modifica diretta dei campi giorno e ora nella struttura puntata da `data_corrente`.

- `ottieni_data_sistema()` → `data_corrente`

Restituisce la data e l'ora correnti del sistema.

Precondizioni:

- `data_corrente` è un puntatore valido a una struttura `DataSistema`.

Postcondizioni:

- Restituisce il valore corrente della struttura `DataSistema` puntata da `data_corrente`.

Side Effect: Nessuno.

- `converti_data_in_timestamp(data)` → $(data \rightarrow \text{giorno} * 24) + data \rightarrow \text{ora}$

Converte una data (giorno e ora) in un timestamp unico, espresso in ore totali dall'inizio della settimana.

Precondizioni:

- `data` è un puntatore valido a una struttura `DataSistema` con campi `giorno` e `ora`.
- `giorno` è compreso tra 0 e 6 (inclusi).
- `ora` è compreso tra 0 e 23 (inclusi).

Postcondizioni:

- Restituisce un intero pari a $data \rightarrow \text{giorno} * 24 + data \rightarrow \text{ora}$.

Side Effect: Nessuno.

- `calcola_priorita_temporale(timestamp_prenotazione)` → `differenza`

Calcola la priorità temporale di una prenotazione rispetto alla data e ora correnti del sistema.

Precondizioni:

- `timestamp_prenotazione` è un intero non negativo.

- `data_corrente` è un puntatore valido a una struttura `DataSistema`.

Postcondizioni:

- Calcola la differenza `differenza = timestamp_prenotazione - converti_data_in_timestamp(data_corrente)`.
- Se `differenza < 0`, restituisce 0 (priorità massima).
- Altrimenti, restituisce `differenza` (più grande è, più bassa è la priorità).

Side Effect: Nessuno

- `ottieni_nome_giorno(giorno) → giorni[giorno]`

Restituisce il nome del giorno della settimana corrispondente all'intero `giorno`.

Precondizioni:

- `giorno` è un intero compreso tra 0 e 6 (0 = Lunedì, ..., 6 = Domenica).

Postcondizioni:

- Se `giorno` è valido, restituisce il nome del giorno corrispondente come stringa costante.
- Altrimenti, restituisce la stringa "Giorno non valido".

Side Effect: Nessuno.

- `ottieni_giorno_sistema(data) → data->giorno`

Restituisce il giorno contenuto nella struttura `DataSistema` passata come parametro.

Precondizioni:

- `data` è un puntatore valido a una struttura `DataSistema` oppure NULL.

Postcondizioni:

- Se `data` è valido, restituisce il valore di `data->giorno`.
- Se `data` è NULL, restituisce -1.

Side Effect: Nessuno

- `ottieni_ora_sistema(data) → data->ora`

Restituisce l'ora contenuta nella struttura `DataSistema` passata come parametro.

Precondizioni:

- `data` è un puntatore valido a una struttura `DataSistema` oppure `NULL`.

Postcondizioni:

- Se `data` è valido, restituisce il valore di `data->ora`.
- Se `data` è `NULL`, restituisce `-1`.

Side Effect: Nessuno.

- `ottieni_giorno_corrente() → data_corrente->giorno`
- Restituisce il giorno corrente del sistema.

Precondizioni:

- `data_corrente` è un puntatore valido a una struttura `DataSistema`.

Postcondizioni:

- Restituisce il valore di `data_corrente->giorno`.

Side effect: Nessuno.

- `ottieni_ora_corrente() → data_corrente->ora`

Restituisce l'ora corrente del sistema.

Precondizioni:

- `data_corrente` è un puntatore valido a una struttura `DataSistema`.

Postcondizioni:

- Restituisce il valore di `data_corrente->ora`.

Side Effect: Nessuno.

F_utili

Specifiche Sintattiche:

- `imposta_colore(int) → void`
- `svuota_buffer(void) → void`
- `pulisci_schermo(void) → void`
- `salvataggio(void) → void`
- `stampa_separatore(void) → void`
- `stampa_bordo_superiore(void) → void`
- `stampa_bordo_inferiore(void) → void`
- `stampa_data_sistema(void) → void`

Specifiche Semantiche:

- `void imposta_colore(int color)`

Imposta il colore del testo stampato nel terminale, compatibile sia con Windows che con sistemi Unix-like.

