Specifiche Sintattiche e Semantiche
Progetto PSD

Progetto: Gestione Prenotazioni Di Una Palestra
“Traccia 2”
Autore: Giovanni Pierno, Tedesco Simone
Data: Maggio 2025

File: util.c

Specifiche Sintattiche

Include di:

- **stdio.h**
- **stdlib.h**
- **util.h**
- **data.h**
- **orario.h**

Funzioni definite:

- **int soloLettere(char *)**
- **void pulisciSchermo()**
- **void pulisciSchermoCliente(const char[50], const char[50])**
- **void messaggio_errore()**
- **void schermata_arrivederci()**

Specifiche Semantiche

soloLettere

- **Sintassi: int soloLettere(char *str)**
- **Semantica: Verifica che la stringa contenga solo lettere alfabetiche (maiuscole e minuscole), spazi e apostrofi.**
- **Precondizioni: str deve essere una stringa terminata da \0.**
- **Postcondizioni: Restituisce 1 se la stringa contiene solo caratteri validi, 0 altrimenti.**

pulisciSchermo

- **Sintassi: void pulisciSchermo()**
- **Semantica: Pulisce lo schermo usando il comando appropriato per il sistema operativo, quindi stampa la data odierna in colore blu.**
- **Precondizioni: Funzione oggi() restituisce una data valida e stampaData() può stampare la data.**
- **Postcondizioni: Schermo pulito e data odierna visualizzata.**

pulisciSchermoCliente

- **Sintassi: void pulisciSchermoCliente(const char nome[50], const char cognome[50])**
- **Semantica: Pulisce lo schermo come pulisciSchermo() e stampa la data odierna seguita da un messaggio di benvenuto personalizzato con nome e cognome in colore verde.**
- **Precondizioni: Parametri nome e cognome devono essere stringhe valide.**
- **Postcondizioni: Schermo pulito, data odierna e messaggio di benvenuto visualizzati.**

messaggio_errore

- **Sintassi: void messaggio_errore()**
- **Semantica: Stampa un messaggio di errore in rosso indicando che l'input dell'utente non è valido e invitando a riprovare.**
- **Precondizioni: Nessuna.**
- **Postcondizioni: Messaggio di errore mostrato a video.**

schermata_arrivederci

- **Sintassi: void schermata_arrivederci()**
- **Semantica: Stampa un messaggio di ringraziamento e arrivederci racchiuso in una cornice gialla.**
- **Precondizioni: Nessuna.**
- **Postcondizioni: Messaggio di saluto mostrato a video.**

Esempio di codice pulisciSchermo():

```
void pulisciSchermo() {           //comando che pulisce lo schermo in caso per rendere l'interfaccia piÃ¹ agevole
    #ifdef _WIN32
        system("cls");  // Comando per Windows
    #else
        system("clear");  // Comando per Linux e macOS
    #endif
    printf("%sDATA ODIERNA:  %s", BLU, RESET);
    stampaData(oggi());  /// Chiama la funzione 'oggi()' per ottenere la data odierna e la passa alla funzione 'stampaData()', che la visualizza a schermo.
    printf("\n");         /// PoichÃ© questa riga Ã¨ inserita nella funzione 'pulisciSchermo()',
                          ///  la data verrÃ  stampata ogni volta che il programma pulisce lo schermo, aggiornando cosÃ¬ l'utente sulla data corrente.
}
```

File main_program.c:

Specifiche sintattiche

- **Include di:**
- **stdio.h**
- **stdlib.h**
- **string.h**
- **menu.h**
- **util.h**
- **list.h**

Variabili globali esterne:

- **extern list* abbonati;**
  Lista globale degli abbonati utilizzata nel programma.

Funzioni definite:

- **int main_program(void)**
- **int main(void)**

Specifiche Semantiche

main_program()

- **Sintassi: int main_program(void)**
- **Semantica: Funzione principale che gestisce il ciclo di menu dell'applicazione. Visualizza il menu iniziale e permette all'utente di scegliere tra uscita (0), menu gestore (1) e menu cliente (2). Gestisce input da tastiera in modo sicuro, con controllo degli errori e pulizia dello schermo tra le operazioni.**
  Alla scelta di uscita libera la lista degli abbonati e termina il programma.
- **Precondizioni: La lista globale abbonati deve essere correttamente inizializzata (anche NULL se vuota).**
- **Postcondizioni: Esegue il menu selezionato, o termina liberando la memoria. Lo schermo è pulito ad ogni operazione.**

Comportamento:

- **Ciclo infinito fino a selezione uscita (0).**
- **Input gestito tramite fgets e sscanf con validazione.**
- **In caso di errore di input, mostra messaggio di errore e ripropone il menu.**
- **Chiama le funzioni menu_iniziale(), menu_gestore(), menu_cliente() per la gestione delle scelte.**
- **Alla terminazione libera la lista con freeList(abbonati).**

main()

- **Sintassi: int main(void)**
- **Semantica:**
  Funzione di avvio del programma, che richiama main_program() per gestire l'intera esecuzione.
- **Precondizioni: Nessuna.**
- **Postcondizioni:**
  Restituisce il valore di ritorno di main_program()
- **Postcondizioni: Il programma rimane in esecuzione fino alla scelta di uscita da parte dell'utente.**

File data.c

Specifiche sintattiche

Include di:

- **data.h**
- **stdio.h**
- **stdlib.h**
- **time.h**

Funzioni definite:

- **int giorniNelMese(int, int)**
- **int dataValida(Data)**
- **Data* newData(void)**
- **Data* creaData(int, int, int)**
- **void setGiorno(Data*, int)**
- **void setMese(Data*, int)**
- **void setAnno(Data*, int)**
- **int getGiorno(Data*)**
- **int getMese(Data*)**
- **int getAnno(Data*)**
- **void distruggiData(Data*)**
- **Data* leggiData()**
- **void stampaData(Data*)**
- **int confrontaDate(Data*, Data*)**
- **Data* aggiungiMesi(Data*, int)**
- **Data* oggi()**
- **Data* sottraiGiorni(Data*, int)**

Specifiche Semantiche

giorniNelMese(int mese, int anno)

- **Sintassi:**
  static int giorniNelMese(int mese, int anno)
- **Semantica:**
  Restituisce il numero di giorni presenti nel mese specificato, considerando gli anni bisestili per febbraio.
- **Precondizioni:**
  mese compreso tra 1 e 12, anno valido.

