Sistema di Gestione delle Prenotazioni per una Palestra

Documentazione del Progetto

Giulia Sabatino - 0512122824

Viola Varone - 0512122523

Sommario

Introduzione

Questo documento illustra le fasi di progettazione e sviluppo del nostro programma in linguaggio C ideato per supportare la gestione delle prenotazioni in una palestra.

Lo scopo principale è stato quello di realizzare un sistema semplice e funzionale, in grado di gestire efficacemente diverse funzionalità per il corretto funzionamento delle attività di una palestra generica.

Il programma infatti permette agli utenti di prenotare le lezioni in base agli orari disponibili e al numero massimo di partecipanti, visualizzare le disponibilità delle varie lezioni, controllare gli abbonamenti e generare report mensili.

Di seguito verranno analizzate le scelte progettuali adottate, la struttura generale del programma, le funzionalità implementate e la modalità di esecuzione dei test in modo da fornire una panoramica completa e dettagliata del nostro lavoro.

Motivazione della scelta dell'ADT

Nel progetto è stato fondamentale scegliere strutture dati (*Abstract Data Types*) che garantissero flessibilità, efficienza e semplicità nella gestione di insiemi dinamici di dati. Tra le varie strutture dati studiate durante il corso abbiamo scelto di utilizzare principalmente le **liste semplicemente collegate** (*singly linked lists*), dove ogni elemento (nodo) contiene un dato e un puntatore al nodo successivo.

Gestione di dati dinamici e della memoria

Le liste sono ideali per memorizzare anche grandi quantità di dati dinamici, soprattutto quando **la loro dimensione o il numero di elementi non è nota ed è variabile nel tempo** (proprio come il numero di clienti o di prenotazioni di una palestra). Inoltre, le liste **evitano lo spreco di spazio in memoria** poiché viene allocata solo per gli elementi presenti.

Flessibilità del sistema

Le liste permettono di **implementare facilmente operazioni di inserimento e cancellazione** semplici da usare anche in modo frequente, per cui risulta particolarmente utile per rimuovere clienti non attivi, cancellare prenotazioni o aggiornare abbonamenti.

Mantenibilità del programma

Le liste ci consentono di **gestire un numero crescente di dati** (nel nostro caso lezioni, clienti o abbonamenti) senza modificare la logica delle operazioni, di conseguenza si ha una navigazione flessibile tra i vari elementi.

Progettazione

Il nostro progetto è stato strutturato in diversi moduli distinti, ognuno responsabile di una specifica funzionalità del sistema in modo da individuare, correggere e migliorare singole parti più facilmente e soprattutto senza modificare la struttura dell'intero codice.

Questa scelta è strettamente collegata ai concetti di **astrazione e information hiding**, secondo cui ogni modulo deve esporre solo ciò che è necessario attraverso le sue interfacce pubbliche (file .h) senza mostrare i dettagli implementativi.

La progettazione modulare, oltre a rendere il codice più ordinato, garantisce anche una maggiore sicurezza e facilità d'uso perché si riducono notevolmente le dipendenze e i rischi di errori dovuti a modifiche indesiderate o accessi non controllati.

Moduli e file header

Il sistema è suddiviso in diversi moduli, ognuno con un proprio file header .h che dichiara le strutture dati e le funzioni associate. Questa divisione migliora la modularità e la manutenibilità del codice.

- Il modulo **cliente.h** contiene la definizione della **struttura Cliente**, con i dati personali dei clienti iscritti, e le funzioni per aggiungere, cercare e stampare i clienti registrati. Nel dettaglio, le funzioni principali di cliente.h sono:

1) *aggiungi_cliente*, che aggiunge un nuovo cliente alla lista;
2) *cerca_cliente*, che cerca un cliente tramite il suo ID;
3) *stampa_clienti*, che stampa la lista dei clienti.

- Il modulo **lezione.h** definisce la **struttura Lezione**, con i dettagli di ogni lezione, e le funzioni per gestire le lezioni, inclusa la disponibilità e gli orari. Nel dettaglio, le funzioni principali di lezione.h sono:

1) *aggiungi_lezione*, che crea una nuova lezione e la aggiunge alla lista;
2) *visualizza_lezione*, che stampa le informazioni sulle lezioni disponibili;
3) *prenota_lezione*, che verifica la disponibilità della lezione e aggiorna il numero di partecipanti.