Precondizioni:

- `colore` è un intero, definite anche MACRO con diversi colori, compatibile con i codici colore standard di Windows (`SetConsoleTextAttribute`) o codici ANSI su Unix/Linux/Mac.

Postcondizioni:

- Il colore del testo stampato successivamente cambia secondo il valore di `color`.

Side Effect:

- Modifica temporaneamente l'aspetto dell'output sul terminale.
- Emette sequenze ANSI o chiama API di Windows.

- `void svuota_buffer()`

Pulisce il buffer di input standard (stdin), scartando tutti i caratteri fino al prossimo newline incluso.

Precondizioni:

- Nessuna, ma ha senso solo se chiamata dopo una lettura da stdin che potrebbe lasciare caratteri residui (es. scanf).

Postcondizioni:

- Il buffer di input viene svuotato fino al primo newline ('\n'), evitando problemi nelle successive letture da input.

Side effect:

- Consuma dati da stdin.

`void pulisci_schermo()`

Cancella il contenuto visibile del terminale, in modo cross-platform.

Precondizioni:

- Nessuna.

Postcondizioni:

- Il contenuto dello schermo viene cancellato.
- Il cursore ritorna in cima.

Side effect:

- Esegue un comando di sistema (`system("cls")` o `system("clear")`).

- `void salvataggio()`

Salva lo stato del sistema (lista veicoli e coda prenotazioni, tabella utenti) su file, e libera la memoria dinamica associata alla lista dei veicoli.

Precondizioni:

- Le funzioni `salva_lista_veicoli()`, `ottieni_coda_prenotazioni()` e `salva_prenotazioni_su_file()` e `salva_utenti_file()` devono essere definite e funzionanti.
- `pulisci_lista_veicoli()` deve liberare correttamente la memoria.

Postcondizioni:

- I dati vengono persistiti su file.
- La memoria della lista veicoli è liberata.

Side effect:

- Scrive su file.
- Libera memoria dinamica.

- `void stampa_bordo_superiore()`

Stampa una riga decorativa (bordo superiore), colorata in ciano.

Precondizioni: Nessuna.

Postcondizioni:

- Una linea ===== viene stampata con colore ciano, poi il colore viene resettato.

Side effect:

- Output su console con colore.

- `void stampa_bordo_inferiore()`

Stampa una riga decorativa (bordo inferiore), uguale alla superiore.

Precondizioni: Nessuna.

Postcondizioni:

- Una linea ===== viene stampata in ciano.

Side effect:

- Output su console con colore.

- void stampa_separatore()

Stampa una riga separatrice decorativa in ciano, più corta del bordo.

Precondizioni: Nessuna.

Postcondizioni:

- Una linea ----- viene stampata.

Side effect:

- Output su console con colore.

- stampa_data_sistema() → void

La funzione stampa sullo standard output la data di sistema corrente, ottenuta tramite `ottieni_data_sistema()`, in formato leggibile: il giorno della settimana (come stringa) e l'ora in formato `hh:00`.

Precondizioni: Nessuna.

Postcondizioni:

- Viene stampata sul terminale una riga con la data corrente del sistema nel formato " `Giorno, ore hh:00`".

Side effect:

- Output su `stdout`.

vi. Razionale dei Casi di Test

Documentazione Errori Test Case

Questo documento descrive tutti i possibili errori che possono essere generati durante l'esecuzione dei test case.

Errori di Fascia Oraria

...

ERRORE_FASCIA_ORARIA

...

Generato quando:

- Il giorno di inizio è < 0 o > 6
- Il giorno di fine è < 0 o > 6
- L'ora di inizio è < 0 o > 23
- L'ora di fine è < 0 o > 23
- Il timestamp di inizio è \geq timestamp di fine

Errori di Data

...

ERRORE_DATA_1

...

