**Report**

# **File**

## main.c

**void menu()**

Specifica sintattica:

void menu();

Specifica semantica:

* Input
  + Nessun parametro in ingresso.
* Output
  + Nessuno.
* Pre Condizione
  + Nessuna
* Post Condizione
  + Viene visualizzato a schermo un menu con le opzioni disponibili
* Side Effect
  + Output a schermo

**int main();**

Specifica sintattica:

int main();

Specifica semantica:

* Input
  + Nessuno
* Output
  + Restituisce 0 a fine esecuzione per indicare la terminazione corretta del programma
* Pre Condizione
  + Nessuna
* Post Condizione
  + I file vengono letti (se esistono) e le strutture dati vengono popolate.
  + L’utente può interagire con il sistema tramite il menu per:
    - Inserire un cliente.
    - Visualizzare clienti ordinati.
    - Verificare la validità di un abbonamento.
    - Prenotare una lezione.
    - Visualizzare le prenotazioni.
    - Inserire nuove lezioni.salvato
* Side Effect
  + Lettura/scrittura su file JSON.
  + Allocazione e deallocazione di memoria dinamica.
  + Stampa su schermo.
  + Interazione con l’utente via scanf, fgets e printf.

## prenotazioni.c

**void visualizza\_prenotazione(Prenotazione prenotazione)**

Specifica sintattica:

void visualizza\_prenotazione(Prenotazione prenotazione);

Specifica semantica:

* Input
  + Una struttura Prenotazione, che contiene informazioni su:
    - ID della prenotazione,
    - Cliente (nome e cognome),
    - Lezione (nome e data).
* Output
  + Nessuno
* Pre Condizione
  + La struttura Prenotazione passata deve essere correttamente inizializzata, inclusi i campi partecipante, lezione e lezione.data.
* Post Condizione
  + Sul terminale verranno mostrati:
    - ID della prenotazione,
    - Nome e cognome del cliente,
    - Nome della lezione,
    - Data e ora della lezione in formato gg/mm/aaaa--hh:mm. salvato
* Side Effect
  + Output a schermo tramite printf

## prenotazioni.h

## abbonamenti.c

**Nodo\* crea\_nodo(Cliente c)**

Specifica sintattica:

Nodo\* crea\_nodo(Cliente c);

Specifica semantica:

* Input
  + Cliente c – la struttura con i dati del nuovo cliente che dovremmo aggiungere nel nodo che creeremo
* Output
  + Puntatore al nodo appena creato con all’ interno i dati del cliente c ed i puntatori ai nodi sx e dx
* Pre Condizione
  + Nessuna
* Post Condizione
  + Viene restituito un puntatore ad un nuovo nodo che viene allocato dinamicamente con i puntatori a dx e sx vuoti ed il cliente nuovo salvato
* Side Effect
  + Viene occupato uno spazio di memoria con la chiamata del malloc quindi dovremmo ricordarci di effettuare una free successivamente

**Nodo\* inserisci\_cliente(Nodo\* radice, Cliente c)**

Specifica sintattica:

Nodo\* inserisci\_cliente(Nodo\* radice, Cliente c)

Specifica semantica:

* Input
  + Nodo\* radice: il puntatore alla radice dell’ albero che avviamo creato(può essere anche NULL se si tratta del primo inserimento)
  + Cliente nuovo: i dati del cliente che andremo ad inserire nell’ albero
* Output
  + Puntatore alla radice aggiornata dell’ albero che include anche i dati del nuovo cliente
* Pre Condizione
  + L’albero deve essere binario e si deve basare sull’ id per la ricerca
* Post Condizione
  + Se l’ id non esiste già viene inserito correttamente rispettando l’ ordine dell’ albero
  + Se è gia presente, non ci sono modifiche è viene stampato un errore
* Side Effect
  + Potrebbero essere salvati dei duplicati

**Int abbonamento\_valido(Cliente c);**

Specifica sintattica:

Int abbonamento\_valido(Cliente c)

Specifica semantica:

* Input
  + Cliente c – la struttura con i dati del cliente
* Output
  + Restituisce 1 se l’abbonamento è ancora valido, 0 altrimenti
* Pre Condizione
  + Il tempo di attivazione deve essere stato inizializzato correttamente
* Post Condizione
  + La funzione calcola il tempo trascorso dalla mdata d’ attivazione alla data odierna e lo confronta con la durata dell’abbonamento (confrontando i giorni)
* Side Effect
  + Nessuno la funzione legge solo i dati non li modifica

**Void stampa\_clienti\_ordinati(Nodo\* radice)**

Specifica sintattica:

void stampa\_clienti\_ordinati(Nodo\* radice)

Specifica semantica:

* Input
  + Puntatore alla radice dell’ albero binario dove sono salvati i clienti
* Output
  + Nessuno la funzione è void
* Pre Condizione
  + L’albero deve essere già ordinato
* Post Condizione
  + Tutti i clienti ssalvati nell’ albero vengono stampati in ordine crescente di codice fiscale, con i dati riguardanti i loro abbonamenti tramite una funzione ricorsiva
* Side Effect
  + Viene stampato un output a schermo

**Void libera\_clienti(Nodo\* radice)**

Specifica sintattica:

void libera\_clienti(Nodo\* radice)

Specifica semantica:

* Input
  + Puntatore alla radice dell’ albero binario dove sono salvati i clienti
* Output
  + Nessuno la funzione è void
* Pre Condizione
  + L’albero deve essere stato gia allocato correttamente
* Post Condizione
  + Tutta la memoria allocata per l’ albero dei clienti viene liberata
* Side Effect
  + Dopo l’ esecuzione il puntatore a radice e tutti i nodi non potreanno più essere usati

**ricerca\_cliente**

Ricerca un cliente nell'albero binario di ricerca utilizzando il codice fiscale come chiave.

Specifica sintattica:

NodoAlbero\* ricerca\_cliente(NodoAlbero\* radice, const char\* codice\_fiscale)

Specifica semantica:

* Input
  + Puntatore alla radice dell’albero binario.
  + stringa rappresentante il codice fiscale del cliente da cercare
* Output
  + Restituisce il puntatore al nodo contenente il cliente cercato, oppure NULL se non trovato
* Pre Condizione
  + Nessuna
* Post Condizione
  + Se il cliente è presente nell’albero, viene restituito il nodo corrispondente; altrimenti, NULL
* Side Effect
  + Nessuno.

**void ricerca\_e\_verifica\_cliente(NodoAlbero\* radice, const char\* codice\_fiscale)**

Cerca un cliente nell'albero binario e ne stampa i dettagli, compresa la validità dell'abbonamento

Specifica sintattica:

void ricerca\_e\_verifica\_cliente(NodoAlbero\* radice, const char\* codice\_fiscale)

Specifica semantica:

* Input
  + puntatore alla radice dell’albero binario.
  + stringa contenente il codice fiscale del cliente da cercare.
* Output
  + Nessun valore restituito; stampa i dati del cliente su stdout.
* Pre Condizione
  + radice deve puntare a un albero valido (può essere anche NULL se vuoto).
* Post Condizione
  + Se il cliente è trovato, stampa i dettagli; altrimenti, stampa un messaggio di errore.
* Side Effect
  + Stampa a video.

**trova\_minimo**

Trova il nodo con il valore minimo (in base al codice fiscale) in un sottoalbero binario

Specifica sintattica:

NodoAlbero\* trova\_minimo(NodoAlbero\* nodo)

Specifica semantica:

* Input
  + puntatore alla radice del sottoalbero.
* Output
  + Restituisce il puntatore al nodo con il valore minimo (più a sinistra).
* Pre Condizione
  + nodo non deve essere NULL
* Post Condizione
  + Viene restituito il nodo con codice fiscale minimo nel sottoalbero
* Side Effect
  + Nessuno

**elimina\_cliente**

Elimina un cliente dall'albero binario di ricerca in base al codice fiscale

Specifica sintattica:

NodoAlbero\* elimina\_cliente(NodoAlbero\* radice, const char\* codice\_fiscale)

Specifica semantica:

* Input
  + puntatore alla radice dell’albero binario
  + codice fiscale del cliente da eliminare
* Output
  + Restituisce il nuovo puntatore alla radice dell’albero dopo l’eliminazione
* Pre Condizione
  + L’albero deve essere strutturato come un albero binario di ricerca
* Post Condizione
  + Se esiste un nodo con il codice fiscale specificato, viene eliminato mantenendo la struttura dell’albero
* Side Effect
  + Deallocazione della memoria per il nodo rimosso

## abbonamenti.h

## lezioni.c

**void inizializza\_catalogo(Catalogo\_Lezioni\* catalogo);**

Specifica sintattica:

void inizializza\_catalogo(Catalogo\_Lezioni\* catalogo);

Specifica semantica:

* Input
  + Puntatore alla struttura Catalogo\_Lezioni.
* Output
  + Nessuno
* Pre Condizione
  + catalogo deve essere un puntatore valido
* Post Condizione
  + La struttura viene inizializzata con:
    - capacità = CAPACITÀ\_INIZIALE (es. 4),
    - numero\_lezioni = 0,
    - memoria allocata per l’array di lezioni.
* Side Effect
  + Allocazione dinamica di memoria

**void aggiungi\_lezione(Catalogo\_Lezioni\* catalogo, const Lezione nuova\_lezione);**

Specifica sintattica:

void aggiungi\_lezione(Catalogo\_Lezioni\* catalogo, const Lezione nuova\_lezione);

Specifica semantica:

* Input
  + Puntatore a Catalogo\_Lezioni, struttura Lezione da inserire.
* Output
  + Nessuno
* Pre Condizione
  + catalogo deve essere un puntatore valido e inizializzato
* Post Condizione
  + La lezione viene aggiunta in coda all’array. Se necessario, la memoria viene raddoppiata
* Side Effect
  + Possibile riallocazione di memoria tramite realloc

**void elimina\_lezione(Catalogo\_Lezioni\* catalogo, const Lezione lezione\_da\_eliminare);**

Specifica sintattica:

void elimina\_lezione(Catalogo\_Lezioni\* catalogo, const Lezione lezione\_da\_eliminare);

Specifica semantica:

* Input
  + Puntatore al catalogo e lezione da eliminare (identificata da ID e data)
* Output
  + Nessuno
* Pre Condizione
  + catalogo deve essere inizializzato e deve contenere almeno una lezione
* Post Condizione
  + La lezione viene rimossa, gli elementi successivi vengono shiftati, e se la capacità supera di 4 volte il numero di lezioni, viene ridotta a metà.
* Side Effect
  + Possibile riallocazione di memoria tramite realloc.
  + Stampa di errori se necessario

**void elimina\_catalogo(Catalogo\_Lezioni\* catalogo);**

Specifica sintattica:

void elimina\_catalogo(Catalogo\_Lezioni\* catalogo);

Specifica semantica:

* Input
  + Puntatore alla struttura Catalogo\_Lezioni
* Output
  + Nessuno
* Pre Condizione
  + Il campo lezione deve essere stato allocato
* Post Condizione
  + Memoria deallocata, valori azzerati
* Side Effect
  + free() sulla memoria allocata.

**void mostra\_lezioni(const Catalogo\_Lezioni catalogo);**

Specifica sintattica:

void mostra\_lezioni(const Catalogo\_Lezioni catalogo);

Specifica semantica:

* Input
  + Copia del catalogo
* Output
  + Stampa su schermo
* Pre Condizione
  + Il catalogo deve contenere lezioni
* Post Condizione
  + Ogni lezione viene stampata a video (ID, nome, posti, data/ora). Se vuoto, viene mostrato un errore
* Side Effect
  + Output su stdout con printf

**const Lezione\* trova\_lezione(const Catalogo\_Lezioni\* catalogo, const unsigned int id);**

Specifica sintattica:

const Lezione\* trova\_lezione(const Catalogo\_Lezioni\* catalogo, const unsigned int id);