- Il modulo **abbonamento.h** contiene invece la **struttura Abbonamento**, che gestisce le informazioni sugli abbonamenti dei clienti, e le funzioni per creare e gestire gli abbonamenti associati ai clienti. Nel dettaglio, le funzioni principali di abbonamento.h sono:

1) *crea_abbonamento*, che permette la creazione di un abbonamento;
2) *gestisci_abbonamento*, consente di modificare o cancellare abbonamenti esistenti;

3) *visualizza_abbonamento*, che visualizza tutti gli abbonamenti attivi presenti nella lista.

- Il modulo **prenotazione.h** definisce la **struttura Prenotazione**, con le informazioni sulle prenotazioni effettuate dai clienti, e le funzioni per gestire le prenotazioni delle lezioni da parte dei clienti. Nel dettaglio, le funzioni principali di prenotazione.h sono:

1) *gestione_prenotazioni*, che permette di visualizzare e cancellare prenotazioni;
2) *aggiungi_prenotazione*, che aggiunge una prenotazione alla lista.

- Il modulo **report.h** infine contiene la definizione della **struttura Report** e le funzioni per generare vari report mensili basati sulle prenotazioni e lezioni. Nel dettaglio, le funzioni principali di report.h sono:

1) *report_mensile*, che produce un report sull'utilizzo delle lezioni in un dato mese.

Operazioni e funzioni del programma

Per facilitare la descrizione e la comprensione delle funzionalità implementate, è possibile suddividere le funzioni in sezioni tematiche in base al loro ruolo specifico all'interno del sistema di gestione della palestra.

- **Le operazioni di aggiunta dati** sono effettuate dalle funzioni *aggiungi_cliente, aggiungi_lezione* e *aggiungi_prenotazione*, che allocano memoria dinamica per i nuovi elementi e li inseriscono all'inizio delle rispettive liste concatenate.
- **Le operazioni di ricerca dati** sono effettuate attraverso le funzioni *cerca_cliente* e *cerca_lezione*, che iterano sulle liste per trovare un elemento con un ID specifico.
- **Le operazioni di gestione dati** sono effettuate attraverso le funzioni *gestione_prenotazioni*, che consente di visualizzare le prenotazioni di un cliente e di eliminarle se richiesto, *gestisci_abbonamento*, che permette di modificare o cancellare abbonamenti associati ai clienti, e *prenota_lezione*, che infine verifica se la lezione ha posti disponibili e aggiorna il conteggio dei partecipanti.
- **Le operazioni di visualizzazione dati** sono effettuate attraverso le funzioni *stampa_clienti* e *visualizza_lezione*, che stampano a video le informazioni degli elementi contenuti nelle liste.
- **L'operazione di report** è effettuata attraverso la funzione *report_mensile*, che raccoglie i dati di prenotazione e lezione per fornire una panoramica sull'utilizzo del servizio in un dato periodo.

Funzioni di supporto

Oltre alle funzioni principali, sono state implementate alcune funzioni di supporto per facilitare l'interazione tra moduli e migliorare l'integrità del sistema.

- La funzione ***cerca_lezione*** permette di individuare una lezione nella lista tramite il suo ID, scorrendo la lista concatenata delle lezioni e confrontando ogni ID fino a trovare una corrispondenza. Se la lezione esiste restituisce un puntatore, altrimenti restituisce NULL.
- La funzione ***trova_id_libero*** serve a trovare un ID univoco e non ancora assegnato per una nuova lezione. Partendo dal valore 1, si incrementa l'ID controllando se è già presente nella lista delle lezioni: non appena trova un ID libero, lo restituisce assicurando così che non ci siano duplicati.
- La funzione ***data_minore_uguale*** confronta due date nel formato "gg/mm/aaaa" e determina se la prima data è minore o uguale alla seconda. Il confronto avviene suddividendo le date in giorno, mese e anno e confrontando in ordine gerarchico.
- La funzione ***stato_abbonamento***, sfruttando la precedente funzione di confronto delle date, determina lo stato di un abbonamento rispetto alla data odierna.