Generato quando:

- La data di inizio è nel passato rispetto alla data di sistema
- Esempio: data sistema = Lunedì 8:00, prenotazione inizia Lunedì 7:00

...

ERRORE_DATA_2

...

Generato quando:

- La data di fine è precedente o uguale alla data di inizio
- Esempio: inizio = Lunedì 10:00, fine = Lunedì 9:00

Errori di Utente

...

ERRORE_UTENTE_NON_TROVATO

...

Generato quando:

- L'ID utente specificato non esiste nel sistema
- Esempio: tentativo di prenotazione con id_utente = 999 quando non esiste

Errori di Veicolo

...

ERRORE_VEICOLO_NON_TROVATO

...

Generato quando:

- L'ID veicolo specificato non esiste nel sistema
- Esempio: tentativo di prenotazione con id_veicolo = 999 quando non esiste

Errori di posizione non valida

...

ERRORE_POSIZIONE_RICONSEGNA

...

Generato quando:

- La posizioni di riconsegna non è una di quelle esistenti
- Esempio: tentativo di prenotazione con posizione_riconsegna = 999 quando non esiste

Test Crea Prenotazione

Formato Input

Ogni test case accetta 8 valori in input:

...

id_utente id_veicolo giorno_inizio ora_inizio giorno_fine ora_fine priorit
posizione_riconsegna

...

Dove:

- id_utente: intero positivo
- id_veicolo: intero positivo
- giorno_inizio: 0-6 (Lunedì-Domenica)
- ora_inizio: 0-23
- giorno_fine: 0-6 (Lunedì-Domenica)
- ora_fine: 0-23
- priorit: -1 per calcolo automatico, altrimenti intero positivo
- posizione_riconsegna: intero positivo

Formato Output

In caso di successo, l'output sarà:

...

id_prenotazione id_utente id_veicolo timestamp_inizio timestamp_fine stato priorit
posizione_riconsegna

...

In caso di errore, l'output sarà uno dei messaggi di errore descritti sopra.

Test Costo Noleggio

Formato Input

Ogni test accetta 6 valori in input:

...

id_prenotazione id_veicolo giorno_inizio ora_inizio giorno_fine ora_fine

...

Dove:

- id_prenotazione: intero positivo
- id_veicolo: intero positivo
- giorno_inizio: 0-6 (Lunedì-Domenica)
- ora_inizio: 0-23
- giorno_fine: 0-6 (Lunedì-Domenica)
- ora_fine: 0-23

Formato Output

In caso di successo l'output sarà:

...

costo

...

In caso di errore, l'output sarà uno dei messaggi di errore descritti sopra.

Test Visualizza Disponibilità

Formato Input

Ogni Test accetta 4 valori:

...

giorno_inizio ora_inizio giorno_fine ora_fine

...

Dove:

- giorno_inizio: 0-6 (Lunedì-Domenica)
- ora_inizio: 0-23
- giorno_fine: 0-6 (Lunedì-Domenica)
- ora_fine: 0-23

Setup Veicoli

Prenotazioni Confermate:

Veicolo 1 (Fiat500, id=1)

- giorno 1, 10-12 (martedì 10-12)
- giorno 3, 6-7 (giovedì 6-7)

Veicolo 2 (HondaCBR, id=2)

- giorno 2, 14-16 (mercoledì 14-16)
- giorno 3, 2-3 (giovedì 2-3)
- giorno 3, 6-7 (giovedì 6-7)

Veicolo 3 (JeepWrangler, id=3)

- giorno 3, 2-3 (giovedì 2-3)
- giorno 3, 6-7 (giovedì 6-7)

Veicolo 4 (TeslaModelS, id=4)

- giorno 3, 3-4 (giovedì 3-4)
- giorno 3, 6-7 (giovedì 6-7)

Formato Output

In caso di successo l'output sarà:

...

1. Descrizione veicoli disponibili:

id_veicolo modello posizione

2. Nessun veicolo se non disponibili

...

In caso di errore, l'output sarà uno dei messaggi di errore descritti sopra.

