REPORT PROGETTO P.S.D. A.A. 2024/2025 - PEU-Z

Traccia 1: Sistema di gestione delle prenotazioni per servizi di car-sharing

IMPLEMENTAZIONE DEGLI ABSTRACT DATA TYPES (ADT)

Nell'elaborazione del codice sono stati utilizzati i seguenti ADT:

- *Orario*
- *Veicolo*
- *Prenotazione*
- *Tabella Hash*

ADT ORARIO

Per l'ADT *Orario* è stata scelta come implementazione una struttura i cui campi descrivono i dettagli della fascia oraria quali:

- **inizio**: definisce l'orario di inizio della fascia oraria.
- **fine**: definisce l'orario di fine della fascia oraria.
- **Disponibilità**: definisce lo stato della disponibilità riferita alla fascia oraria.

MOTIVAZIONE ADT ORARIO

L'ADT *Orario* è stato implementato attraverso un **array statico** di strutture, a cui ogni indice corrisponde ad una fascia oraria.

La dimensione dell'array rappresenta il numero di fasce orarie selezionabili durante l'esecuzione del programma, ed è scelta basandosi sulle fasce presenti su file "*Orari.txt*".

L'utilizzo di un array statico deriva dalla scelta implementativa che non prevede l'aggiunta di nuove fasce orarie per il servizio, da cui la dimensione fissa "*8*".

ADT VEICOLO

Per l'ADT *Veicolo* è stata scelta come implementazione una struttura i cui campi descrivono i dettagli del Veicolo quali:

- **tipoVeicolo**: definisce la categoria del veicolo in base alle sue funzioni caratteristiche.
- **modello**: definisce il produttore e il prototipo del veicolo.
- **colore**: definisce la tonalità del veicolo.
- **targa**: definisce l'identificativo univoco rappresentativo del veicolo.
- **orari**: definisce l'insieme delle fasce orarie prenotabili del veicolo.
- **postiOmologati**: definisce il limite di passeggeri del veicolo.

- **Combustibile**: definisce la tipologia di carburante del veicolo.
- **annoDiImmatricolazione**: definisce l'anno di registrazione del veicolo.
- **costoNoleggioOrario**: definisce il costo per ora del veicolo.

MOTIVAZIONE ADT VEICOLO

L'ADT *Veicolo* è stato implementato attraverso di un array statico di strutture, a cui ogni indice corrisponde un Veicolo.

La dimensione dell'array rappresenta i Veicoli presenti sull'ipotetico mercato ed è scelta al fine di simulare un database basandosi sulle vetture presenti su file *"Veicoli.txt"*.

L'utilizzo di un array statico deriva dalla scelta implementativa che non prevede l'aggiunta di nuovi Veicoli per il servizio, da cui la dimensione fissa "*VEICOLI_TAGLIA*".

ADT PRENOTAZIONE

Per l'ADT *Prenotazione* è stata scelta l'implementazione di una struttura i cui campi descrivono i dettagli della prenotazione quali:

- **ID**: definisce l'identificativo univoco della prenotazione.
- **nomeUtente**: definisce il nome dell'utente che effettua la prenotazione.
- **v**: definisce il veicolo prenotato.
- **data**: definisce la data corrispondente alla creazione della prenotazione.
- **CostoNoleggioFinale**: definisce il calcolo finale del noleggio in base alla fascia oraria scelta.
- **OrarioSceltoInizio**: indica l'orario di inizio della prenotazione.
- **OrarioSceltoFine**: indica l'orario di fine della prenotazione.
- **next**: puntatore alla prossima prenotazione, per la gestione in caso di collisioni.

MOTIVAZIONE ADT PRENOTAZIONE

L'ADT *Prenotazione* è stato implementato come oggetto che verrà poi utilizzato per la costruzione della Tabella hash contentene le prenotazioni degli utenti.

Tra i campi dell'ADT troviamo *next*, un campo vantaggioso che è un puntatore a tipo *Prenotazione*, utilizzato in caso di collisioni all'interno della tabella hash.

ADT TABELLA HASH

Per l'ADT *Tabella Hash* è stata scelta l'implementazione di una struttura i cui campi definiscono una tabella hash archetipa:

- **taglia**: definisce la dimensione della tabella hash.
- **tabella**: puntatore ad un array di puntatori a tipo Prenotazione.

MOTIVAZIONE ADT TABELLA HASH

L'ADT *Tabella Hash* è stato implementato attraverso l'archetipa tabella hash che utilizza il metodo delle liste concatenate per la gestione di collisioni.

La scelta della tabella hash per gestire le prenotazioni è vantaggiosa per la sua *facilità di ricerca* utilizzando una chiave (**ID**) e per la sua *gestione dinamica della memoria*, che permette di ridurre gli sprechi di memoria.

PROGETTAZIONE DEL SISTEMA E DINAMICHE TRA COMPONENTI

MENU DI ACCESSO

Nella fase iniziale di avvio del programma, viene chiamata la funzione **menuAccesso** che mostra sullo stdout il menu di accesso al servizio di car sharing. All'utente vengono presentate 3 opzioni:

1. **Registrazione**
2. **Login**
3. **Esci**

Opzione 1: Viene chiesto all'utente di inserire un nome utente da registrare nel sistema (viene salvato nel file "*utente.txt*").

Opzione 2: Viene chiesto all'utente di inserire il suo nome utente. Se l'utente non risulta registrato tra i nomi utenti salvati nel sistema, viene riportato al menu di accesso per scegliere una nuova operazione.

Opzione 3: Interrompe l'esecuzione del programma.

CARICAMENTO INPUT

In seguito, vengono caricati i dati dei veicoli e delle prenotazioni dai rispettivi file ("*Veicoli.txt*", "*Orari.txt*" e "*StoricoPrenotazioni.txt*") nelle rispettive strutture. In particolare, per i veicoli vengono chiamate iterativamente le funzioni **creaVeicolo**, che si occupa di allocare la memoria per un nuovo Veicolo, e **riempiVeicoli** che va a inserire i dati del veicolo nella struct.

Successivamente viene chiamata la funzione **RiempiTabellaHashDaFile** che carica le prenotazioni già effettuate (se non esiste nessuna prenotazione, alloca una tabella hash di dimensioni "*HASH_TAGLIA*").