- **Postcondizioni:**
  Restituisce un intero con il numero di giorni del mese.

dataValida(Data d)

- **Sintassi:**
  int dataValida(Data d)
- **Semantica:**
  Verifica se la data passata è valida controllando anno, mese e giorno.
- **Precondizioni:**
  Nessuna specifica.
- **Postcondizioni:**
  Restituisce 1 se la data è valida, 0 altrimenti.

newData(void)

- **Sintassi:**
  Data* newData(void)
- **Semantica:**
  Alloca dinamicamente memoria per una nuova struttura Data.
- **Precondizioni:**
  Disponibilità di memoria.
- **Postcondizioni:**
  Restituisce un puntatore a Data allocata dinamicamente.

creaData(int giorno, int mese, int anno)

- **Sintassi:**
  Data* creaData(int giorno, int mese, int anno)
- **Semantica:**
  Crea e inizializza una struttura Data con i valori forniti.
- **Precondizioni:**
  Valori validi per giorno, mese e anno.
- **Postcondizioni:**
  Restituisce un puntatore a Data inizializzata, o NULL in caso di errore di allocazione.

setGiorno(Data* d, int giorno)

- **Sintassi:**
  void setGiorno(Data* d, int giorno)
- **Semantica:**
  Imposta il giorno della data.
- **Precondizioni:**
  Puntatore a Data valido.
- **Postcondizioni:**
  Giorno aggiornato.

setMese(Data* d, int mese)

- **Sintassi:**
  void setMese(Data* d, int mese)
- **Semantica:**
  Imposta il mese della data.
- **Precondizioni:**
  Puntatore a Data valido.
- **Postcondizioni:**
  Mese aggiornato.

setAnno(Data* d, int anno)

- **Sintassi:**
  void setAnno(Data* d, int anno)
- **Semantica:**
  Imposta l'anno della data.
- **Precondizioni:**
  Puntatore a Data valido.
- **Postcondizioni:**
  Anno aggiornato.

getGiorno(Data* d)

- **Sintassi:**
  int getGiorno(Data* d)
- **Semantica:**
  Restituisce il giorno della data.

- **Precondizioni:**
  Puntatore a Data valido.
- **Postcondizioni:**
  Giorno restituito.

getMese(Data* d)

- **Sintassi:**
  int getMese(Data* d)
- **Semantica:**
  Restituisce il mese della data.
- **Precondizioni:**
  Puntatore a Data valido.
- **Postcondizioni:**
  Mese restituito.

getAnno(Data* d)

- **Sintassi:**
  int getAnno(Data* d)
- **Semantica:**
  Restituisce l'anno della data.
- **Precondizioni:**
  Puntatore a Data valido.
- **Postcondizioni:**
  Anno restituito.

distruggiData(Data* d)

- **Sintassi:**
  void distruggiData(Data* d)
- **Semantica:**
  Libera la memoria allocata per la struttura Data.
- **Precondizioni:**
  Puntatore a Data allocata dinamicamente.

- **Postcondizioni:**
  Memoria liberata.

leggiData()

- **Sintassi:**
  Data* leggiData()
- **Semantica:**
  Legge una data da input standard, ripetendo la richiesta finché la data inserita non è valida.
- **Precondizioni:**
  Nessuna.
- **Postcondizioni:**
  Restituisce un puntatore a Data valida inserita dall'utente.

stampaData(Data* d)

- **Sintassi:**
  void stampaData(Data* d)
- **Semantica:**
  Stampa la data nel formato gg/mm/aaaa.
- **Precondizioni:**
  Puntatore a Data valido.
- **Postcondizioni:**
  Data stampata su standard output.

confrontaDate(Data* d1, Data* d2)

- **Sintassi:**
  int confrontaDate(Data* d1, Data* d2)
- **Semantica:**
  Confronta due date: restituisce -1 se d1 < d2, 1 se d1 > d2, 0 se uguali.
- **Precondizioni:**
  Puntatori a Data validi.
- **Postcondizioni:**
  Valore di confronto restituito.

aggiungiMesi(Data* d, int mesi)

- **Sintassi:**
  Data* aggiungiMesi(Data* d, int mesi)
- **Semantica:**
  Aggiunge un numero di mesi alla data fornita, gestendo overflow/underflow e correggendo il giorno se necessario. Restituisce una nuova Data modificata.
- **Precondizioni:**
  Puntatore a Data valido, mesi intero.
- **Postcondizioni:**
  Restituisce un nuovo puntatore a Data con la data modificata, o NULL in caso di errore.

oggi()

- **Sintassi:**
  Data* oggi()
- **Semantica:**
  Restituisce una nuova Data corrispondente alla data odierna.
- **Precondizioni:**
  Disponibilità di memoria.
- **Postcondizioni:**
  Puntatore a Data con data odierna restituito.

sottraiGiorni(Data* data, int giorni)

- **Sintassi:**
  Data* sottraiGiorni(Data* data, int giorni)
- **Semantica:**
  Sottrae un numero di giorni dalla data fornita, restituendo la data risultante. Modifica la data passata direttamente.
- **Precondizioni:**
  Puntatore a Data valido, giorni non negativo.
- **Postcondizioni:**
  Data modificata e restituita.

File:menu.c

Include di:

- **stdio.h**
- **stdlib.h**
- **string.h**
- **ctype.h**
- **menu.h**
- **gestione_Abbonamenti.h**
- **util.h**
- **data.h**
- **list.h**
- **gestione_prenotazioni.h**
- **lezione.h**
- **gestione_lezioni.h**
- **file.h**

Variabili globali esterne:

- **extern listP* lista_prenotazioni;**
- **extern list* abbonati;**
- **extern listL* lezioni;**

Funzioni definite:

- **void menu_iniziale(void)**
- **void menu_gestore(void)**
- **list* controlloCliente(void)**
- **void menu_cliente(void)**

Specifiche Semantiche

menu_iniziale()

- **Sintassi: void menu_iniziale(void)**
- **Semantica:**
  Visualizza il menu principale dell'applicazione, proponendo le opzioni di accesso al menu gestore, al menu cliente o di uscita dal programma.
- **Precondizioni: Nessuna.**
- **Postcondizioni: Stampa a video il menu iniziale e attende la selezione da parte dell'utente.**

menu_gestore()

- **Sintassi: void menu_gestore(void)**
- **Semantica:**
  Gestisce il ciclo del menu del gestore con le opzioni per:

1. **Gestione abbonamenti**
2. **Gestione prenotazioni e report**
3. **Gestione lezioni**
4. **Ritorno al menu principale**
   La funzione gestisce in modo robusto l'input utente, pulisce lo schermo ad ogni iterazione, mostra messaggi di errore in caso di input non valido e invoca le funzioni di gestione specifiche per ogni scelta.