Specifica semantica:

* Input
  + Puntatore al catalogo e ID della lezione da cercare
* Output
  + Puntatore alla lezione trovata o NULL se non esiste
* Pre Condizione
  + Catalogo valido e inizializzato
* Post Condizione
  + Puntatore alla lezione desiderata (se presente)
* Side Effect
  + Nessuno

**bool conflitto\_orario\_lezione(const Catalogo\_Lezioni\* catalogo, time\_t orario)**

Specifica sintattica:

bool conflitto\_orario\_lezione(const Catalogo\_Lezioni\* catalogo, time\_t orario)

Specifica semantica:

* Input
  + Puntatore al catalogo, orario da verificare.
* Output
  + true se esiste già una lezione con lo stesso orario, altrimenti false
* Pre Condizione
  + Catalogo valido e inizializzato
* Post Condizione
  + Valutazione della presenza di conflitto temporale
* Side Effect
  + Nessuno

## lezioni.h

## utilities.c

**Orario\_Tm\* converti\_orario\_in\_struct\_tm(time\_t orario)**

Specifica sintattica:

Orario\_Tm\* converti\_orario\_in\_struct\_tm(time\_t orario)

Specifica semantica:

* Input
  + valore temporale da convertire
* Output
  + Puntatore a una struttura Orario\_Tm corrispondente all’orario specificato
* Pre Condizione
  + orario deve essere un valore valido ed accettabile
* Post Condizione
  + Restituisce la struttura compilata correttamente per visualizzazione
* Side Effect
  + La struttura restituita punta a memoria gestita da localtime, quindi verrá deallocata automaticamente

**time\_t converti\_orario\_in\_time\_t(Orario\_Tm\* tm\_orario, int giorno, int mese, int anno, int ora, int minuto)**

Specifica sintattica:

time\_t converti\_orario\_in\_time\_t(Orario\_Tm\* tm\_orario, int giorno, int mese, int anno, int ora, int minuto)

Specifica semantica:

* Input
  + struttura da compilare
  + valore temporale da convertire
* Output
  + Valore corrispondente alla data e ora indicata
* Pre Condizione
  + orario deve essere un valore valido ed accettabile
* Post Condizione
  + Restituisce un null o altri valori coerenti
* Side Effect
  + La struttura tm\_orario viene sovrascritta

**unsigned int genera\_id\_univoco(const char filepath)**

Specifica sintattica:

unsigned int genera\_id\_univoco(const char filepath)

Specifica semantica:

* Input
  + percorso al file JSON in cui cercare ID esistenti
* Output
  + ID univoco non presente nel file
* Pre Condizione
  + filepath valido; se il file non esiste, viene considerato vuoto
* Post Condizione
  + Restituito un ID numerico unico rispetto agli altri ID già presenti nel JSON
* Side Effect
  + Allocazione dinamica di memoria temporanea (poi liberata); apertura e lettura del file JSON

**bool file\_vuoto(char filepath)**

Specifica sintattica:

bool file\_vuoto(char\* filepath)

Specifica semantica:

* Input
  + percorso al file da controllare
* Output
  + estituisce true se il file è vuoto o non apribile, false altrimenti
* Pre Condizione
  + filepath valido (anche se il file non esiste)
* Post Condizione
  + Verifica effettuata correttamente
* Side Effect
  + Apertura e chiusura del file specificato

## utilities.h

## Persistenza\_dati.c

**Catalogo\_Lezioni carica\_catalogo\_da\_file(const char\* filepath)**

Specifica sintattica:

Catalogo\_Lezioni carica\_catalogo\_da\_file(const char\* filepath)

Specifica semantica:

* Input
  + percorso del file JSON contenente il catalogo delle lezioni
* Output
  + Ritorna una struttura ‘Catalogo\_Lezioni’ popolata con i dati letti dal file.
* Pre Condizione
  + Il parametro ‘filepath’ deve essere un puntatore non nullo e il file deve essere accessibile.
* Post Condizione
  + Se il file esiste e contiene un array JSON valido, il catalogo viene correttamente popolato con tutte le lezioni. In caso contrario, viene restituito un catalogo vuoto.
* Side Effect
  + Allocazione dinamica temporanea per la lettura del file, liberata prima del termine.
  + Eventuali errori di lettura vengono stampati su stderr.

