

REPORT PROGETTO

“GESTIONE DELLE PRENOTAZIONI PER UNA PALESTRA”

I. MOTIVAZIONE DELLA SCELTA DELL'ADT

Nel progetto sono stati utilizzati diversi tipi di ADT (Abstract Data Type), ognuno scelto in base alla necessità dell'entità. Ogni struttura dati è stata progettata al fine di rispettare i principi di buona progettazione del software, quali:

Astrazione

Un ADT deve fornire una rappresentazione concettuale dei dati, separando il *cosa fa* dal *come lo fa*. L'utente deve poter utilizzare l'ADT senza conoscere i dettagli interni dell'implementazione. Questo facilita l'uso e permette di cambiare l'implementazione senza modificare il codice che lo utilizza.

Modularità

Ogni ADT deve essere progettato come un componente autonomo e indipendente. Questo approccio modulare permette di dividere il sistema in parti gestibili, rendendo più semplice lo sviluppo, il testing e la manutenzione del software.

Coerenza

Le operazioni fornite da un ADT devono garantire che lo stato interno dell'oggetto rimanga sempre valido. Ad esempio, una pila non deve mai permettere un'estrazione da una struttura vuota senza gestire opportunamente l'errore. La coerenza assicura l'integrità logica dei dati.

Efficienza

Anche se l'astrazione è importante, l'implementazione sottostante deve comunque essere efficiente. Questo significa scegliere le strutture dati e gli algoritmi più adatti per garantire buone prestazioni in termini di tempo di esecuzione e uso della memoria, in relazione ai requisiti dell'applicazione.

Information Hiding

È strettamente legato all'incapsulamento e significa nascondere all'esterno tutti i dettagli implementativi non necessari. L'utente dell'ADT non deve sapere come i dati sono organizzati o come le operazioni sono realizzate. Questo consente di modificare l'implementazione interna senza impattare il codice che utilizza l'ADT.

LEZIONE

```
struct lezione
{
    char* id;           // ID univoco della lezione
    char nome[50];       // es. "Zumba", "Pilates"
    Data data;          // La data della lezione (giorno, mese, anno)
    Orario ora_lezione;  // Orario di inizio lezione, in formato HH:MM
    int posti_max;       // capacità totale
    int posti_occupati;  // numero attuale di iscritti alla lezione
};
```

Motivazione

All'interno del sistema gestionale sviluppato, la necessità di rappresentare e manipolare in modo efficiente le lezioni programmate (come corsi di ginnastica o attività sportive) ha portato alla progettazione di un tipo di dato strutturato. La struttura `Lezione` nasce per modellare un'entità reale, fornendo un insieme coerente di informazioni e operazioni che consentano di gestire la vita completa di una lezione: dalla creazione, alla visualizzazione, alla memorizzazione permanente su file.

Vantaggi

- Le informazioni delle lezioni possono essere salvate su file per un uso successivo
- La Gestione dinamica della memoria consente maggiore flessibilità e ottimizzazione nell'uso delle risorse
- È possibile aggiungere nuovi campi o funzionalità senza modificare il resto del programma

DATA

```
struct data
{
    int giorno;
    int mese;
    int anno;
};
```

Motivazione

La struttura `Data` è stata creata per rappresentare una data completa (giorno, mese, anno) in modo chiaro e centralizzato, rendendo più semplice la gestione, il confronto e la validazione di date nel programma.

Vantaggi

- Validazione robusta (inclusi anni bisestili)
 - Operazioni frequenti semplificate (confronto, copia, oggi)
 - Uso sicuro con allocazione dinamica e gestione esplicita della memoria
-

ORARIO

```
struct orario
{
    int minuti;
    int ore;
};
```

Motivazioni

La struttura `Orario` è stata introdotta per rappresentare in modo autonomo e modulare un orario del giorno nel formato 24 ore. Inoltre, l'incapsulamento tramite funzioni dedicate migliora la leggibilità del codice, ne facilita il riutilizzo e centralizza la gestione della validità dei dati.

Vantaggi

- Permette un controllo centralizzato sulla validità di ore e minuti
 - Favorisce la modularità del programma
 - Facilita operazioni comuni come confronto, stampa e copia dell'orario
-

PRENOTAZIONE

```
struct prenotazione
{
    char *id_prenotazione; // identificativo univoco della prenotazione
    char *id_cliente;       // identificativo univoco del cliente
    char *id_lezione;       // identificativo univoco del cliente
    Data data_prenotazione; // data in cui viene effettuata la prenotazione

    struct prenotazione *next;
};
```

Motivazioni

La struttura `Prenotazione` è stata progettata per rappresentare in modo completo e flessibile l'associazione tra un cliente, una lezione e una data di prenotazione. Il campo `next` è stato incluso per consentire l'utilizzo della struttura all'interno di una tabella hash, dove le prenotazioni vengono memorizzate in base a una chiave. In caso di collisione tra chiavi hash, il campo `next` permette di collegare più prenotazioni nella stessa posizione della tabella, creando una lista collegata. Questo approccio consente una gestione efficiente delle collisioni e mantiene ottime prestazioni anche con un alto numero di prenotazioni.

Vantaggi

- Gestione efficiente delle collisioni nella tabella hash: grazie alla lista collegata creata tramite il campo `next`, si evitano sovrascritture e si garantisce l'integrità dei dati
 - Consente il salvataggio su file e la visualizzazione sia completa che essenziale
 - Riduce la duplicazione del codice e migliora la separazione tra logica di gestione e visualizzazione dei dati
-

CLIENTE

```
struct cliente
{
    char *id;           // identificativo univoco del cliente
    char nome[50];       // nome del cliente
    char cognome[50];    // cognome del cliente
    int durata_abbonamento; // durata dell'abbonamento in mesi
    Data data_iscrizione; // data d'iscrizione dell'abbonamento
    Data data_scadenza;   // data di scadenza dell'abbonamento
    struct cliente *next;
};
```

Motivazioni

La struttura `Cliente` è stata ideata per rappresentare in modo chiaro e completo le informazioni relative a un cliente, includendo dati identificativi, personali e legati all'abbonamento. L'inserimento del campo `next` è funzionale all'utilizzo della struttura all'interno di una tabella hash: in caso di collisione tra chiavi hash, i clienti che occupano la stessa posizione vengono collegati tra loro tramite una lista collegata. Questo approccio consente una gestione efficiente dei dati anche in presenza di molteplici clienti con chiavi hash coincidenti.

Vantaggi

- Gestione efficace delle collisioni nella tabella hash: grazie al campo `next`, è possibile mantenere l'integrità dei dati anche in caso di chiavi duplicate
 - Consente operazioni modulari di visualizzazione, salvataggio e confronto dei dati
 - Favorisce la manutenzione e il riutilizzo del codice tramite funzioni di accesso (getter/setter)
-

LISTA LEZIONI

```
struct node
{
    Lezione lez;
    struct node *next;
};
```

Motivazione

Nel contesto dell'applicazione sviluppata, l'utilizzo di una struttura dati di tipo lista per la gestione delle lezioni risulta essere una scelta consigliata e funzionalmente adeguata. Le lezioni, come evidenziato nel modulo `lezione.c`, vengono create dinamicamente tramite allocazione di memoria, il che implica che il numero totale di lezioni non è noto a priori e può cambiare nel tempo. Una lista consente di superare questa limitazione, offrendo una gestione dinamica della memoria, senza i vincoli imposti da strutture statiche come gli array. Inoltre, la possibilità di ordinare le lezioni in base alla data permette una visualizzazione più utile e intuitiva per l'utente, facilitando la pianificazione e la prenotazione delle attività.

Vantaggi

- Supporta la gestione dinamica della memoria, adattandosi a un numero variabile di lezioni
 - Permette l'ordinamento delle lezioni in base alla data per migliorare l'esperienza utente
 - Facilita operazioni di consultazione, aggiornamento e salvataggio su file grazie alla natura sequenziale
-

HASH TABLE PER CLIENTI E PRENOTAZIONI

```
struct hash
{
    int size;           // Dimensione della tabella hash.
    Cliente *table;     // Vettore di puntatori a Cliente.
};
```

```
struct hash_p
{
    int size;           // Dimensione della tabella hash.
    Prenotazione *table; // Vettore di puntatori a Prenotazione.
};
```

Motivazione

Nel contesto della gestione dei clienti e delle prenotazioni rappresentati rispettivamente dalla struttura definita nel file `cliente.c` e `prenotazione.c`, è consigliato utilizzare una **hash table (tabella hash)** per migliorare significativamente l'efficienza delle operazioni di ricerca, inserimento e cancellazione, soprattutto quando il numero di clienti e di prenotazioni diventa elevato. Poiché sia cliente che prenotazioni hanno un identificativo univoco (`id`) rappresentato da una stringa, è opportuno utilizzare tale identificativo come **chiave** per la struttura.

Vantaggi

- **Ricerca più veloce:**
In una lista collegata, la ricerca di un cliente o di prenotazione specifica richiede un'operazione lineare ($O(n)$), dove n è il numero di elementi. Al contrario, una hash table consente di accedere direttamente all'elemento tramite il suo ID.
- **Inserimento e cancellazione efficienti:**
Le operazioni di inserimento e rimozione possono anch'esse essere eseguite in tempo costante nella hash table, evitando la necessità di scorrere la lista per trovare il punto esatto dove intervenire.
- **Scalabilità:**
Man mano che il sistema cresce e il numero di clienti o prenotazioni aumenta, una hash table mantiene prestazioni elevate senza degradazioni significative, a differenza della lista collegata che tende a rallentare.

- **Accesso diretto tramite chiave (ID):**

Dal momento che l'ID è univoco e sempre presente, risulta perfetto come chiave per la funzione di hash. Questo consente un accesso diretto senza dover confrontare ogni campo della struttura.

- **Migliore organizzazione dei dati:**

Con una hash table è più facile implementare operazioni avanzate, come il controllo rapido di duplicati, l'accesso rapido ai dati salvati su file, o l'integrazione con sistemi di caching.

ii. COME USARE E INTERAGIRE CON IL SISTEMA

Il programma è strutturato per simulare la **gestione di una palestra**, con due macro-aree operative: una per il **gestore** e una per il **cliente**. Il menù iniziale consente di accedere a una delle due aree, oppure di uscire dal programma.

- ✓ **1** → Accesso area **gestore**
- ✓ **2** → Accesso area **cliente**
- ✓ **0** → **Uscita**

```
=====
BENVENUTO NELLA TUA PALESTRA !
=====
1) Per il gestore della palestra
2) Per l'area cliente (Devi conoscere il tuo ID)
0) Per uscire
Scelta: █
```

AREA CLIENTE:

Quando un cliente vuole accedere alla sua area riservata, viene avviata la funzione `login_cliente`. Questo è il punto di partenza per l'interazione personalizzata con il sistema.

Il programma richiede l'inserimento dell'ID del cliente. Una volta ricevuto, vengono eseguiti dei controlli importanti:

1. **Controllo dell'esistenza:** l'ID viene cercato nella tabella hash dei clienti. Se non esiste, il sistema mostra un messaggio di errore e non consente l'accesso.

```
=====
BENVENUTO NELLA TUA AREA !
=====
Inserisci il tuo ID cliente: C018
Cliente non trovato. Verifica l'ID inserito.

Premi INVIO per tornare al menu...█
```

2. **Controllo della validità dell'abbonamento:** se il cliente esiste ma il suo abbonamento è scaduto (verificato confrontando la data odierna con la data di scadenza), viene mostrato un **messaggio di avviso**.

```
=====
BENVENUTO NELLA TUA AREA !
=====
Inserisci il tuo ID cliente: C003
Attenzione, abbonamento scaduto!

Premi INVIO per andare avanti...█
```

Tuttavia, il **login viene comunque completato**: il cliente può accedere al menu, ma alcune funzionalità saranno limitate (es. non potrà prenotare lezioni).

Appena si entra nel `menu_cliente`, il sistema saluta il cliente mostrando nome e cognome (estratti dall'oggetto `Cliente`), e presenta un elenco numerato di opzioni.

```

=====
Benvenuto, Michele Vitiello!
=====
1) Visualizza informazioni sul tuo abbonamento
2) Rinnova abbonamento
3) Prenota una lezione
4) Disdici una prenotazione
5) Visualizza lezioni
6) Visualizza le tue prenotazioni
0) Esci
=====
Scegli un'opzione: █

```

Il menu è strutturato in un **ciclo do-while**, il che significa che viene mostrato **ripetutamente**, finché l'utente non sceglie l'opzione "0" per uscire. Ogni scelta dell'utente viene interpretata tramite uno **switch-case**.

1. VISUALIZZA INFORMAZIONI SULL'ABBONAMENTO

Se il cliente sceglie l'opzione **1**, viene mostrata una schermata con i dettagli dell'abbonamento.

Subito dopo, viene controllata la **validità dell'abbonamento**: se è ancora valido, viene stampato "Stato abbonamento: Valido", altrimenti "Scaduto".

Questa verifica è fatta tramite la funzione `abbonamento_valido`, che confronta la **data di oggi** con la **data di scadenza** dell'abbonamento.

```

=====
INFORMAZIONI SULL'ABBONAMENTO
=====
Durata abbonamento: 20 mesi
Data iscrizione: 17/05/2025
Data scadenza: 17/01/2027
Stato abbonamento: Valido

Premi INVIO per tornare al menu... █

```

2. RINNOVA ABBONAMENTO

Se il cliente seleziona l'opzione **2**, può procedere al rinnovo del proprio abbonamento. Il sistema richiederà all'utente di inserire la durata del rinnovo, verificando che il valore inserito sia maggiore di zero. Successivamente, verrà chiamata la funzione `rinnova_abbonamento`, che aggiornerà la data di scadenza e la durata (espressa in mesi) dell'abbonamento direttamente nell'oggetto cliente memorizzato nella tabella hash.

```

=====
Benvenuto, Michele Vitiello!
=====
1) Visualizza informazioni sul tuo abbonamento
2) Rinnova abbonamento
3) Prenota una lezione
4) Disdici una prenotazione
5) Visualizza lezioni
6) Visualizza le tue prenotazioni
0) Esci
=====
Scegli un'opzione: 2
Inserisci la durata del rinnovo in mesi (numero intero positivo): 1
Il tuo abbonamento e' stato rinnovato fino al 17/02/2027

Premi INVIO per tornare al menu... █

```


3. PRENOTA UNA LEZIONE

Quando il cliente sceglie l'opzione 3 dal menu, ha la possibilità di prenotare una lezione tra quelle disponibili. Il processo, però, è un po' più complesso e comprende vari controlli per garantire che tutto avvenga correttamente.

Prima di tutto, il sistema verifica se l'abbonamento del cliente è ancora valido. Se l'abbonamento è scaduto, il sistema blocca immediatamente la possibilità di prenotare, mostrando un messaggio chiaro che invita a rinnovare l'abbonamento per poter procedere: "Abbonamento scaduto, rinnovalo per effettuare prenotazioni". Questo passaggio è fondamentale per assicurarsi che solo clienti attivi possano accedere alle lezioni.

Se invece l'abbonamento è valido, il sistema procede mostrando all'utente una lista di lezioni disponibili. Queste lezioni sono selezionate accuratamente: sono quelle che non si sono ancora svolte e che hanno ancora posti liberi, quindi il cliente può scegliere solo tra lezioni prenotabili.

A questo punto, il cliente inserisce l'ID della lezione che desidera prenotare. Inserito questo dato, il sistema effettua una serie di verifiche per assicurarsi che la prenotazione possa effettivamente essere effettuata:

- Verifica che la lezione esista davvero nella lista delle lezioni.
- Controlla che la lezione non si sia già svolta, perché non avrebbe senso prenotare una lezione passata.
- Controlla che ci siano ancora posti disponibili, ovvero che la lezione non sia già al completo.
- Verifica infine che il cliente non sia già prenotato a quella stessa lezione, per evitare prenotazioni duplicate.

Se tutte queste condizioni vengono soddisfatte, il sistema procede con la prenotazione vera e propria. Incrementa il numero di posti occupati nella lezione, segnalando così che un posto è stato preso. Crea quindi una nuova prenotazione associata al cliente e alla lezione scelta.

Questa nuova prenotazione viene inserita nella tabella hash che gestisce tutte le prenotazioni, mantenendo così l'organizzazione interna dei dati. Inoltre, tutte le modifiche, compresa la nuova prenotazione e l'aggiornamento dei posti occupati, vengono salvate su file di testo, per garantire la persistenza delle informazioni anche dopo la chiusura del programma.

Come ultima cosa, il sistema comunica all'utente che la prenotazione è andata a buon fine con un messaggio semplice e diretto: "Prenotazione effettuata con successo!".

LISTA DELLE LEZIONI DISPONIBILI				
ID	NOME	ORA	DATA	POSTI
L003	Crossfit	18:00	27/05/2025	2/20
L004	Sala pesi	19:30	29/05/2025	1/22
L005	Zumba	17:00	02/06/2025	0/20
L006	Pilates	15:30	05/06/2025	0/15
L007	Danza Classica	14:00	10/06/2025	0/10
L008	Pugilato	12:00	12/06/2025	4/12
L009	Pugilato	18:00	15/06/2025	0/18
L010	Crossfit	16:00	20/06/2025	1/25
Inserisci l'ID della lezione da prenotare: <input type="text"/>				

4. DISDICI UNA PRENOTAZIONE

Quando l'utente seleziona l'opzione 4 nel menu, il sistema gli permette di annullare una prenotazione che ha già effettuato in precedenza. La prima cosa che il programma fa è mostrare all'utente una lista completa di tutte le prenotazioni attive associate al suo profilo. In questo modo, il cliente ha sott'occhio tutte le sue prenotazioni attuali e può scegliere con facilità quale desidera cancellare.

A questo punto, viene richiesto all'utente di inserire l'ID della prenotazione che intende disdire. Questa operazione serve a identificare in modo univoco la prenotazione specifica, evitando confusione o cancellazioni accidentali.