Test Storico Prenotazioni

Formato Input

Ogni test accetta 1 valore in input:

...

id_utente

...

dove:

id_utente = intero positivo

Formato Output

In caso di successo l'output sarà:

...

id_prenotazione id_utente id_veicolo timestamp_inizio timestamp_fine stato priorita
posizione_riconsegna

id_prenotazione id_utente id_veicolo timestamp_inizio timestamp_fine stato priorita
posizione_riconsegna

ecc...

...

In caso di errore, l'output sarà uno dei messaggi di errore descritti sopra.

Schema degli Input per Tipo di Test

Test Creazione Prenotazione (TC01, TC13)

Input: 8 argomenti

1. id_utente (int)
2. id_veicolo (int)
3. giorno_inizio (int)
4. ora_inizio (int)
5. giorno_fine (int)
6. ora_fine (int)
7. tipo_prenotazione (int)
8. id_prenotazione (int)

Test Calcolo Costo Noleggio (TC14, TC28)

Input: 6 argomenti

1. id_prenotazione (int)
2. id_veicolo (int)
3. giorno_inizio (int)
4. ora_inizio (int)
5. giorno_fine (int)
6. ora_fine (int)

Test Visualizza Disponibilita (TC29, TC38)

Input: 4 argomenti

1. giorno_inizio (int)
2. ora_inizio (int)
3. giorno_fine (int)

4. ora_fine (int)

Test Storico Prenotazioni (TC39, TC42)

Input: 1 argomento

1. id_utente (int)

Test Case

TC01: Creazione Prenotazione Valida

- Input:

id_utente: 1

id_veicolo: 2

giorno_inizio: 0

ora_inizio: 10

giorno_fine: 1

ora_fine: 10

priorità: -1

posizione_riconsegna: 0

- Output Atteso: Dettagli della prenotazione creata

- Verifica: Il sistema deve creare correttamente la prenotazione

TC02: Creazione Prenotazione nel Passato

- Input:

id_utente: 2

id_veicolo: 2

giorno_inizio: 0

ora_inizio: 0

giorno_fine: 1

ora_fine: 0

priorità: -1

posizione_riconsegna: 0

- Output Atteso: "ERRORE_DATA_1" (data nel passato)

- Verifica: Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC03: Creazione Prenotazione Data di fine precedente a Data inizio

- Input:

id_utente: 1

id_veicolo: 2

giorno_inizio: 1

ora_inizio: 12

giorno_fine: 1

ora_fine: 10

priorità: -1

posizione_riconsegna: 0

- Output Atteso: "ERRORE_FASCIA_ORARIA"

- **Verifica:** Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC04: Creazione Prenotazione Utente non esistente

- **Input:**

id_utente: 999
id_veicolo: 2
giorno_inizio: 1
ora_inizio: 10
giorno_fine: 1
ora_fine: 12
priorità: -1
posizione_riconsegna: 0

- **Output Atteso:** "ERRORE_UTENTE_NON_TROVATO"

- **Verifica:** Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC05: Creazione Prenotazione Veicolo non esistente

- **Input:**

id_utente: 1
id_veicolo: 999
giorno_inizio: 1
ora_inizio: 10
giorno_fine: 1
ora_fine: 12
priorità: -1
posizione_riconsegna: 0

- **Output Atteso:** "ERRORE_VEICOLO_NON_TROVATO"

- **Verifica:** Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC06: Creazione Prenotazione con Giorno fuori range

- **Input:**

id_utente: 1
id_veicolo: 2
giorno_inizio: 7
ora_inizio: 10
giorno_fine: 8
ora_fine: 12
priorità: -1
posizione_riconsegna: 0

- **Output Atteso:** "ERRORE_FASCIA_ORARIA"

- **Verifica:** Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC07: Creazione Prenotazione con Ora fuori range

- **Input:**

id_utente: 1
id_veicolo: 2
giorno_inizio: 1
ora_inizio: 24
giorno_fine: 1
ora_fine: 26
priorità: -1
posizione_riconsegna: 0

- **Output Atteso:** "ERRORE_FASCIA_ORARIA"

- **Verifica:** Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC08: Creazione Prenotazione con priorità manuale