**void carica\_prenotazioni\_da\_file(const char\* filepath, Lista\_Prenotazioni\* lista);**

Specifica sintattica:

void carica\_prenotazioni\_da\_file(const char\* filepath, Lista\_Prenotazioni\* lista);

Specifica semantica:

* Input
  + percorso del file JSON da cui leggere i dati
  + puntatore a una lista inizializzata delle prenotazioni.
* Output
  + Nessun valore di ritorno. La lista passata viene popolata con i dati letti dal file.
* Pre Condizione
  + ‘filepath’ e ‘lista’ devono essere puntatori validi e non nulli. lista deve essere inizializzata a NULL.
* Post Condizione
  + La lista viene popolata con tutte le prenotazioni presenti nel file, se il file è valido.
* Side Effect
  + Allocazioni dinamiche temporanee per la lettura e parsing del file JSON. Eventuali errori di lettura stampati su stderr.

**void carica\_abbonamenti\_da\_file(const char\* filepath, NodoAlbero\*\* radice\_BST)**

Specifica sintattica:

Specifica semantica:

* Input
  + percorso del file JSON contenente gli abbonamenti.
  + doppio puntatore alla radice dell’albero BST da aggiornare.
* Output
  + Nessun valore di ritorno. L’albero viene aggiornato in loco.
* Pre Condizione
  + ‘filepath’ e ‘radice\_BST’ devono essere validi. L’albero puntato da ‘radice\_BST’ deve essere inizializzato a NULL.
* Post Condizione
  + Tutti gli abbonamenti presenti nel file vengono inseriti nell’albero binario.
* Side Effect
  + Analisi del file JSON con allocazioni dinamiche temporanee.
  + Output su stderr in caso di errore.

**bool salva\_lezioni\_su\_file(const Catalogo\_Lezioni\* catalogo, const char\* filepath);**

Specifica sintattica:

bool salva\_lezioni\_su\_file(const Catalogo\_Lezioni\* catalogo, const char\* filepath);

Specifica semantica:

* Input
  + puntatore alla struttura contenente le lezioni da salvare.
  + percorso del file JSON di destinazione.
* Output
  + true se il salvataggio è avvenuto correttamente
  + false in caso contrario.
* Pre Condizione
  + catalogo e filepath devono essere puntatori validi.
* Post Condizione
  + Il contenuto del catalogo viene salvato in formato JSON nel file specificato.
* Side Effect
  + Scrittura su file e allocazione temporanea di stringhe JSON.
  + Stampa di errori su stderr.

**bool salva\_prenotazioni\_su\_file(const Lista\_Prenotazioni lista, const char\* filepath);**

Specifica sintattica:

bool salva\_prenotazioni\_su\_file(const Lista\_Prenotazioni lista, const char\* filepath);

Specifica semantica:

* Input
  + puntatore alla struttura contenente le lezioni da salvare.
  + percorso del file JSON di destinazione.
* Output
  + true se il salvataggio è avvenuto correttamente
  + false in caso contrario.
* Pre Condizione
  + catalogo e filepath devono essere puntatori validi.
* Post Condizione
  + Il contenuto del catalogo viene salvato in formato JSON nel file specificato.
* Side Effect
  + Scrittura su file e allocazione temporanea di stringhe JSON.
  + Stampa di errori su stderr.

**bool salva\_abbonamenti\_su\_file(const NodoAlbero\* nodo, const char\* filepath);**

Specifica sintattica:

bool salva\_abbonamenti\_su\_file(const NodoAlbero\* nodo, const char\* filepath);

Specifica semantica:

* Input
  + radice dell’albero binario bilanciato contenente i clienti abbonati.
  + percorso del file JSON di destinazione.
* Output
  + true se il salvataggio è avvenuto correttamente
  + false in caso contrario.
* Pre Condizione
  + nodo e filepath devono essere puntatori validi.
* Post Condizione
  + I dati dell’albero dei clienti vengono salvati in formato JSON nel file specificato.
* Side Effect
  + Scrittura su file e allocazione temporanea di stringhe JSON.
  + Stampa di errori su stderr.