MENU DI SERVIZIO UTENTE

In seguito si accede al menu principale del Car-sharing con il corrispettivo nome utente. All'utente vengono presentate 6 opzioni:

1. **Nuova Prenotazione**
2. **Visualizza storico prenotazioni**

3. **Visualizza Sconti**
4. **Visualizza Veicoli**
5. **Trova Prenotazione**
6. **Esci**

Opzione 1

Viene stampato a stdout il catalogo dei veicoli. Viene chiesto all'utente di scegliere uno dei veicoli mediante indice (*da 0 a 9*). Se la scelta dell' indice non è valida, mediante opportuni controlli verrà chiesto di inserirne uno valido.

Successivamente, attraverso la funzione **LimitaOrariDisponibili**, vengono rese non disponibili tutte le fasce orarie antecedenti all'orario locale del pc. In seguito, vengono mostrate le fasce orarie del veicolo scelto e viene chiesto all'utente di scegliere una fascia orari mediante indice. Se la scelta dell'indice è valida e il veicolo disponibile (**verificaDisponibilita**), viene generato l'ID della prenotazione (**rand**) e la data della prenotazione (**ottieniData**) mediante la libreria "*time.h*".

Viene chiamata la funzione **NuovaPrenotazione** che salvare tutti gli input dell'utente in un oggetto di tipo Prenotazione, e viene stampato su stdout all'utente il riepilogo della prenotazione (**stampaPrenotazione**).

All'utente è concessa, poi, la scelta di *confermare* o *annullare* la prenotazione. In caso di conferma, il programma aggiorna la disponibilità del veicolo scelto per l'orario scelto (**modificaDisponibilita**) e registra la prenotazione nello storico (**AggiornaStorico**) su file "*StoricoPrenotazioni.txt*". In seguito l'utente viene chiesto all'utente se tornare al menu principale e solo dopo aver confermato, viene stampato un messaggio di feedback contenente l'ID della prenotazione confermata.

In caso di *annullamento*, la memoria della prenotazione viene deallocata e l'utente viene reindirizzato al menu principale con un messaggio di feedback di conferma annullamento.

D'altro canto, se la scelta relativa alla conferma o all'annullamento *non* è valida, mediante opportuni controlli verrà chiesto di inserirne uno valido.

Altrimenti se una fascia oraria risulta già *occupata* all'utente viene viene chiesto all'utente se tornare al menu principale e solo dopo aver confermato, viene stampato un messaggio di feedback relativo all'impossibilità di uso del veicolo scelta per la fascia oraria scelta.

Opzione 2

L'utente viene reindirizzato alla schermata di visualizzazione dello storico prenotazioni e viene chiamata la funzione **StampaPrenotazioneTabellaHash**, che stampa solo le prenotazioni associate all'utente che sta utilizzando il programma. In seguito alla stampa dello storico, viene chiesto all'utente se tornare al menu principale.

Opzione 3

L'utente viene reindirizzato alla schermata di visualizzazione degli sconti disponibili con annesse stampe su stdout. Se si prenota un veicolo in una fascia oraria dopo le ore *20:00*, si ha diritto a uno sconto del **30%** sul costo totale, invece se si sceglie una fascia oraria prima delle ore *9:00*, si ha diritto a uno sconto del **15%**. Una volta stampato e visualizzato gli sconti a schermo, viene chiesto all'utente se tornare al menu principale.

Opzione 4

L'utente viene reindirizzato alla schermata di visualizzazione del catalogo veicoli con annesse stampe su stdout. Viene chiamata iterativamente la funzione **stampaVeicolo** che mostra su stdout i dettagli di tutti i veicoli presenti nel catalogo. Una volta stampati e visualizzati i veicoli, viene chiesto all'utente se tornare al menu principale.

Opzione 5

Viene chiesto all'utente di inserire l'ID della prenotazione che sta trovare. Successivamente viene chiamata la funzione **TrovaPrenotazione** che effettua una ricerca nella tabella hash utilizzando come chiave l'ID. Se viene trovata, la funzione **stampaPrenotazione** ne mostra i dettagli, altrimenti viene stampato un messaggio su stdout che la prenotazione non è stata trovata. Dopo aver visionato il risultato della ricerca, viene chiesto all'utente se tornare al menu principale.

Opzione 6

Viene interrotta l'esecuzione del programma, vengono deallocati in modo iterativo tutti i veicoli con l'ausilio della funzione **liberaVeicolo** e poi viene deallocata la tabella hash con la funzione **LiberaTabellaHash** che chiama la funzione **LiberaLista** iterativamente per deallocare tutte le prenotazioni**.**

Opzione 7

Nel caso in cui, l'utente riesca a bypassare i controlli relativi alla scelta tra le opzioni del menu principale, il programma considererà l'opzione di *default* che prevede la chiusura forzata del programma in **errore** dopo aver deallocato nello stesso modus operandi dell'*opzione 6*.

SPECIFICA SINTATTICA E SEMANTICA

FUNZIONI “Veicolo.h”

riempiVeicoli

Riempie la struct veicolo con i dati contenuti nel file Veicolo.txt

Specifica sintattica:

riempiVeicoli(veicolo) -> int

Parametri:

v: struct veicolo

Specifica semantica:

riempiVeicoli(v) -> 0 se chiusura del file corretta altrimenti 1

Pre-condizione:

- Veicolo.txt deve esistere e contenere i dati dei veicoli
- La struct veicolo deve esistere
- controlloToken, chiudiFile, riempiOrari devono essere implemetate correttamente

Post-condizione:

La struct veicolo e` riempita con successo

Ritorna:

0 se chiusura del file corretta altrimenti 1

Effetti collaterali:

- Modifica il contenuto nella struct veicolo
- Se il file è vuoto, la struct veicolo risulta NULL
- Stampa errore per apertura del file fallita, per l'allocazione dei vari campi della struct e riempimento fallito.

---

stampaVeicolo

Stampa a video i dati della struct veicolo

Specifica sintattica:

stampaVeicoli(veicolo) -> void

Parametri:

v: struct veicolo

Specifica semantica:

stampaVeicoli(v) -> void

Pre-condizione:

La struct veicolo deve contenere dei dati diversi da NULL

Post-condizione:

Non ritorna nessun valore

Ritorna:

Nessun valore

Effetti collaterali:

Stampa a video i dati della struct veicolo

---

liberaVeicolo

Libera la memoria della struct veicolo

Specifica sintattica:

liberaVeicolo(veicolo) -> void

Parametri:

v: struct veicolo

Specifica semantica:

liberaVeicolo(v) -> void

Pre-condizione:

Memoria allocata per la struct veicolo

Post-condizione:

Memoria liberata correttamente

Ritorna:

Nessun valore

Effetti collaterali:

La struct veicolo non ha più dati presenti in memoria

---

riempiOrari

- Prende in input la struct veicolo
- Riempi la struct annidata Orari con i dati contenuti nel file Orari.txt

