**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

## 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