Gestione palestra

Raffaele Sorrentino, Lucio Robustelli

Maggio 2025

1 Introduzione

Il progetto ha come scopo quello di gestire a tutto campo una palestra, dagli abbonamenti alle prenotazioni. Abbiamo preso in considerazione un prototipo di palestra che apre alle ore 8:00 e chiude alle ore 22:00, ha a disposizione a tutti gli orari la sala attrezzi con 40 posti per ogni ora, mentre ha una sala in disparte in cui le attività variano in base all'orario e al giorno e possono essere prenotate da un massimo di 20 clienti per ora. Le attività che abbiamo messo a disposizione del cliente sono:

-Sala attrezzi
-Yoga
-Karate
-Pilates
-Funzionale
-Zumba
-Boxe

2 Scelte ADT

2.1 Cliente

2.1.1 Struttura

La struttura cliente è formata da 5 campi:

-Codice fiscale (identificativo)
-Nome
-Cognome
-Data di nascita
-Settimane di abbonamento

Di questi, i campi fondamentali sono 2: il codice fiscale, che ci permette di identificare ogni cliente e interagire col programma, e poi le settimane di abbonamento. Abbiamo scelto di usare le settimane come parametro invece dei mesi per una gestione più semplice dei dati.

2.1.2 Lista

Per la gestione dei clienti abbiamo utilizzato una lista che viene caricata appena il programma va in esecuzione, in modo da avere a disposizione un elenco con tutti gli abbonati alla palestra. Abbiamo scelto una lista per gli abbonati per vari motivi:

- È una struttura dati dinamica, che ci permette di rimuovere o aggiungere nuovi abbonati in qualsiasi momento.
- Ci permette di accedere a qualsiasi elemento nell'elenco tramite l'identificativo.
- È la struttura dati che ci permette di collegarci alla coda.

2.1.3 Coda

Per gestire l'ordine delle prenotazioni, abbiamo usato una coda, perché rappresenta appieno il comportamento di una lista di attesa. Abbiamo vari vantaggi come:

-La sua logica di funzionamento (FIFO), che simula la realtà.

-Evita problemi di disordine nell'accesso alla prenotazione.

2.2 Lezione

2.2.1 Struttura

La struttura lezione è composta da 4 campi:

- Nome corso
- Orario
- Numero di prenotazioni
- Numero massimo di prenotazioni

In questa struttura si identifica la giusta lezione tramite la combinazione di Nome corso e orario.

2.2.2 Lista

Per memorizzare le lezioni abbiamo utilizzato una lista che viene riempita nel momento in cui si deve effettuare una prenotazione. In base al giorno viene riempita da file diversi definiti nel codice. La scelta è andata sulla lista perché:

- Il numero di lezioni è limitato ed è gestito dalla memoria.
- Ci permette di accedere sequenzialmente a tutte le lezioni.
- Struttura semplice ed accessibile

3 Progettazione

3.1 Main

Il programma principale inizia con l'inizializzazione di tutte le variabili contatore e della lista contenente gli abbonati. Poi entriamo in un'iterazione che permette l'avanzare delle settimane tramite una risposta dell'utente. Ogni volta

che l'iterazione inizia, viene fatto un elenco delle persone che hanno annullato l'abbonamento o che semplicemente gli è scaduto quello in corso. Ogni 4 iterazioni verrà generato il report mensile, contenente le informazioni di maggiore rilievo all'interno di una palestra. Per rendere il tutto più reale, viene generato un numero casuale tra 0 e 10 che sarà il numero di persone in coda. Verrà chiesto per ognuno il codice fiscale; se presente nella lista degli abbonati, lo aggiunge alla coda, altrimenti viene registrato come nuovo cliente, aggiunto alla lista abbonati e poi alla coda.

Una volta caricati in coda tutti i clienti, andiamo a presentare il menù di scelta per le operazioni disponibili. Nel caso in cui venga scelta la prenotazione di una lezione, si va ad aprire un nuovo menù per scegliere la giornata in modo da andare ad aprire il file delle lezioni giusto.
Inoltre nel main sono state usate due funzioni:
-void init(int * contcliente, int * contnewabbonamento, int * contrinnovaabbonamento, int * contannullaabbonamento, int * contprenotalezione,int contLezioni[])
Serve ad inizializzare tutte le variabili contatore.
-int maxLezioni(int contLezioni[])
Serve ad ottenere l'indice della lezione con più prenotazioni effettuate.

3.2 Makefile

Il nostro makefile presenta le seguenti funzioni:
-**run**: Compila ed esegue il programma principale e il file utils.
-**run_TC1**:Compila ed esegue il programma che si occupa del primo caso di test e il file utils.
-**run_TC2**:Compila ed esegue il programma che si occupa del secondo caso di test e il file utils.
-**run_TC3**:Compila ed esegue il programma che si occupa del terzo caso di test e il file utils.c.
-**clean**: Rimuove i file oggetto e eseguibili del main e di utils.