- **Precondizioni: Funzioni di gestione (es. gestione_abbonamenti(), gestione_prenotazioni()) devono essere definite e operative.**
- **Postcondizioni: L'utente può navigare tra i sottomenu gestore finché non sceglie di tornare al menu principale.**

controlloCliente()

- **Sintassi: list* controlloCliente(void)**
- **Semantica:**
  Richiede all'utente l'inserimento di nome, cognome e codice abbonamento per verificare l'esistenza di un cliente nella lista abbonati.
  Carica la lista degli abbonati da file se necessario.
  Permette di uscire inserendo "exit" o "0".
  Valida i dati inseriti, mostra messaggi di errore in caso di input errato o abbonamento non trovato e consente ripetizioni fino a un inserimento valido o uscita.
- **Precondizioni: Lista abbonati eventualmente inizializzata o caricata da file.**
- **Postcondizioni:**
  Restituisce il puntatore al nodo della lista corrispondente al cliente trovato, oppure NULL in caso di uscita o mancato riconoscimento.

menu_cliente()

- **Sintassi: void menu_cliente(void)**
- **Semantica:**
  Gestisce il menu cliente, consentendo al cliente autenticato di:

1. **Controllare la validità del proprio abbonamento**
2. **Prenotare una lezione**
3. **Visualizzare report mensile delle lezioni**
4. **Visualizzare le proprie prenotazioni**
5. **Disdire una prenotazione**
6. **Tornare al menu principale**
   La funzione carica da file, se necessario, le liste di prenotazioni, abbonati e lezioni, gestisce l'autenticazione tramite controlloCliente() e poi mostra un menu

ricorsivo per interagire con le funzionalità dedicate al cliente, con validazione dell'input e gestione errori.

- **Precondizioni: Liste lista_prenotazioni, abbonati e lezioni correttamente inizializzate o caricate da file.**
- **Postcondizioni: Consente al cliente di interagire con il sistema tramite le opzioni disponibili o di tornare al menu principale.**

File: abbonamento.c

Include di:

• stdlib.h
• string.h
• stdio.h
• list.h
• abbonamento.h
• file.h
• data.h
• util.h

Variabili globali esterne:

• abbonamento* vuoto — Puntatore a un abbonamento "vuoto" predefinito
• list* abbonati — Lista degli abbonamenti registrati

Funzioni definite:

• abbonamento* creaAbbonamento(void)
• void distruggiAbbonamento(abbonamento* a)
• void init_vuoto(void)
• int getCodiceAbbonamento(const abbonamento* a)
• const char* getNome(const abbonamento* a)
• const char* getCognome(const abbonamento* a)
• Data* getDataInizio(const abbonamento* a)
• int getDurata(const abbonamento* a)
• void setCodiceAbbonamento(abbonamento* a, int codice)
• void setNome(abbonamento* a, const char* nome)
• void setCognome(abbonamento* a, const char* cognome)
• void setDataInizio(abbonamento* a, Data* data)
• void setDurata(abbonamento* a, int durata)
• int getMaxCodiceAbbonamento(list* abbonati)
• abbonamento* getAbbonamentoByID(list* abbonati, int codice)

• int abbonamento_isScaduto(abbonamento* ab)
• void stampaDettagliAbbonamento(abbonamento* a)

Specifiche semantiche

creaAbbonamento()

• Sintassi: abbonamento* creaAbbonamento(void)
• Semantica:
Crea un nuovo oggetto abbonamento allocando memoria dinamica e inizializzandolo con valori di default: codice 0, nome e cognome stringhe vuote, data NULL, durata 0 mesi.
• Precondizioni: La memoria deve essere allocabile.
• Postcondizioni: Restituisce un puntatore a un abbonamento inizializzato o NULL in caso di errore di allocazione.

distruggiAbbonamento()

• Sintassi: void distruggiAbbonamento(abbonamento* a)
• Semantica:
Libera la memoria occupata da un abbonamento e dalla sua data associata se presenti.
• Precondizioni: a deve essere un puntatore valido o NULL.
• Postcondizioni: La memoria occupata da a e dalla data associata è liberata; se a è NULL la funzione non fa nulla.

init_vuoto()

• Sintassi: void init_vuoto(void)
• Semantica:
Inizializza la variabile globale vuoto con un abbonamento "vuoto" contenente valori predefiniti (es. data 01/01/2000).
• Precondizioni: Memoria disponibile per allocazione.
• Postcondizioni: vuoto punta a un abbonamento valido inizializzato.

getCodiceAbbonamento()

• Sintassi: int getCodiceAbbonamento(const abbonamento* a)
• Semantica:
Restituisce il valore del codice identificativo dell'abbonamento a.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Restituisce il codice di abbonamento.

getNome()

• Sintassi: const char* getNome(const abbonamento* a)
• Semantica:
Restituisce una stringa contenente il nome del cliente associato all'abbonamento a.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Restituisce puntatore alla stringa interna nome.

getCognome()
• Sintassi: const char* getCognome(const abbonamento* a)
• Semantica:
Restituisce una stringa contenente il cognome del cliente associato all'abbonamento a.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Restituisce puntatore alla stringa interna cognome.

getDataInizio()
• Sintassi: Data* getDataInizio(const abbonamento* a)
• Semantica:
Restituisce il puntatore alla data di inizio dell'abbonamento a.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Restituisce puntatore a data_inizio.

getDurata()
• Sintassi: int getDurata(const abbonamento* a)
• Semantica:
Restituisce la durata in mesi dell'abbonamento a.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Restituisce valore intero della durata.

setCodiceAbbonamento()
• Sintassi: void setCodiceAbbonamento(abbonamento* a, int codice)
• Semantica:
Imposta il codice identificativo dell'abbonamento a al valore codice.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Il campo codice_abbonamento di a è aggiornato.

setNome()
• Sintassi: void setNome(abbonamento* a, const char* nome)
• Semantica:
Copia in modo sicuro la stringa nome nel campo nome dell'abbonamento a.
• Precondizioni: a e nome non devono essere NULL.
• Postcondizioni: Il campo nome è aggiornato con la nuova stringa, terminata correttamente.

setCognome()
• Sintassi: void setCognome(abbonamento* a, const char* cognome)
• Semantica:
Copia in modo sicuro la stringa cognome nel campo cognome dell'abbonamento a.
• Precondizioni: a e cognome non devono essere NULL.

• Postcondizioni: Il campo cognome è aggiornato con la nuova stringa, terminata correttamente.

setDataInizio()

• Sintassi: void setDataInizio(abbonamento* a, Data* data)
• Semantica:
Imposta il puntatore data_inizio dell'abbonamento a con il puntatore data.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Il campo data_inizio è aggiornato.

setDurata()

• Sintassi: void setDurata(abbonamento* a, int durata)
• Semantica:
Imposta la durata in mesi dell'abbonamento a.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Il campo durata è aggiornato.

getMaxCodiceAbbonamento()

• Sintassi: int getMaxCodiceAbbonamento(list* abbonati)
• Semantica:
Restituisce il valore massimo del codice identificativo presente nella lista di abbonamenti abbonati.
• Precondizioni: abbonati deve essere una lista valida (può essere vuota).
• Postcondizioni: Ritorna il massimo codice trovato o 0 se lista vuota.

getAbbonamentoByID()

• Sintassi: abbonamento* getAbbonamentoByID(list* abbonati, int codice)
• Semantica:
Cerca e restituisce il puntatore all'abbonamento nella lista abbonati con codice uguale a codice.
• Precondizioni: abbonati deve essere una lista valida.
• Postcondizioni: Restituisce puntatore all'abbonamento trovato o NULL se non esiste.

abbonamento_isScaduto()

• Sintassi: int abbonamento_isScaduto(abbonamento* ab)
• Semantica:
Determina se un abbonamento è scaduto confrontando la data odierna con la data di scadenza (data_inizio + durata).
• Precondizioni: ab può essere NULL.
• Postcondizioni: Restituisce 1 se scaduto o NULL, 0 se valido e non scaduto.

stampaDettagliAbbonamento()

• Sintassi: void stampaDettagliAbbonamento(abbonamento* a)
• Semantica:
Stampa a video tutte le informazioni dettagliate dell'abbonamento a, incluse data

inizio, durata e data scadenza.
• Precondizioni: a non deve essere NULL.
• Postcondizioni: Dati dell'abbonamento stampati in modo leggibile.