Specifica sintattica:

riempiOrari(veicolo) -> int

Parametri:

v: struct veicolo

Specifica semantica:

riempiOrari(v) -> 0 se chiusura del file corretta altrimenti 1

Pre-condizione:

- La struct veicolo esistere
- Orari.txt deve esistere e contenere i dati degli intervalli orari e il loro status di disponibilita`
- controlloToken, chiudiFile devono essere implemetate correttamente

Post-condizione:

La struct annidata Orari riempito con successo

Ritorna:

0 se chiusura del file corretta altrimenti 1

Effetti collaterali:

- Modifica il contenuto nella struct annidata orari
- Se il file è vuoto, la struct veicolo risulta NULL
- File viene chiuso in caso in cui controllotoken fallisce

---

stampaDisponibilita

Stampa a video lo status di disponibilita di un veicolo

Specifica sintattica:

stampaDisponibilita(veicolo, int) -> void

Parametri:

- v: struct veicolo
- indiceOrario: indice orario scelto

Specifica semantica:

stampaDisponibilità(v, indiceOrario) -> void

Pre-condizione:

La struct veicolo deve esistere e contenere dati della struct annidata Orari

Post-condizione:

Non ritorna nessun valore

Ritorna:

Nessun valore

Effetti collaterali:

Stampa a video la disponibilita ("Non disponibile" o "Disponibile")

---

modificaDisponibilita

Va a modificare nel campo disponibilità il suo valore a 1 quando chiamata

Specifica sintattica:

modificaDisponibilità(veicolo, int) -> void

Parametri:

- v: struct veicolo

- indiceOrario: indice orario scelto

Specifica semantica:

modificaDisponibilità(v, indiceOrario) -> void

Pre-condizione:

La struct veicolo deve esistere e contenere dati nella struct annidata Orari

Post-condizione:

Non ritorna nessun valore, Campo Disponibilità cambiato

Ritorna:

Nessun valore

Effetti collaterali:

Cambiato il valore nel campo Disponibilità della struct annidata Orari

---

ottieniModello

Restituisce la stringa del campo modello della struct veicolo

Specifica sintattica:

ottieniModello(veicolo) -> char*

Parametri:

v: struct veicolo

Specifica semantica:

ottieniModello(v) -> stringa del modello veicolo

Pre-condizione:

La struct veicolo deve esistere

Post-condizione:

Ottenuta la stringa nel campo modello della struct veicolo

Ritorna:

Una stringa del campo modello della struct veicolo altrimenti NULL

Effetti collaterali:

Nessun effetto collaterale

---

ottieniTarga

Restituisce la stringa del campo targa della struct veicolo

Specifica sintattica:

ottieniTarga(veicolo) -> char*

Parametri:

v: struct veicolo

Specifica semantica:

ottieniTarga(v) -> stringa della targa veicolo

Pre-condizione:

La struct veicolo deve esistere

Post-condizione:

Ottenuta la stringa nel campo targa della struct veicolo

Ritorna:

Una stringa del campo targa della struct veicolo altrimenti NULL

Effetti collaterali:

Nessun effetto collaterale

---

ottieniOrarioInizio

Restituisce il float del campo inizio della struct annidata orari

Specifica sintattica:

ottieniOrariInizio(veicolo, int) -> float

Parametri:

- v: struct veicolo

- indiceOrario: indice orario scelto

Specifica semantica:

ottieniOrariInizio(v , indiceOrario) -> float della struct annidata orari

Pre-condizione:

La struct veicolo deve esistere

Post-condizione:

Ottenuto il float del campo inizio della struct annidata orari

Ritorna:

Un float del campo inzio della struct annidata orari altrimenti -1

Effetti collaterali:

Nessun effetto collaterale

---

ottieniOrarioFine

Restituisce il float del campo fine della struct annidata orari

Specifica sintattica:

ottieniOrariFine(veicolo, int) -> float

Parametri:

v: struct veicolo

indiceOrario: indice orario scelto

Specifica semantica:

ottieniOrariFine(v , indiceOrario) -> float della struct annidata orari

Pre-condizione:

La struct veicolo deve esistere

Post-condizione:

Ottenuto il float del campo fine della struct annidata orari

Ritorna:

Un float del campo fine della struct annidata orari altrimenti -1

Effetti collaterali:

Nessun effetto collaterale

---

ottieniDisponibilita

Restituisce l'int del campo Disponibilita della struct annidata orari

Specifica sintattica:

ottieniDisponibilita(veicolo, int) -> int

Parametri:

- v: struct veicolo
- indiceOrario: indice orario scelto

Specifica semantica:

ottieniDisponibilita(v , indiceOrario) -> int della struct annidata orari

Pre-condizione:

La struct veicolo deve esistere

Post-condizione:

Ottenuto l'int del campo Disponibilita della struct annidata orari

Ritorna:

Un int del campo Disponibilita della struct annidata orari altrimenti -1

Effetti collaterali:

Nessun effetto collaterale

---

ottieniCostoOrario

Restituisce il float del campo CostoNoleggioOrario della struct veicolo

Specifica sintattica:

ottieniCostoOrario(veicolo) -> float

Parametri:

v: struct veicolo

Specifica semantica:

ottieniCostoOrario(v) -> float della struct veicolo

Pre-condizione:

La struct veicolo deve esistere

Post-condizione:

Ottenuto il float del campo CostoNoleggioOrario della struct veicolo

Ritorna:

Un float del campo CostoNoleggioOrario della struct veicolo altrimenti -1

Effetti collaterali:

Nessun effetto collaterale

---

creaVeicolo

Alloca memoria per la struct veicolo

Specifica sintattica:

creaVeicolo() -> veicolo

Parametri:

Nessuno.

Specifica semantica:

creaVeicolo() -> struct veicolo

Pre-condizione:

Nessuna pre-condizione

Post-condizione:

Memoria allocata correttamente per la struct veicolo

Ritorna:

La struct veicolo allocata correttamente altrimenti NULL

Effetti collaterali:

Stampa un messaggio di errore in caso di allocazione fallita

FUNZIONI "TabellaHash.h"

ottieniTaglia

Restituisce la taglia della tabella hash.

Specifica sintattica:

ottieniTaglia (TabellaHash) -> int

Parametri:

t: Tabella hash

Specifica semantica:

ottieniTaglia (t) -> int

Pre-condizione:

t deve esistere e diversa da NULL

Post-condizione:

Taglia della tabella hash ottenuta correttamente

Ritorna:

un intero che è la taglia della tabella hash.

Effetti collaterali:

Nessun effetto collaterale.