3.3 Cliente

La struttura cliente presenta funzioni di base (get/set) per ogni campo della struttura ed inoltre una funzione clienteNULL, in cui verifichiamo se il dato passato è uguale alla sua versione nulla definita.

3.3.1 Lista cliente

Con la lista implementiamo funzioni di base come:
-void newListaC()
-int emptyListaC(listacliente)
-listacliente consListaC(listacliente,cliente)
-int sizeListaC(listacliente)
-void freeListaC(listacliente)

-cliente newAbbonamento(listacliente)
A queste aggiungiamo invece funzioni specifiche per il progetto sviluppato:

-listacliente LoadInizio(listacliente l)

- Precondizione: la lista deve essere possibilmente vuota, inoltre deve esistere un file definito come "NOMEFILE".
- Postcondizione: restituisce una lista contenente tutti gli abbonati con abbonamento maggiore di 0 letti dal file.
- Funzionamento: Apriamo il file degli abbonati in modalità lettura, leggiamo riga per riga ed estraggo i dati. Controllo la durata degli abbonamenti per vedere se tutti hanno ancora accesso alla lista abbonati e se viene trovato qualcuno con l abbonamento scaduto andiamo a chiamare una funzione che aggiorna anche il file. Chiudiamo il file e restituiamo la lista caricata.

-cliente trovaCliente(listacliente l,char cod[])

- Precondizione: la lista deve contenere dei clienti, o eventualmente essere vuota. Cod è una stringa che contiene il codice fiscale da ricercare.
- Postcondizione: se esiste un cliente nella lista con quel codice fiscale lo restituisce. Se invece non viene trovato, restituiamo NULLCLIENTE.
- Funzionamento: Controlliamo se la lista è vuota, poi viene creato un puntatore alla testa della lista in modo da visitare l'intera lista, se trova il cliente con il codice fiscale passato lo restituisce altrimenti passa la versione NULL di cliente.

-listacliente rimuoviAbbonamenti(listacliente l)

- Precondizione: la lista deve contenere dei clienti.
- Postcondizione: Tutti i clienti con abbonamento minore o uguale a 0 vengono rimosso.
- Funzionamento: Usiamo due puntatori, corrente per scorrere la lista, precedente invece punta all'elemento prima per collegare la lista nel caso in cui ci sia una rimozione nel mezzo. Restituiamo la lista aggiornata.

-void updateFileAbb(listacliente l)

- Precondizione: la lista **DEVE** contenere tutti gli abbonati validi, deve essere definita una macro "NOMEFILE".
- Postcondizione:Il file passato verrà riempito da capo con gli elementi della lista.
- Funzionamento: Apriamo il file in scrittura, scorriamo la lista tramite un puntatore e ogni nodo viene scritto sul file.

-listaCliente updateSettimanale(listaCliente l)

- Precondizione:La lista l è correttamente riempita, anche se può essere vuota.
- Postcondizione: ad ogni cliente viene decrementato di 1 l'abbonamento.
- Funzionamento: Controllo se la lista è vuota, se non lo è scorro la lista e decremento il valore dell'abbonamento. Restituisco la lista con i valori aggiornati.

-int RinnovaAbbonamento(listacliente l,cliente c, int rinn

- Precondizione:la lista deve essere caricata, cliente c non deve essere NULL, rinn deve essere maggiore di 0.
- Postcondizione: Incrementa le settimane di abbonamento del cliente passato di rinn settimane.
- Funzionamento: Usiamo un puntatore per scorrere la lista, cerchiamo il cliente nella lista, se lo troviamo sommiamo rinn alle sue settimane di abbonamento e ritorniamo 1 perche è andata a buon fine l'operazione. Se non trova il cliente o altro ritorna 0.

-int annullaAbbonamento(listaCliente l,cliente c)

- Precondizione: la lista può contenere 0 o più nodi, la struttura cliente passata deve essere valida.
- Postcondizione: al cliente passato viene settato il campo abbonamento a 0, restituisce 1 se è andata a buon fine la funzione, 0 altrimenti.
- Funzionamento: Scorriamo la lista fino a quando non troviamo un incrocio tra nodo e cliente, se lo trova setta l'abbonamento a 0 e ritorna 1. Se non lo trova ritorna 0.

3.3.2 Coda cliente

Con la coda abbiamo implementato solo funzioni basilari:
-codaCliente newCoda()
-int emptyCoda(codaCliente)
-int enqueue(codaCliente,cliente)
-cliente dequeue(codaCliente)
-void freeCoda(codaCliente)

3.4 Lezione

La struttura lezione presenta le funzioni get/set per ogni campo, e vengono definite le macro per assegnare i file .txt in base al giorno.