File: gestione_abbonamenti.c

1. Specifiche Sintattiche

- **Include di: stdio.h, stdlib.h, string.h, gestione_abbonamenti.h, list.h, abbonamento.h, file.h data.h util.h**

Definizioni di macro per colori:

Verde: Conferma, successo, abbonamento valido
Rosso: Errore, avvisi, abbonamento scaduto
Giallo: input, attenzione, prompt
Viola: Intestazioni di menu
Blu: Titoli secondari o sezioni
Reset: Ripristina il colore normale

Funzioni definite:

- **Controllo_abbonamento(abbonamento *a)**
- **Gestione_abbonamenti()**
- **Aggiungi_abbonamento(list** abbonati)**
- **eliminaAbbonamento(list* abbonati)**
- **eliminaPerID(list* abbonati)**
- **eliminaPerCognome(list* abbonati)**
- **ricercaAbbonamento(list* abbonati)**
- **ricercaEModificaAbbonamento(list* abbonati)**
- **modificaAbbonamento(list* nodo, list* abbonati)**

2. Specifiche Semantiche

aggiungiAbbonamento()

- **Sintassi: void aggiungiAbbonamento(ListaAbbonamenti *lista);**
- **Semantica: Aggiunge un nuovo abbonamento alla lista, richiedendo i dati all'utente.**
- **Precondizioni: La lista lista deve essere inizializzata correttamente.**
- **Postcondizioni: Un nuovo abbonamento è stato allocato e inserito in lista.**

modificaAbbonamento()

- **Sintassi: void modificaAbbonamento(ListaAbbonamenti *lista);**
- **Semantica: Modifica un abbonamento esistente nella lista, identificato da un codice fornito dall'utente.**
- **Precondizioni: La lista lista deve contenere almeno un abbonamento.**
- **Postcondizioni: L'abbonamento selezionato è stato modificato con i nuovi dati forniti.**

eliminaAbbonamento(), eliminaPerId(), eliminaPerNomeCognome()

- **Sintassi: list eliminaAbbonamento(*list abbonati), list* eliminaPerID(list*), list* eliminaPerNomeCognome(list*)**
- **Semantica: Gestiscono l'eliminazione di Abbonamenti tramite ID o nome e cognome**
- **Precondizioni: La lista non deve essere vuota**
- **Postcondizioni: Viene Eliminato l'elemento selezionato**

Gestione_abbonamenti()

- **Sintassi: void gestione_abbonamenti()**
- **Semantica: Mostra un menu interattivo per la gestione degli abbonamenti**
- **Precondizioni: La lista lista deve contenere almeno un abbonamento.**
- **Postcondizioni: La lista può essere modificata, visualizzata e salvata**

ricercaAbbonamento()

- **Sintassi: list* ricercaAbbonamento(list*)**
- **Semantica: Ricerca un abbonamento per Id o nome/cognome**
- **Precondizioni: La lista di abbonati deve esistere**
- **Postcondizioni: Restituisce l'abbonamento trovato o NULL**

ricercaEModificaAbbonamento()

- **Sintassi: list* ricercaEModificaAbbonamento(list*)**
- **Semantica: Ricerca e modifica un abbonamento esistente**
- **Precondizioni: La lista non deve essere vuota**
- **Postcondizioni: L'elemento selezionato viene aggiornato**

Controllo_abbonamento()

- **Sintassi: void controllo_abbonamento(abbonamento*)**
- **Semantica: mostra lo stato dell' abbonamento (valido o scaduto)**
- **Precondizioni: Oggetto abbonamento valido**

- **Postcondizioni: Mostra il messaggio colorato corrispondente allo stato dell'abbonamento**

File: gestione_lezioni.c

1. Specifiche Sintattiche

- **Include di: stdio.h, stdlib.h, string.h, gestione_lezioni.h, lezione.h, list_lezioni.h, data.h, file.h, orario.h, util.h;**

Funzioni definite:

- **aggiungiLezione()**
- **gestione_Lezioni()**
- **lezione_printByDate()**
- **lezione_printByDisciplina()**
- **lezioe_printByID()**
- **lezione_checkByID()**
- **visualizzaLezioni()**

2. Specifiche Semantiche

aggiungiLezione()

- **Sintassi: void aggiungiLezione(ListaLezioni *lezioni);**
- **Semantica: Aggiunge una nuova lezione alla lista, richiedendo i dati all'utente.**
- **Precondizioni: La lista deve essere valida.**
- **Postcondizioni: Una nuova lezione è aggiunta alla lista.**

Gestione_lezioni()

- **Sintassi: void aggiungi_lezione()**
- **Semantica: Gestisce il menu delle lezioni per l'amministratore**
- **Precondizioni: La lista deve essere inizializzata correttamente.**
- **Postcondizioni: L'utente può aggiungere, eliminare o visualizzare lezioni**

Lezione_printByDate()

- **Sintassi: void lezione_printByDate(ListaLezioni Lezioni)**
- **Semantica: La funzione mostra a video tutte le lezioni ordinate per data**
- **Precondizioni: La lista deve contenere almeno una lezione**
- **Postcondizioni: Le lezioni sono stampate in ordine cronologico**

Lezione_printByDisciplina()

- **Sintassi: void lezione_printByDisciplina(ListaLezioni Lezioni)**

- **Semantica: La funzione visualizza le lezioni raggrupandole per tipo di disciplina**
- **Precondizioni: La lista deve essere inizializzata e contenere i dati**
- **Postcondizioni: Le lezioni vengono visualizzate per tipologia**

Lezione_printByID()

- **Sintassi: void lezione_printByID(ListaLezioni Lezioni)**
- **Semantica: La funzione consente di cercare e visualizzare una specifica lezione inserendo il suo codice identificativo**
- **Precondizioni: L'ID immesso deve corrispondere ad una lezione presente**
- **Postcondizioni: viene visualizzata la lezione corrispondente all'ID**

Lezione_checkByID()