Il sistema quindi effettua due controlli fondamentali:

- ✓ Verifica che la prenotazione esista effettivamente all'interno del sistema. Se l'ID inserito non corrisponde a nessuna prenotazione, viene notificato all'utente che la prenotazione non è stata trovata.
- ✓ Controlla che la prenotazione appartenga proprio al cliente che sta tentando di disdirla. Questo è un passaggio cruciale per garantire la sicurezza e la correttezza delle operazioni, evitando che un cliente possa cancellare prenotazioni di altri.

Se questi controlli vengono superati con successo, il sistema procede a rimuovere la prenotazione dalla struttura dati interna, ovvero dalla tabella hash che gestisce tutte le prenotazioni. Inoltre, poiché una prenotazione annullata libera un posto nella lezione corrispondente, il sistema aggiorna anche il conteggio dei posti occupati, decrementandolo di uno.

Infine, tutte queste modifiche vengono salvate sui file di testo per mantenere aggiornati i dati anche al di fuori della sessione di esecuzione del programma.

Come conferma finale per l'utente, viene stampato il messaggio "Prenotazione disdetta con successo!", così da fornire un feedback immediato e positivo sull'operazione appena effettuata.

```
=====
LISTA DELLE PRENOTAZIONI DI: Michele Vitiello!
=====
ID          LEZIONE          DATA          ORA
P003        Crossfit          27/05/2025    18:00
P002        Pugilato              12/06/2025    12:00
-----
Inserisci l'ID della prenotazione da disdire: █
```

5. VISUALIZZA LEZIONI

L'opzione **5** mostra al cliente le lezioni disponibili. Quando il cliente sceglie questa opzione, il programma chiama la funzione `visualizza_lezioni`. Questa funzione innanzitutto mostra un semplice menu con le due opzioni, oltre a un'opzione per tornare indietro senza visualizzare nulla.

```
=====
                        LISTA DELLE LEZIONI
=====
Cosa vuoi visualizzare?
1) Tutte le lezioni
2) Solo lezioni disponibili (non passate e con posti liberi)
0) Torna indietro
Scelta: █
```

Se il cliente sceglie di visualizzare tutte le lezioni, il programma scorre tutta la lista delle lezioni registrate e le stampa una per una, senza filtrare nulla. Questo significa che il cliente potrà vedere sia lezioni passate che future, indipendentemente dalla disponibilità di posti.

Se invece il cliente sceglie di visualizzare solo le lezioni disponibili, la funzione applica due filtri fondamentali:

- La data della lezione deve essere uguale o successiva alla data odierna, quindi la lezione non deve essere già passata.
- La lezione deve avere ancora posti disponibili, cioè il numero di posti occupati deve essere inferiore al numero massimo consentito.

La funzione scorre quindi la lista, controlla questi due criteri e mostra solo le lezioni che li soddisfano. Questo aiuta il cliente a concentrarsi solo sulle opportunità effettivamente ancora prenotabili.

Nel caso in cui, con il filtro selezionato, non risultino lezioni da mostrare, il programma comunica chiaramente che non ci sono lezioni disponibili con quei criteri. Questo evita confusione e permette al cliente di capire subito che non ci sono opzioni prenotabili.

6. VISUALIZZA LE TUE PRENOTAZIONI

Con l'opzione **6**, il cliente può vedere tutte le lezioni a cui **è prenotato**.

Viene semplicemente chiamata la funzione `stampa_prenotazioni_cliente`, che attraversa la hashtable delle prenotazioni e cerca tutte quelle **associate all'ID del cliente**.

```
=====
LISTA DELLE PRENOTAZIONI DI: Michele Vitiello!
=====
ID      LEZIONE      DATA      ORA
P003    Crossfit      27/05/2025 18:00
P002    Pugilato      12/06/2025 12:00

Premi INVIO per tornare al menu...█
```

0. ESCI

Infine, selezionando "0", l'utente esce dal menu. Il programma esce dal ciclo `do-while` e termina l'interazione col cliente, tornando così al menu di partenza.

AREA PER LA GESTIONE DELLA PALESTRA:

Una volta entrato nell'area della gestione della palestra, verrà visualizzata una schermata con il titolo *"BENVENUTO NELLA AREA GESTORE!"* e un elenco numerato delle operazioni disponibili:

```
=====
BENVENUTO NELLA AREA GESTORE!
=====
1) Inserisci un nuovo cliente
2) Inserisci una nuova lezione
3) Visualizza la lista dei clienti
4) Rinnova l'abbonamento di un cliente
5) Visualizza la lista delle lezioni
6) Rimuovi un cliente
7) Rimuovi una lezione
8) Ricerca un cliente
9) Visualizza le prenotazioni di un cliente
10) Visualizza le prenotazioni di una lezione
11) Genera il report mensile
0) Esci
=====
Scegli un'opzione: █
```

1. INSERISCI UN NUOVO CLIENTE

Quando si seleziona l'opzione **1 – Inserisci un nuovo cliente**, il sistema avvia una procedura guidata per registrare un nuovo cliente all'interno dell'archivio.

La prima informazione richiesta è il **nome** del cliente. Il sistema effettua un controllo per assicurarsi che il nome sia composto esclusivamente da lettere. Qualora venissero inseriti numeri o simboli, verrà mostrato un messaggio d'errore e sarà necessario reinserire correttamente il dato.

Successivamente viene richiesto il **cognome**, anch'esso soggetto allo stesso tipo di controllo: deve contenere solo lettere. Anche in questo caso, in presenza di caratteri non validi, il sistema inviterà a ripetere l'inserimento.

Dopo aver acquisito nome e cognome, verrà chiesta la **durata dell'abbonamento**, espressa in mesi. È necessario inserire un numero intero positivo. Se il valore inserito è nullo, negativo o non numerico, il sistema lo segnala come errore e chiede di reinserirlo.

Una volta raccolte tutte le informazioni in modo corretto, il sistema genera automaticamente un **codice identificativo univoco** per il cliente e procede con la registrazione. Il nuovo cliente viene così inserito all'interno della struttura dati e salvato anche nel file dedicato, così da garantire la conservazione delle informazioni.

Infine, viene mostrato un messaggio che conferma l'avvenuto inserimento, e verrà stampato l'ID del cliente appena inserito. In caso di duplicazione dell'identificativo, il sistema segnala l'anomalia e non esegue l'inserimento.

```
=====
INSERISCI UN CLIENTE
=====
Inserisci il nome: Antonio
Inserisci il cognome: Cimmino
Inserisci la durata dell'abbonamento (in mesi): 3
Cliente inserito con successo, il suo ID e': C015

Premi INVIO per tornare al menu...█
```

2. INSERISCI UNA NUOVA LEZIONE

Se si seleziona l'opzione **"2) Inserisci una nuova lezione"**, il sistema attiva una procedura guidata che consente di creare una nuova attività da inserire nel calendario delle lezioni. Durante questa fase, verranno richieste alcune informazioni essenziali, con controlli integrati per garantire che i dati inseriti siano corretti e coerenti.

La prima richiesta riguarda il **nome della lezione**. Il sistema mostra l'elenco delle attività consentite (Sala pesi - Pugilato - Danza Classica - Yoga - Zumba - Pilates - Crossfit), invitando a scegliere uno di questi nomi. È importante che il nome sia composto esclusivamente da lettere: eventuali numeri o simboli comportano la ripetizione dell'inserimento. Inoltre, se il nome non appartiene alla lista di lezioni ammesse, viene notificato un errore.

Successivamente, viene chiesto di inserire l'**orario della lezione**, nel formato HH:MM. Il sistema verifica che l'orario sia inserito correttamente e che i valori per ore e minuti siano validi. In caso di errore di formato (ad esempio, lettere o separatori errati), viene chiesto di ripetere l'inserimento.

Una volta definito l'orario, viene richiesto di indicare la **data** della lezione. Qui il sistema impone una regola: la data deve essere compresa **tra il mese corrente e il mese successivo**. Qualsiasi data al di fuori di questo intervallo non viene accettata, per evitare di pianificare lezioni troppo lontane nel tempo.

Infine, si deve specificare il **numero massimo di partecipanti**. È necessario inserire un numero positivo e non superiore a **30**, che rappresenta il limite massimo consentito per ogni lezione. Qualora il numero inserito sia nullo, negativo o superiore al limite, il sistema mostrerà un messaggio d'errore e richiederà l'inserimento corretto.

Una volta raccolte tutte le informazioni, il sistema procede a generare automaticamente un **codice identificativo univoco** per la nuova lezione. A operazione completata, viene mostrato un messaggio che conferma l'avvenuta registrazione della lezione, e viene mostrato l'ID della lezione appena inserita.

```
=====
INSERISCI UNA LEZIONE
=====
Le lezioni consentite sono:
- Sala pesi - Pugilato - Danza Classica - Yoga
- Zumba - Pilates - Crossfit
-----
Inserisci il nome: Sala pesi
Inserisci l'orario della lezione (HH:MM): 16:00
Inserisci la data (GG/MM/AAAA): 30/06/2025
Inserisci il numero massimo di posti (max 30): 18
Lezione inserita con successo, con ID: L011

Premi INVIO per tornare al menu...█
```

3. VISUALIZZA LA LISTA DEI CLIENTI

Se si seleziona l'opzione **"3) Visualizza la lista dei clienti"**, il sistema mostra a schermo l'elenco completo di tutti i clienti attualmente all'interno della struttura, e della somma di clienti registrati alla palestra. Questa funzione consente al gestore di avere una panoramica immediata e ordinata di tutti gli utenti presenti nel sistema.

4. RINNOVA L'ABBONAMENTO DI UN CLIENTE

Se si seleziona l'opzione **"4) Rinnova l'abbonamento di un cliente"**, il sistema consente di prolungare la validità dell'abbonamento per uno degli utenti già iscritti.

La procedura inizia mostrando una **lista semplificata di tutti i clienti presenti**, con le informazioni essenziali (ID, Nome, Cognome). Questo aiuta il gestore a identificare il cliente corretto, evitando errori durante l'inserimento.

Una volta visualizzato l'elenco, il sistema richiede di **inserire manualmente l'ID del cliente** per il quale si desidera effettuare il rinnovo. Dopo l'inserimento, il sistema verifica che l'ID corrisponda effettivamente a un cliente esistente.

- **Se il cliente non viene trovato**, viene mostrato un messaggio di errore e non viene eseguita alcuna operazione.
- **Se il cliente esiste**, il sistema procede con il **rinnovo dell'abbonamento**, aggiornando la durata e la data di scadenza.

Al termine dell'operazione, viene mostrato un messaggio di conferma che indica l'avvenuto rinnovo, specificando anche il nome del cliente interessato.

```
=====
                        LISTA DEI CLIENTI
=====
C014  Vitiello    Ferdinando
C006  Scarico    Giovanni
C005  Vitiello    Rosa
C012  Vitiello    Michele
C008  Boy        Tony
C007  Cesarano   Tobia
C002  Vitiello    Domenico
C015  Cimmino    Antonio
C001  Scoffield  Michael
C009  Vertolomo  Gerardo
C004  Auditore   Ezio
C013  testing    ClienteTest
C003  Auditore   Ezio

Inserisci l'ID del cliente a cui rinnovare: C014
Inserisci la durata del rinnovo in mesi (numero intero positivo): 1
Il tuo abbonamento e' stato rinnovato fino al 06/11/2025

Abbonamento rinnovato con successo per Ferdinando.

Premi INVIO per tornare al menu...|
```

5. VISUALIZZA LA LISTA DELLE LEZIONI

Se si seleziona l'opzione **"5) Visualizza la lista delle lezioni"**, il sistema mostra a schermo tutte le lezioni attualmente registrate nel sistema.

Questa funzionalità permette al gestore di consultare in modo chiaro e ordinato l'elenco delle lezioni inserite, con i dettagli principali relativi a ciascuna attività. Le informazioni visualizzate includono tipicamente:

- Il **nome della lezione**;
- La **data e l'orario di svolgimento**;
- Il **numero massimo di posti disponibili**;
- L'**ID univoco** associato alla lezione.

Le lezioni vengono stampate **in ordine cronologico**, partendo da quelle più imminenti fino ad arrivare a quelle programmate più avanti nel tempo. In questo modo, il gestore

ha immediatamente sotto controllo le attività in programma nel futuro prossimo, facilitando l'organizzazione.

6. RIMUOVI UN CLIENTE

Se si seleziona l'opzione **"6) Rimuovi un cliente"**, il sistema permette al gestore di eliminare un cliente dall'elenco registrato, insieme a tutte le prenotazioni a lui collegate. Inizialmente, viene mostrata una **lista semplificata di tutti i clienti**, utile per aiutare il gestore a individuare l'ID del cliente da rimuovere. Dopo questa visualizzazione, il sistema richiede di **inserire l'ID del cliente** che si desidera eliminare.

Una volta inserito l'ID, il sistema verifica che questo corrisponda a un cliente effettivamente esistente:

- Se il cliente **non viene trovato**, viene mostrato un messaggio di errore e l'operazione viene annullata.
- Se invece il cliente **viene trovato**, viene rimosso dalla tabella dei clienti e cancellato anche dal file di salvataggio permanente.

Inoltre, il sistema si occupa automaticamente di **eliminare tutte le prenotazioni associate a quel cliente**, così da evitare incongruenze nei dati (ad esempio, prenotazioni "orfane" legate a un cliente non più esistente).

```
=====
LISTA DEI CLIENTI
=====
C014  Vitiello    Ferdinando
C006  Scarico     Giovanni
C005  Vitiello    Rosa
C012  Vitiello    Michele
C008  Boy         Tony
C007  Cesarano   Tobia
C002  Vitiello    Domenico
C015  Cimmino    Antonio
C001  Scoffield  Michael
C009  Vertolomo  Gerardo
C004  Auditore   Ezio
C013  testing    ClienteTest
C003  Auditore   Ezio
=====
RIMUOVI UN CLIENTE
=====
Inserisci l'ID del cliente da rimuovere: C015
Cliente rimosso!

Premi INVIO per tornare al menu...|
```

7. RIMUOVI UNA LEZIONE

Quando viene selezionata l'opzione **"7) Rimuovi una lezione"**, il sistema consente al gestore di eliminare una lezione già programmata, insieme a tutte le prenotazioni effettuate per essa.

All'avvio dell'opzione, viene mostrata una **lista sintetica di tutte le lezioni attualmente in programma**, ognuna identificata dal proprio ID univoco. Questo permette al gestore di identificare con precisione la lezione che intende cancellare.

Successivamente, viene richiesto di **inserire l'ID della lezione da rimuovere**. Una volta ricevuto l'input, il sistema:

- Controlla che l'ID corrisponda a una lezione esistente.**
- Se la lezione viene trovata, **la rimuove dalla lista**, liberando la memoria e aggiornando il file contenente tutte le lezioni salvate.

- Se invece l'ID non è valido o non esiste, viene mostrato un messaggio di errore e l'operazione viene interrotta.

Oltre alla rimozione della lezione, il sistema effettua automaticamente anche la **cancellazione di tutte le prenotazioni collegate a quella lezione**, evitando di mantenere riferimenti a lezioni non più esistenti.

```
=====
LISTA DELLE LEZIONI
=====
L001    Yoga          10:00    15/05/2025    0/15
L002    Sala pesi     09:00    17/05/2025    0/18
L003    Crossfit      18:00    27/05/2025    2/20
L004    Sala pesi     19:30    29/05/2025    1/22
L007    Danza Classica 14:00    17/06/2025    0/10
L006    Pilates       15:30    18/06/2025    0/15
L008    Pugilato      12:00    19/06/2025    4/12
L010    Crossfit      16:00    20/06/2025    2/25
L005    Zumba         17:00    24/06/2025    0/20
L009    Pugilato      18:00    28/06/2025    0/18
L011    Sala pesi     16:00    30/06/2025    0/18
=====
RIMUOVI UNA LEZIONE
=====
Inserisci l'ID della lezione da rimuovere: L011
Lezione rimossa!

Premi INVIO per tornare al menu...|
```

8. RICERCA UN CLIENTE

Selezionando l'opzione **"8) Ricerca un cliente"**, il gestore può visualizzare in dettaglio le informazioni di un cliente specifico, partendo dal suo **ID univoco**.

Il sistema apre la schermata dedicata alla ricerca e **richiede l'inserimento dell'ID del cliente da cercare**. Se l'ID inserito è vuoto o non corrisponde ad alcun cliente registrato nella tabella, viene stampato un messaggio di errore e l'operazione si conclude senza modificare nulla.

Se invece il cliente viene trovato:

- Vengono **stampati a video tutti i suoi dati anagrafici** (nome, cognome, data di iscrizione, ecc.).
- Viene anche mostrato lo **stato attuale dell'abbonamento**, specificando se è **ancora valido oppure scaduto**, in base alla data attuale e alla data di scadenza dell'abbonamento del cliente.

Questa funzione è utile per fornire al gestore un **accesso rapido e puntuale alle informazioni di un cliente**, ad esempio in caso di dubbi sulla validità dell'abbonamento o per verifiche durante il check-in in palestra.

```
=====
RICERCA UN CLIENTE
=====
Inserisci l'ID del cliente da cercare: C012
Cliente trovato:
=====
ID: C012
Nome: Michele
Cognome: Vitiello
Durata abbonamento: 21 mesi
Data iscrizione: 17/05/2025
Data scadenza: 17/02/2027
Stato abbonamento: Valido

Premi INVIO per tornare al menu...|
```


9. VISUALIZZA LE PRENOTAZIONI DI UN CLIENTE

Selezionando l'opzione “9) Visualizza prenotazioni”, il sistema consente al gestore di **consultare l'elenco completo delle prenotazioni effettuate da un determinato cliente**.

Come avviene l'interazione:

1. Viene inizialmente mostrata la **lista sintetica dei clienti** registrati, così da aiutare il gestore a individuare rapidamente l'ID del cliente interessato.
2. Dopo la visualizzazione, il sistema richiede l'**inserimento dell'ID del cliente** da esaminare.
3. Se l'ID inserito **non è valido o non corrisponde a nessun cliente**, viene stampato un messaggio di errore e l'operazione si interrompe.