- **Input:**

id_utente: 1
id_veicolo: 2
giorno_inizio: 1
ora_inizio: 10
giorno_fine: 1
ora_fine: 12
priorità: 5
posizione_riconsegna: 0

- **Output Atteso:** Dettagli della prenotazione creata con priorità impostata a 5

- **Verifica:** Il sistema deve creare correttamente la prenotazione

TC09: Creazione Prenotazione che copre tutta la settimana da data avvio sistema

- **Input:**

id_utente: 1
id_veicolo: 2
giorno_inizio: 0
ora_inizio: 8
giorno_fine: 6
ora_fine: 23
priorità: -1
posizione_riconsegna: 0

- **Output Atteso:** Dettagli della prenotazione creata

- **Verifica:** Il sistema deve creare correttamente la prenotazione

TC10: Creazione Prenotazione che copre tutta la settimana

- **Input:**

id_utente: 1
id_veicolo: 2
giorno_inizio: 0
ora_inizio: 0
giorno_fine: 6
ora_fine: 23

priorità: -1
posizione_riconsegna: 0
- **Output Atteso:** "ERRORE_DATA_1" (Data inizio precedente a data sistema)
- **Verifica:** Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC11: Creazione Prenotazione con durata zero

- **Input:**
id_utente: 1
id_veicolo: 2
giorno_inizio: 2
ora_inizio: 15
giorno_fine: 2
ora_fine: 15
priorità: -1
posizione_riconsegna: 0
- **Output Atteso:** "ERRORE_FASCIA_ORARIA"
- **Verifica:** Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC12: Creazione Prenotazione con posizione riconsegna diversa

- **Input:**
id_utente: 1
id_veicolo: 2
giorno_inizio: 3
ora_inizio: 8
giorno_fine: 3
ora_fine: 10
priorità: -1
posizione_riconsegna: 2
- **Output Atteso:** 18 1 2 80 82 0 72 2
- **Verifica:** Il sistema deve creare correttamente la prenotazione, con la posizione di riconsegna nuova

TC13: Creazione Prenotazione con posizione riconsegna non esistente

- **Input:**
id_utente: 1
id_veicolo: 2
giorno_inizio: 3
ora_inizio: 8
giorno_fine: 3
ora_fine: 10
priorità: -1
posizione_riconsegna: 999
- **Output Atteso:** ERRORE_POSIZIONE_RICONSEGNA
- **Verifica:** Il sistema deve rifiutare la prenotazione e restituire un messaggio di errore appropriato

TC14: Calcolo Costo Noleggio Valido Utilitaria (tipo: 0)

- Input:

id_utente: 1
id_veicolo: 1
giorno_inizio: 0
ora_inizio: 10
giorno_fine: 0
ora_fine: 12

- Output Atteso: "10.00" (5€/ora × 2 ore)

- Verifica: Il sistema deve calcolare correttamente il costo del noleggio di un'utilitaria

TC15: Calcolo Costo Noleggio con Utente Non Esistente

- Input:

id_utente: 1
id_veicolo: 999
giorno_inizio: 0
ora_inizio: 10
giorno_fine: 0
ora_fine: 12

- Output Atteso: "ERRORE_UTENTE_NON_TROVATO"

- Verifica: Il sistema deve gestire correttamente il caso di utente non trovato e restituire un messaggio di errore appropriato

TC16: Calcolo Costo Noleggio su più giorni

- Input:

id_utente: 1
id_veicolo: 1
giorno_inizio: 0
ora_inizio: 22
giorno_fine: 1
ora_fine: 2

- Output Atteso: "20.00"

- Verifica: Il sistema deve calcolare correttamente il costo del noleggio posto su due giorni diversi

TC17: Calcolo Costo Noleggio con durata zero

- Input:

id_utente: 1
id_veicolo: 1
giorno_inizio: 0
ora_inizio: 10
giorno_fine: 0
ora_fine: 10