- **Sintassi: void lezione_checkById(ListaLezioni Lezioni, int id)**
- **Semantica: La funzione controlla se una lezione con codice id è presente nella lista e ne restituisce il puntatore**
- **Precondizioni: La lista deve contenere almeno una lezione**
- **Postcondizioni: Restituisce il puntatore alla lezione se esistente, altrimenti NULL**

visualizzaLezioni()

- **Sintassi: void visualizzaLezioni(ListaLezioni Lezioni)**
- **Semantica: Fornisce un menu per la visualizzazione delle lezioni presenti nella lista, filtrabili per data, disciplina o codice identificativo**
- **Precondizioni: La lista delle lezioni deve contenere elementi**
- **Postcondizioni: Le lezioni vengono visualizzate secondo il criterio scelto**

Include di:

• stdlib.h
• string.h
• stdio.h
• lezione.h
• data.h
• orario.h
• util.h
• prenotazioni.h
• list_Lezioni.h

Funzioni definite:

• creaLezione()
• distruggiLezione()
• getCodiceLezione()
• getDisciplinaLezione()

- **getNomeLezione()**
- **getPostiMax()**
- **getPostiOccupati()**
- **getDataLezione()**
- **getOrarioLezione()**
- **getDurataLezione()**
- **setCodiceLezione()**
- **setDisciplinaLezione()**
- **setNomeLezione()**
- **setPostiMax()**
- **addPostiOccupati()**
- **removePostiOccupati()**
- **setDataLezione()**
- **setOrarioInizio()**
- **setDurataLezione()**
- **stampaDettagliLezione()**
- **getLezioneByID()**
- **getMaxCodiceLezione()**

Specifiche semantiche

- ### creaLezione()
  • Sintassi: lezione* creaLezione(void)
  • Semantica: Alloca dinamicamente una nuova struttura lezione, inizializzandola con valori di default (codice 0, stringhe vuote, data 1/1/2000, orario mezzanotte, posti occupati 0, durata 0, posti max 0).
  • Precondizioni: Memoria allocabile.
  • Postcondizioni: Restituisce un puntatore valido o NULL se l'allocazione fallisce.

- ### distruggiLezione()
  • Sintassi: void distruggiLezione(lezione* l)
  • Semantica: Libera la memoria occupata dalla struttura lezione (ma non dai puntatori a data e orario).
  • Precondizioni: Il puntatore deve essere valido (non NULL).
  • Postcondizioni: Memoria liberata, puntatore non più utilizzabile.

- ### getCodiceLezione()
  • Sintassi: int getCodiceLezione(const lezione* l)
  • Semantica: Restituisce il codice identificativo della lezione.

- ### getDisciplinaLezione()
  • Sintassi: const char* getDisciplinaLezione(const lezione* l)
  • Semantica: Restituisce la stringa disciplina associata alla lezione.

- ### getNomeLezione()
  • Sintassi: const char* getNomeLezione(const lezione* l)
  • Semantica: Restituisce la stringa nome associata alla lezione.

- **getPostiMax()**
  - **Sintassi: short int getPostiMax(const lezione* l)**
  - **Semantica: Restituisce il numero massimo di posti disponibili per la lezione.**
- **getPostiOccupati()**
  - **Sintassi: int getPostiOccupati(const lezione* l)**
  - **Semantica: Restituisce il numero di posti attualmente occupati.**

- **getDataLezione()**
  - **Sintassi: Data* getDataLezione(const lezione* l)**
  - **Semantica: Restituisce il puntatore alla data associata alla lezione.**

- **getOrarioLezione()**
  - **Sintassi: Orario* getOrarioLezione(const lezione* l)**
  - **Semantica: Restituisce il puntatore all'orario di inizio della lezione.**

- **getDurataLezione()**
  - **Sintassi: short int getDurataLezione(const lezione* l)**
  - **Semantica: Restituisce la durata della lezione in minuti.**

- **setCodiceLezione()**
  - **Sintassi: void setCodiceLezione(lezione* l, int codice)**
  - **Semantica: Imposta il codice identificativo della lezione.**
  - **Precondizioni: Lezione valida, codice positivo.**
  - **Postcondizioni: Codice aggiornato.**

- **setDisciplinaLezione()**
  - **Sintassi: void setDisciplinaLezione(lezione* l, const char* disciplina)**
  - **Semantica: Copia la stringa disciplina nella struttura, proteggendo da overflow.**
  - **Precondizioni: Puntatori validi.**
  - **Postcondizioni: Disciplina aggiornata.**

- **setNomeLezione()**
  - **Sintassi: void setNomeLezione(lezione* l, const char* nome)**
  - **Semantica: Copia la stringa nome nella struttura.**
  - **Precondizioni: Puntatori validi.**
  - **Postcondizioni: Nome aggiornato.**

- **setPostiMax()**
  - **Sintassi: void setPostiMax(lezione* l, short int max)**
  - **Semantica: Imposta il numero massimo di posti disponibili.**

- **addPostiOccupati()**
  - **Sintassi: void addPostiOccupati(lezione* l)**
  - **Semantica: Incrementa di 1 il numero di posti occupati.**
  - **Precondizioni: Numero posti occupati < posti max.**
  - **Postcondizioni: Posti occupati incrementati di 1.**

- **removePostiOccupati()**
  - **Sintassi: void removePostiOccupati(lezione* l)**
  - **Semantica: Decrementa di 1 il numero di posti occupati se > 0.**

• **Precondizioni: Posti occupati > 0.**
• **Postcondizioni: Posti occupati diminuiti di 1 o mantenuti a 0.**

- **setDataLezione()**
  • **Sintassi: void setDataLezione(lezione* l, Data* data)**
  • **Semantica: Imposta la data della lezione.**

- **setOrarioInizio()**
  • **Sintassi: void setOrarioInizio(lezione* l, Orario* o)**
  • **Semantica: Imposta l'orario di inizio della lezione.**
  • **Precondizioni: Puntatori validi.**
  • **Postcondizioni: Orario aggiornato.**

- **setDurataLezione()**
  • **Sintassi: void setDurataLezione(lezione* l, short int durata)**
  • **Semantica: Imposta la durata della lezione in minuti.**

- **stampaDettagliLezione()**
  • **Sintassi: void stampaDettagliLezione(lezione* l)**
  • **Semantica: Stampa a video tutti i dettagli della lezione, incluso calcolo posti rimanenti e orario di fine lezione.**
  • **Precondizioni: Lezione valida.**
  • **Postcondizioni: Output sul terminale.**

- **getLezioneByID()**
  • **Sintassi: lezione* getLezioneByID(listL* lezioni, int codice)**
  • **Semantica: Scorre la lista delle lezioni per trovare quella con codice uguale a quello dato, e la restituisce.**
  • **Precondizioni: Lista inizializzata e codice presente nella lista.**
  • **Postcondizioni: Puntatore alla lezione trovata o NULL.**