NuovaTabellaHash

Alloca memoria per una nuova tabella hash.

Specifica sintattica:

NuovaTabellaHash (int) -> TabellaHash

Parametri:

taglia: grandezza della tabella hash.

Specifica semantica:

NuovaTabellaHash (taglia) -> tabellahash se allocata correttamente altrimenti NULL

Pre-condizione:

La taglia deve essere maggiore di 0.

Post-condizione:

La tabella hash allocata correttamente altrimenti NULL

Ritorna:

TabellaHash allocata correttamente

Effetti collaterali:

Stampa un messaggio di errore se l'allocazione fallisce.

FunzioneHash

Calcola l'indice della tabella hash.

Specifica sintattica:

FunzioneHash (int, int) -> int

Parametri:

- ID: codice univoco della prenotazione
- taglia: grandezza della tabella hash

Specifica semantica:

FunzioneHash (ID, taglia) -> indice tabella hash

Pre-condizione:

La taglia deve essere maggiore di zero.

Post-condizione:

Ottenuto l'indice della tabella hash

Ritorna:

Intero che è l'indice della tabella hash

Effetti collaterali:

Nessun effetto collaterale.

InserisciPrenotazione

Inserisce una prenotazione nella tabella hash

Specifica sintattica:

InserisciPrenotazione (TabellaHash, Prenotazione) -> int

Parametri:

t: Tabella hash

p: struct prenotazione

Specifica semantica:

InserisciPrenotazione (t, p) -> 1 a buon fine altrimenti 0 gia` presente

Pre-condizione:

- t deve essere allocata correttamente
- p deve essere diversa da NULL
- ottieniID, ottieniNext, FunzioneHash, assegnaNext devono essere implemetate correttamente

Post-condizione:

Se l'inserimento ha avuto successo restituisce un 1 (inserimento effettuato correttamente) altrimenti 0

Ritorna:

0 se la prenotazione è già presente nella tabella hash, 1 altrimenti.

Effetti collaterali:

Modifica il contenuto della tabella hash aggiungendo una prenotazione.

LiberaTabellaHash

Dealloca la memoria occupata dalla tabella hash.

Specifica sintattica:

LiberaTabellaHash (TabellaHash) -> void

Parametri:

t: tabella hash

Specifica semantica:

LiberaTabellaHash (t) -> void

Pre-condizione:

- TabellaHash deve esistere e diversa da NULL
- LiberaLista implementata correttamente

Post-condizione:

Memoria liberata correttamente.

Ritorna:

Non ritorna nessun valore.

Effetti collaterali:

La tabella hash non contiene più i dati presenti in memoria.

StampaPrenotazioneTabellaHash

Stampa a video le prenotazioni dell'utente

Specifica sintattica:

StampaPrenotazioneTabellaHash (TabellaHash, char*) -> void

Parametri:

- t: tabella hash
- nomeUtente: stringa nomeUtente

Specifica semantica:

StampaPrenotazioneTabellaHash (t, nomeUtente) -> void

Pre-condizione:

- TabellaHash deve esistere e diversa da NULL
- nomeUtente allocato correttamente
- ottieniTaglia, ottieniNomeUtente, stampaPrenotazione, ottieniNext implementate correttamente

Post-condizione:

Nessuna post-condizione.

Ritorna:

Non ritorna nessun valore.

Effetti collaterali:

Stampa a video i dati della prenotazione dell'utente, se non presenti un avviso

---

TrovaPrenotazione

Cerca all'interno della tabella hash la prenotazione attraverso l'ID

Specifica sintattica:

TrovaPrenotazione (TabellaHash, int, int) -> Prenotazione

Parametri:

- t: tabella hash
- ID: chiave Prenotazione
- taglia: grandezza tabella hash

Specifica semantica:

TrovaPrenotazione: (t, ID, taglia) -> prenotazione se ID trovato altrimenti NULL

Pre-condizione:

- TabellaHash deve esistere e diverso da NULL
- Taglia deve essere maggiore di 0.
- FunzioneHash, ottieniID, ottieniNext implementate correttamente

Post-condizione:

Restituisce prenotazione se l'ID coincide altrimenti NULL

Ritorna:

Prenotazione se l'ID coincide altrimenti NULL

Effetti collaterali:

Prenotazione contiene dati della struct Prenotazione

FUNZIONI “Utile.h”

ottieniData

Funzione per ottenere la data locale della macchina formattata gg/mm/aaaa

Specifica sintattica:

ottieniData()-> char*

Parametri:

Nessun parametro

Specifica semantica:

ottieniData() -> data locale formattata

Pre-condizione:

Includere la libreria time.h

Post-condizione:

Data locale della macchina ottenuta con successo

Ritorna:

Una stringa della data locale formattata gg/mm/aaaa, altrimenti NULL.

Effetti collaterali:

Stampa un messaggio di errore in caso di allocazione della memoria per la stringa fallita

---

AggiornaStorico

Aggiorna il file StoricoPrenotazioni.txt in appending con una nuova prenotazione

Specifica sintattica:

AggiornaStorico(Prenotazione, int, int)-> int

Parametri:

- p: struct Prenotazione
- indiceVeicolo: indice del veicolo
- indiceOrario: indice dell'orario

Specifica semantica:

AggiornaStorico(p, indiceVeicolo, indiceOrario) -> 0 se chiudiFile fallito altrimenti 1

Pre-condizione:

- Struct Prenotazione deve esistere e diversa da NULL
- Indici validi
- chiudiFile, ottieniNomeUtente, ottieniDataPrenotazione, ottieniInizioPrenotazione, ottieniFinePrenotazione, ottieniID, ottieniModelloPrenotazione, ottieniTargaPrenotazione devono essere implementate correttamente

Post-condizione:

- Dati scritti correttamente su file
- Chiusura file avvenuta con successo

Ritorna:

0 se la chiusura del file e` fallita altrimenti 1

Effetti collaterali:

In caso di fallimento nell'apertura del file viene stampato un messaggio di errore

---

RiempiTabellaHashDaFile

- La funzione legge il file StoricoPrenotazioni.txt, conta il numero di prenotazioni e crea una tabella hash per contenere tutte le prenotazioni

- Se la data corrente corrisponde alla data locale viene aggiornata la disponibilita

Specifica sintattica:

RiempiTabellaHashDaFile(veicolo) -> TabellaHash

Parametri:

*v: puntatore a un array di struct veicolo

Specifica semantica:

RiempiTabellaHashDaFile(*v) -> tabella hash aggiornata o NULL (non ci sono prenotazioni)

Pre-condizione:

- Il puntatore deve essere diverso da NULL e deve puntare all'array di struct veicolo
- La struct veicolo deve esistere e diversa da NULL
- Tabella hash allocata correttamente
- Formattazione del file corretta

Post-condizione:

- Tabella hash creata correttamente con le prenotazioni
- Disponibilita` aggiornata in caso data corrente sia uguale a quella locale
- Chiusura del file avvenuta correttamente
- NuovaTabellaHash, chiudiFile, ottieniData, LiberaTabellaHash, controllotoken, NuovaPrenotazione, modificaDisponibilita devono essere implemetate correttamente

Ritorna:

La tabella hash aggiornata con i dati o a NULL in caso non vi siano prenotazioni

Effetti collaterali:

- Allocata memoria per la tabella hash
- Stampa messaggi di errore in caso di: lettura fallita del file, memoria non allocata correttamente, ottenimento data locale fallita, inserimento fallito
- Viene liberata memoria in caso di tokenizzazione non corretta

---

LimitaOrariDisponibili

Modifica la disponibilita` degli orari disponibili durante una prenotazione confrontandoli con l'orario locale

Specifica sintattica:

LimitaOrariDisponibili(veicolo) -> void

Parametri:

*v: puntatore a un array di struct veicolo

Specifica semantica:

LimitaOrariDisponibili(*v) -> void

Pre-condizione:

- Il puntatore deve essere diverso da NULL e deve puntare all'array di struct veicolo
- La struct veicolo deve esistere e diversa da NULL
- modificaDisponibilita, ottieniOrarioInizio implementate correttamente

Post-condizione:

Disponibilita` modificata in tempo reale

Ritorna:

Nessun valore

Effetti collaterali:

La disponibilita` viene modificata

---

verificaSconto

La funzione verifica la possibilità di un sconto in determinati intervalli orari prestabiliti e restituisce un tipo float riguardante lo sconto da applicare al calcolo del costo totale del noleggio

Specifica sintattica:

verificaSconto(veicolo, int) -> float

Parametri:

- v: oggetto veicolo
- indiceOrario: indice dell'orario scelto

Specifica semantica:

verificaSconto(v, indiceOrario) -> Percentuale di Sconto

Pre-condizione:

- L'oggetto veicolo deve esistere e contenere i dati sugli intervalli orari
- IndiceOrario deve essere valido
- OttieniOrarioInizio implementato correttamente

Post-condizione:

- Restituisce la percentuale di sconto se l'orario della prenotazione corrisponde all'intervallo
- Altrimenti restituisce 1

Ritorna:

- Restituisce un tipo float della percentuale di sconto
- Altrimenti float di 1.0

Effetti collaterali:

Nessun effetto collaterale

---

verificaDisponibilita

Verifica la disponibilita` di un veicolo in un determinato orario

Specifica sintattica:

int verificaDisponibilità (veicolo, int) -> void

Parametri:

- v: oggetto veicolo
- indiceOrario: indice orario scelto

Specifica semantica:

verificaDisponibilità(v, indiceOrario) -> 1 se ottieniDisponibilita restituisce 0 altrimenti 0

Pre-condizione:

- L'oggetto veicolo deve esistere e contenere i dati dell'oggetto orari
- IndiceOrario valido
- ottieniDisponibilita implemetata correttamente

Post-condizione:

Restituisce un intero se ottieniDisponibilita e` 0 o diverso da 0

Ritorna:

1 se ottieniDisponibilita restituisce 0 altrimenti 0

Effetti collaterali:

Nessun effetto collaterale

---

stampaOrari

Stampa a video l'intervallo orario scelto dall'utente con la sua disponibilità (Non disponibile o Disponibile)

Specifica sintattica:

stampaOrari(veicolo) -> void

Parametri:

v: struct veicolo

Specifica semantica:

stampaOrari(v) -> void

Pre-condizione:

- La struct veidolo deve esistere e contenere dati
- stampaDisponibilita, ottieniOrarioInizio e ottieniOrarioFine devono esssere implementate correttamente

Post-condizione:

Nessun valore di ritorno

Ritorna:

Nessun valore

Effetti collaterali:

Stampa a video la lista degli orari e la loro disponibilita del veicolo

---

controllotoken

Controlla se il token e` diverso da NULL (tokenizzazione corretta)

Specifica sintattica:

void controllotoken(char*)-> int

Parametri:

token: stringa

Specifica semantica:

controllotoken(token)-> 1 se token diverso da zero altrimenti 0

Pre-condizione:

token deve esistere

Post-condizione:

Se token diverso NULL restituisce 1 altrimenti 0

Ritorna:

1 se token è diverso da NULL altrimenti 0

Effetti collaterali:

Stampa un messaggio di errore in caso il token sia NULL

---

chiudiFile

Funzione che chiude un file

Specifica sintattica:

chiudiFile(FILE) -> int

Parametri:

*file: puntatore a file

Specifica semantica:

chiudiFile(*file) -> 1 se avvenuto con successo altrimenti 0

Pre-condizione:

Puntatore a file diverso da NULL e deve puntare al file

Post-condizione:

File chiuso correttamente

Ritorna:

1 se avvenuto con successo altrimenti 0

Effetti collaterali:

Stampa un messaggio di errore in caso di fallimento nella chiusura del file

---

FUNZIONI "Menu.h"

operazioneAccesso

Gestisce le operazioni per l'accesso scelto dall'utente (registrazione login uscita), aggiornando "utente.txt" se necessario

Specifica sintattica:

operazioneAccesso(char, char**) -> int

Parametri:

- operazione: carattere per l'operazione da effettuare ("1, 2, 3")
- nomeUtente: doppio puntatore a char per nomeUtente

Specifica semantica:

operazioneAccesso(operazione, **nomeUtente) -> int:

- -1: Errore nell'esecuzione
- 1: Accesso corretto
- 0: Utente già registrato
- 2: Utente non registrato
- 3: Operazione non valida

Pre-condizione:

il file "utente.txt" deve esistere.