Se il cliente viene trovato:

- ✓ Lo schermo viene pulito, e viene mostrata una **tabella riepilogativa delle prenotazioni effettuate da quel cliente**.
- ✓ Per ogni prenotazione vengono riportate:
 - L'**ID della prenotazione**
 - Il **nome della lezione**
 - La **data e l'orario** della lezione prenotata

Il sistema recupera questi dati incrociando le informazioni contenute nella **tabella delle prenotazioni** con quelle presenti nella **lista delle lezioni**.

Se il cliente **non ha prenotazioni registrate**, viene visualizzato un messaggio esplicito.

VISUALIZZA PRENOTAZIONI			
C014	Vitiello	Ferdinando	
C006	Scarico	Giovanni	
C005	Vitiello	Rosa	
C012	Vitiello	Michele	
C008	Boy	Tony	
C007	Cesarano	Tobia	
C002	Vitiello	Domenico	
C001	Scofield	Michael	
C009	Vertolomo	Gerardo	
C004	Auditore	Ezio	
C013	testing	ClienteTest	
C003	Auditore	Ezio	
Inserisci l'ID del cliente: C012			

LISTA DELLE PRENOTAZIONI DI: Michele Vitiello!			
ID	LEZIONE	DATA	ORA
P010	Crossfit	20/06/2025	16:00
P002	Pugilato	19/06/2025	12:00

10. VISUALIZZA LE PRENOTAZIONI DI UNA LEZIONE

Selezionando l'opzione "**10) Visualizza prenotazioni lezione**", il sistema permette al gestore di vedere tutte le prenotazioni relative a una specifica lezione.

Come avviene l'interazione:

1. Viene mostrata una lista sintetica delle lezioni disponibili, così da facilitare l'individuazione dell'ID della lezione di interesse.
2. Il sistema richiede l'inserimento dell'ID della lezione di cui si vogliono visualizzare le prenotazioni.
3. Se l'ID inserito non è valido o non corrisponde a nessuna lezione presente nella lista, viene mostrato un messaggio di errore e l'operazione termina.

Se la lezione viene trovata:

- Lo schermo viene pulito e viene mostrato il dettaglio della lezione selezionata, con nome, data e orario.
- Successivamente viene visualizzata una tabella riepilogativa delle prenotazioni per quella lezione.
- Per ogni prenotazione vengono riportati:
 - L'ID della prenotazione
 - Il nome e il cognome del cliente che ha effettuato la prenotazione

Se non sono presenti prenotazioni per la lezione selezionata, viene visualizzato un messaggio chiaro che indica l'assenza di prenotazioni. In questo modo il gestore può consultare rapidamente chi ha prenotato una certa lezione e organizzare meglio le attività.

=====

VISUALIZZA PRENOTAZIONI

=====

L001	Yoga	10:00	15/05/2025	0/15
L002	Sala pesi	09:00	17/05/2025	0/18
L003	Crossfit	18:00	27/05/2025	2/20
L004	Sala pesi	19:30	29/05/2025	1/22
L007	Danza Classica	14:00	17/06/2025	0/10
L006	Pilates	15:30	18/06/2025	0/15
L008	Pugilato	12:00	19/06/2025	4/12
L010	Crossfit	16:00	20/06/2025	2/25
L005	Zumba	17:00	24/06/2025	0/20
L009	Pugilato	18:00	28/06/2025	0/18

=====

Inserisci l'ID della lezione: L008

=====

LISTA DELLE PRENOTAZIONI DELLA LEZIONE DI: Pugilato!

DATA: 19/06/2025 ORARIO: 12:00

=====

ID	CLIENTE	COGNOME
P004	Michael	Scoffield
P009	Giovanni	Scarico
P011	Ferdinando	Vitiello
P002	Michele	Vitiello

Premi INVIO per tornare al menu...

11. GENERA REPORT MENSILE

Selezionando l'opzione "**11) Genera report mensile**", il sistema crea un riepilogo delle prenotazioni effettuate nel mese corrente.

Il programma analizza tutte le prenotazioni fatte in questo mese, aggiornando una struttura che conta quante volte ogni lezione è stata prenotata.

Dopo aver ordinato le lezioni in base al numero di prenotazioni, viene generato un file di testo con:

- Il totale delle prenotazioni del mese.
- Le prime 3 lezioni più prenotate che si svolgono nel mese, con nome, data e orario.

Alla fine, viene mostrato un messaggio che conferma la creazione del report.

Se si verifica un errore, viene segnalato a video.

```
Report mensile generato correttamente in 'report_Giugno_2025.txt'  
  
Premi INVIO per tornare al menu...
```

```
≡ report_Giugno_2025.txt  
1  REPORT MENSILE - Giugno 2025  
2  
3  Totale prenotazioni effettuate: 7  
4  
5  Lezioni più frequentate (solo quelle di Giugno):  
6  1. Pugilato - 4 prenotazioni  
7    | Data: 19/06/2025 | Ora: 12:00  
8  2. Crossfit - 2 prenotazioni  
9    | Data: 20/06/2025 | Ora: 16:00  
10 3. Pugilato - 1 prenotazioni  
11 | Data: 28/06/2025 | Ora: 18:00  
12
```

Al termine dell'esecuzione del programma, cioè quando si sceglie di uscire, il sistema genera automaticamente il report mensile delle prenotazioni. In questo modo, il gestore riceve sempre un riepilogo aggiornato e completo delle attività del mese appena concluso, senza dover richiedere manualmente il report.

III. PROGETTAZIONE

CLIENTE

- **crea_cliente**
 1. Alloca memoria per il cliente e per la stringa ID
 2. Copia i dati passati nella nuova struttura
 3. Calcola la data di scadenza dell'abbonamento con `calcolo_scadenza_abbonamento`
 4. Imposta il puntatore next a NULL
- **distruggi_cliente**
 1. Se il cliente è NULL non fa nulla
 2. Libera memoria per ID, date e struttura cliente
- **salva_cliente_file**
 1. Apre il file in modalità appending
 2. Scrive i dati nel file.
 3. Chiude il file
- **visualizza_cliente**
 1. Se c è NULL, stampa "Cliente non valido"
 2. Stampa ID, nome, cognome e chiama la funzione `visualizza_abbonamento_cliente`
- **visualizza_abbonamento_cliente**
 1. Se c == NULL, stampa "Cliente non valido"
 2. Altrimenti stampa durata abbonamento, data iscrizione e data scadenza
- **visualizza_essenziale_cliente**
 1. Se c == NULL, stampa "Cliente non valido"
 2. Altrimenti stampa ID, cognome e nome in formato tabellare
- **confronta_clienti**
 1. Se uno dei due è NULL, restituisce 0
 2. Altrimenti confronta gli ID dei due clienti tramite il comando `strcmp`
- **get_id_cliente**
 1. Se c == NULL, restituisce NULL
 2. Altrimenti, restituisce l'id del cliente
- **get_nome_cliente**
 1. Se c == NULL, restituisce NULL
 2. Altrimenti, restituisce il nome del cliente
- **get_cognome_cliente**
 1. Se c == NULL, restituisce NULL
 2. Altrimenti, restituisce il cognome del cliente
- **get_durata_abbonamento**
 1. Se c == NULL, restituisce -1
 2. Altrimenti, restituisce la durata dell'abbonamento del cliente
- **get_data_iscrizione**
 1. Se c == NULL, restituisce NULL
 2. Altrimenti, restituisce la data d'iscrizione dell'abbonamento del cliente
- **get_data_scadenza**
 1. Se c == NULL, restituisce NULL
 2. Altrimenti, restituisce la data di scadenza dell'abbonamento del cliente
- **get_next_cliente**
 1. Se c == NULL, restituisce NULL
 2. Altrimenti, restituisce il puntatore al cliente successivo
- **set_next_cliente**
 1. Aggiorna il campo next del cliente

- **set_data_scadenza**
 1. Se `c == NULL` e `nuova_data == NULL`, stampa a schermo un errore
 2. Altrimenti, aggiorna la data di scadenza dell'abbonamento del cliente
 - **set_durata**
 1. Se `c == NULL`, stampa a schermo un errore
 2. Altrimenti, aggiorna la durata dell'abbonamento del cliente
-

DATA

- **crea_data**
 1. Alloca memoria
 2. Se `malloc` fallisce, stampa errore e termina il programma
 3. Inizializza giorno, mese e anno
 4. Ritorna la nuova data creata
 - **visualizza_data**
 1. Stampa a schermo la data secondo il formato GG/MM/AAAA
 - **confronta_date**
 1. Confronta anno, poi mese, poi giorno in ordine
 2. Restituisce un valore intero in base all'esito del confronto (-1 se `d1 < d2` / 0 se `d1 == d2` / 1 se `d1 > d2`)
 - **leggi_data**
 1. Richiede input in formato GG/MM/AAAA
 2. Controlla che la data sia valida e successiva ad oggi
 3. Ripete la richiesta in caso di errore
 - **copia_data**
 1. Alloca nuova struttura Data
 2. Termina il programma in caso di errore di allocazione
 3. Copia i campi giorno, mese, anno
 4. Ritorna la nuova data
 - **libera_data**
 1. Se `d` è NULL non fa nulla
 2. Altrimenti libera la memoria
 - **data_oggi**
 1. Usa funzioni della libreria `time.h` per ottenere la data corrente.
 2. Restituisce una Data allocata dinamicamente
 - **get_giorno**
 1. Se `d == NULL`, restituisce -1
 2. Altrimenti, restituisce il giorno, in formato numerico, della data
 - **get_mese**
 1. Se `d == NULL`, restituisce -1
 2. Altrimenti, restituisce il mese, in formato numerico, della data
 - **get_anno**
 1. Se `d == NULL`, restituisce -1
 2. Altrimenti, restituisce l'anno, in formato numerico, della data
-

ORARIO

- **crea_orario**
 1. Verifica che ore e minuti siano nei range validi
 2. Alloca memoria per un nuovo Orario
 3. Assegna i valori e restituisce il puntatore

- **copia_orario**
 1. Alloca nuova memoria per un nuovo Orario
 2. Copia ore e minuti dall'orario originale passato in input
 3. Restituisce la nuova copia
 - **stampa_orario**
 1. Se l'orario è NULL, stampa messaggio di errore
 2. Altrimenti, stampa ore e minuti con formato oo:mm
 - **confronta_orario**
 1. Confronta prima le ore tra i due orari.
 2. Se le ore sono uguali, confronta i minuti.
 3. Restituisce il risultato della comparazione (-1 se o1 è prima di o2 / 0 se o1 == o2 / 1 se o1 è successivo a o2)
 - **libera_orario**
 1. Se il puntatore è diverso da NULL, chiama free(o)
 - **get_ora**
 1. Ritorna valore intero corrispondente alle ore (0–23), -1 se o è non valido
 - **get_minuti**
 1. Ritorna valore intero corrispondente ai minuti (0–59), -1 se o è non valido
-

LEZIONE

- **crea_lezione**
 1. Alloca memoria per una nuova struttura lezione
 2. Copia dinamicamente l'ID
 3. Copia nome, data e orario usando funzioni dedicate
 4. Imposta i posti occupati inizialmente a zero
- **salva_lezione_file**
 1. Apre il file in modalità appending
 2. Scrive ID, nome, data, orario, posti max e posti occupati sul file
 3. Chiude il file
- **libera_lezione**
 1. Libera l'ID
 2. Libera data e orario con funzioni specifiche (libera_data e libera_orario)
 3. Libera la struttura stessa
- **visualizza_lezione**
 1. Stampa ID, nome, data, orario, posti occupati e posti disponibili
 2. Controlla se la lezione è al completo
- **visualizza_essenziale_lezione**
 1. Estrae orario e data in formato stringa.
 2. Stampa ID, nome, orario, data, posti occupati/posti max
- **confronta_lezione**
 1. Confronta prima la data delle due lezioni
 2. Se le date sono uguali, confronta l'orario tramite la funzione confronta_orario
- **get_id_lezione**
 1. Se l == NULL, restituisce NULL
 2. Altrimenti, restituisce l'ID della lezione
- **get_posti_occupati**
 1. Se l == NULL, restituisce -1
 2. Altrimenti, restituisce il numero di posti occupati nella lezione

- **get_posti_max**
 1. Se l == NULL, restituisce -1
 2. Altrimenti, restituisce la capacità massima della lezione
 - **get_nome_lezione**
 1. Se l == NULL, restituisce NULL
 2. Altrimenti, restituisce il nome della lezione
 - **get_data_lezione**
 1. Se l == NULL, restituisce NULL
 2. Altrimenti, restituisce la data della lezione
 - **get_ora_lezione**
 1. Se l == NULL, restituisce NULL
 2. Altrimenti, restituisce l'orario della lezione
 - **set_posti_occupati**
 1. Controlla che il valore sia nel range consentito
 2. Aggiorna posti_occupati o stampa errore se fuori range
-

PRENOTAZIONE

- **crea_prenotazione**
 1. Alloca memoria per la struttura e per ciascuna stringa ID
 2. Copia i valori dei parametri nella struttura
 3. Imposta il campo next a NULL
- **libera_prenotazione**
 1. Libera ogni campo stringa e la data.
 2. Infine libera la struttura stessa
- **visualizza_prenotazione**
 1. Controlla se la prenotazione è NULL
 2. Se valida, stampa l'ID prenotazione, cliente, lezione e data della prenotazione
- **visualizza_essenziale_prenotazione**
 1. Verifica se p è NULL
 2. Se valida, stampa i tre ID (prenotazione, cliente, lezione)
- **salva_prenotazione_file**
 1. Apre il file in modalità appending
 2. Scrive i campi della prenotazione nel file
 3. Chiude il file
- **get_id_prenotazione**
 1. Se p == NULL, restituisce NULL
 2. Altrimenti, restituisce l'ID della prenotazione
- **get_id_cliente_prenotazione**
 1. Se p == NULL, restituisce NULL
 2. Altrimenti, restituisce l'ID del cliente associato alla prenotazione
- **get_id_lezione_prenotazione**
 1. Se p == NULL, restituisce NULL
 2. Altrimenti, restituisce l'ID della lezione associata alla prenotazione
- **get_data_prenotazione**
 1. Se p == NULL, restituisce NULL
 2. Altrimenti, restituisce la data in cui è stata effettuata la prenotazione
- **get_next_prenotazione**
 1. Se p == NULL, restituisce NULL

2. Altrimenti, restituisce il puntatore alla prossima prenotazione

- **set_next_prenotazione**

1. Se $p \neq \text{NULL}$, viene assegnato a $p \rightarrow \text{next}$ il puntatore next
-

AREA CLIENTI

- **login_cliente**

1. Legge l'ID cliente tramite input dall'utente
2. Usa hashSearch per trovare il cliente nel hashtable tramite l'id
3. Controlla la validità dell'abbonamento con abbonamento_valido
4. Attende un invio per tornare al menu

- **visualizza_lezione**

1. Chiede se visualizzare tutte o solo le lezioni disponibili (non svolte e con posti liberi)
2. Scorre la lista controllando data e disponibilità
3. Visualizza solo le lezioni che soddisfano il filtro
4. Se non trova nessuna lezione che soddisfa il filtro stampa un avviso

- **prenota_lezione**

1. Mostra la lista delle lezioni libere
2. Legge l'ID lezione da prenotare
3. Verifica l'esistenza, disponibilità e validità di prenotazione
4. Controlla che il cliente non sia già prenotato per quella lezione
5. Prenota la lezione al cliente e incrementa i posti occupati e decrementa i posti disponibili
6. Crea una nuova prenotazione e la inserisce nella hashtable e nel file di testo

- **disdici_prenotazione**

1. Mostra le prenotazioni attive del cliente
2. Legge l'ID prenotazione da disdire
3. Verifica che la prenotazione appartenga al cliente
4. Rimuove la prenotazione dalla hashtable
5. Decrementa i posti occupati della lezione corrispondente e incrementa i posti disponibili
6. Aggiorna il file di testo

- **menu_cliente**

1. Mostra un menu con varie opzioni:
Visualizza abbonamento, rinnova, prenota lezione, disdici prenotazione, visualizza lezioni, visualizza le tue prenotazioni.
 2. Legge l'input dell'utente.
 3. Richiama le funzioni appropriate in base alla scelta.
 4. Continua finché l'utente non sceglie di uscire
-

AREA GESTORE

- **inserisci_cliente**

1. Legge nome, cognome e durata abbonamento da input, con controlli di validità
2. Genera un ID univoco per il cliente con genera_id_generico
3. Crea il cliente e lo salva su file
4. Inserisce il cliente nella hashtable e nel file di testo
5. Stampa l'ID Cliente

- **nome_lezione_valido**

1. Confronta il nome fornito con ogni nome nella lista delle lezioni permesse
2. Restituisce 1 se trova una corrispondenza, altrimenti 0