- **getMaxCodiceLezione()**
  • **Sintassi: int getMaxCodiceLezione(listL* lezioni)**
  • **Semantica: Scorre la lista delle lezioni e restituisce il codice massimo presente.**
  • **Precondizioni: Lista inizializzata.**
  • **Postcondizioni: Valore massimo del codice restituito, 0 se lista vuota.**

Specifiche Semantiche di list.c

Include di:

- **stdio.h**
- **stdlib.h**
- **string.h**
- **abbonamento.h**
- **data.h**
- **list.h**

- **util.h**

Funzioni definite:

- **list* newList(void)**
- **int emptyList(list* l)**
- **list* consList(abbonamento* val, list* l)**
- **abbonamento* getFirst(list* l)**
- **list* tailList(list* l)**
- **list* getNext(list* nodo)**
- **void setNext(list* nodo, list* next)**
- **abbonamento* getValue(list* l)**
- **void setValue(list* l, abbonamento* a)**
- **void printList(list* l)**
- **void freeList(list* l)**
- **void visualizzaListaOrdinata(list* abbonati)**
- **void ordinaPerNome(list** head)**
- **void ordinaPerDataScadenza(list** abbonati)**
- **void ordinaPerDataInizio(list** abbonati)**
- **void ordinaPerCodiceDecrescente(list** head)**
- **void ordinaPerCodiceCrescente(list** head)**

2. Specifiche Semantiche

newList()

- **Sintassi: list* newList(void)**
- **Semantica: Crea e restituisce una nuova lista vuota (NULL).**
- **Precondizioni: Nessuna.**
- **Postcondizioni: Lista vuota restituita.**

*emptyList(list l)**

- **Sintassi: int emptyList(list* l)**
- **Semantica: Verifica se la lista l è vuota (NULL).**
- **Precondizioni: Lista l valida (anche NULL).**
- **Postcondizioni: Ritorna 1 se lista vuota, 0 altrimenti.**

consList(abbonamento val, list l)***

- **Sintassi: list* consList(abbonamento* val, list* l)**
- **Semantica: Crea un nuovo nodo con valore val e lo inserisce in testa alla lista l.**
- **Precondizioni: val non NULL, lista l inizializzata o NULL.**
- **Postcondizioni: Restituisce la nuova testa della lista con il nodo inserito.**

*getFirst(list l)**

- **Sintassi: abbonamento* getFirst(list* l)**
- **Semantica: Restituisce il valore del primo nodo della lista l.**
- **Precondizioni: Lista l non vuota.**
- **Postcondizioni: Ritorna puntatore a abbonamento contenuto nel primo nodo, oppure NULLITEM se lista vuota.**

*tailList(list l)**

- **Sintassi: list* tailList(list* l)**
- **Semantica: Restituisce la coda della lista l, cioè la lista senza il primo elemento.**
- **Precondizioni: Lista l non vuota.**
- **Postcondizioni: Ritorna la lista senza il primo nodo, o NULL se lista vuota.**

*getNext(list nodo)**

- **Sintassi: list* getNext(list* nodo)**

- **Semantica: Restituisce il nodo successivo a nodo.**
- **Precondizioni: Nodo valido.**
- **Postcondizioni: Ritorna puntatore al nodo successivo, o NULL se non esiste.**

setNext(list nodo, list next)**

- **Sintassi: void setNext(list* nodo, list* next)**
- **Semantica: Imposta il puntatore successivo del nodo nodo a next.**
- **Precondizioni: Puntatori validi.**
- **Postcondizioni: Nodo aggiornato con nuovo puntatore al nodo successivo.**

*getValue(list l)**

- **Sintassi: abbonamento* getValue(list* l)**
- **Semantica: Restituisce il valore contenuto nel nodo l.**
- **Precondizioni: Nodo valido.**
- **Postcondizioni: Ritorna il puntatore all'abbonamento contenuto nel nodo.**

setValue(list l, abbonamento a)**

- **Sintassi: void setValue(list* l, abbonamento* a)**
- **Semantica: Imposta il valore del nodo l a a.**
- **Precondizioni: Nodo e valore validi.**
- **Postcondizioni: Nodo aggiornato con nuovo valore.**

*printList(list l)**

- **Sintassi: void printList(list* l)**
- **Semantica: Stampa tutti gli abbonamenti contenuti nella lista l.**
- **Precondizioni: Lista inizializzata (anche vuota).**
- **Postcondizioni: Abbonamenti stampati a video.**

*freeList(list l)**

- **Sintassi: void freeList(list* l)**
- **Semantica: Libera tutta la memoria occupata dalla lista l.**

- **Precondizioni: Lista allocata in memoria.**
- **Postcondizioni: Memoria liberata e lista invalidata.**

*visualizzaListaOrdinata(list abbonati)**

- **Sintassi: void visualizzaListaOrdinata(list* abbonati)**
- **Semantica: Visualizza un menu per l'ordinamento e stampa la lista abbonati ordinata secondo il criterio scelto.**
- **Precondizioni: Lista inizializzata.**
- **Postcondizioni: Lista visualizzata ordinata.**

ordinaPerNome(list head)**

- **Sintassi: void ordinaPerNome(list** head)**
- **Semantica: Ordina la lista puntata da head in ordine crescente per nome, cognome e data iscrizione.**
- **Precondizioni: Lista valida.**
- **Postcondizioni: Lista ordinata in loco.**

ordinaPerDataScadenza(list abbonati)**

- **Sintassi: void ordinaPerDataScadenza(list** abbonati)**
- **Semantica: Ordina la lista in ordine crescente per data di scadenza.**
- **Precondizioni: Lista valida.**
- **Postcondizioni: Lista ordinata in loco.**

ordinaPerDataInizio(list abbonati)**

- **Sintassi: void ordinaPerDataInizio(list** abbonati)**
- **Semantica: Ordina la lista in ordine crescente per data di inizio abbonamento.**
- **Precondizioni: Lista valida.**
- **Postcondizioni: Lista ordinata in loco.**

ordinaPerCodiceDecrescente(list head)**

- **Sintassi: void ordinaPerCodiceDecrescente(list** head)**
- **Semantica: Ordina la lista in ordine decrescente per codice abbonamento.**

- **Precondizioni: Lista valida.**
- **Postcondizioni: Lista ordinata in loco.**

ordinaPerCodiceCrescente(list head)**

- **Sintassi: void ordinaPerCodiceCrescente(list** head)**
- **Semantica: Ordina la lista in ordine crescente per codice abbonamento.**
- **Precondizioni: Lista valida.**
- **Postcondizioni: Lista ordinata in loco.**

list_prenotazioni.c

1. Specifiche Sintattiche

- **Include di: stdio.h, stdlib.h, string.h, list_prenotazioni.h, prenotazioni.h, lezione.h data.h, orario.h util.h list_Lezioni.h, gestione_prenotazioni.h**

Funzioni definite:

- **Prenotazione_setNext()**
- **Prenotazione_setValue()**
- **Prenotazione_emptyList()**
- **Prenotazione_printList()**
- **Prenotazione_freeList()**

2. Specifiche Semantiche

Prenotazione_setNext()

- **Sintassi: void prenotazione_setNext(ListaPrenotazioni p, ListaPrenotazioni next)**
- **Semantica: imposta il puntatore al nodo successivo nella lista delle prenotazione**
- **Precondizioni: Entrambi i parametri devono essere nodi validi**
- **Postcondizioni: il campo next del nodo p punta a next**

Prenotazione_setValue()

- **Sintassi: void prenotazione_setValue(ListaPrenotazioni p, Prenotazione *prenotazione)**
- **Semantica: Associa una struttura Prenotazione al nodo della lista**
- **Precondizioni: Il puntatore prenotazione deve essere valido**
- **Postcondizioni: il nodo p contiene la prenotazione specificata**

Prenotazione_emptyList()

- **Sintassi: int prenotazione_emptyList(Listaprenotazione *lista)**
- **Semantica: Verifica se la lista delle prenotazioni è vuota**
- **Precondizioni: Lista inizializzata**
- **Postcondizioni: Ritorna 1 se la lista è vuota, 0 altrimenti**

Prenotazione_printList()

- **Sintassi: void prenotazione_printList(ListaPrenotazioni *lista)**
- **Semantica: Stampa tutte le prenotaizioni presenti nella lista**
- **Precondizioni: Lista valida e contenente almeno un elemento**
- **Postcondizioni: Le prenotazioni vengono stampate a video in ordine di inserimento**

inserisciPrenotazione()

- **Sintassi: void prenotazione_freeList(ListaPrenotazione *lista)**
- **Semantica: Libera la memoria occupata da tutti i nodi della lista delle prenotazioni**
- **Precondizioni: Lista inizializzata**
- **Postcondizioni: Tutti i nodi vengono deallocati e la lista diventa vuota**

orario.c

1. Specifiche Sintattiche

• Include di:

- **stdio.h,**
- **stdlib.h,**
- **time.h**
- **orario.h**

Definizioni di Macro per colori:

Verde: Conferma, successo, abbonamento valido
Rosso: Errore, avvisi, abbonamento scaduto
Giallo: input, attenzione, prompt
Viola: Intestazioni di menu
Blu: Titoli secondari o sezioni
Reset: Ripristina il colore normale

Funzioni definite:

- **GetOra() GetMinuto()**
- **SetOra() SetMinuto()**
- **ConfrontaOrari()**
- **StampaOrario()**

2. Specifiche Semantiche

GetOra() / GetMinuto()

- **Sintassi: int getOra(Orario o), int getMinuto(Orario o)**
- **Semantica: Restituiscono rispettivamente l'ora e il minuto da una struttura Ora**
- **Precondizioni: Orario Valido**
- **Postcondizioni: Ritorna il valore richiesto**

SetOra() / SetMinuto

- **Sintassi: void SetOra(Orario o, int ora), void setMinuto(Orario *o, int minuto);**
- **Semantica: Impostano rispettivamente l'ora e i minuti in una struttura Orario**
- **Precondizioni: Puntatore valido**
- **Postcondizioni: Orario aggiornato**

confrontaOrari()

- **Sintassi: int confrontaOrari(Orario o1, Orario o2)**
- **Semantica: Confronta due orari restituendo -1, 0, o 1.**
- **Precondizioni: Orari validi**
- **Postcondizioni: Valore del confronto restituito**

stampaOrario()

- **Sintassi: void stampaOrario(Orario o)**
- **Semantica: Stampa l'orario in formato HH:MM**
- **Precondizioni: Orario valido**
- **Postcondizioni: Valori Stampati**

Specifiche Sintattiche

Prenotazione.c

Include di:

- **stdio.h**
- **stdlib.h**
- **string.h**

- **prenotazioni.h**
- **abbonamento.h**
- **lezione.h**
- **data.h**
- **orario.h**
- **util.h**
- **file.h**
- **list.h**
- **list_Lezioni.h**
- **list_prenotazioni.h**

Funzioni definite:

- **gestione_prenotazioni()**
- **aggiungi_prenotazione(int, int)**
- **aggiungi_prenotazioneCliente(abbonamento*)**
- **disdici_prenotazione()**
- **disdici_prenotazioneCliente(abbonamento*)**
- **visualizzaPrenotazioni(listP*, list*)**
- **visualizzaPrenotazioniLezione(int, listP*, list*)**
- **visualizzaPrenotazioniCliente(int, listP*, listL*)**
- **reportLezioniUltimoMeseCliente(int, listP*, listL*)**
- **reportDisciplineUltimoMese(listP*, listL*)**
- **reportGestore(listP*, listL*)**

Specifiche Semantiche:

gestione_prenotazioni()

- **Sintassi: void gestione_prenotazioni()**
- **Semantica: Menu principale per gestire tutte le funzionalità relative alle prenotazioni: aggiunta, cancellazione, visualizzazione, report.**

- **Precondizioni: Liste correttamente inizializzate.**
- **Postcondizioni: L'utente può gestire e consultare le prenotazioni.**

aggiungi_prenotazione(int codice_abbonamento, int codice_lezione)

- **Sintassi: void aggiungi_prenotazione(int, int)**
- **Semantica: Permette all'amministratore di creare una prenotazione per un cliente.**
- **Precondizioni: Codici validi e liste non vuote.**
- **Postcondizioni: Prenotazione inserita nella lista.**

aggiungi_prenotazioneCliente(abbonamento* abbonato_corrente)

- **Sintassi: void aggiungi_prenotazioneCliente(abbonamento*)**
- **Semantica: Il cliente può aggiungere una prenotazione a una lezione usando il proprio codice abbonamento.**
- **Precondizioni: Codice abbonato valido.**
- **Postcondizioni: Prenotazione registrata.**

disdici_prenotazione()

- **Sintassi: void disdici_prenotazione()**
- **Semantica: L'amministratore può eliminare una prenotazione specificando i codici.**
- **Precondizioni: La prenotazione deve esistere.**
- **Postcondizioni: Prenotazione rimossa dalla lista.**

disdici_prenotazioneCliente(abbonamento*)

- **Sintassi: void disdici_prenotazioneCliente(abbonamento*)**
- **Semantica: Il cliente può annullare una propria prenotazione.**
- **Precondizioni: Codici validi e prenotazione esistente.**
- **Postcondizioni: Prenotazione eliminata.**

visualizzaPrenotazioni(listP*, list*)

- **Sintassi: void visualizzaPrenotazioni(listP*, list*)**
- **Semantica: Visualizza tutte le prenotazioni registrate nel sistema.**

- **Precondizioni: Lista inizializzata.**
- **Postcondizioni: Prenotazioni mostrate a video.**

visualizzaPrenotazioniLezione(int codice_lezione, listP*, list*)