Post-condizione:

- operazione 1: se il nome utente non risulta registrato ed è valido, viene allocato, aggiunto in appending su file "utente.txt", altrimenti restituisce 0
- operazione 2: se il nome utente esiste su file "utente.txt", viene allocato e viene effettuato l'accesso, altrimenti restituisce 2
- operazione 3: il programma viene terminato senza errori tramite exit(0)

Ritorna:

- -1: Nel caso di errore di allocazione, e/o apertura file
- 0: Se l'utente nel caso di registrazione risulta già presente su file "utente.txt"
- 1: Se l'utente effettua correttamente il login e/o registrazione
- 2: Se l'utente sceglie fare il login ma non risulta registrato
- 3: Se l'utente sceglie un'operazione non valida (Es. fuori all'intervallo [1;3])

Effetti collaterali:

- Nel caso di errori di allocazioni, apertura file, chiusura file, stampa un messaggio di ERRORE
- Stampa un messaggio di feedback, nel caso di uscita da programma
- Viene chiesto all'utente di inserire il nome utente, con un messaggio di avviso per il corretto inserimento
- Se l'inserimento non è valido, stampa un messaggio di avviso
- Se il nomeUtente passa tutti i controlli, allora modifica "utente.txt"

---

menuAccesso

Stampa le opzioni di accesso/uscita per l'utente

Specifica sintattica:

menuAccesso() -> char*

Parametri:

Nessuno

Specifica semantica:

menuAccesso() -> stringa nomeUtente

Pre-condizione:

La funzione operazioneAccesso deve essere implementata correttamente.

Post-condizione:

- opzione 1: Se l'utente effettua correttamente la registrazione in operazioneAccesso, allora restituisce il puntatore al nomeUtente
- opzione 2: Se l'utente effettua correttamente il login in operazioneAccesso, allora restituisce il puntatore al nomeUtente
- opzione 3: Se l'utente inserisce quest'opzione, viene chiamata la funzione operazioneAccesso che chiude il programma e non ritorna a menuAccesso

Ritorna:

stringa nomeUtente se l'accesso/registrazione da operazioneAccesso viene effettuato/a correttamente

Effetti collaterali:

- Allocazione del nomeUtente in seguito alla chiamata di operazioneAccesso
- Nel caso di input non validi, viene stampato un messaggio di feedback, e si ripete il menu
- Nel caso in cui operazioneAccesso ritorna 0, stampa un messaggio di registrazione già avvenuta
- Nel caso in cui operazioneAccesso ritorna 2, stampa un messaggio di registrazione assente
- Nel caso in cui operazioneAccesso ritorna 3, stampa un messaggio di operazione non valida
- Nel caso in cui operazioneAccesso ritorna un valore fuori dall'intervallo [0;3], stampa un messaggio di errore generico e ripete il menu

---

menuPrincipale

Funzione ausiliaria per il ritorno al menu Principale (scelte relative al servizio)

Specifica sintattica:

menuPrincipale() -> int

Parametri:

Nessuno

Specifica semantica:

menuPrincipale() -> 0

Pre-condizione:

Nessuna.

Post-condizione:

Restituisce 0 se l'utente inserisce 'Y' o 'y', altrimenti resta nel ciclo in attesa di un nuovo input, attendendo in loop una scelta positiva

Ritorna:

0, ossia quando l'utente inserisce 'Y' o 'y'

Effetti collaterali:

- Pulisce il buffer con getchar
- Stampa un messaggio di richiesta input
- Stampa messaggi di avviso nel caso di input non validi
- Stampa un messaggio di feedback nel caso in cui l'utente inserisce 'N' o 'n'

FUNZIONI "Prenotazione.h"

NuovaPrenotazione

Crea una nuova prenotazione nella struct item (prenotazione)

Specifica sintattica:

NuovaPrenotazione(int, char*, veicolo, int, char*) -> Prenotazione

Parametri:

- ID: indice (key) prenotazione
- NomeUtente: nome utente prenotazione
- c: struct veicolo
- i: indice orario
- dataPrenotazione: data prenotazione

Specifica semantica:

NuovaPrenotazione(ID, NomeUtente, c, i, dataPrenotazione) -> Prenotazione altrimenti NULL

Pre-condizione:

- Struct veicolo deve esistere
- NomeUtente e dataPrenotazione allocate correttamente
- ottieniOrarioInizio, ottieniOrarioFine, costoNoleggio devono essere implementate correttamente

Post-condizione:

Prenotazione allocata e creata correttamente altrimenti NULL

Ritorna:

Prenotazione

Effetti collaterali:

Stampa a video dei messaggi di errore in caso di mancata allocazione di NomeUtente, dataPrenotazione e prenotazione stessa

---

LiberaLista

Libera memoria dalla struct prenotazione

Specifica sintattica:

LiberaLista(Prenotazione) -> void

Parametri:

p: struct prenotazione

Specifica semantica:

LiberaLista(p) -> void

Pre-condizione:

Memoria allocata per la struct prenotazione

Post-condizione:

Memoria liberata correttamente

Ritorna:

Nessun valore

Effetti collaterali:

La struct prenotazione non ha piu` dati presenti in memoria

---

stampaPrenotazione

Stampa a video i dati della struct prenotazione

Specifica sintattica:

stampaPrenotazione(Prenotazione) -> void

Parametri:

p: struct prenotazione

Specifica semantica:

stampaPrenotazione(p) -> void

Pre-condizione:

- La struct prenotazione deve contenere dei dati diversi da NULL
- ottieniModelloPrenotazione deve essere implementata correttamente

Post-condizione:

Non ritorna nessun valore

Ritorna:

Nessun valore

Effetti collaterali:

Stampa a video i dati della struct veicolo

---

ottieniID

Restituisce l'intero del campo ID della struct prenotazione

Specifica sintattica:

ottieniID(Prenotazione) -> int

Parametri:

p: struct prenotazione

Specifica semantica:

ottieniID(p) -> intero dell'ID prenotazione

Pre-condizione:

La struct prenotazione deve esistere

Post-condizione:

Ottenuto l'ID della prenotazione

Ritorna:

Un intero del campo ID della struct prenotazione altrimenti -1

Effetti collaterali:

Nessun effetto collaterale

---

ottieniNext

Restituisce il puntatore al prossimo elemento nella lista concatenata

Specifica sintattica:

ottieniNext(Prenotazione p) -> struct item*

Parametri:

p: struct prenotazione

Specifica semantica:

ottieniNext(p) -> puntatore al prossimo elemento nella lista concatenata altrimenti NULL

Pre-condizione:

Struct prenotazione deve esistere

Post-condizione:

Ottenuto il puntatore al prossimo elemento nella lista concatenata

Ritorna:

puntatore al prossimo elemento nella lista concatenata altrimenti NULL

Effetti collaterali:

Nessun effetto collaterale

---

ottieniNomeUtente

Restituisce la stringa del campo data della struct prenotazione

Specifica sintattica:

ottieniDataPrenotazione(Prenotazione) -> char*

Parametri:

p: struct prenotazione

Specifica semantica:

ottieniDataPrenotazione(p) -> stringa della data prenotazione

Pre-condizione:

La struct prenotazione deve esistere

Post-condizione:

Ottenuta la stringa del campo data della struct prenotazione altrimenti NULL

Ritorna:

Una stringa del campo data della struct prenotazione altrimenti NULL

Effetti collaterali:

Nessun effetto collaterale

---

ottieniInizioPrenotazione

Restituisce il float del campo OrarioSceltoInizio della struct prenotazione

Specifica sintattica:

ottieniInizioPrenotazione(Prenotazione) -> float

Parametri:

p: struct prenotazione

Specifica semantica:

ottieniInizioPrenotazione(p) -> float dell'inizio prenotazione

Pre-condizione:

La struct prenotazione deve esistere

Post-condizione:

Ottenuto il float del campo OrarioSceltoInizio della struct prenotazione

Ritorna:

Un float del campo OrarioSceltoInizio della struct prenotazione altrimenti -1

Effetti collaterali:

Nessun effetto collaterale

---

ottieniFinePrenotazione

Restituisce il float del campo OrarioSceltoFine della struct prenotazione

Specifica sintattica:

ottieniFinePrenotazione(Prenotazione) -> float

Parametri:

p: struct prenotazione

Specifica semantica:

ottieniFinePrenotazione(p) -> float della fine prenotazione

Pre-condizione:

La struct prenotazione deve esistere

Post-condizione:

Ottenuto il float del campo OrarioSceltoFine della struct prenotazione

Ritorna:

Un float del campo OrarioSceltoFine della struct prenotazione altrimenti -1

Effetti collaterali:

Nessun effetto collaterale

---

ottieniModelloPrenotazione

Restituisce la stringa del modello del veicolo

Specifica sintattica:

ottieniModelloPrenotazione(Prenotazione) -> char*

Parametri:

p: struct veicolo

Specifica semantica:

ottieniModelloPrenotazione(p) -> stringa del modello veicolo

Pre-condizione:

- La struct prenotazione deve esistere
- ottieniModello deve essere implementato correttamente

Post-condizione:

Ottenuta la stringa del modello del veicolo

Ritorna:

Una stringa del modello del veicolo dalla struct veicolo altrimenti NULL

Effetti collaterali:

Nessun effetto collaterale

---

ottieniTargaPrenotazione

Restituisce la stringa della targa del veicolo

Specifica sintattica:

ottieniTargaPrenotazione(Prenotazione) -> char*

Parametri:

p: struct veicolo

Specifica semantica:

ottieniTargaPrenotazione(p) -> stringa della targa veicolo

Pre-condizione:

- La struct prenotazione deve esistere
- ottieniTarga deve essere implementato correttamente

Post-condizione:

Ottenuta la stringa della targa del veicolo

Ritorna:

Una stringa della targa del veicolo dalla struct veicolo altrimenti NULL

Effetti collaterali:

Nessun effetto collaterale

---

costoNoleggio

Calcola il costo noleggio di un intervallo orario (con eventuale sconto)

Specifica sintattica:

costoNoleggio(veicolo, int) -> float

Parametri:

v: struct veicolo

indiceOrario: indice dell'orario scelto

Specifica semantica:

costoNoleggio(v, indiceOrario) -> float del costo noleggio

Pre-condizione:

- La struct veicolo deve esistere
- ottieniOrarioInizio, ottieniOrarioFine, ottieniCostoOrario, verificaSconto devono essere implementate correttamente

Post-condizione:

Restituisce il costo totale del noleggio dell'intervallo orario (con eventuale sconto)

Ritorna:

Un float del costo totale

Effetti collaterali:

Nessun effetto collaterale

---

assegnaNext

Assegna prenotazione next al campo next di p

Specifica sintattica:

assegnaNext(Prenotazione, Prenotazione) -> void

Parametri:

- p: struct prenotazione
- next: struct prenotazione

Specifica semantica:

assegnaNext(p, next) -> void

Pre-condizione:

p deve esistere e diversa da NULL

Post-condizione:

Next di p punterà alla prenotazione next

Ritorna:

Nessun valore

Effetti collaterali:

Nessun effetto collaterale

FUNZIONI RELATIVE AL TESTING

StampaVeicoliFile

Stampa su file di output le rispettive disponibilità per tutte le fasce orarie del veicolo (riferimento indice)

Specifica sintattica:

StampaVeicoliFile() -> int

Parametri:

Nessuno.

Specifica semantica:

StampaVeicoliFile() -> 0/1

Pre-condizione:

Il file "TC3_input.txt" deve esistere ed essere formattato come: indiceVeicolo indiceOrario Disponibilità

Post-condizione:

Il file "TC3_output.txt" conterrà i veicoli con i rispettivi indici per gli orari e la rispettive disponibilità

Ritorna:

- 0 se i file non vengono aperti o chiusi correttamente.
- 1 se i file vengono chiusi correttamente

Effetti collaterali:

- Il file "TC3_output.txt" viene modificato.
- Stampa su schermo eventuali messaggi di errore nel caso in cui i file non vengono aperti o chiusi correttamente.

---

confrontoFile

Confronta i rispettivi file di output e file di oracolo

Specifica sintattica:

confrontoFile(char*, char*, int) -> int

Parametri:

- nomefileoutput: nome del file di output
- nomefileoracle: nome del file di oracolo
- OPERAZIONE: numero per il corretto riferimento di stampa su result.txt

Specifica semantica:

confrontoFile(nomefileoutput, nomefileoracle, OPERAZIONE) -> 0/1

Pre-condizione:

- Il file "nomefileoutput.txt" deve esistere.
- Il file "nomefileoracle.txt" deve esistere.

Post-condizione:

Il file "result.txt" conterrá il PASS o FAIL del Test Case.

Ritorna:

- 0 se i file non vengono aperti o chiusi correttamente.
- 1 se i file vengono chiusi correttamente

Effetti collaterali:

- Il file "result.txt" viene modificato.
- Stampa su schermo eventuali messaggi di errore nel caso in cui i file non vengono aperti o chiusi correttamente.
- Lo status del test viene aggiornato ogni riga letta da file.

---

stampaCostofile

Scrive il costo del noleggio del veicolo

Specifica sintattica:

stampaCostofile(Prenotazione) -> int

Parametri:

p: prenotazione di cui trovare il costo noleggio

Specifica semantica:

stampaCostofile(p) -> 0/1

Pre-condizione:

La prenotazione deve esistere, essere allocata correttamente.

Post-condizione:

Il file "TC2_output.txt" conterrà il costo del noleggio della prenotazione considerata.