Descrizione del main

Il file main.c costituisce il punto di ingresso del programma e **coordina l'interazione con l'utente tramite un menu testuale.** In particolare, il main gestisce un *ciclo do-while* che **mostra il menu e permette di scegliere tra le diverse funzionalità** a cui è associato un numero specifico da 0 a 9.

Per ogni scelta il main **chiama le funzioni appropriate dei vari moduli**, passando i puntatori alle liste concatenate, e poi dopo ogni operazione il programma attende che l'utente prema Invio per proseguire.

Il main si occupa inoltre di **mantenere aggiornate le liste dinamiche**, salvando le modifiche fatte attraverso le varie operazioni.

Specifica sintattica e semantica

In questa sezione vengono descritti formalmente gli *Abstract Data Types* (ADT) utilizzati nel progetto, insieme alle **operazioni associate a ciascuno di essi**. Per ogni tipo di dato sono riportati:

1) il **nome** del tipo;
2) il **dominio**, cioè l'insieme dei valori che può rappresentare;
3) i **tipi di dato** utilizzati internamente.

Per ogni funzione vengono inoltre indicate le condizioni di utilizzo (*pre-condizioni*), gli effetti attesi al termine dell'esecuzione (*post-condizioni*) e gli eventuali *effetti collaterali*.

Specifica degli ADT

	Nome del tipo	*Dominio*	*Tipi usati*
ADT Cliente	Cliente	Insieme dei clienti iscritti in palestra, dove ciascun cliente è composto da ID univoco, nome e cognome.	int, char[], Cliente*
ADT Lezione	Lezione	Insieme delle lezioni disponibili nella palestra, dove ogni lezione ha ID univoco, nome, giorno, orario, durata, numero di partecipanti e numero massimo di partecipanti.	int, char[], Lezione*
ADT Abbonamento	Abbonamento	Insieme degli abbonamenti associati a un cliente, ognuno formato dal tipo, data di inizio e data di fine.	int, char[], Abbonamento*, Cliente
ADT Prenotazione	Prenotazione	Insieme delle prenotazioni registrate, dove ciascuna prenotazione collega un cliente a una o più lezioni.	int, Prenotazione*, Cliente, Lezione
ADT Report	Report	Insieme dei conteggi delle prenotazioni per ciascuna lezione offerte dalla palestra.	char[], int, Report*, Prenotazione, Lezione

Specifica delle funzioni principali

Funzioni di cliente.h

- *aggiungi_cliente(Cliente* head) → Cliente**

Descrizione: Aggiunge un nuovo cliente in fondo alla lista concatenata dei clienti, assegnando automaticamente al nuovo cliente un ID crescente.

Pre-condizioni: head è un puntatore valido a una lista concatenata di clienti (o NULL se la lista è vuota).

- Gli input nome e cognome devono essere delle stringa di massimo 49 lettere.

Post-condizioni: La lista clienti è aggiornata con un nuovo cliente con ID univoco.

- In caso di errore (memoria o input non valido), la lista rimane invariata.

Effetti collaterali: Alloca dinamicamente memoria per un nuovo nodo Cliente.

- *cerca_cliente(Cliente *head, int id) → Cliente**

Descrizione: Scorre la lista clienti partendo alla ricerca di un cliente con ID uguale a id, se lo trovo restituisce il suo puntatore altrimenti se non lo trova NULL.

Pre-condizioni: head è un puntatore valido a una lista concatenata di clienti (o NULL se la lista è vuota).

- L’ID del cliente deve essere un intero positivo.

Post-condizioni: Nessuna modifica alla lista clienti.

Effetti collaterali: Nessuno, essendo una funzione di sola lettura.

- *stampa_clienti(Cliente* head) → Cliente**

Descrizione: Stampa la lista di tutti i clienti con ID valido e permette all’utente di scegliere se cancellare un cliente specifico indicandone l’ID.

Pre-condizioni: head è un puntatore valido a una lista concatenata di clienti (o NULL).

- L’input per la scelta 's'/'n' e per l’ID da cancellare deve essere valido.