- **Sintassi: void visualizzaPrenotazioniLezione(int, listP*, list*)**
- **Semantica: Mostra tutte le prenotazioni associate a una determinata lezione.**
- **Precondizioni: Codice lezione esistente.**
- **Postcondizioni: Prenotazioni visualizzate.**

visualizzaPrenotazioniCliente(int codice_abbonamento, listP*, listL*)

- **Sintassi: void visualizzaPrenotazioniCliente(int, listP*, listL*)**
- **Semantica: Elenca tutte le prenotazioni effettuate da un cliente.**
- **Precondizioni: Codice abbonato esistente.**
- **Postcondizioni: Lista delle prenotazioni del cliente stampata.**

reportLezioniUltimoMeseCliente(int codice_abbonamento, listP*, listL*)

- **Sintassi: void reportLezioniUltimoMeseCliente(int, listP*, listL*)**
- **Semantica: Mostra tutte le lezioni frequentate da un cliente nell'ultimo mese.**
- **Precondizioni: Codice abbonato valido, lista aggiornata.**
- **Postcondizioni: Report stampato.**

reportDisciplineUltimoMese(listP*, listL*)

- **Sintassi: void reportDisciplineUltimoMese(listP*, listL*)**
- **Semantica: Riporta le discipline più frequentate nel mese corrente.**
- **Precondizioni: Prenotazioni e lezioni aggiornate.**
- **Postcondizioni: Dati aggregati visualizzati.**

Report Gestore()

- **Sintassi: void reportGestore(listP*, listL*)**
- **Semantica: Riassunto generale dell'attività: lezioni, prenotazioni e discipline per il gestore.**

- **Precondizioni: Liste aggiornate.**
- **Postcondizioni: Report amministrativo generato.**

Gesitone_prenotazioni.c

1. Specifiche Sintattiche

- **Include di: stdio.h, stdlib.h, string.h, prenotazioni.h, lezione.h, data.h, orario.h, list.h util.h, abbonamento.h gestione_abbonamenti.h, gestione_lezioni.h list_lezioni.h**

Funzioni definite:

- **gestione_prenotazioni()**
- **aggiungi_prenotazione(int, int)**
- **aggiungi_prenotazioneCliente(abbonamento*)**
- **disdici_prenotazione()**
- **disdici_prenotazioneCliente(abbonamento*)**
- **visualizzaPrenotazioni(listP*, list*)**
- **visualizzaPrenotazioniLezione(int, listP*, list*)**
- **visualizzaPrenotazioniCliente(int, listP*, listL*)**
- **reportLezioniUltimoMeseCliente(int, listP*, listL*)**
- **reportDisciplineUltimoMese(listP*, listL*)**
- **reportGestore(listP*, listL*)**

Specifiche Semantiche

Gestione_prenotazioni()

- **Sintassi: void gestione_prenotazioni()**
- **Semantica: il menu principale per la gestione delle prenotazioni. Permette di aggiungere, disdire e visualizzare prenotazioni, oltre a generare repost**
- **Precondizioni: Liste inizializzate correttamente**
- **Postcondizioni: L'utente può modificare o consultare le prenotazioni**

Aggiungi_prenotazione(int,int)

- **Sintassi: void aggiungi_prenotazione(int codice_abbonamento,int codice_prenotazione)**
- **Semantica: Permette all'amministratore di inserire una nuova prenotazione specificando codice abbonamento e codice lezione**
- **Precondizioni: Liste non vuote, codici esistenti**
- **Postcondizioni: I dettagli sono visualizzati a video.**

Aggiungi_prenotazioneCliente(abbonamento)

- **Sintassi: void aggiungi_prenotazioneCliente(abbonamento* abbonato_corrente)**
- **Semantica: Permette al cliente di prenotarsi a una lezione inserendo solo il codice della lezione, associato automaticamente al suo codice abbonamento**
- **Precondizioni: codice abbonato esistente**
- **Postcondizioni: Prenotazione inserita nella lista**

Disdici_prenotazione()

- **Sintassi: void disdici_prenotazione()**
- **Semantica: Permette all'amministratore di annullare una prenotazione inserendo i codici di abbonato e lezione**
- **Precondizioni: La prenotazione deve esistere**
- **Postcondizioni: Prenotazione eliminata, lista aggiornata**

Disdici_prenotazioneCliente(abbonamento*)

- **Sintassi: void disdici_prenotazioneCliente(abbonamento*)**
- **Semantica: Permette a un cliente di annullare una propria prenotazione specificando il codice della lezione**
- **Precondizioni: Codice abbonato e lezione validi**
- **Postcondizioni: Prenotazione Rimossa**

visualizzaPrenotazioni(listP*, list*)

- **Sintassi: void visualizzaPrenotazioni(ListaP*,list*)**
- **Semantica: stampa tutte le prenotazioni presenti nella lista**
- **Precondizioni: Lista inizializzata**
- **Postcondizioni: Tutte le prenotazioni vengono stampate**

VisualizzaPrenotazioniLezione(int, listP*, list*)

- **Sintassi: void visualizzaPrenotazioniLezione(int, listP*, list*)**
- **Semantica: Mostra tutte le prenotazioni relative ad una lezione specifica**
- **Precondizioni: Codice lezione esistente**
- **Postcondizioni: Prenotazioni visualizzate**

VisualizzaPrenotazioniCliente(abbonamento*)

- **Sintassi: void visualizzaPrenotazioniCliente(abbonamento*)**
- **Semantica: Mostra tutte le prenotazioni effettuate da un determinato abbonato**
- **Precondizioni: codice abbonato esistente**
- **Postcondizioni: Prenotazioni del cliente stampate a video**

reportLezioniUltimoMeseCliente(int ,ListaP *, ListaL*)

- **Sintassi: void reportLezioniUltimoMeseCliente(int codiceAbbonato,ListaP *prenotazioni, ListaL*lezioni)**
- **Semantica: Genera un report con l'elenco delle lezioni frequentate da un abbonato nell'ultimo mese**
- **Precondizioni: Codice abbonato valido**
- **Postcondizioni: Report stampato a video**

reportDisciplineUltimoMese()

- **Sintassi: void reportDisciplineUltimoMese(ListaPrenotazioni *prenotazioni, ListaLezioni *lezioni)**
- **Semantica: Stampa un report delle discipline più frequentate nell'ultimo mese**
- **Precondizioni: Liste lezioni e prenotazioni devono essere valide**
- **Postcondizioni: Report mostrato a video**

ReportGestore(listP*, listL*)

- **Sintassi: void ReportGestore(ListaP*prenotazioni, list*lezioni)**
- **Semantica: Genera un report dettagliato per il gestore con statistiche sulle prenotazioni per disciplina e lezione**
- **Precondizioni: Liste valide e non vuote**
- **Postcondizioni: Report stampato a video**