- Output Atteso: ERRORE_FASCIA_ORARIA

- Verifica: Il sistema deve gestire il caso in cui ora_inizio == ora_fine

TC18: Calcolo Costo Noleggio con ora_fine < ora_inizio nello stesso giorno**- Input:**

id_utente: 1
id_veicolo: 1
giorno_inizio: 0
ora_inizio: 12
giorno_fine: 0
ora_fine: 10

- Output Atteso: ERRORE_FASCIA_ORARIA

- **Verifica:** Il sistema deve gestire il caso in cui ora_fine < ora_inizio nello stesso giorno (caso limite)

TC19: Calcolo Costo Noleggio con giorno_fine < giorno_inizio**- Input:**

id_utente: 1
id_veicolo: 1
giorno_inizio: 2
ora_inizio: 10
giorno_fine: 1
ora_fine: 12

- Output Atteso: ERRORE_FASCIA_ORARIA

- **Verifica:** Il sistema deve gestire il caso in cui giorno_fine < giorno_inizio (caso limite)

TC20: Calcolo Costo Noleggio con veicolo non esistente**- Input:**

id_utente: 1
id_veicolo: 999
giorno_inizio: 0
ora_inizio: 10
giorno_fine: 0
ora_fine: 12

- Output Atteso: ERRORE_VEICOLO_NON_TROVATO

- **Verifica:** Il sistema deve gestire il caso di veicolo non trovato.

TC21: Calcolo Costo Noleggio con orari fuori range**- Input:**

id_utente: 1
id_veicolo: 1
giorno_inizio: 0
ora_inizio: 25
giorno_fine: 0
ora_fine: 27

- Output Atteso: ERRORE_FASCIA_ORARIA

- **Verifica:** Il sistema deve gestire orari non validi.

TC22: Calcolo Costo Noleggio con giorni fuori range**- Input:**

id_utente: 1
id_veicolo: 1
giorno_inizio: 0
ora_inizio: 25
giorno_fine: 0
ora_fine: 27

- **Output Atteso:** ERRORE_FASCIA_ORARIA

- **Verifica:** Il sistema deve gestire giorni non validi.

TC23: Calcolo Costo Noleggio Valido SUV (tipo: 1)

- **Input:**

id_utente: 1
id_veicolo: 3
giorno_inizio: 0
ora_inizio: 10
giorno_fine: 0
ora_fine: 15

- **Output Atteso:** "40.00" ($8\text{€/ora} \times 5 \text{ ore}$)

- **Verifica:** Il sistema deve calcolare correttamente il costo del noleggio di un SUV

TC24: Calcolo Costo Noleggio Valido SPortiva (tipo: 2)TC24: Calcolo Costo Noleggio Valido Sportiva (tipo: 2)

- **Input:**

id_utente: 1
id_veicolo: 4
giorno_inizio: 0
ora_inizio: 10
giorno_fine: 0
ora_fine: 15

- **Output Atteso:** "62.50" ($12.50\text{€/ora} \times 5 \text{ ore}$)

- **Verifica:** Il sistema deve calcolare correttamente il costo del noleggio di una Sportiva

TC25: Calcolo Costo Noleggio Valido Moto (tipo: 3)

- **Input:**

id_utente: 1
id_veicolo: 2
giorno_inizio: 0
ora_inizio: 10
giorno_fine: 0
ora_fine: 15

- **Output Atteso:** "20.00" ($4\text{€/ora} \times 5 \text{ ore}$)

- **Verifica:** Il sistema deve calcolare correttamente il costo del noleggio di una moto

TC26: Calcolo Costo Noleggio Valido Utilitaria (tipo: 0) per 24 ore

- **Input:**

id_utente: 1
id_veicolo: 1

giorno_inizio: 1
ora_inizio: 0
giorno_fine: 2
ora_fine: 0

- **Output Atteso:** "100.00" ($5\text{€}/\text{ora} \times (24 \text{ ore} - 4 \text{ ore di sconto})$), si applica lo sconto "PACCHETTI ORARI", in cui ogni 5 ore noleggiate la sesta è gratuita, quindi su 24 ore 4 saranno gratuite, e se ne pagheranno 20)
- **Verifica:** Il sistema deve calcolare correttamente il costo del noleggio di una Utilitaria applicando lo sconto "PACCHETTI ORARI"