Post-condizioni: La lista clienti viene aggiornata solo se l’utente sceglie di cancellare un cliente già esistente. In questo caso, quel nodo viene rimosso dalla lista e la memoria viene liberata.

Effetti collaterali: Libera memoria in caso di cancellazione di un cliente.

Funzioni di lezione.h

- *aggiungi_lezione(Lezione* head) → Lezione**

Descrizione: Aggiunge una nuova lezione con dati forniti da input e assegna il più piccolo ID libero.

Pre-condizioni: head è un puntatore valido a lista lezioni o NULL (se è vuota).

- Gli input nome, data, orario, durata e massimo partecipanti devono essere validi.

Post-condizioni: La lista viene aggiornata con nuova lezione in coda.

- In caso di errore di input o memoria non disponibile, la lista rimane invariata.

Effetti collaterali: Alloca memoria dinamicamente.

- *visualizza_lezione(Lezione* head) → Lezione**

Descrizione: Visualizza tutte le lezioni con posti disponibili e permette di cancellare una lezione specifica tramite il suo ID associato.

Pre-condizioni: head è un puntatore valido a lista lezioni o NULL

- L’input utente per cancellazione (s/n) e l’ID lezione da cancellare sono validi.

Post-condizioni: La lista è aggiornata se una lezione viene cancellata, altrimenti la lista resta invariata.

Effetti collaterali: Se si elimina una lezione, la memoria viene liberata.

- *prenota_lezione(Lezione* head_lezioni, int id_lezione, int id_cliente) → int*

Descrizione: Prenota un cliente in una lezione specifica se ci sono posti liberi.

Pre-condizioni: head_lezioni è un puntatore valido a lista lezioni.

- id_lezione e id_cliente sono interi positivi.

Post-condizioni: Il contatore partecipanti nella lezione viene incrementato solo se c'è disponibilità. Se la lista è piena o non esiste, non avviene nessuna modifica.

Effetti collaterali: Il campo partecipanti della lezione prenotata viene modificato.

- Vengono stampati messaggi in caso di errore.

Funzioni di abbonamento.h

- *crea_abbonamento(Abbonamento* abbonamenti, Cliente* clienti) → Abbonamento**

Descrizione: Crea un nuovo abbonamento associato a un cliente già esistente.

Pre-condizioni: abbonamenti e clienti sono dei puntatori validi (a liste o NULL).

Gli input utente (ID cliente, tipo abbonamento, date) devono essere validi e coerenti.

Post-condizioni: Il nuovo abbonamento viene aggiunto in testa alla lista, se i dati sono corretti e il cliente è iscritto.

- Se si verifica errore o il cliente non viene trovato, la lista originale rimane la stessa.

Effetti collaterali: Alloca memoria dinamicamente.

- *gestisci_abbonamento(Abbonamento* head, int id_cliente) → Abbonamento**

Descrizione: Visualizza lo stato di un abbonamento per un cliente specifico e permette di cancellarlo.

Pre-condizioni: head lista di abbonamenti è valida o NULL, id_cliente è un intero positivo.

- L'input utente per data odierna e la scelta della cancellazione devono essere validi.

Post-condizioni: La lista si aggiorna se un abbonamento viene cancellato, altrimenti è invariata.

Effetti collaterali: La memoria viene liberata se l'abbonamento viene cancellato.

- *visualizza_abbonamento(Abbonamento* head) → void*

Descrizione: Visualizza tutti gli abbonamenti attivi presenti nella lista, mostrando le relative informazioni.

Pre-condizioni: head è un puntatore valido a una lista di abbonamenti (può essere NULL per lista vuota).

Post-condizioni: Nessuna modifica alla lista o agli abbonamenti.

Effetti collaterali: Stampa a schermo la lista degli abbonamenti.

Funzioni di prenotazione.h

- *gestione_prenotazioni(Prenotazione* head, int id_cliente, Lezione* lezioni_head) → Prenotazione**

Descrizione: Visualizza tutte le prenotazioni di un cliente, mostrando i dettagli della lezione associata e permette di cancellarle.

Pre-condizioni: head lista e lezione_head sono validi o NULL.

- L'id_cliente deve essere un numero intero positivo.

- L'input da tastiera per la scelta della cancellazione deve essere valido.