**void elimina\_elem\_da\_persistenza(const char\* tipo, const unsigned int id);**

Specifica sintattica:

void elimina\_elem\_da\_persistenza(const char\* tipo, const unsigned int id);

Specifica semantica:

* Input
  + tipo di elemento da rimuovere ("cliente", "lezione", "prenotazione").
  + identificativo numerico dell’elemento da eliminare.
* Output
  + Nessun valore di ritorno.
* Pre Condizione
  + ‘tipo’ deve essere una stringa valida.
  + ‘id’ deve corrispondere a un valore valido.
* Post Condizione
  + Se esiste un elemento nel file corrispondente al tipo e all’ID forniti, esso viene rimosso.
* Side Effect
  + Lettura, modifica e riscrittura del file JSON specificato.
  + Stampa di messaggi informativi o di errore su stdout/stderr.

**.static void aggiung\_clienti\_array\_json(const NodoAlbero\* nodo, cJSON\* array\_json);**

Specifica sintattica:

static void aggiung\_clienti\_array\_json(const NodoAlbero\* nodo, cJSON\* array\_json);

Specifica semantica:

* Input
  + puntatore alla radice o a un sottoalbero dell’albero binario dei clienti.
  + array JSON su cui serializzare i clienti.
* Output
  + Nessun valore di ritorno.
* Pre Condizione
  + array\_json deve essere un oggetto JSON valido.
* Post Condizione
  + Tutti i clienti dell’albero vengono aggiunti all’array JSON in ordine simmetrico (in-order).
* Side Effect
  + Allocazione dinamica di oggetti JSON per ogni cliente.

## Persistenza\_dati.h

## Lista\_prenotazioni.h

## Lista\_Prenotazioni.c

**Lista\_Prenotazioni crea\_lista\_prenotazioni();**

Cosa fa:

Crea una nuova lista di prenotazioni vuota.

Specifica sintattica:

Lista\_Prenotazioni crea\_lista\_prenotazioni();

Specifica semantica:

* Input
  + Nessuno
* Output
  + Puntatore a una lista vuota.
* Pre Condizione
  + Nessuna.
* Post Condizione
  + Viene restituito un puntatore a una lista vuota.
* Side Effect
  + Nessuno.

**void aggiungi\_prenotazione(Lista\_Prenotazioni\* lista, const Prenotazione prenotazione);**

Aggiunge una prenotazione alla lista, verificando disponibilità e conflitti.

Specifica sintattica:

void aggiungi\_prenotazione(Lista\_Prenotazioni\* lista, const Prenotazione prenotazione);

Specifica semantica:

* Input
  + Puntatore alla lista delle prenotazioni, struttura prenotazione da aggiungere.
* Output
  + Nessuno.
* Pre Condizione
  + lista deve essere un puntatore valido.
* Post Condizione
  + La prenotazione viene aggiunta in fondo alla lista se non ci sono conflitti.
* Side Effect
  + Allocazione dinamica di memoria e scrittura su stderr in caso di errore o conflitto.

**.bool disdici\_prenotazione(Lista\_Prenotazioni\* lista, const Lezione\* 1 lezione);**

Rimuove una prenotazione da una lista, cercandola per ID e data lezione.

Specifica sintattica:

bool disdici\_prenotazione(Lista\_Prenotazioni\* lista, const Lezione\* lezione);

Specifica semantica:

* Input
  + Puntatore alla lista delle prenotazioni, puntatore alla lezione da disdire.
* Output
  + true se la prenotazione è stata trovata e rimossa
  + false altrimenti
* Pre Condizione
  + lista e lezione devono essere puntatori validi.
* Post Condizione
  + La prenotazione corrispondente alla lezione viene rimossa se trovata
* Side Effect
  + Scrittura su stderr e deallocazione della memoria del nodo rimosso

**void visualizza\_prenotazioni(const Lista\_Prenotazioni lista);**

Stampa a video tutte le prenotazioni presenti nella lista.