- **data_valida_per_lezione**

1. Ottiene la data odierna
 2. Estrae mese e anno sia dalla data odierna che dalla data da controllare
 3. Calcola mese e anno del mese successivo
 4. Controlla se la data fornita appartiene al mese corrente o a quello successivo
 5. Libera la memoria della data odierna
- **inserisci_lezione**
 1. Richiede nome lezione, orario, data e numero max di posti con controlli di validità
 2. Verifica che il nome lezione sia consentito
 3. Verifica che la data sia nel mese corrente o successivo
 4. Crea la lezione con un ID generato
 5. Inserisce la lezione in testa alla lista
 6. Salva la lezione nel file
 7. Stampa l'ID della lezione
 - **rimuovi_cliente**
 1. Legge l'ID cliente da input
 2. Cerca e rimuove il cliente dalla hashtable
 3. Scorre la hash table delle prenotazioni rimuovendo quelle collegate a quel cliente
 4. Aggiorna i file di clienti e prenotazioni
 - **rimuovi_lezione**
 1. Legge l'ID lezione da input
 2. Rimuove la lezione dalla lista, liberando la memoria
 3. Scorre la tabella delle prenotazioni eliminando quelle collegate alla lezione
 4. Aggiorna i file di lezioni e prenotazioni
 - **ricerca_cliente**
 1. Legge l'ID cliente da input
 2. Cerca il cliente nella hashtable
 3. Se trovato, mostra i dati del cliente
 - **visualizza_prenotazioni_cliente**
 1. Legge l'ID cliente da input
 2. Verifica se il cliente esiste nella hashtable
 3. Visualizza tutte le prenotazioni di quel cliente
 - **visualizza_prenotazioni_lezione**
 1. Chiede all'utente di inserire l'ID della lezione
 2. Cerca la lezione nella lista
 3. Se la lezione esiste, pulisce lo schermo e stampa le prenotazioni associate
 4. Se la lezione non esiste, mostra un messaggio di errore
 - **menu_gestore**
 1. Mostra un menu con le opzioni disponibili
 2. Legge la scelta dell'utente
 3. Esegue la funzione corrispondente alla scelta
 4. Dopo ogni operazione, attende che l'utente prema INVIO prima di tornare al menu
 5. Continua finché l'utente non sceglie di uscire
-

HASH PRENOTAZIONI

- **fnv1aHash32**
 1. Inizializza il valore hash con l'offset basis
 2. Per ogni carattere nella stringa:
 - Applica un XOR tra l'hash e il carattere corrente
 - Moltiplica il risultato per il FNV prime
 3. Restituisce l'hash risultante

- **hashfun**
 1. Calcola l'hash a 32 bit della chiave usando FNV-1a e lo riduce entro il range [0, size-1]
- **newHashtable_p**
 1. Alloca memoria per la struttura hash_p
 2. Se fallisce ritorna NULL
 3. Imposta la dimensione della tabella hash
 4. Inizializza la tabella con calloc e quindi impostando tutti gli elementi a NULL
- **insertHash_p**
 1. Calcola l'indice della tabella hash usando hashFun
 2. Inserisce la prenotazione in testa alla lista nella posizione hash
 3. Controlla l'ID della prenotazione per verificare che non ci siano duplicati nella lista
 4. Ritorna 1 se l'inserimento ha successo, 0 altrimenti
- **hashDelete_p**
 1. Calcola l'indice della tabella hash usando hashFun
 2. Gestisce l'eliminazione in testa e in mezzo alla lista.
 3. La prenotazione, se trovata, viene rimossa, o NULL se non la trova
- **hashSearch_p**
 1. Calcola l'indice della tabella hash usando hashFun
 2. Scorre la lista delle prenotazioni di quell'indice
 3. Ritorna la prenotazione ricercata, oppure NULL se non la trova
- **deleteList_p**
 1. Dealloca la memoria dedicata ad una lista collegata di prenotazioni
- **destroyHashtable_p**
 1. Elimina tutte le liste collegate chiamando la funzione deleteList_p
 2. Libera l'array e la struttura
- **stampaHash_p**
 1. Scorre l'intera tabella e chiama la funzione visualizza_prenotazione.
 2. Conta e stampa il numero totale di prenotazioni
- **stampaMinimaHash_p**
 1. Scorre l'intera tabella e chiama la funzione visualizza_essenziale_prenotazione
- **get_size_hash_p**
 1. Restituisce la dimensione della tabella hash
- **get_table_hash_p**
 1. Restituisce l'array delle liste di prenotazioni

HASH

- **fnv1aHash32**
 1. Inizializza il valore hash con l'offset basis
 2. Per ogni carattere nella stringa:
 - Applica un XOR tra l'hash e il carattere corrente
 - Moltiplica il risultato per il FNV prime
 3. Restituisce l'hash risultante
- **hashfun**
 1. Calcola l'hash a 32 bit della chiave usando FNV-1a e lo riduce entro il range [0, size-1]
- **newHashtable**
 1. Alloca memoria per la struttura hash
 2. Se fallisce ritorna NULL
 3. Imposta la dimensione della tabella hash

4. Inizializza la tabella con calloc e quindi impostando tutti gli elementi a NULL
- **insertHash**
 1. Calcola l'indice della tabella hash usando hashFun
 2. Inserisce il cliente in testa alla lista nella posizione hash
 3. Controlla l'ID del cliente per verificare che non ci siano duplicati nella lista
 4. Ritorna 1 se l'inserimento ha successo, 0 altrimenti
 - **hashDelete**
 1. Calcola l'indice della tabella hash usando hashFun
 2. Gestisce l'eliminazione in testa e in mezzo alla lista.
 3. Il cliente, se trovato, viene rimosso, o NULL se non lo trova
 - **hashSearch**
 1. Calcola l'indice della tabella hash usando hashFun
 2. Scorre la lista dei clienti di quell'indice
 3. Ritorna il cliente ricercato, oppure NULL se non lo trova
 - **deleteList**
 1. Dealloca la memoria dedicata ad una lista collegata di clienti
 - **destroyHashtable**
 1. Elimina tutte le liste collegate chiamando la funzione deleteList
 2. Libera l'array e la struttura
 - **stampaHash**
 1. Scorre l'intera tabella e chiama la funzione visualizza_cliente.
 2. Conta e stampa il numero totale di clienti
 - **stampaMinimaHash**
 1. Scorre l'intera tabella e chiama la funzione visualizza_essenziale_cliente
 - **get_size_hash**
 1. Restituisce la dimensione della tabella hash
 - **get_table_hash**
 1. Restituisce l'array delle liste di clienti
-

LISTA LEZIONI

- **newList**
 1. Crea una lista vuota (NULL)
- **emptyList**
 1. Controlla se il puntatore alla lista è NULL
 2. Restituisce 1 se la lista è vuota, 0 altrimenti
- **consList**
 1. Alloca dinamicamente un nuovo nodo
 2. Assegna la lezione al nuovo nodo
 3. Collega il nuovo nodo alla lista esistente
 4. Ritorna il nuovo nodo come testa della lista
- **tailList**
 1. Controlla se la lista è NULL, se lo è ritorna NULL.
 2. Altrimenti, ritorna il campo "next" del primo nodo
- **getFirst**
 1. Controlla se la lista è NULL, se lo è ritorna NULL.
 2. Altrimenti, ritorna il campo "lez" del primo nodo
- **sizeList**
 1. Scorre tutta la lista contando i nodi uno ad uno tramite una variabile contatore
 2. Ritorna la lunghezza della lista

- **reverseList**
 1. Scorre la lista originale
 2. Aggiunge ciascun elemento in testa a una nuova lista
 3. Ritorna la nuova lista
 - **removeList**
 1. Copia i nodi precedenti alla posizione in una lista temporanea
 2. Salta il nodo da rimuovere
 3. Ricompone la lista con i nodi rimanenti e la restituisce
 - **copiaLista**
 1. Copia la lista in ordine inverso in una lista temporanea
 2. Inverte la lista temporanea per ottenere l'ordine originale della lista iniziale
 - **ordina_Lista**
 1. Copia la lista originale per non modificarla
 2. Seleziona ripetutamente la lezione "minima" (più imminente)
 3. La rimuove dalla copia e la inserisce in una nuova lista ordinata
 4. Alla fine, inverte la lista ordinata per ottenere l'ordine corretto
 - **outputList**
 1. Ordina la lista
 2. Scorre la lista e stampa ogni lezione tramite la funzione ``visualizza_lezione``
 - **stampaMinimaList**
 1. Ordina la lista
 2. Scorre la lista e stampa ogni lezione tramite la funzione ``visualizza_essenziale_lezione``
-

REPORT

- **aggiorna_statistica**
 1. Cerca tramite ID la lezione già presente
 2. Se trovata, incrementa il conteggio
 3. Se non trovata e lo spazio è insufficiente, raddoppia l'array con `realloc` e inserisce la nuova voce
 - **genera_report_mensile**
 1. Ottiene la data odierna e filtra le prenotazioni effettuate nel mese corrente
 2. Per ciascuna prenotazione del mese corrente, aggiorna un array di statistiche sul numero di prenotazioni per ciascuna lezione
 3. Ordina l'array di statistiche in ordine decrescente di prenotazioni
 4. Tra queste, seleziona fino a 3 lezioni che si svolgono effettivamente nel mese corrente
 5. Crea un file nel formato ``report_<mese>_<anno>.txt`` e scrive al suo interno: il numero totale di prenotazioni del mese corrente e le 3 lezioni più frequentate (che si svolgono nel mese).
 6. Stampa un messaggio di conferma a schermo.
 - **genera_report_test**
 1. Ottiene la data odierna e ne identifica mese e anno correnti
 2. Scorre tutte le prenotazioni nella tabella hash temporanea
 3. Conta tutte le prenotazioni fatte nel mese corrente e aggiorna le statistiche per ogni lezione
 4. Ordina le lezioni in base al numero di prenotazioni in ordine decrescente
 5. Scrive su file le informazioni riassuntive: Numero totale di prenotazioni nel mese e fino a 3 lezioni più frequentate che si svolgono effettivamente nel mese corrente
 6. Se non ci sono lezioni prenotate per il mese corrente, lo segnala nel report
-

UTILS

- **rinnova_abbonamento**
 1. Controlla la validità dei parametri
 2. Chiede all'utente la durata del rinnovo in mesi
 3. Calcola la nuova data di scadenza e aggiorna i dettagli del cliente
 4. Calcola la nuova durata dell'abbonamento
 5. Aggiorna il file con i dati modificati
- **carica_contatore_generico**
 1. Apre il file e cerca righe che iniziano con il prefisso selezionato
 2. Estrae la parte numerica e aggiorna il contatore massimo
 3. Ritorna il valore massimo trovato
- **genera_id_generico**
 1. Identifica quale contatore usare in base al prefisso
 2. Se non inizializzato, lo carica da file
 3. Incrementa il contatore e genera l'ID nel formato 'Prefisso + numero'
- **carica_clienti_da_file**
 1. Apre il file 'clienti.txt' e legge i dati dei clienti
 2. Crea le strutture cliente e le inserisce nella tabella hash
- **carica_prenotazioni_da_file**
 1. Apre il file 'prenotazioni.txt' e legge le informazioni delle prenotazioni
 2. Crea le strutture prenotazione e le inserisce nella hashtable
- **aggiorna_file_clienti**
 1. Apre il file 'clienti.txt' in modalità scrittura ("w")
 2. Itera su ogni cliente della tabella hash e scrive i corrispondenti dati sul file
- **aggiorna_file_lezioni**
 1. Apre il file 'lezioni.txt' in modalità scrittura ("w")
 2. Itera sulla lista e scrive le informazioni di ogni lezione sul file
- **aggiorna_file_prenotazioni**
 1. Apre il file 'prenotazioni.txt' in modalità scrittura ("w")
 2. Itera su ogni prenotazione della tabella hash e scrive le corrispondenti informazioni di ogni lezione sul file
- **pulisci_schermo**
 1. Utilizza 'cls' su Windows o 'clear' su Linux e macOS tramite 'system (...)' al fine di pulire lo schermo
- **carica_lezioni_da_file**
 1. Apre il file 'lezione.txt' in modalità lettura ("r") e legge le informazioni delle lezioni
 2. Converte le stringhe lette in strutture dati e aggiorna la lista
- **calcola_durata_in_mesi**
 1. Calcola la differenza in anni e mesi tra la data di inizio e di fine dell'abbonamento
 2. Se la data finale è prima nel giorno rispetto a quella iniziale, sottrae un mese
- **stampa_prenotazioni_cliente**
 1. Scorre la tabella delle prenotazioni cercando quelle con l'ID del cliente indicato
 2. Per ogni prenotazione trovata, cerca nella lista la lezione corrispondente
 3. Stampa ID, nome della lezione, data e orario corrispettivi
- **stampa_prenotazioni_lezione**
 1. Ottiene l'ID, nome, data e orario della lezione indicata
 2. Scorre tutta la tabella delle prenotazioni
 3. Per ogni prenotazione che corrisponde all'ID della lezione, recupera il cliente
 4. Stampa ID prenotazione, nome e cognome cliente
 5. Se non ci sono prenotazioni per la lezione, stampa un messaggio di avviso

- **cerca_lezione_per_id**
 1. Scorre la lista iterativamente
 2. Confronta l'ID della lezione corrente con quello cercato
 3. Restituisce la lezione appena viene trovata
 - **stampa_lezioni_libere**
 1. Ordina la lista delle lezioni
 2. Per ogni lezione verifica se ci sono posti disponibili e se la data è valida
 3. Stampa i dettagli essenziali della lezione
 - **calcolo_scadenza_abbonamento**
 1. Estrae giorno, mese e anno dalla data di inizio
 2. Somma i mesi della durata, gestendo anche il caso di avanzamento dell'anno
 3. Crea e restituisce la data di scadenza
 - **abbonamento_valido**
 1. Confronta la data di scadenza con quella corrente
 2. Restituisce 1 se la data corrente è precedente o uguale a quella di scadenza, 0 altrimenti
 - **leggi_intero**
 1. Legge una riga da stdin
 2. Converte la riga in intero
 3. Se fallisce, chiede nuovamente l'input
 - **solo lettere**
 1. Scorre la stringa carattere per carattere
 2. Controlla che ogni carattere sia una lettera o uno spazio
 3. Se trova numeri o caratteri non ammessi, ritorna 0
 4. Se almeno una lettera è presente e nessun carattere non ammesso, ritorna 1.
 - **nome_mese**
 1. Converte il mese da formato numerico al nome del mese tramite un array statico di stringhe dove sono presenti i nomi di tutti i mesi
 2. Ritorna il nome corrispondente all'indice
-

MAINTESTING

- **confronta_file**
 1. Apre entrambi i file in lettura
 2. Confronta ogni riga
 3. Se trova una differenza, ritorna 0
 4. Se tutti i confronti sono uguali, ritorna 1
- **salva_prenotazione_test**
 1. Apre un file in scrittura
 2. Scrive l'ID della prenotazione, cliente, lezione, data e posti occupati sul file
 3. Chiude il file
- **prenota_lezione_test_1**
 1. Verifica la validità dell'abbonamento e dell'ID lezione
 2. Controlla se ci sono posti disponibili
 3. Evita duplicati di prenotazione
 4. Genera un ID per la prenotazione
 5. Inserisce temporaneamente la prenotazione nella hashtable
 6. Salva i dettagli su file e rimuove la prenotazione dall'hashtable
- **esegui_test_prenotazione**
 1. Legge le righe di input e tenta la prenotazione per ogni coppia cliente-lezione
 2. Scrive l'esito sul file output

3. Confronta l'output con il file oracle
- **salva_cliente_test**
 1. Apre il file
 2. Scrive durata abbonamento e data di scadenza.
 3. Chiude il file
 - **rinnova_abbonamento_test**
 1. Calcola nuova scadenza
 2. Aggiorna cliente
 3. Salva i dati aggiornati su file
 - **esegui_test_abbonamenti**
 1. Legge i parametri (cliente e durata)
 2. Esegue test di rinnovo o verifica validità
 3. Scrive esito su file
 4. Confronta con file oracle
 - **prenotazioni_per_report**
 1. Controlla validità abbonamento, disponibilità e duplicati
 2. Inserisce la prenotazione in struttura temporanea
 - **esegui_test_report**
 1. Legge prenotazioni da file input
 2. Inserisce in hashtable temporanea
 3. Genera report
 4. Confronta con file oracle
 - **run_test_case**
 1. Costruisce i nomi dei file input/output/oracle
 2. Chiama la funzione di test in base al tipo
-

INTERAZIONE TRA I MODULI

Il progetto è organizzato in più moduli, ognuno con responsabilità chiare e specifiche, che lavorano insieme per realizzare un sistema completo di gestione delle prenotazioni, delle iscrizioni e delle lezioni. I moduli principali del sistema sono:

- Data
- Orario
- Cliente
- Lezione
- Prenotazione

MODULO DATA:

Il modulo **data** è il punto di riferimento per tutto ciò che riguarda la gestione delle date nel programma. Gli altri moduli si appoggiano a questo per lavorare con le informazioni temporali in modo preciso e uniforme.

Ad esempio, il modulo **cliente** usa data per tenere traccia della validità degli abbonamenti, controllando se sono ancora attivi in base alla data odierna. Il modulo **prenotazioni** invece fa affidamento su data per verificare che le prenotazioni siano fatte per lezioni con date valide e future, e per gestire correttamente le scadenze.

Anche il modulo **lezioni** utilizza il modulo data per programmare le lezioni su date corrette e per mostrarne la disponibilità. Infine, il modulo **orario** si serve di data per assicurarsi che gli orari siano associati a giorni e mesi reali e coerenti.

In sostanza, il modulo data fornisce a tutto il sistema un modo sicuro e coerente per lavorare con le date, permettendo agli altri moduli di concentrarsi sulle loro funzionalità specifiche senza preoccuparsi dei dettagli di gestione temporale. Questo rende il progetto più chiaro, modulare e facile da mantenere.

MODULO ORARIO:

Il modulo **orario** si occupa di gestire l'orario preciso di inizio delle lezioni all'interno del sistema. Gli altri moduli si appoggiano a questo per trattare in modo uniforme e sicuro le informazioni sugli orari.

Per esempio, il modulo **lezioni** usa orario per definire gli orari delle singole lezioni, assicurandosi che gli orari siano validi e coerenti. In sostanza, il modulo fornisce un supporto fondamentale per tutto ciò che riguarda il tempo nel sistema, consentendo agli altri moduli di concentrarsi sulle funzionalità specifiche senza preoccuparsi dei dettagli di gestione degli orari.

MODULO CLIENTE:

Il modulo cliente è il punto di riferimento per tutto ciò che riguarda la gestione degli utenti registrati nel sistema. Rappresenta ogni cliente con una struttura dati completa che include informazioni anagrafiche e dati di abbonamento.