3.4.1 Lista lezione

Come per cliente andiamo ad elencare le funzioni base:
-listaLezioni newListaL()
-int emptyListaL(listaLezioni)
-listaLezioni consListaL(listaLezioni,lezione)
-void freeListaL(listaLezioni)

Ora invece andiamo a spiegare le funzioni create da noi:

-listalezioni loadListaL(listalezioni l,char file[])

- Precondizione: file deve essere il nome corretto del file da aprire e avere il formato corretto dei dati mentre la lista l deve essere inizializzata.
- Postcondizione: La lista verrà riempita con tutte le lezioni scritte nel file.
- Funzionamento: Apriamo il file in lettura, leggiamo riga per riga il file e inseriamo in coda alla lista ogni lezione. Chiudiamo il file e ritorno la lista caricata.

-int prenotaLezione(listaLezioni l,char file[],int contLezioni[])

- Precondizione: la lista deve essere caricata con le lezioni disponibili, file deve contenere il nome del file da aprire, contLezioni è un array di 7 interi usato per contare le prenotazioni di ogni lezione.
- Postcondizione:il numero di prenotati della lezione scelta viene incrementato e scritto sul file, il contatore relativo a quella specifica lezione viene aumentato, restituisco 1 se la prenotazione è stata effettuata.
- Funzionamento: Stampiamo a schermo le lezioni disponibili, e attraverso un menù numerato facciamo scegliere la lezione e poi facciamo inserire anche l'orario. Cerchiamo nella lista la combinazione inserita dall'utente, verifichiamo la disponibilità se conferma la scelta, aggiorniamo il numero di prenotati, contLezione, salviamo sul file i cambiamenti e ritorniamo 1. Se la lezione non viene trovata o è piena ritorniamo 0.

-void mostraLezioniDisponibili(listaLezioni l)

- Precondizione:la lista deve contenere almeno una lezione.
- Postcondizione: vengono stampate a schermo tutte le lezioni.
- Funzionamento: Usiamo un puntatore per scorrere la lista, e stamperemo ogni nodo fino alla fine della lista.

-int disponibilitaLezione(lezione lez)

- Precondizione: lez deve essere correttamente riempito e quindi lez.pren deve essere minore di lez.maxpren.

- Postcondizione: la funzione restituisce il numero di posti disponibili.
- Funzionamento: Ritorniamo la differenza tra il massimo di posti prenotabili e quelli prenotati.

-void salvaListaLezioni(listaLezioni l,char file[])

- Precondizione: la lista l deve essere valida e caricata, file deve essere il nome del file da aprire.
- Postcondizione: tutte le lezioni presenti nella lista vengono scritte nel file.
- Funzionamento: apriamo il file in scrittura, usiamo un puntatore per scorrere la lista e ogni nodo viene scritto sul file nel formato predefinito, e infine chiudiamo il file.

-void resetFile()

- Precondizione: le macro sui file devono essere corrette.
- Postcondizione: a tutti i file dal lunedi al sabato verrà impostato il campo pren a 0.
- Funzionamento: Definisco un array contenente tutti i nomi dei file, creo e carico una lista con tutte le lezioni, setto il numero di prenotati a 0 e salvo su file la lista.

4 Testing

4.1 Test Case 1

Verifica della corretta registrazione delle prenotazioni e dell'aggiornamento delle disponibilità
Per mostrare il corretto funzionamento, abbiamo creato e caricato la lista con un giorno a caso, fatto visualizzare a schermo il contenuto, compreso il numero di prenotati. Infine, effettuiamo una prenotazione e ristampiamo il contenuto della lista per mostrare che effettivamente tiene conto delle nuove prenotazioni.

4.2 Test Case 2

Test della gestione degli abbonamenti e della verifica della validità.
Per verificare il comportamento del programma, abbiamo deciso di testarlo con la creazione e il riempimento della lista abbonati.Poi abbiamo aggiunto un cliente con 0 settimane di abbonamento, visualizziamo l'intera lista e richiamiamo la funzione rimuoviAbbonamenti che si occupa proprio della verifica di validità per tutti i clienti. Infine visualizziamo nuovamente la lista e il nome del cliente precedentemente inserito, non comparirà più.

4.3 Test Case 3

Verifica che il report generato contenga informazioni corrette e complete sulle prenotazioni.
Per simulare il funzionamento del report mensile, inizializziamo tutte le variabili che ci serviranno, aggiungo un cliente a caso agli abbonati, effettuo 4 prenotazioni, e infine stampo il numero di clienti prima e dopo il test, e quale tipo di lezione ha avuto più prenotazioni.