Specifica sintattica:

void visualizza\_prenotazioni(const Lista\_Prenotazioni lista);

Specifica semantica:

* Input
  + Lista delle prenotazioni.
* Output
  + Nessuno.
* Pre Condizione
  + Nessuna.
* Post Condizione
  + Le prenotazioni vengono stampate a video.
* Side Effect
  + Scrittura su stdout o stderr.

**void libera\_lista\_prenotazioni(Lista\_Prenotazioni\* lista);**

Libera tutta la memoria allocata per la lista delle prenotazioni.

Specifica sintattica:

void libera\_lista\_prenotazioni(Lista\_Prenotazioni\* lista);

Specifica semantica:

* Input
  + Puntatore alla lista delle prenotazioni.
* Output
  + Nessuno.
* Pre Condizione
  + lista deve essere un puntatore valido.
* Post Condizione
  + Tutti i nodi della lista vengono deallocati.
* Side Effect
  + Deallocazione di memoria.

**int conteggia\_prenotazioni(const Lista\_Prenotazioni\* lista, const Lezione\* lezione);**

Conta il numero di prenotazioni per una lezione specifica.

Specifica sintattica:

int conteggia\_prenotazioni(const Lista\_Prenotazioni\* lista, const Lezione\* lezione);

Specifica semantica:

* Input
  + Puntatore alla lista delle prenotazioni e puntatore alla lezione da cercare.
* Output
  + Numero intero di prenotazioni trovate, -1 in caso di errore
* Pre Condizione
  + lista e lezione devono essere puntatori validi
* Post Condizione
  + Viene restituito il numero di prenotazioni per la lezione specificata
* Side Effect
  + Scrittura su stderr in caso di errore

**bool lezione\_piena(const Lista\_Prenotazioni lista, const Lezione lezione\_da\_analizzare);**

Verifica se una lezione ha raggiunto la capienza massima

Specifica sintattica:

bool lezione\_piena(const Lista\_Prenotazioni lista, const Lezione lezione\_da\_analizzare);

Specifica semantica:

* Input
  + Lista delle prenotazioni e struttura lezione
* Output
  + true se la lezione è piena
  + false altrimenti
* Pre Condizione
  + lista deve essere valido
* Post Condizione
  + Viene restituito il risultato del controllo di capienza
* Side Effect
  + Nessuno (usa conteggia\_prenotazioni, che può stampare su stderr)

**bool controllo\_conflitto\_orario(const Lista\_Prenotazioni lista, const Lezione lezione, const Cliente partecipante);**

Controlla se un partecipante ha già una prenotazione in conflitto orario

Specifica sintattica:

Specifica semantica:

* Input
  + Lista delle prenotazioni, struttura lezione, struttura cliente
* Output
  + true se esiste un conflitto
  + false altrimenti
* Pre Condizione
  + lista, lezione e partecipante devono essere validi
* Post Condizione
  + Viene restituito se esiste un conflitto di orario
* Side Effect
  + Nessuno

**.Prenotazione\* trova\_prenotazione(const Lista\_Prenotazioni lista, const Lezione lezione, const Cliente partecipante);**

Cerca una prenotazione tramite data, ID, nome e cognome

Specifica sintattica:

Prenotazione\* trova\_prenotazione(const Lista\_Prenotazioni lista, const Lezione lezione, const Cliente partecipante);

Specifica semantica:

* Input
  + Lista delle prenotazioni, struttura lezione, struttura cliente
* Output
  + Puntatore alla prenotazione trovata o NULL se non trovata
* Pre Condizione
  + lista, lezione e partecipante devono essere validi
* Post Condizione
  + Viene restituito un puntatore alla prenotazione, se presente
* Side Effect
  + Scrittura su stderr in caso di errore
  + Prenotazione non trovata