TC27: Calcolo Costo Noleggio Valido Utilitaria (tipo: 0) per 6 giorni

- Input:

id_utente: 1
id_veicolo: 1
giorno_inizio: 0
ora_inizio: 23
giorno_fine: 6
ora_fine: 23

- **Output Atteso:** "600.00" ($5\text{€}/\text{ora} \times (144 \text{ ore} - 24 \text{ ore di sconto})$), si applica lo sconto "PACCHETTI ORARI", in cui ogni 5 ore noleggiate la sesta è gratuita, quindi su 144 ore 24 saranno gratuite, e se ne pagheranno 120)
- **Verifica:** Il sistema deve calcolare correttamente il costo del noleggio di una Utilitaria applicando lo sconto "PACCHETTI ORARI"

TC28: Calcolo Costo Noleggio Valido Utilitaria (tipo: 0) per 6 giorni, avendo completato in precedenza già 10 noleggi

- Input:

id_utente: 2
id_veicolo: 1
giorno_inizio: 0
ora_inizio: 23
giorno_fine: 6
ora_fine: 23

- **Output Atteso:** "540.00" ($5\text{€}/\text{ora} \times (144 \text{ ore} - 24 \text{ ore di sconto}) - 10\%$), si applica lo sconto "PACCHETTI ORARI", in cui ogni 5 ore noleggiate la sesta è gratuita, quindi su 144 ore 24 saranno gratuite, inoltre si applica un 10% di sconto siccome l'utente in questione ha 10 noleggi completati, per cui si applica lo sconto "SCONTO FEDELTA'" che applica un 10% di sconto all'11 noleggio effettuato, dopo 10 completati)
- **Verifica:** Il sistema deve calcolare correttamente il costo del noleggio di una Utilitaria applicando lo sconto "PACCHETTI ORARI" e "SCONTO FEDELTA'"

TC29: Visualizzazione Disponibilità con tutti i Veicoli Disponibili

- Input:

- giorno_inizio: 2
- ora_inizio: 8
- giorno_fine: 2

- ora_fine: 10
- **Output Atteso:**
 - 4 TeslaModelS Posizione D
 - 3 JeepWrangler Posizione C
 - 2 HondaCBR Posizione B
 - 1 Fiat500 Deposito
- **Verifica:** Il sistema deve:
 - Mostrare correttamente i veicoli disponibili nel periodo richiesto.

TC30: Visualizzazione Disponibilità con nessun Veicolo Disponibile

- **Input:**
 - giorno_inizio: 3
 - ora_inizio: 6
 - giorno_fine: 3
 - ora_fine: 7
- **Output Atteso:**
- **Verifica:** Il sistema deve:
 - Non mostrare nessun Veicolo.

TC31: Visualizzazione Disponibilità con 3 Veicoli Disponibili

- **Input:**
- **Input:**
 - giorno_inizio: 2
 - ora_inizio: 14
 - giorno_fine: 2
 - ora_fine: 16
- **Output Atteso:**
 - 4 TeslaModelS Posizione D
 - 3 JeepWrangler Posizione C
 - 1 Fiat500 Deposito
- **Verifica:** Il sistema deve:
 - Mostrare correttamente i veicoli disponibili nel periodo richiesto

TC32: Visualizzazione Disponibilità con 3 Veicoli Disponibili

- **Input:**
 - giorno_inizio: 1
 - ora_inizio: 10
 - giorno_fine: 1
 - ora_fine: 12
- **Output Atteso:**
 - 4 TeslaModelS Posizione D
 - 3 JeepWrangler Posizione C
 - 2 HondaCBR Posizione B
- **Verifica:** Il sistema deve:
 - Mostrare correttamente i veicoli disponibili nel periodo richiesto

TC33: Visualizzazione Disponibilità con 3 Veicoli Disponibili

- **Input:**

- **Input:**

- giorno_inizio: 3

- ora_inizio: 3

- giorno_fine: 3

- ora_fine: 4

- **Output Atteso:**

3 JeepWrangler Posizione C

2 HondaCBR Posizione B

1 Fiat500 Deposito

- **Verifica:** Il sistema deve:

Mostrare correttamente i veicoli disponibili nel periodo richiesto

TC34: Visualizzazione Disponibilità con 3 Veicoli Disponibili

- **Input:**

- **Input:**

- giorno_inizio: 3

- ora_inizio: 2

- giorno_fine: 3

- ora_fine: 3

- **Output Atteso:**

4 TeslaModelS Posizione D

2 HondaCBR Posizione B

1 Fiat500 Deposito

- **Verifica:** Il sistema deve:

Mostrare correttamente i veicoli disponibili nel periodo richiesto

TC35: Visualizzazione Disponibilità con 2 Veicoli Disponibili

- **Input:**

- **Input:**

- giorno_inizio: 3

- ora_inizio: 2

- giorno_fine: 3

- ora_fine: 4

- **Output Atteso:**

2 HondaCBR Posizione B

1 Fiat500 Deposito

- **Verifica:** Il sistema deve:

Mostrare correttamente i veicoli disponibili nel periodo richiesto

TC36: Visualizzazione Disponibilità con Giorno fuori range

- **Input**

- **Input:**

- giorno_inizio: 7

- ora_inizio: 10

- giorno_fine: 8

- ora_fine: 12

- **Output Atteso:** ERRORE_FASCIA_ORARIA
- **Verifica:** Il sistema deve gestire orari non validi

TC37: Visualizzazione Disponibilità con Giorno fuori range

- **Input:**
- **Input:**
 - giorno_inizio: 1
 - ora_inizio: 24
 - giorno_fine: 1
 - ora_fine: 25
- **Output Atteso:** ERRORE_FASCIA_ORARIA
- **Verifica:** Il sistema deve gestire orari non validi

TC38: Visualizzazione Disponibilità con stessa ora

- **Input:**
 - giorno_inizio: 2
 - ora_inizio: 4
 - giorno_fine: 2
 - ora_fine: 4
- **Output Atteso:** ERRORE_FASCIA_ORARIA
- **Verifica:** Il sistema deve gestire orari non validi

TC39: Visualizzazione Storico Prenotazioni Utente 1

- **Input:**
 - id_utente: 1
- **Output Atteso:** Tutte le prenotazioni dell'utente con id_utente 1, vengono stampate
- **Verifica:** Il sistema deve:
 1. Mostrare correttamente tutte le prenotazioni dell'utente con id_utente 1
 2. Non mostrare le prenotazioni di altri utenti
 3. Includere tutti i dettagli della prenotazione (id, veicolo, date, stato, priorità, posizione)
 4. Gestire correttamente il formato di output per ogni prenotazione

TC40: Visualizzazione Storico Prenotazioni Utente 2

- **Input:**
 - id_utente: 2
- **Output Atteso:** Tutte le prenotazioni dell'utente con id_utente 2, vengono stampate
- **Verifica:** Il sistema deve:
 1. Mostrare correttamente tutte le prenotazioni dell'utente con id_utente 2
 2. Non mostrare le prenotazioni di altri utenti
 3. Includere tutti i dettagli della prenotazione (id, veicolo, date, stato, priorità, posizione)
 4. Gestire correttamente il formato di output per ogni prenotazione

TC41: Visualizzazione Storico Prenotazioni con Utente non esistente

- **Input:**
 - id_utente: 999
- **Output Atteso:** "ERRORE_UTENTE_NON_TROVATO"

- **Verifica:** Il sistema deve stampare un messaggio di errore

TC42: Visualizzazione Storico Prenotazioni da Admin

- **Input:**

- id_utente: 0

- **Output Atteso:** Tutte le prenotazioni degli utenti registrati, vengono stampate

- **Verifica:** Il sistema deve:

1. Mostrare correttamente tutte le prenotazioni degli utenti registrati
2. Includere tutti i dettagli della prenotazione (id, veicolo, date, stato, priorità, posizione)
3. Gestire correttamente il formato di output per ogni prenotazione