Tutti gli altri moduli che hanno bisogno di interagire con i clienti si appoggiano a questo modulo per accedere, visualizzare o aggiornare i dati relativi a un utente in modo sicuro e centralizzato.

Il modulo prenotazioni si appoggia a cliente per verificare che solo clienti con abbonamento attivo possano prenotare lezioni.

In sintesi, cliente fornisce agli altri moduli un'interfaccia coerente e ben definita per lavorare con gli utenti. Questo consente di affidarsi a cliente per tutto ciò che riguarda l'identificazione, la registrazione e il tracciamento dell'utente. Il risultato è un progetto più solido, modulare e facilmente espandibile.

MODULO LEZIONE:

Il modulo lezione è il cuore della gestione delle attività nel sistema. Ogni lezione viene rappresentata come una struttura autonoma, completa di identificativo, nome della lezione, data, orario e disponibilità di posti. Il suo scopo è quello di offrire un modello coerente e facilmente gestibile per tutto ciò che riguarda le singole sessioni di lezione offerte ai clienti.

Gli altri moduli del progetto si appoggiano a questo per pianificare, prenotare e visualizzare le lezioni. Ad esempio, il modulo prenotazioni interagisce strettamente con lezione per verificare la disponibilità di posti prima di confermare una prenotazione.

Il modulo cliente può usare lezione per mostrare al cliente un riepilogo delle lezioni prenotate, utile per fornire una panoramica sintetica ma completa, ma anche per visualizzare tutte le lezioni disponibili. Inoltre, ogni volta che un cliente si iscrive o cancella da una lezione, viene aggiornato il numero di partecipanti.

Il modulo data, invece, gioca un ruolo fondamentale nel garantire la coerenza temporale: ogni lezione include una data che viene gestita proprio grazie al modulo data. Allo stesso modo, il modulo orario permette di associare con precisione l'orario della lezione.

In sintesi, il modulo lezione fornisce agli altri componenti del progetto un modello strutturato e facilmente accessibile per lavorare con le lezioni.

MODULO PRENOTAZIONE:

Il **modulo prenotazione** rappresenta il fulcro della gestione delle iscrizioni alle lezioni da parte dei clienti. Ogni prenotazione è modellata come una struttura autonoma, che associa in modo univoco un cliente a una lezione. Il suo scopo principale è quello di **registrare le richieste di partecipazione** da parte degli utenti alle lezioni.

Questo modulo lavora a stretto contatto con altri componenti del sistema, fungendo da ponte tra clienti, lezioni e gestione temporale. In particolare, interagisce in modo diretto con il modulo **lezione** per **verificare la disponibilità dei posti** prima di consentire l'inserimento di una nuova prenotazione. Solo se la lezione ha posti liberi, il sistema può procedere con la registrazione.

Il modulo **cliente** si appoggia a prenotazione per visualizzare l'elenco delle lezioni prenotate da un determinato utente. Questo consente al sistema di offrire una panoramica personalizzata dell'attività svolta, utile sia per motivi informativi sia per gestire eventuali cancellazioni o modifiche.

IV. SPECIFICA SINTATTICA E SEMANTICA

CLIENTE

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o : Cliente

Tip i u s a t i: char, int, Data, FILE*

Op e r a t o r i:

- **crea_cliente (char, char, char, int, Data) → Cliente**
Crea la scheda del cliente
 - **distruggi_cliente (Cliente) → void**
Rimuove il cliente
 - **salva_cliente_file (Cliente) → void**
Salva il cliente passato in input in un file di testo
 - **visualizza_cliente (Cliente) → void**
Stampa a schermo i dati del cliente passato in input
 - **visualizza_abbonamento_cliente (Cliente) → void**
Stampa a schermo i dettagli dell'abbonamento del cliente passato in input
 - **visualizza_essenziale_cliente (Cliente) → void**
Stampa a schermo le generalità del Cliente e il suo id
 - **confronta_clienti (Cliente, Cliente) → int**
Confronta i due clienti passati in input
 - **get_id_cliente (Cliente) → char***
Ritorna l'ID del Cliente passato in input
 - **get_nome_cliente (Cliente) → char***
Ritorna il nome del Cliente passato in input
 - **get_cognome_cliente (Cliente) → char***
Ritorna il cognome del Cliente passato in input
 - **get_durata_abbonamento (Cliente) → int**
Ritorna la durata dell'abbonamento del Cliente passato in input
 - **get_data_iscrizione (Cliente) → Data**
Ritorna la data di iscrizione dell'abbonamento del Cliente passato in input
 - **get_data_scadenza (Cliente) → Data**
Ritorna la data di scadenza dell'abbonamento del Cliente passato in input
 - **get_next_cliente (Cliente) → Cliente**
Ritorna il puntatore al Cliente successivo
 - **set_next_cliente (Cliente, Cliente) → void**
Imposta il puntatore al Cliente successivo
 - **set_data_scadenza (Cliente, Data) → void**
Aggiorna la data di scadenza dell'abbonamento del Cliente passato in input
 - **set_durata (Cliente, int) → void**
Aggiorna la durata dell'abbonamento del Cliente passato in input
-

SPECIFICA SEMANTICA

T i p o d i r i f e r i m e n t o :

Cliente

Cliente è una struct contenente le informazioni principali di un abbonato in una Palestra, come il nome, il cognome, durata dell'abbonamento, la data di iscrizione dell'abbonamento, la data di scadenza dell'abbonamento e infine un identificativo univoco per il Cliente.

O p e r a t o r i:

- **crea_cliente (id, nome, cognome, durata, data_is) → C**
Side-effects: Alloca memoria dinamica
- **distruggi_cliente (c) → void**
Precondizione: c ≠ NULL
Side-effects: Libera memoria dinamica
- **salva_cliente_file (c) → void**
Precondizione: c ≠ NULL
Side-effects: Scrittura su file "clienti.txt"
- **visualizza_cliente (c) → void**
Precondizione: c ≠ NULL
Side-effects: Stampa a schermo le informazioni del cliente
- **visualizza_abbonamento_cliente (c) → void**
Precondizione: c ≠ NULL
Side-effects: Stampa a schermo le informazioni dell'abbonamento
- **visualizza_essenziale_cliente (c) → void**
Side-effects: Stampa a schermo le generalità del cliente
- **confronta_clienti (c1, c2) → int**
Precondizione: c1 ≠ NULL and c2 ≠ NULL
Postcondizione: Ritorna 1 se gli ID coincidono, 0 altrimenti.
- **get_id_cliente (c) → c->id**
Precondizione: c ≠ NULL
Postcondizione: Se c == NULL, return NULL
- **get_nome_cliente (c) → c->nome**
Precondizione: c ≠ NULL
Postcondizione: Se c == NULL, return NULL
- **get_cognome_cliente (c) → c->cognome**
Precondizione: c ≠ NULL
Postcondizione: Se c == NULL, return NULL
- **get_durata_abbonamento (c) → c->durata_abbonamento**
Precondizione: c ≠ NULL
Postcondizione: Se c == NULL, return -1
- **get_data_iscrizione (c) → c->data_iscrizione**
Precondizione: c ≠ NULL
Postcondizione: Se c == NULL, return NULL
- **get_data_scadenza (c) → c->data_scadenza**
Precondizione: c ≠ NULL

- Postcondizione: Se $c == \text{NULL}$, return NULL
 - **get_next_cliente (c) → c->next**
Precondizione: $c \neq \text{NULL}$
Postcondizione: Se $c == \text{NULL}$, return NULL
 - **set_next_cliente (c, next) → void**
Precondizione: $c \neq \text{NULL}$
Side-effects: Modifica il campo next della struttura Cliente
 - **set_data_scadenza (c, nuova_data) → void**
Precondizione: $c \neq \text{NULL}$ and $\text{nuova_data} \neq \text{NULL}$
Side-effects: Aggiorna la data di scadenza dell'abbonamento
 - **set_durata (c, nuova_durata) → void**
Precondizione: $c \neq \text{NULL}$
Side-effects: Aggiorna la durata dell'abbonamento
-

DATA

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o : Data

Tip i u s a t i: int, time_t,

Op e r a t o r i:

- **crea_data (int, int, int) → Data**
Crea una nuova data con i campi passati in input
- **visualizza_data (Data) → void**
Stampa la data
- **confronta_date (Data, Data) → int**
Confronta due date passate in input
- **leggi_data () → Data**
Chiede all'utente una data nel formato GG/MM/AAAA e se viene inserita correttamente crea la data e la restituisce in output
- **copia_data (Data) → Data**
Copia la data passata in input
- **libera_data (Data) → void**
Rimuove una data
- **data_oggi () → Data**
Restituisce la data corrente
- **get_giorno (Data) → int**
Restituisce il giorno della data passata in input
- **get_mese (Data) → int**
Restituisce il mese della data passata in input
- **get_anno (Data) → int**
Restituisce l'anno della data passata in input

SPECIFICA SEMANTICA

T i p o d i r i f e r i m e n t o : Data

Data è una struttura contenente i campi giorno, mese e anno

O p e r a t o r i:

- **crea_data (gg, mm, aa) → nuova_data**
Precondizione: $0 < gg \leq 31$ and $0 < mm \leq 12$
Side-effects: Alloca memoria dinamica e termina il programma con exit in caso di fallimento.
 - **visualizza_data (data) → void**
Precondizione: data \neq NULL
Postcondizione: Stampa la data in formato GG/MM/AAAA
Side-effects: Stampa su schermo la data passata in input
 - **confronta_date (d1, d2) → int**
Precondizione: d1 \neq NULL and d2 \neq NULL
Postcondizione: Restituisce -1 se $d1 < d2$
Restituisce 0 se $d1 == d2$
Restituisce 1 se $d1 > d2$
 - **leggi_data () → inserita**
Postcondizione: Ritorna una data valida e successiva alla data corrente
Side-effects: Alloca nuova memoria, possibili ripetizioni
 - **copia_data (originale) → nuova_data**
Precondizione: origine \neq NULL
Postcondizione: Restituisce una copia indipendente della data originale
Side-effects: Alloca nuova memoria
 - **libera_data (d) → void**
Precondizione: d \neq NULL
Postcondizione: La memoria associata a d viene deallocata
Side-effects: Libera memoria
 - **data_oggi () → crea_data (giorno, mese, anno)**
Postcondizione: Ritorna una data che rappresenta il giorno corrente del sistema
Side-effects: Chiama funzioni di sistema (time, localtime), alloca memoria
 - **get_giorno (d) → d->giorno;**
Precondizione: d \neq NULL
Postcondizione: Ritorna il giorno della data passata in input, -1 altrimenti
 - **get_mese (d) → d->mese**
Precondizione: d \neq NULL
Postcondizione: Ritorna il mese della data passata in input, -1 altrimenti
 - **get_anno (d) → d->anno**
Precondizione: d \neq NULL
Postcondizione: Ritorna l'anno della data passata in input, -1 altrimenti
-

ORARIO

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o: Orario

Tip i u s a t i: int

Op e r a t o r i:

- **crea_orario (int, int) → Orario**
Crea un nuovo orario con i campi passati in input
- **copia_orario (Orario) → Orario**
Crea una copia dell'orario passato in input
- **stampa_orario (Orario) → void**
Stampa l'orario passato in input
- **confronta_orario (Orario, Orario) → int**
Confronta i due orari passati in input e ritorna un valore in base al confronto
- **libera_orario (Orario) → void**
Rimuove l'orario passato in input
- **get_ora (Orario) → int**
Ritorna l'ora dall'orario passato in input
- **get_minuti (Orario) → int**
Ritorna i minuti dall'orario passato in input

SPECIFICA SEMANTICA

Tip o d i r i f e r i m e n t o: Orario

Orario è una struttura contenente i campi minuti e ore

Op e r a t o r i:

- **crea_orario (ore, min) → o**
Precondizione: $0 < \text{ore} \leq 23$ and $0 < \text{min} \leq 59$
Side-effects: Alloca nuova memoria
- **copia_orario (originale) → copia**
Precondizione: originale \neq NULL
Postcondizione: Ritorna una copia indipendente dell'orario originale
Side-effects: Alloca nuova memoria dinamica. Se originale == NULL ritorna NULL.
- **stampa_orario (o) → void**
Precondizione: o \neq NULL
Postcondizione: Se o \neq NULL, stampa l'orario in formato HH:MM
Se o == NULL, stampa "Orario non valido"
Side-effects: Stampa a schermo l'orario
- **confronta_orario (o1, o2) → int** Precondizione: o1 \neq NULL and o2 \neq NULL Postcondizione: Restituisce
-1 se o1 è prima di o2
Restituisce 0 se gli orari coincidono
Restituisce 1 se o1 è dopo o2
Restituisce 0 e stampa un errore se uno dei due è null

Side-effects: Stampa a schermo un avviso di errore se uno dei parametri è NULL

- **libera_orario (o) → void**

Precondizione: o ≠ NULL

Postcondizione: La memoria associata a o viene deallocata

Side-effects: Libera memoria

- **get_ora (o) → o->ore**

Precondizione: o ≠ NULL

Postcondizione: Restituisce il campo ore dall'orario passato in input, -1 altrimenti

- **get_minuti (o) → o->minuti**

Precondizione: o ≠ NULL

Postcondizione: Restituisce il campo minuti dall'orario passato in input, -1 altrimenti

LEZIONE

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o : Lezione

Tip i u s a t i : const char, Data, char, Cliente, int, Orario, FILE*

O p e r a t o r i:

- **crea_lezione (const char*, const char*, Data, Orario, int) → Lezione**

Crea una nuova lezione con i dati passati in input

- **salva_lezione_file (Lezione) → void**

Salva i campi della lezione passata in input all'interno di un file di testo "lezione.txt"

- **libera_lezione (Lezione) → void**

Rimuove la lezione passata in input e dealloca la memoria

- **visualizza_lezione (Lezione) → void**

Stampa a schermo tutti i dettagli della lezione passata in input

- **visualizza_essenziale_lezione (Lezione) → void**

Stampa i dati principali della lezione passata in input

- **confronta_lezione (Lezione, Lezione) → int**

Confronta la data e l'orario delle due lezioni passate in input

- **get_id_lezione (Lezione) → char***

Ritorna l'id della lezione passata in input

- **get_posti_occupati (Lezione) → int**

Ritorna i posti occupati della lezione passata in input

- **get_posti_max (Lezione) → int**

Ritorna il massimo numero di posti occupabili della lezione passata in input

- **get_nome_lezione (Lezione) → char ***

Ritorna il nome della lezione passata in input

- **get_data_lezione (Lezione) → Data**

Ritorna la data della lezione passata in input

- **get_ora_lezione (Lezione) → Orario**
Ritorna l'orario della lezione passata in input
- **set_posti_occupati (Lezione, int) → void**
Aggiorna il numero di posti prenotati della lezione selezionata con il valore intero passato in input.

SPECIFICA SEMANTICA

T i p o d i r i f e r i m e n t o :

Lezione

Lezione è una struttura che presenta tutti i dettagli di un corso di una palestra come il nome del corso, la data e l'orario in cui si svolgerà il corso, il numero massimo di posti prenotabili e il numero di posti già prenotati. Ogni singola lezione è identificata univocamente da un id lezione nel formato "L000".

O p e r a t o r i :

- **crea_lezione (id, nome, dat, orario, posti_max) → nuova_lezione**
Precondizione: id ≠ NULL and nome ≠ NULL and posti_max > 0
Postcondizione: Ritorna un oggetto di tipo lezione
Side-effects: Alloca memoria e in caso di errore, termina il programma con exit(1).
- **salva_lezione_file (l) → void**
Precondizione: l ≠ NULL and fp ≠ NULL
Postcondizione: Scrive su file i dati della lezione in formato testuale
Side-effects: Scrittura sul file "lezione.txt"
- **libera_lezione (l) → void**
Precondizione: l ≠ NULL
Postcondizione: Libera la memoria dedicata alla lezione passata in input
Side-effects: Dealloca memoria
- **visualizza_lezione (l) → void**
Precondizione: l ≠ NULL
Side-effects: Stampa a schermo i dati della lezione
- **visualizza_essenziale_lezione (l) → void**
Precondizione: l ≠ NULL
Side-effects: Stampa a schermo i dati della lezione
- **confronta_lezione (l1, l2) → int** Precondizione: l1 ≠ NULL and l2 ≠ NULL Postcondizione:
Restituisce -1 se l1 è prima di l2
Restituisce 0 se l1 e l2 hanno stessa data e orario
Restituisce 1 se l1 è dopo l2
- **get_id_lezione (l) → char***
Precondizione: l ≠ NULL
Postcondizione: Ritorna la stringa id, NULL altrimenti
- **get_posti_occupati (l) → int**
Precondizione: l ≠ NULL
Postcondizione: Ritorna il numero di posti occupati nella lezione, -1 altrimenti

- **get_posti_max (I) → int**
Precondizione: I ≠ NULL
Postcondizione: Ritorna il numero massimo di posti prenotabili nella lezione, -1 altrimenti
 - **get_nome_lezione (I) → char ***
Precondizione: I ≠ NULL
Postcondizione: Ritorna il nome della lezione, NULL altrimenti
 - **get_data_lezione (I) → Data**
Precondizione: I ≠ NULL
Postcondizione: Ritorna la data della lezione, NULL altrimenti
 - **get_ora_lezione (I) → Orario**
Precondizione: I ≠ NULL
Postcondizione: Ritorna l'orario della lezione, NULL altrimenti
 - **set_posti_occupati (I, pos_occupati) → void**
Precondizione: I ≠ NULL and $0 \leq \text{nuovi_posti} \leq \text{posti_max}$
Postcondizione: Viene aggiornato il campo posti occupati
Side-effects: Se il valore è fuori dal range, il programma termina con exit (1).
-

PRENOTAZIONE

SPECIFICA SINTATTICA

T i p o d i r i f e r i m e n t o : Prenotazione

T i p i u s a t i: char*, Data, FILE*

O p e r a t o r i:

- **crea_prenotazione (char*, char*, char*, Data) → Prenotazione**
Crea la prenotazione con i dati passati in input
- **libera_prenotazione (Prenotazione) → void**
Rimuove la prenotazione selezionata e dealloca la memoria dedicata ad essa
- **visualizza_prenotazione (Prenotazione) → void**
Stampa a schermo tutti i dettagli della prenotazione
- **visualizza_essenziale_prenotazione (Prenotazione) → void**
Stampa a schermo i dettagli essenziali della prenotazione
- **salva_prenotazione_file (Prenotazione) → void**
Salva la prenotazione passata in input nel file di testo "prenotazioni.txt"
- **get_id_prenotazione (Prenotazione) → char***
Ritorna l'id della prenotazione
- **get_id_cliente_prenotazione (Prenotazione) → char***
Ritorna l'id del cliente che ha eseguito la prenotazione
- **get_id_lezione_prenotazione (Prenotazione) → char***
Ritorna l'id della lezione prenotata
- **get_data_prenotazione (Prenotazione) → Data**
Ritorna la data di quando è stata eseguita la prenotazione
- **get_next_prenotazione (Prenotazione) → Prenotazione**
Ritorna il puntatore alla prenotazione successiva

- **set_next_prenotazione (Prenotazione, Prenotazione) → void**

Aggiorna il puntatore alla prenotazione successiva

SPECIFICA SEMANTICA

T i p o d i r i f e r i m e n t o :

Prenotazione è una struttura che presenta la data di quando verrà effettuata la prenotazione e gli identificatori univoci del cliente, lezione e della prenotazione.