Post-condizioni: La lista è aggiornata se le prenotazioni sono cancellate, altrimenti resta invariata.

Effetti collaterali: Modifica della lista prenotazioni e dei partecipanti della lezione.

- *aggiungi_prenotazione(Prenotazione* head, int id_cliente, int id_lezione) → Prenotazione**

Descrizione: Crea una nuova prenotazione e la inserisce in testa alla lista.

Pre-condizioni: head lista deve essere valida o NULL.

- id_cliente e id_lezione devono essere interi validi.

Post-condizioni: Il nuovo nodo viene aggiunto in testa alla lista prenotazioni.

Effetti collaterali: Alloca memoria dinamicamente.

Funzioni di report.h

- *report_mensile(Prenotazione prenotazioni, Lezione lezioni, const char* mese_anno) → void*

Descrizione: Genera e stampa un report mensile con il numero di prenotazioni per ogni lezione effettuate nel mese specificato nel formato "mm/aaaa".

Pre-condizioni: prenotazioni è un puntatore valido alla lista delle prenotazioni.

- lezioni è un puntatore valido alla lista delle lezioni (o NULL se lista vuota).

- mese_anno è una stringa valida nel formato "mm/aaaa".

Post-condizioni: Viene stampato un report ordinato per numero di prenotazioni decrescente con il dettaglio delle lezioni nel mese specificato.

- Le liste prenotazioni e lezioni non vengono modificati.

Effetti collaterali: Alloca e dealloca dinamicamente memoria per la lista temporanea report.

Specifica delle funzioni di supporto

- *cerca_lezione(Lezione* head, int id_lezione) → Lezione**

Descrizione: Cerca e restituisce il puntatore alla lezione con l'ID specificato, o NULL se non viene trovata.

Pre-condizioni: head è un puntatore valido (lista o NULL).

- id_lezione è un numero intero positivo.

Post-condizioni: Nessuna modifica alla lista.

Effetti collaterali: Nessuno.

- *trova_id_libero(Lezione* head) → int*

Descrizione: Restituisce il più piccolo ID intero positivo non ancora usato nella lista di lezioni.

Pre-condizioni: head è un puntatore valido (lista di lezioni o NULL).

Post-condizioni: Nessuna modifica alla lista.

Effetti collaterali: Nessuno.

- *data_minore_uguale(const char* data1, const char* data2) → int*

Descrizione: Confronta due date restituendo 1 se data1 ≤ data2, 0 se data1 > data2 e -1 se il formato è errato.

Pre-condizioni: data1 e data2 sono due stringhe nel formato "gg/mm/aaaa".

Post-condizioni: Nessuna modifica.

Effetti collaterali: Nessuno.

- *stato_abbonamento(const char* data_inizio, const char* data_fine, const char* data_oggi) → int*

Descrizione: Determina lo stato dell'abbonamento rispetto alla data odierna tra non attivo (0), attivo (1) e scaduto (-1).

Pre-condizioni: Le date sono stringhe valide nel formato "gg/mm/aaaa".

Post-condizioni: Nessuna modifica.

Effetti collaterali: Nessuno

Razionale dei casi di test

Il *processo di testing* del sistema di gestione delle prenotazioni per la palestra è stato progettato per garantire il corretto funzionamento di tutte le funzionalità fondamentali previste dal progetto. In particolare, i casi di test sono stati scelti per verificare la

completezza e l’affidabilità delle operazioni di prenotazione, gestione abbonamenti, visualizzazione delle disponibilità e generazione dei report.

1) Il primo gruppo di test riguarda la **registrazione delle prenotazioni** e il conseguente aggiornamento delle disponibilità delle lezioni. Questa verifica è essenziale per assicurare che il sistema impedisca prenotazioni oltre la capienza massima e mantenga aggiornato lo stato delle lezioni.
2) Il secondo gruppo è dedicato alla **gestione degli abbonamenti**, con particolare attenzione alla verifica della validità e durata: solo gli utenti con abbonamenti attivi possano effettuare prenotazioni.
3) Infine il sistema deve essere in grado di **generare un report di utilizzo mensile** accurato e completo, che riepiloghi le prenotazioni effettuate e l’affluenza alle lezioni che offre la palestra.