Ritorna:

- 0 se i file non vengono aperti o chiusi correttamente.
- 1 se i file vengono chiusi correttamente

Effetti collaterali:

- Il file "TC2_output.txt" viene modificato.
- Stampa su schermo eventuali messaggi di errore nel caso in cui i file non vengono aperti o chiusi correttamente.
- Stampa su schermo il resoconto relativo al costo del noleggio per la prenotazione.

---

controlloStorico

Scorre la tabella Hash al fine di vedere tutte le prenotazioni effettivamente caricate.

Specifica sintattica:

controlloStorico(TabellaHash) -> int

Parametri:

t: tabella hash contenente le prenotazioni effettuate dai clienti

Specifica semantica:

controlloStorico(t) -> 0/1

Pre-condizione:

La tabella hash deve essere allocata correttamente.

Post-condizione:

Il file "TC4_output.txt" conterrà lo storico delle prenotazioni caricate effettivamente nella tabella Hash.

Ritorna:

- 0 se i file non vengono aperti o chiusi correttamente.
- 1 se i file vengono chiusi correttamente

Effetti collaterali:

- Il file "TC4_output.txt" viene modificato.

- Stampa su schermo eventuali messaggi di errore nel caso in cui i file non vengono aperti o chiusi correttamente.

---

stampaDisponibilitaFile

Stampa su file di output le rispettive disponibilità per tutte le fasce orarie del veicolo

Specifica sintattica:

stampaDisponibilitaFile(veicolo, int) -> int

Parametri:

- v: veicolo di cui considerare le fasce orarie
- indiceOrario: indica di quale orario considerare le disponibilità

Specifica semantica:

stampaDisponibilitaFile(v,indiceOrario) -> 0/1

Pre-condizione:

Il veicolo deve essere allocato correttamente, insieme alle sue fasce orarie e quindi anche le rispettive disponibilità

Post-condizione:

Il file "TC1_output.txt" conterrà le fasce orarie e le rispettive disponibilità del veicolo

Ritorna:

- 0 se i file non vengono aperti o chiusi correttamente.
- 1 se i file vengono chiusi correttamente

Effetti collaterali:

- Il file "TC1_output.txt" viene modificato.
- Stampa su schermo eventuali messaggi di errore nel caso in cui i file non vengono aperti o chiusi correttamente.

---

ottieniCostoNoleggio

Restituisce il valore del costo noleggio della prenotazione

Specifica sintattica:

ottieniCostoNoleggio(prenotazione) -> float

Parametri:

p: prenotazione da considerare

Specifica semantica:

ottieniCostoNoleggio(p) -> costo della prenotazione

Pre-condizione:

La struct prenotazione deve esistere, essere allocata correttamente e contenere il costo del noleggio

Post-condizione:

Restituisce il costo totale del noleggio della prenotazione

Ritorna:

Un float del costo del noleggio altrimenti -1

Effetti collaterali:

Nessun effetto collaterale

---

ottieniPrenotazione

Restituisce la prenotazione secondo l'indice

Specifica sintattica:

ottieniPrenotazione (TabellaHash, int) -> Prenotazione

Parametri:

- t: tabella hash
- indice: indice tabella hash

Specifica semantica:

ottieniPrenotazione: (t, indice) -> prenotazione

Pre-condizione:

- TabellaHash deve esistere e diverso da NULL
- Indice valido

Post-condizione:

Restituisce prenotazione

Ritorna:

Prenotazione

Effetti collaterali:

Nessun effetto collaterale

RAZIONALE CASI DI TEST

TEST CASE 1

Riguarda la verifica della corretta creazione delle prenotazioni e dell'aggiornamento della disponibilità dei veicoli.

- INPUT

  Il file "TC1_input.txt" contiene l'indice del veicolo e della fascia oraria da prenotare.

- OUTPUT

  Il file "TC1_output.txt" conterrà le fasce orarie e le rispettive disponibilità antecedenti alla prenotazione, il resoconto della prenotazione (senza ID), le fasce orarie e le rispettive disponibilità posteriormente alla prenotazione.

- ORACOLO

  Il file "TC1_oracle.txt" contiene le fasce orarie e le rispettive disponibilità antecedenti alla prenotazione, il resoconto della prenotazione (senza ID), le fasce orarie e le rispettive disponibilità posteriormente alla prenotazione.

- RISULTATO

  Il programma stamperà "PASS" nel file "result.txt" se la prenotazione viene creata correttamente e la disponibilità aggiornata. Altrimenti stamperà "FAIL".

TEST CASE 2

Riguarda il calcolo del costo del noleggio in base al tempo di utilizzo

- INPUT

Il file “TC2_input.txt” contiene l’indice del veicolo e della fascia oraria di cui calcolare il costo del noleggio.

- OUTPUT

  Il file “TC2_output.txt” conterrà il costo finale del noleggio in base alla fascia oraria.

- ORACOLO

  Il file “TC2_oracle.txt” contiene il costo finale del noleggio del veicolo e della fascia oraria specificata in input.

- RISULTATO

  Il programma stamperà “PASS” se il costo finale del noleggio è stato calcolato correttamente, altrimenti stamperà “FAIL”.

TEST CASE 3

Riguarda la verifica della visualizzazione corretta dei veicoli disponibili.

- INPUT

  Il file “TC3_input.txt” contiene su ogni riga l’indice del veicolo, le fasce orarie e le loro disponibilità nel formato indiceVeicolo indiceOrario Disponibilità. “10” indica l’interruzione dell’inserimento dei veicoli.

- OUTPUT

  Il file “TC3_output.txt” contiene su ogni riga l’indice del veicolo, le fasce orarie e le loro disponibilità nel formato indiceVeicolo indiceOrario, se lette correttamente sul file di input.

- ORACOLO

  Il file “TC3_oracle.txt” contiene su ogni riga l’indice del veicolo, le fasce orarie e le loro disponibilità nel formato atteso.

- RISULTATO

  Il programma stamperà “PASS” i veicoli vengono stampati con le rispettive fasce orarie e disponibilità correttamente, altrimenti stamperà “FAIL”.

TEST CASE 4

Riguarda la corretta gestione dello storico prenotazioni.

- INPUT

  Il file “TC4_input.txt” contiene uno storico generico di prenotazioni effettuate.

- OUTPUT

  Il file “TC4_output.txt” conterrà le prenotazioni presenti nello storico nell’ordine in cui vengono inserite nella Tabella Hash.

- ORACOLO

  Il file “TC4_oracle.txt” contiene le prenotazioni presenti nello storico nell’ordine in cui vengono inserite nella Tabella Hash.

- RISULTATO

  Il programma stamperà “PASS” se lo storico viene gestito correttamente dalla tabella Hash e di conseguenza stampate correttamente su file, altrimenti stamperà “FAIL”.