O p e r a t o r i:

- **crea_prenotazione (id_pren, id_cli, id_lez, data_prenotazione) → nuova_prenotazione**
Precondizione: id_pren, id_cli, id_lez ≠ NULL
Postcondizione: Ritorna un puntatore alla nuova prenotazione
Side-effects: Alloca memoria e in caso di errore, termina il programma con exit (1).
- **libera_prenotazione (p) → void**
Precondizione: p ≠ NULL
Postcondizione: Libera la memoria dedicata alla prenotazione passata in input
Side-effects: Dealloca memoria
- **visualizza_prenotazione (p) → void**
Precondizione: p ≠ NULL
Side-effects: Stampa a schermo i dettagli della prenotazione
- **visualizza_essenziale_prenotazione (p) → void**
Precondizione: p ≠ NULL
Side-effects: Stampa a schermo i dettagli essenziali della prenotazione
- **salva_prenotazione_file (p) → void**
Precondizione: p ≠ NULL
Postcondizione: Scrive su file i dati della prenotazione in formato testuale
Side-effects: Scrittura sul file "prenotazioni.txt"
- **get_id_prenotazione (p) → p->id_prenotazione**
Precondizione: p ≠ NULL
Postcondizione: Restituisce il valore del campo id_prenotazione se p è valido, altrimenti NULL
- **get_id_cliente_prenotazione (p) → p->id_cliente**
Precondizione: p ≠ NULL
Postcondizione: Restituisce il valore del campo id_cliente se p è valido, altrimenti NULL
- **get_id_lezione_prenotazione (p) → p->id_lezione**
Precondizione: p ≠ NULL
Postcondizione: Restituisce il valore del campo id_lezione se p è valido, altrimenti NULL
- **get_data_prenotazione (p) → p->data_prenotazione**
Precondizione: p ≠ NULL

Postcondizione: Restituisce il campo data_prenotazione se p è valido, altrimenti NULL

- **get_next_prenotazione (p) → p->next**

Precondizione: p ≠ NULL

Postcondizione: Restituisce il puntatore alla prenotazione successiva alla lista, altrimenti NULL

- **set_next_prenotazione (p, next) → void**

Precondizione: p ≠ NULL

Postcondizione: Imposta il campo next della prenotazione p al valore next

Side-effects: Modifica lo stato interno della struttura Prenotazione

AREA CLIENTI

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o :

Tip i u s a t i : hashtable, Cliente, list, hashtable_p, int, char*, Lezione, Data, Prenotazione

Op e r a t o r i :

- **login_cliente (hashtable) → Cliente**
Effettua il login richiedendo l'ID cliente, cercandolo nella hash table e verificando l'abbonamento
- **visualizza_lezione (list) → static void**
Visualizza tutte le lezioni oppure solo quelle disponibili
- **prenota_lezione (Cliente, list, hashtable_p) → static void**
Consente a un cliente di prenotare una lezione tra quelle disponibili
- **disdici_prenotazione (Cliente, hashtable, list) → static void**
Permette ad un cliente di annullare una propria prenotazione
- **menu_cliente (Cliente, hashtable, list, hashtable_p) → void**
Gestisce il menu principale per il lato del cliente

SPECIFICA SEMANTICA

Tip o d i r i f e r i m e n t o :

Op e r a t o r i :

- **login_cliente (h) → c**
Precondizione: h ≠ NULL and c ≠ NULL
Postcondizione: Se il login del cliente è valido, restituisce il cliente, altrimenti NULL
Side-effects: Stampa a schermo, scrittura su file di testo e lettura input dall'utente
- **visualizza_lezione (l) → static void**
Precondizione: l ≠ NULL
Side-effects: Stampa a schermo, lettura input dall'utente e allocazione e deallocazione di memoria.
- **prenota_lezione (c, l, h) → void**
Precondizione: h ≠ NULL and l ≠ NULL and c ≠ NULL

Postcondizione: Se la prenotazione è riuscita, verrà aggiornata la lezione selezionata e la prenotazione verrà aggiunta alla hash table

Side-effects: Stampa a schermo, scrittura su file "lezione.txt" e "prenotazioni.txt", aggiornamento della hashtable, lettura input dall'utente e allocazione dinamica.

- **disdici_prenotazione (c, hp, l) → static void**

Precondizione: hp ≠ NULL and l ≠ NULL and c ≠ NULL

Postcondizione: Prenotazione rimossa dalla hashtable, decrementati i posti occupati nelle lezioni e incrementati i posti disponibili

Side-effects: Stampa a schermo, scrittura su file "lezione.txt" e "prenotazioni.txt", aggiornamento della hashtable e della lista lezioni, lettura input dall'utente e deallocazione.

- **menu_cliente (c, h, l, hp) → void**

Precondizione: hp ≠ NULL and l ≠ NULL and c ≠ NULL and h ≠ NULL

Postcondizione: Le varie strutture dati presenti possono essere aggiornate in base all'iterazione dell'utente

Side-effects: Stampa a schermo, lettura input dall'utente e chiamate ricorsive.

AREA GESTORE

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o :

Tip i u s a t i : hashtable, Cliente, list, hashtable_p, int, char, Data, const char*, Orario, Prenotazione, Lezione

Op e r a t o r i :

- **inserisci_cliente (hashtable) → static void**

Permette di inserire un nuovo cliente, che successivamente verrà salvato su file e inserito nella tabella hash.

- **nome_lezione_valido (const char*) → static int**

Verifica se il nome della lezione inserito in input è tra quelli predefiniti e ammessi.

- **data_valida_per_lezione (Data) → static int**

Controlla se la data inserita per la lezione è nel mese corrente o successivo

- **inserisci_lezione (list) → static list**

Aggiunge una nuova lezione alla lista, controllando il nome (ammesso), l'orario, la data e i posti disponibili. La lezione viene salvata su file

- **rimuovi_cliente (hashtable, hashtable_p) → static void**

Rimuove un cliente dalla tabella hash e cancella tutte le sue prenotazioni dalla tabella delle prenotazioni

- **rimuovi_lezione (list, hashtable_p) → static list**

Rimuove una lezione dalla lista e cancella tutte le prenotazioni associate ad essa dalla tabella delle prenotazioni

- **ricerca_cliente (hashtable) → static void**

Cerca un cliente per ID nella tabella hash e ne stampa i dettagli se trovato

- **visualizza_prenotazioni_cliente (hashtable, hashtable_p, list) → static void**
Mostra tutte le prenotazioni associate a un cliente, dopo averlo cercato tramite ID
- **visualizza_prenotazioni_lezione (hashtable, hashtable_p, list) → static void**
Mostra tutte le prenotazioni associate a una lezione, cercata per ID
- **menu_gestore (hashtable, list, hashtable_p) → void**
Gestisce il menù testuale per il gestore della palestra, offrendo tutte le funzionalità

SPECIFICA SEMANTICA

T i p o d i r i f e r i m e n t o :

O p e r a t o r i:

- **inserisci_cliente (h) → static void**
Precondizione: h ≠ NULL
Postcondizione: Se i dati inseriti sono validi, un nuovo cliente viene creato, salvato su file clienti.txt e inserito nella tabella hash h.
 Se il cliente è già presente, viene stampato un messaggio di errore.
Side-effects: Stampa a schermo, scrittura su file, lettura input dall'utente e allocazione e deallocazione dinamica di memoria.
- **nome_lezione_valido (nome) → static int**
Precondizione: nome ≠ NULL
Postcondizione: Restituisce 1 se il nome inserito corrisponde ad una delle lezioni ammesse
 Restituisce 0 altrimenti
- **data_valida_per_lezione (data) → valida**
Precondizione: data ≠ NULL
Postcondizione: Restituisce 1 se la data fornita è nel mese corrente oppure nel mese successivo, rispetto alla data corrente
 Restituisce 0 altrimenti
Side-effects: allocazione e deallocazione dinamica di memoria.
- **inserisci_lezione (l) → l**
Precondizione: l ≠ NULL
Postcondizione: Se i dati inseriti sono validi, una nuova lezione viene creata, salvata su file lezione.txt e inserita in testa alla lista. Viene restituita la lista aggiornata con la nuova lezione in testa
Side-effects: Stampa a schermo, scrittura su file, allocazione dinamica di memoria.
- **rimuovi_cliente (h, hp) → static void**
Precondizione: h ≠ NULL and hp ≠ NULL
Postcondizione: Il cliente identificato dall'ID viene rimosso da h, se presente. Se trovato tutte le prenotazioni associate a quest'ultimo verranno eliminate da hp e rimosse dal file prenotazioni.txt

Side-effects: Stampa a schermo, modifica e scrittura su file, deallocazione dinamica di memoria.

- **rimuovi_lezione (l, hp) → nuova**

Precondizione: l ≠ NULL and hp ≠ NULL

Postcondizione: La lezione identificata dall'ID viene rimossa dalla lista, se presente. Se trovata tutte le prenotazioni associate a quest'ultima verranno eliminate da hp e verrà restituita la lista aggiornata.

Side-effects: Stampa a schermo, modifica e scrittura su file, deallocazione dinamica di memoria.

- **ricerca_cliente (h) → static void**

Precondizione: h ≠ NULL

Postcondizione: Viene stampato a schermo il cliente corrispondente all'ID immesso, se esiste.

Side-effects: Stampa a schermo

- **visualizza_prenotazioni_cliente (h, hp, l) → static void**

Precondizione: h ≠ NULL and hp ≠ NULL and l ≠ NULL

Postcondizione: Vengono visualizzate tutte le prenotazioni del cliente specificato, se esiste.

Side-effects: Stampa a schermo

- **visualizza_prenotazioni_lezione (h, hp, l) → static void**

Precondizione: h ≠ NULL and hp ≠ NULL and l ≠ NULL

Postcondizione: Vengono visualizzate tutte le prenotazioni della lezione specificata, se esiste.

Side-effects: Stampa a schermo

- **menu_gestore (h, l, hp) → void**

Precondizione: h ≠ NULL and hp ≠ NULL and l ≠ NULL

Postcondizione: Il gestore può eseguire tutte le operazioni di gestione clienti/lezioni/ prenotazioni tramite un menu interattivo. I dati vengono aggiornati a seconda delle azioni selezionate.

Side-effects: Stampa a schermo, input da terminale, modifiche in memoria e su file, allocazioni e deallocazioni dinamiche.

HASH PRENOTAZIONI

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o: hashtable_p

Tip i u s a t i: int, Prenotazione, char*, const char*, uint32_t,

Op e r a t o r i:

- **fnv1aHash32 (const char*) → uint32_t**

Calcola l'hash FNV-1a a 32 bit per una stringa

- **hashfun (const char*, int) → static int**

Calcola un valore hash limitato alla dimensione di una tabella

- **newHashtable_p (int) → hashtable_p**

Crea e inizializza una nuova hash table per le prenotazioni

- **insertHash_p (hashtable_p, Prenotazione) → int**
Inserisce una prenotazione all'interno della tabella hash
- **hashDelete_p (hashtable_p, char*) → Prenotazione**
Rimuove una prenotazione dalla tabella hash tramite ID
- **hashSearch_p (hashtable_p, char*) → Prenotazione**
Cerca una prenotazione nella tabella hash tramite il suo ID
- **deleteList_p (Prenotazione) → static void**
Dealloca tutta la memoria occupata da una lista collegata alle prenotazioni
- **destroyHashtable_p (hashtable_p) → void**
Dealloca tutta la memoria associata alla tabella hash delle prenotazioni
- **stampaHash_p (hashtable_p) → void**
Stampa a video tutte le prenotazioni contenute nella tabella hash
- **stampaMinimaHash_p (hashtable_p) → void**
Stampa a video le informazioni essenziali delle prenotazioni nella tabella
- **get_size_hash_p (hashtable_p) → int**
Restituisce la dimensione della tabella hash
- **get_table_hash_p (hashtable_p) → Prenotazione**
Restituisce il puntatore alla tabella interna delle prenotazioni

SPECIFICA SEMANTICA

Tip o d i r i f e r i m e n t o : hashtable_p è una struttura dati contenente la lista dei clienti con i campi size (dimensione della tabella hash) e *table (vettore di puntatori a Prenotazione)

Op e r a t o r i :

- **fnv1aHash32 (key) → hash**
Precondizione: key ≠ NULL
Postcondizione: Restituisce il valore hash a 32 bit calcolato con l'algoritmo FNV-1a
- **hashfun (key, size) → fnv1aHash32(key) % size**
Precondizione: key ≠ NULL and size > 0
Postcondizione: Restituisce un intero nell'intervallo [0, size-1] ottenuto da fnv1aHash32(key) % size
- **newHashtable_p (size) → h**
Precondizione: size > 0
Postcondizione: viene restituito un nuovo oggetto hashtable_p correttamente allocato
Side-effects: Allocazione dinamica di memoria
- **insertHash_p (h, p) → int**
Precondizione: h ≠ NULL and p ≠ NULL
Postcondizione: La prenotazione p viene inserita nella tabella, in testa alla lista della posizione hashfun. Se inserita con successo ritorna 1, 0 altrimenti
Side-effects: Modifica della struttura interna della tabella hash h
- **hashDelete_p (h, id) → curr**
Precondizione: h ≠ NULL and id ≠ NULL and size > 0

Postcondizione: Se trova la prenotazione con ID, la rimuove dalla tabella e restituisce la tabella aggiornata. NULL altrimenti.

Side-effects: Modifica della struttura interna della tabella hash h

- **hashSearch_p (h, id) → curr**

Precondizione: h ≠ NULL and id ≠ NULL and size > 0

Postcondizione: Se trova la prenotazione con ID, la restituisce. NULL altrimenti

- **deleteList_p (p) → static void**

Precondizione: p ≠ NULL

Postcondizione: Tutti i nodi nella lista a partire da p sono stati deallocati dinamicamente e la lista viene distrutta completamente

Side-effects: Deallocazione dinamica della memoria

- **destroyHashtable_p (h) → void**

Precondizione: h ≠ NULL

Postcondizione: Tutti gli elementi e la tabella stessa vengono deallocati

Side-effects: Distrugge la tabella hash e libera ogni nodo Prenotazione

- **stampaHash_p (h) → void**

Precondizione: h ≠ NULL

Postcondizione: Le informazioni di tutte le prenotazioni vengono stampate

Side-effects: Stampa a schermo

- **stampaMinimaHash_p (h) → void**

Precondizione: h ≠ NULL

Postcondizione: Le informazioni minime di tutte le prenotazioni vengono stampate

Side-effects: Stampa a schermo

- **get_size_hash_p (h) → h->size**

Precondizione: h ≠ NULL

Postcondizione: Restituisce la dimensione della hash table

- **get_table_hash_p (h) → h->table**

Precondizione: h ≠ NULL

Postcondizione: Restituisce il puntatore all'array di prenotazioni

HASH

SPECIFICA SINTATTICA

T i p o d i r i f e r i m e n t o : hashtable

T i p i u s a t i: int, Cliente, char*, const char*, hashtable, uint32_t

O p e r a t o r i:

- **fnv1aHash32 (const char*) → uint32_t**

Calcola l'hash FNV-1a a 32 bit per una stringa

- **hashfun (const char*, int) → static int**

Calcola un valore hash limitato alla dimensione di una tabella

- **newHashtable (int) → hashtable**

Crea e inizializza una nuova hash table per i clienti

- **insertHash (hashtable, Cliente) → int**
Inserisce un cliente all'interno della tabella hash
- **hashDelete (hashtable, char*) → Cliente**
Rimuove un cliente dalla tabella hash tramite ID
- **hashSearch (hashtable, char*) → Cliente**
Cerca un cliente nella tabella hash tramite il suo ID
- **deleteList (Cliente) → static void**
Dealloca tutta la memoria occupata da una lista collegata alle prenotazioni
- **destroyHashtable (hashtable) → void**
Dealloca tutta la memoria associata alla tabella hash dei clienti
- **stampaHash (hashtable) → void**
Stampa a video tutti i clienti contenuti nella tabella hash
- **stampaMinimaHash (hashtable) → void**
Stampa a video le informazioni essenziali dei clienti nella tabella
- **get_size_hash (hashtable) → int**
Restituisce la dimensione della tabella hash
- **get_table_hash (hashtable) → Cliente**
Restituisce il puntatore alla tabella interna dei clienti

SPECIFICA SEMANTICA

T i p o d i r i f e r i m e n t o : hashtable è una struttura dati contenente i campi size (dimensione della tabella hash) e *table (vettore di puntatori a Cliente)

O p e r a t o r i:

- **fnv1aHash32 (key) → hash**
Precondizione: key ≠ NULL
Postcondizione: Restituisce il valore hash a 32 bit calcolato con l'algoritmo FNV-1a
- **hashfun (key, size) → fnv1aHash32(key) % size**
Precondizione: key ≠ NULL and size > 0
Postcondizione: Restituisce un intero nell'intervallo [0, size-1] ottenuto da fnv1aHash32(key) % size
- **newHashtable (size) → h**
Precondizione: size > 0
Postcondizione: viene restituito un nuovo oggetto hashtable correttamente allocato
Side-effects: Allocazione dinamica di memoria
- **insertHash (h, c) → int**
Precondizione: h ≠ NULL and c ≠ NULL
Postcondizione: Il cliente c viene inserito nella tabella, in testa alla lista della posizione hashfun. Se inserita con successo ritorna 1, 0 altrimenti
Side-effects: Modifica della struttura interna della tabella hash h
- **hashDelete (h, id) → curr**
Precondizione: h ≠ NULL and id ≠ NULL and size > 0

Postcondizione: Se trova il cliente con ID, lo rimuove dalla tabella e restituisce la tabella aggiornata. NULL altrimenti.

Side-effects: Modifica della struttura interna della tabella hash h

- **hashSearch (h, id) → curr**

Precondizione: h ≠ NULL and id ≠ NULL and size > 0

Postcondizione: Se trova il cliente con ID, la restituisce. NULL altrimenti

- **deleteList (p) → static void**

Precondizione: p ≠ NULL

Postcondizione: Tutti i nodi nella lista a partire da p sono stati deallocati dinamicamente e la lista viene distrutta completamente

Side-effects: Deallocazione dinamica della memoria

- **destroyHashtable (h) → void**

Precondizione: h ≠ NULL

Postcondizione: Tutti gli elementi e la tabella stessa vengono deallocati

Side-effects: Distrugge la tabella hash e libera ogni nodo Cliente

- **stampaHash (h) → void**

Precondizione: h ≠ NULL

Postcondizione: Le informazioni di tutte i clienti vengono stampate

Side-effects: Stampa a schermo

- **stampaMinimaHash (h) → void**

Precondizione: h ≠ NULL

Postcondizione: Le informazioni minime di tutte i clienti vengono stampate

Side-effects: Stampa a schermo

- **get_size_hash (h) → h->size**

Precondizione: h ≠ NULL

Postcondizione: Restituisce la dimensione della hash table

- **get_table_hash (h) → h->table**

Precondizione: h ≠ NULL

Postcondizione: Restituisce il puntatore all'array di prenotazioni

LISTA LEZIONI

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o : list

Tip i u s a t i : Lezione, int

Op e r a t o r i:

- **newList (void) → list**

Crea una nuova lista vuota

- **emptyList (list) → int**

Controlla se la lista è vuota

- **consList (Lezione, list) → list**

Aggiunge una nuova lezione in testa alla lista

- **tailList (list) → list**

Restituisce la coda della lista

- **getFirst (list) → Lezione**

Restituisce il primo nodo in testa alla lista di lezione

- **sizeList (list) → int**

Restituisce la lunghezza della lista

- **reverseList (list) → list**

Restituisce la lista invertita

- **removeList (list, int) → list**

Rimuove la lezione, in posizione passata in input, dalla lista

- **copiaLista (list) → static list**

Copia la lista originale in una lista temporanea

- **outputList (list) → void**

Stampa tutte le lezioni della lista

- **stampaMinimaList (list) → void**

Stampa i dettagli essenziali di tutte le lezioni della lista

- **ordina_Lista (list) → list**

Ordina la lista di lezioni in ordine cronologico crescente

SPECIFICA SEMANTICA

Tipo di riferimento: list è una lista contenente le lezioni

Operatori:

- **newList (void) → list**

Postcondizione: Restituisce una lista vuota

- **emptyList (l) → int**

Postcondizione: Restituisce 1 se l è vuota, 0 altrimenti

- **consList (Le, l) → list**

Precondizione: le ≠ NULL

Postcondizione: Restituisce una nuova lista con le come primo elemento e l come coda

Side-effects: Alloca dinamicamente memoria per un nuovo nodo

- **tailList (l) → temp**

Precondizione: l ≠ NULL

Postcondizione: Restituisce la coda della lista

- **getFirst (l) → l->lez**

Precondizione: l ≠ NULL

Postcondizione: Restituisce la lezione contenuta nel primo nodo

- **sizeList (l) → n**

Precondizione: l ≠ NULL

Postcondizione: Restituisce il numero di elementi nella lista

- **reverseList (l) → rev**

Precondizione: l ≠ NULL

Postcondizione: Restituisce una nuova lista con gli elementi di l, ma in ordine inverso

Side-effects: Alloca memoria per i nuovi nodi

- **removeList (l, pos) → l2**
Precondizione: $0 \leq \text{pos} < \text{sizeList}(L)$
Postcondizione: Restituisce una nuova lista ottenuta rimuovendo l'elemento nella posizione pos
Side-effects: Alloca memoria per i nuovi nodi
- **copiaLista (l) → copia**
Precondizione: $l \neq \text{NULL}$
Postcondizione: Restituisce una copia della lista l, mantenendo l'ordine
Side-effects: Alloca memoria per i nuovi nodi
- **outputList (l) → void**
Precondizione: $l \neq \text{NULL}$
Postcondizione: Stampa tutte le lezioni presenti nella lista l, ordinate per data
Side-effects: Stampa a schermo
- **stampaMinimaList (l) → void**
Precondizione: $l \neq \text{NULL}$
Postcondizione: Stampa i dettagli essenziali delle lezioni presenti nella lista l, ordinate per data
Side-effects: Stampa a schermo
- **ordina_Lista (l) → reverseList(l_ord)**
Precondizione: $l \neq \text{NULL}$
Postcondizione: Restituisce una nuova lista contenente tutte le lezioni presenti in l, ordinate per data crescente
Side-effects: Alloca nuove strutture dati

REPORT

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o: Stats

Tip i u s a t i: char, int, Data, hashtable_p, list, Prenotazione, FILE*, Lezione, const char*

Op e r a t o r i:

- **aggiorna_statistica (Stats*, int*, int*, const char*) → static int**
Trova o aggiunge una lezione all'array di statistiche. Cerca se esiste già una statistica per quella lezione. Se la trova, incrementa, altrimenti aggiunge la lezione all'array.
- **genera_report_mensile (hashtable_p, list) → void**
Genera e stampa un report mensile (file di testo) sulle prenotazioni e sulle tre lezioni più frequentate
- **genera_report_test (hashtable_p, list, const char*) → void**
Genera un report testuale sulle prenotazioni contenute nella tabella hash temporanea
usata per i test, con le statistiche delle lezioni più prenotate nel mese corrente

SPECIFICA SEMANTICA

T i p o d i r i f e r i m e n t o : La struttura stats è una struttura contenente l'ID della lezione e il numero di prenotazioni su quelle lezioni

O p e r a t o r i:

- **aggiorna_statistica (stats, n, size_stats, id_lezione) → static int**
Precondizione: stats % NULL and id_lezione % NULL and *n ≤ *size_stats
Postcondizione: Se id_lezione è già presente in stats, incrementa il campo "conteggio" della struttura. Se non è presente, inserisce un nuovo elemento con id_lezione e conteggio = 1 e incrementa n.
Side-effects: Possibile riallocazione di memoria (realloc). In caso di memory fault, il programma termina con exit(EXIT_FAILURE)
- **genera_report_mensile (hp, l) → void**
Precondizione: hp % NULL and l % NULL
Postcondizione: Viene generato un file di testo riepilogativo con le statistiche relative al numero di prenotazioni totali del mese e le tre lezioni più prenotate del mese
Side-effects: Stampa a schermo, allocazione e deallocazione dinamica della memoria, scrittura di file
- **genera_report_test (hp_test, l, output_fname) → void**
Precondizione: hp_test % NULL and l % NULL and output_fname % NULL
Postcondizione: Viene generato un file di testo "output_fname" in modalità "w" (Scrittura). All'interno di questo file devono essere presenti: tutte le prenotazioni effettuate nel mese corrente e le tre lezioni più prenotate che si svolgono nel mese corrente
Side-effects: Stampa a schermo, allocazione e deallocazione dinamica della memoria, scrittura su file

UTILS

SPECIFICA SINTATTICA

T i p o d i r i f e r i m e n t o :

T i p i u s a t i: Cliente, hashtable, int, char, Data, FILE, static int, Prenotazione, Orario, Lezione, list

O p e r a t o r i:

- **rinnova_abbonamento (Cliente, hashtable) → void**
Rinnova l'abbonamento di un cliente per un numero specificato di mesi
- **carica_contatore_generico (const char*, const char*) → int**
Carica il valore massimo di un ID generico da un file, basandosi su un prefisso
- **genera_id_generico (const char*, const char*) → char***
Genera un nuovo ID univoco basato su un prefisso e un file di contatori
- **carica_cliente_da_file (hashtable) → void**
Carica i dati dei clienti da un file e li inserisce nella hashtable
- **carica_prenotazione_da_file (hashtable_p) → void**

Carica tutte le prenotazioni dal file e le inserisce nella hashtable delle prenotazioni

- **carica_lezione_da_file (list) → list**
Carica le lezioni da file e le inserisce in una lista
- **aggiorna_file_clienti (hashtable) → void**
Salva tutti i clienti dalla hashtable su file
- **aggiorna_file_lezione (list) → void**
Salva la lista delle lezioni su file
- **aggiorna_file_prenotazioni (hashtable_p) → void**
Scrive tutte le prenotazioni presenti nella hashtable su file
- **pulisci_schermo () → void**
Pulisce lo schermo della console
- **calcola_durata_in_mesi (Data, Data) → int**
Calcola la durata in mesi tra due date
- **stampa_prenotazioni_cliente (Cliente, hashtable_p, list) → void**
Stampa tutte le prenotazioni associate a un cliente
- **stampa_prenotazioni_lezione (hashtable, Lezione, hashtable_p) → void**
Stampa tutte le prenotazioni associate a una specifica lezione
- **cerca_lezione_per_id (list, const char*) → Lezione**
Cerca una lezione nella lista in base al suo ID
- **stampa_lezioni_libere (list) → void**
Stampa la lista delle lezioni che hanno posti disponibili e che non sono passate
- **calcolo_scadenza_abbonamento (Data, int) → Data**
Calcola la data di scadenza dell'abbonamento dato dalla data d'inizio e la durata in mesi
- **abbonamento_valido (Data, Data) → int**
Verifica se un abbonamento è valido
- **leggi_intero () → int**
Legge un intero da input, gestendo errori di input non valido
- **solo lettere (char*) → int**
Verifica se una stringa contiene solo lettere (maiuscole o minuscole) e spazi
- **nome_mese (int) → const char**
Converte un numero di mese in una stringa con il nome del mese

SPECIFICA SEMANTICA

T i p o d i r i f e r i m e n t o :

O p e r a t o r i:

- **rinnova_abbonamento (c, h) → void**
Precondizione: c ≠ NULL and h ≠ NULL
Postcondizione: L'abbonamento del cliente viene esteso, aggiornato nei dati e salvato su file
Side-effects: La struttura dati Cliente viene modificata: viene aggiornata la data di scadenza e la durata dell'abbonamento, scrittura su file "clienti.txt"
- **carica_contatore_generico (filename, prefix) → max_id**
Precondizione: filename ≠ NULL and prefix ≠ NULL

Postcondizione: Restituisce il valore massimo trovato per il prefisso specificato

Side-effects: Lettura di file

▪ **genera_id_generico (prefix_letter, filename) → id**

Precondizione: filename ≠ NULL and prefix_letter ∈ {"C", "L", "P"}

Postcondizione: Restituisce una nuova stringa ID allocata dinamicamente

Side-effects: Allocazione dinamica della memoria

▪ **carica_clienti_da_file (h) → void**

Precondizione: clienti.txt ≠ NULL and h ≠ NULL

Postcondizione: Tutti i clienti letti dal file vengono inseriti nell'hashtable h tramite insertHash

Side-effects: Lettura da file

▪ **carica_prenotazioni_da_file (h) → void**

Precondizione: prenotazioni.txt ≠ NULL and h ≠ NULL

Postcondizione: Tutte le prenotazioni lette dal file vengono inserite nell'hashtable h tramite insertHash

Side-effects: Lettura da file

▪ **carica_lezioni_da_file (l) → l**

Precondizione: lezione.txt ≠ NULL

Postcondizione: La lista l viene popolata con le lezioni lette da file

Side-effects: Lettura da file

▪ **aggiorna_file_clienti (h) → void**

Precondizione: h ≠ NULL and clienti.txt ≠ NULL

Postcondizione: I dati del cliente vengono scritti nel file "clienti.txt"

Side-effects: Scrittura su file

▪ **aggiorna_file_lezioni (l) → void**

Precondizione: l ≠ NULL

Postcondizione: I dati delle lezioni vengono scritti nel file "lezioni.txt"

Side-effects: Scrittura su file

▪ **aggiorna_file_prenotazioni (h) → void**

Precondizione: h ≠ NULL

Postcondizione: Il file "prenotazione.txt" viene aggiornato con tutte le prenotazioni

Side-effects: Scrittura su file

▪ **pulisci_schermo () → void**

Postcondizione: La console viene ripulita

Side-effects: Chiama system("cls") su windows o system("clear") su Unix o macOS

▪ **calcola_durata_in_mesi (data_inizio, data_fine) → durata_in_mesi**

Precondizione: data_inizio ≠ NULL and data_fine ≠ NULL and data_fine ≥ data_inizio

Postcondizione: Restituisce la differenza in mesi interi tra le due date

- **stampa_prenotazioni_cliente (c, hp, l) → void**
Precondizione: c ≠ NULL and hp ≠ NULL
Postcondizione: Vengono stampate le prenotazioni del cliente, se presenti
Side-effects: Stampa a schermo
 - **stampa_prenotazioni_lezione (h, lezione, hp) → void** Precondizione: h ≠ NULL and hp ≠ NULL and lezione ≠ NULL Postcondizione: Vengono stampate le prenotazioni della lezione specificata, se presenti
Side-effects: Stampa a schermo
 - **cerca_lezione_per_id (l, id_lezione) → le**
Precondizione: l ≠ NULL and id_lezione ≠ NULL
Postcondizione: Restituisce la lezione con l'ID specificato, oppure NULL se non trovata
 - **stampa_lezioni_libere (l) → void**
Precondizione: l ≠ NULL
Postcondizione: Vengono stampate tutte le lezioni con posti liberi e data odierna o futura
Side-effects: Stampa a schermo
 - **calcolo_scadenza_abbonamento (data_inizio, durata_abbonamento) → data_scadenza**
Precondizione: data_inizio ≠ NULL and durata_abbonamento ≥ 0
Postcondizione: Restituisce la data di scadenza calcolata
 - **abbonamento_valido (oggi, scadenza) → int**
Precondizione: oggi ≠ NULL and scadenza ≠ NULL
Postcondizione: Restituisce 1 se l'abbonamento è ancora valido (cioè non scaduto), 0 altrimenti
 - **leggi_intero () → valore**
Precondizione: l'input deve essere da stdin
Postcondizione: Restituisce il valore intero letto dall'utente
Side-effects: Lettura da stdin, stampa su stdout
 - **solo lettere (s) → contiene_lettera**
Precondizione: s ≠ NULL
Postcondizione: Ritorna 1 se la stringa contiene almeno una lettera e nessun carattere non ammesso. Ritorna 0 altrimenti
 - **nome_mese (mese) → mesi**
Precondizione: $1 \leq \text{mese} \leq 12$
Postcondizione: Restituisce il nome del mese come stringa. Se il mese è fuori dal range, restituisce "Mese sconosciuto"
-

MAIN TESTING

SPECIFICA SINTATTICA

Tip o d i r i f e r i m e n t o :

Tip i u s a t i: char*, FILE, int, Prenotazione, Lezione, list, Cliente, const char*, Data, hashtable, hashtable_p

Op e r a t o r i:

- **confronta_file (char*, char*) → int**
Confronta due file riga per riga per verificarne l'identità
- **salva_prenotazione_test (Prenotazione, int, const char*) → void**
Salva su file i dati relativi a una prenotazione, in formato leggibile
- **prenota_lezione_test (Cliente, list, hashtable_p, char*, const char*) → int**
Permette ad un cliente di prenotare una lezione in un ambiente di test
- **esegui_test_prenotazione (char*, char*, char*, hashtable, list, hashtable_p) → int**
Esegue un test automatico per la prenotazione lezioni
- **salva_cliente_test (Cliente, const char*) → void**
Salva i dati relativi all'abbonamento di un cliente su file
- **rinnova_abbonamento_test (Cliente, hashtable, int, const char*) → int**
Rinnova l'abbonamento di un cliente per un numero specificato di mesi
- **esegui_test_abbonamenti (char*, char*, char*, hashtable) → int**
Esegue un test automatico su abbonamenti: validità o rinnovo
- **prenotazioni_per_report (Cliente, list, hashtable_p, char*) → int**
Prenota una lezione solo se valida per generare il report
- **esegui_test_report (char*, char*, char*, hashtable, list) → int**
Esegue test per la generazione del report delle prenotazioni
- **run_test_case (char*, int, hashtable, list, hashtable_p) → int**
Avvia l'esecuzione di un caso di test specificato

SPECIFICA SEMANTICA

Tip o d i r i f e r i m e n t o :

Op e r a t o r i:

- **confronta_file (oracle, output) → int**
Precondizione: fp_oracle ≠ NULL and fp_output ≠ NULL
Postcondizione: Restituisce 1 se i file combaciano, 0 altrimenti
Side-effects: Lettura da file, stampa a schermo
- **salva_prenotazione_test (p, po, output_fname) → void**
Precondizione: p ≠ NULL and output_fname ≠ NULL
Postcondizione: I dati della prenotazione vengono scritti su file
Side-effects: Scrittura su file
- **prenota_lezione_test (c, l, hp, id_lezione_scelta, output_fname) → int**
Precondizione: c ≠ NULL and hp ≠ NULL and l ≠ NULL
Postcondizione: Se la prenotazione è possibile, viene effettuata e salvata su file. Al termine del test, lo stato viene ripristinato

Side-effects: Scrittura su file, deallocazione di memoria, modifica delle strutture

- **esegui_test_prenotazione (input_fname, output_fname, oracle_fname, h, l, hp) → z**

Precondizione: fp_input ≠ NULL and fp_output ≠ NULL

Postcondizione: Il risultato del test viene confrontato con il file oracle

Side-effects: Lettura da file, scrittura su file, chiamata di funzione 'prenota_lezione_test' e 'confronta_file'

- **salva_cliente_test (c, output_fname) → void**

Precondizione: c ≠ NULL

Postcondizione: I dati vengono scritti su file

Side-effects: Scrittura su file

- **rinnova_abbonamento_test (c, h, durata, output_fname) → int**

Precondizione: c ≠ NULL

Postcondizione: La data di scadenza e la durata dell'abbonamento vengono aggiornate. I dati vengono scritti su file

Side-effects: Modifica oggetto Cliente, scrittura su file, chiamata di funzione 'set_data_scadenza' e 'set_durata'

- **esegui_test_abbonamenti (input_fname, output_fname, oracle_fname, h) → z**

Precondizione: fp_input ≠ NULL and fp_output ≠ NULL

Postcondizione: L'output del test viene confrontato con l'oracle

Side-effects: Scrittura su file, chiamata di funzione 'rinnova_abbonamento_test', 'abbonamento_valido' e 'confronta_file'

- **prenotazioni_per_report (c, l, hp_test, id_lezione) → int**

Precondizione: c ≠ NULL and l ≠ NULL

Postcondizione: La prenotazione viene inserita nella hashtable temporanea

Side-effects: Aggiornamento della struttura lezione, allocazione di memoria, chiamata di funzione 'cerca_lezione_per_id', 'abbonamento_valido', 'confronta_date'

- **esegui_test_report (input_fname, output_fname, oracle_fname, h, l) → z**

Precondizione: fp_input ≠ NULL and fp_output ≠ NULL

Postcondizione: Il report viene generato e confrontato con il file oracle

Side-effects: Allocazione di memoria, scrittura su file, chiamata di funzione 'genera_report_test' e 'prenotazioni_per_report'

- **run_test_case (tc_id, tipo_test, h, l, hp) → esito**

Precondizione: tipo_test = {1, 2, 3}

Postcondizione: Esegue e ritorna il risultato del test

Side-effects: Chiama tutte le funzioni precedenti in base alla scelta, genera dinamicamente i nomi di file

V. RAZIONALE CASI DI TEST

INTRODUZIONE

Nel presente documento vengono descritti i casi di test implementati per verificare il corretto funzionamento delle funzionalità principali del sistema, tra cui:

- La registrazione delle prenotazioni e l'aggiornamento delle disponibilità;
- La gestione degli abbonamenti (verifica validità e rinnovo);
- La generazione del report sulle prenotazioni effettuate.

I test sono stati automatizzati tramite specifiche funzioni C che leggono input da file, producono output in file separati e li confrontano con file **oracle**, contenenti i risultati attesi.

Le funzioni utilizzate corrispondono a quelle presenti nel progetto originale, con alcune modifiche minime. In particolare, sono state sostituite alcune istruzioni di stampa con valori di ritorno, poiché le stampe risultano superflue durante la fase di testing. Inoltre, i dati prodotti vengono salvati in file di testo, che fungono da output e verranno successivamente confrontati con l'oracolo.



Nota: I dati di input utilizzati durante il testing erano validi al momento dell'esecuzione dei test, in relazione agli scenari specifici da verificare. Tutte le operazioni effettuate nell'ambiente di test sono completamente isolate e non influenzano i file del programma principale (ad esempio, una prenotazione creata durante il test non viene salvata nel file prenotazioni.txt effettivo). **Qualora in futuro i dati utilizzati venissero modificati**, sarà necessario **aggiornare i file di input e i file oracle** con valori coerenti rispetto al comportamento che si intende verificare. Per realizzare l'ambiente di testing è stato creato un file main dedicato.

La compilazione del programma di test avviene tramite il comando: **make testing.exe**. Successivamente, per eseguire l'applicativo di test, è sufficiente lanciare il comando: **./testing.exe**

TEST SUITE

La **test suite** sviluppata comprende una serie di casi di test volti a verificare il comportamento corretto delle funzionalità principali del sistema. Ogni test è identificato da un codice univoco e riguarda un'area specifica del sistema: prenotazioni, abbonamenti e reportistica. Di seguito vengono elencati i casi di test implementati:

PRENOTAZIONI

La funzione `esegui_test_prenotazione` è progettata per verificare che il sistema di prenotazione delle lezioni funzioni correttamente in condizioni controllate e ripetibili. L'intero processo è costruito in modo da non alterare i dati reali, offrendo così un ambiente sicuro per testare il comportamento del sistema.

Tutto inizia con la **preparazione dell'ambiente di test**. Vengono predisposti diversi file di input contenente una riga ciascuno, ognuno con una coppia composta da un ID

cliente e un ID lezione. In memoria sono disponibili le strutture necessarie per eseguire il test.

Durante la fase di **esecuzione del test**, la funzione apre il file di input e lo legge riga per riga. Per ogni coppia cliente-lezione, il sistema cerca il cliente corrispondente nella hashtable e poi invoca la funzione `prenota_lezione_test`, che gestisce la logica di validazione e prenotazione.

La funzione `prenota_lezione_test` esegue una serie di controlli per assicurarsi che la prenotazione sia legittima. Se tutte le condizioni risultano soddisfatte, viene generato un ID univoco per la nuova prenotazione. A questo punto viene creata una nuova struttura `Prenotazione`, che viene **inserita temporaneamente nella hashtable**. La prenotazione non viene però salvata nel sistema reale: viene solo scritta nel file di output tramite la funzione `salva_prenotazione_test`.

Questa funzione scrive su file i dettagli della prenotazione in un formato leggibile, includendo:

- L'ID della prenotazione.
- L'ID del cliente.
- L'ID della lezione prenotata.
- La data della prenotazione, che corrisponde alla data corrente.
- Il numero di posti occupati aggiornato per la lezione.

Appena dopo la scrittura su file, la prenotazione viene **rimossa dalla hashtable temporanea** e i posti occupati della lezione vengono **ripristinati** al valore precedente, così da non alterare in alcun modo lo stato effettivo del sistema.

Se la prenotazione fallisce per una qualsiasi delle condizioni, il file di output conterrà un messaggio descrittivo che spiega la causa del fallimento. Alla fine del processo, l'intero contenuto del file di output viene confrontato con un **file oracle**, che contiene il risultato atteso. Questo confronto, effettuato con la funzione `confronta_file`, serve per validare se il sistema ha prodotto l'output corretto.

In conclusione, la funzione `esegui_test_prenotazione` restituisce 1 se l'output prodotto corrisponde esattamente a quello dell'oracolo (indicando che il test è **superato con successo**), oppure 0 se c'è una discrepanza (quindi **test fallito**).

Test Case:

TC1_prenotazione

Verifica che un cliente valido possa effettuare correttamente una prenotazione per una lezione valida. Il test controlla che:

- La prenotazione venga generata correttamente con i dati previsti;
- Il numero di posti occupati della lezione venga incrementato correttamente.



Nota: il file oracle deve essere aggiornato con la **data corrente** al momento del test, poiché la data della prenotazione viene impostata automaticamente al giorno in cui il test viene eseguito.

L'input utilizzato è il seguente:

<pre>TC1_prenotazione_input.txt 1 C012 L009</pre>	<pre>ID CLIENTE: C012 ID LEZIONE: L009</pre>
--	--

L'oracolo atteso è il seguente:

```
TC1_prenotazione_oracle.txt
1  ID: P011
2  ID Cliente: C012
3  ID Lezione: L009
4  Data Prenotazione: 05/06/2025
5  -----
6  Posti occupati: 1
```

Con il numero di posti occupati della lezione incrementato di 1, e con la data odierna della

L'output generato è il seguente:

```
TC1_prenotazione_output.txt
1  ID: P011
2  ID Cliente: C012
3  ID Lezione: L009
4  Data Prenotazione: 05/06/2025
5  -----
6  Posti occupati: 1
```

TC2_prenotazione

Verifica il caso in cui un cliente è già prenotato alla lezione: il sistema non deve creare una nuova prenotazione.

L'input utilizzato è il seguente:

<pre>TC2_prenotazione_input.txt 1 C004 L008</pre>	<pre>ID CLIENTE: C004 ID LEZIONE: L008 Il cliente è già prenotato alla lezione</pre>
--	--

L'oracolo atteso è il seguente:

```
TC2_prenotazione_oracle.txt
1  Prenotazione fallita per C004
```

L'output generato è il seguente:

```
TC2_prenotazione_output.txt
1  Prenotazione fallita per C004
```

TC3_prenotazione

Verifica il comportamento con un cliente che tenta più volte di prenotarsi alla stessa lezione: anche in questo caso, non deve essere creata alcuna nuova prenotazione.

L'input utilizzato è il seguente:

```
TC3_prenotazione_input.txt
1 PIPPO L004
```

ID CLIENTE: PIPPO (non esiste questo cliente)
ID LEZIONE: L008

L'oracolo atteso è il seguente:

```
TC3_prenotazione_oracle.txt
1 Prenotazione fallita per PIPPO
```

L'output generato è il seguente:

```
TC3_prenotazione_output.txt
1 Prenotazione fallita per PIPPO
```

ABBONAMENTI

La funzione `esegui_test_abbonamenti` è progettata per testare in modo automatico due aspetti fondamentali legati agli abbonamenti dei clienti: la validità di un abbonamento alla data attuale e la corretta gestione del rinnovo dell'abbonamento. Il test è concepito in modo tale da poter essere eseguito in maniera isolata e ripetibile, senza influenzare i dati reali del sistema.

Il processo inizia con la preparazione dell'ambiente di test. Vengono predisposti diversi file di input contenenti, una riga con, o solo l'ID di un cliente (per testare se il suo abbonamento è ancora valido), oppure l'ID del cliente seguito da un numero intero che rappresenta i mesi con cui si desidera rinnovare l'abbonamento.

Quando la funzione `esegui_test_abbonamenti` viene invocata, essa apre i file di input e legge la riga. In base al contenuto della riga, viene determinato quale tipo di test eseguire:

- Se la riga contiene un solo parametro (ovvero l'ID del cliente), viene eseguito un test sulla validità dell'abbonamento.
- Se invece contiene due parametri (l'ID e una durata in mesi), si esegue un test sul rinnovo dell'abbonamento.

Nel caso di test sulla validità, il cliente viene cercato. Se non viene trovato, viene scritto nel file di output un messaggio del tipo *"Cliente non trovato"*. Se il cliente esiste, viene controllata la validità del suo abbonamento usando la funzione `abbonamento_valido()`, che confronta la data di scadenza con la data odierna. Il risultato viene quindi riportato su file output, indicando se l'abbonamento è ancora valido o meno.

Nel caso di test di rinnovo, si utilizza la funzione `rinnova_abbonamento_test`, che esegue vari passaggi. Innanzitutto, calcola una nuova data di scadenza partendo da quella attuale, incrementandola del numero di mesi richiesto. Poi aggiorna la scadenza del cliente e ricalcola la durata effettiva dell'abbonamento come intervallo di mesi tra la data di iscrizione e la nuova data di scadenza. Infine, tramite la funzione `salva_cliente_test`, i nuovi dati vengono scritti nel file di output, comprensivi sia della durata aggiornata che della nuova data di scadenza.

Una volta completato il test (sia di validità che di rinnovo), il contenuto del file di output viene confrontato con il file oracle, che rappresenta l'output atteso. Questo confronto è effettuato tramite la funzione `confronta_file`. Infine, la funzione `esegui_test_abbonamenti` restituisce un risultato intero: 1 se l'output generato corrisponde esattamente a quello dell'oracolo, indicando che il test è stato superato con successo; 0 se invece ci sono differenze tra i due file, segnalando un possibile errore nel comportamento della funzione testata.

Test case:

TC1_abbonamento

Verifica il **rinnovo di un abbonamento**: il test controlla che la durata dell'abbonamento e la data di scadenza vengano aggiornate correttamente.

L'input utilizzato è il seguente:

```
TC1_abbonamento_input.txt
1 C012 4
```

ID CLIENTE: C012
RINNOVO ABBONAMENTO (in mesi): 4

L'oracolo atteso è il seguente:

```
TC1_abbonamento_oracle.txt
1 Durata abbonamento: 25
2 Data scadenza: 17/06/2027
```

Con la durata dell'abbonamento incrementata, e con la data di scadenza aggiornata.

L'output generato è il seguente:

```
TC1_abbonamento_output.txt
1 Durata abbonamento: 25
2 Data scadenza: 17/06/2027
```

TC2_abbonamento

Verifica un **abbonamento ancora valido** (data di scadenza non superata): il sistema deve scrivere in output *"abbonamento valido"*.

L'input utilizzato è il seguente:

```
TC2_abbonamento_input.txt
1 C005
```

ID CLIENTE: C005
(cliente con

L'oracolo atteso è il seguente:

```
TC2_abbonamento_oracle.txt
1 Abbonamento valido
```

L'output generato è il seguente:

```
TC2_abbonamento_output.txt
1 Abbonamento valido
```


TC3_abbonamento

Verifica un **abbonamento scaduto** (data di scadenza superata): il sistema deve scrivere in output *"abbonamento non valido"*.

L'input utilizzato è il seguente:

```
TC3_abbonamento_input.txt
1 C003
```

ID CLIENTE: C003 (cliente
con abbonamento)

L'oracolo atteso è il seguente:

```
TC3_abbonamento_oracle.txt
1 Abbonamento non valido
```

L'output generato è il seguente:

```
TC3_abbonamento_output.txt
1 Abbonamento non valido
```

REPORT

La funzione `esegui_test_report` ha lo scopo di verificare automaticamente la correttezza della generazione di un report di prenotazioni, confrontando l'output prodotto con un file oracle che rappresenta il risultato atteso.

Il processo ha inizio con la creazione di una hashtable temporanea, chiamata `hp_test`, che viene utilizzata per raccogliere tutte le prenotazioni valide effettuate durante il test. Successivamente, vengono aperti tre file fondamentali: il file di input, da cui saranno lette le richieste di prenotazione (ognuna composta da un ID cliente e un ID lezione); il file di output, in cui sarà salvato il report generato; e infine il file oracle, contenente il risultato atteso.

Una volta aperti i file, si entra nella fase di lettura e validazione delle prenotazioni. Per ogni riga letta dal file di input, vengono estratti l'ID del cliente e l'ID della lezione a cui vuole iscriversi.

L'effettiva validazione della prenotazione avviene all'interno della funzione `prenotazioni_per_report`. Qui, si eseguono diversi controlli in sequenza.

Se tutti i controlli vengono superati, la prenotazione viene considerata valida: si aggiorna il numero di posti occupati nella lezione, si crea un nuovo oggetto `Prenotazione` e lo si inserisce nella hashtable `hp_test`.

Nel caso in cui anche solo uno dei controlli fallisca, viene scritto un messaggio nel file di output che indica il fallimento della prenotazione per il cliente specificato.

Una volta processate tutte le richieste, viene generato il report finale chiamando la funzione `genera_report_test`. Questa funzione costruisce un riepilogo leggibile delle prenotazioni valide presenti nella hashtable temporanea, e lo salva nel file di output.

Infine, il contenuto del file di output viene confrontato con quello del file oracle, tramite la funzione `confronta_file`. Se i due file coincidono perfettamente, il test viene considerato superato e la funzione restituisce 1. In caso contrario, viene restituito 0, indicando un errore nella generazione del report.

Test case:

▪ TC1_report

Verifica la corretta **generazione del report mensile delle prenotazioni**. Il test crea un totale di 6 prenotazioni:

- 3 relative a una lezione;
- 2 a una seconda lezione;
- 1 a una terza lezione.

Le prenotazioni vengono temporaneamente inserite in una struttura hash, e il sistema deve generare come output il report mensile aggregato.

L'input utilizzato è il seguente:

```
TC1_report_input.txt
1  C013 L007
2  C012 L007
3  C005 L007
4  C013 L006
5  C008 L006
6  C013 L005
```

Servono per creare le 6 prenotazioni:
L007 avrà 3 prenotazioni;
L006 avrà 2 prenotazioni;
L005 avrà 1 prenotazione;

L'oracolo atteso è il seguente:

```
TC1_report_oracle.txt
1  REPORT MENSILE - Giugno 2025
2
3  Totale prenotazioni effettuate: 6
4
5  Lezioni più frequentate (solo quelle di Giugno):
6  1. Danza Classica - 3 prenotazioni
7  |   Data: 17/06/2025 | Ora: 14:00
8  2. Pilates - 2 prenotazioni
9  |   Data: 18/06/2025 | Ora: 15:30
10 3. Zumba - 1 prenotazioni
11 |   Data: 24/06/2025 | Ora: 17:00
12
```

L'output generato è il seguente:

```
TC1_report_output.txt
1  REPORT MENSILE - Giugno 2025
2
3  Totale prenotazioni effettuate: 6
4
5  Lezioni più frequentate (solo quelle di Giugno):
6  ✓ 1. Danza Classica - 3 prenotazioni
7  |   Data: 17/06/2025 | Ora: 14:00
8  ✓ 2. Pilates - 2 prenotazioni
9  |   Data: 18/06/2025 | Ora: 15:30
10 ✓ 3. Zumba - 1 prenotazioni
11 |   Data: 24/06/2025 | Ora: 17:00
12
```