Università degli studi di Salerno

Progetto PSD: Report Traccia 1

Singh Raj Abhaypartap - 0512119154
Tozza Gennaro Carmine - 0512120382
Valva Lorenzo - 0512119639

31 maggio 2024

Indice

Capitolo 1

Motivazione della scelta degli ADT

1.1 ADT Data

Uno dei primi ADT che si incontra in questo progetto è il tipo di dato astratto Data, implementato come un puntatore alla struttuta DataOra.
L'implementazione della struct DataOra è definita nel file data.c in modo da non renderla visibile al client tramite l'include dell'header file data.h, così facendo possiamo trarre una serie di vantaggi relativi l'information hiding.
Discorso equivalente per tutti gli ADT successivamente implementati.

La struttura permette di gestire date e orari di inizio e fine evento, per questo include funzioni per creare, validare, confrontare, leggere e scrivere quest'ultime. Inoltre, utile anche ai fini del testing, usiamo i file per leggere dati di input e scrivere dati di output.

1.2 ADT Sala

L'ADT sala è dotato di un id e di un nome.
La scelta di utilizzare un id univoco per le sale facilità l'assegnazione di quest'ultime ad un evento.

1.3 ADT Lista_sala

La prima struttura dati utilizzata in questo progetto è la lista. In particolare, l'ADT lista_sala permette di gestire una lista di sale, in modo tale da mantenere organizzate le informazioni relative alle diverse sale.

Essa è implementata tramite un array dinamico, ciò permette alla lista di gestire una quantità crescente di dati senza dover specificare una dimensione massima a priori.

Inoltre consente di accedere direttamente agli elementi in tempo costante $O(1)$, a differenza di una lista collegata dove l'accesso agli elementi richiede un tempo lineare $O(n)$ in media.

Infine gli array dinamici utilizzano memoria contigua, il che significa che tutti gli elementi sono memorizzati in un blocco continuo di memoria.
Questo migliora la località dei dati e rende il programma più efficiente dal punto di vista della cache, in quanto la CPU può accedere agli elementi con meno accessi alla memoria principale.

1.4 ADT Evento

Il tipo di dato astratto Evento prevede un'implementazione che include un id univoco, il nome, il tipo, le date e gli orari di inizio e fine evento, che utilizzano l'ADT Data, e l'id della sala.
La scelta di utilizzare un id univoco per gli eventi facilita la loro ricerca per effettuare operazioni su di essi, evitando eventuali omonimie che potrebbero crearsi utilizzando una ricerca basata sul campo nome.

1.5 ADT EventoBst

L'ADT EvenotBst è stato introdotto per gestire in modo efficiente e organizzato la raccolta degli eventi all'interno del sistema di gestione delle conferenze.
Questo ADT rappresenta un albero binario di ricerca in cui ogni nodo contiene un singolo evento e l'ordinamento è basato su proprietà degli eventi, come la data e l'ora di inizio.

L'uso di un albero binario di ricerca consente di aggiungere eventi con una complessità temporale media di $O(\log n)$, il che significa che l'inserimento di un nuovo evento è relativamente rapido anche quando il numero totale di eventi è grande.
Tuttavia, per quanto riguarda le operazioni di modifica e rimozione, non è possibile sfruttare le stesse proprietà di efficienza dell'albero. Di conseguenza, queste operazioni hanno una complessità di $O(n)$, poiché richiedono una ricerca lineare per l'id dell'evento prima di poter essere eseguite.

Nonostante questa limitazione, l'uso di questo ADT è considerato giustificato. Le alternative avrebbero comunque comportato una complessità temporale

di $O(n)$ per le operazioni di modifica e rimozione e avrebbero inoltre reso lineare anche la complessità di aggiunta degli eventi, rendendo il sistema meno e fficiente nel complesso.
Utilizzando l'EventoBst, si mantiene almeno l'operazione di aggiunta degli eventi efficiente, migliorando così le prestazioni generali del sistema di gestione delle conferenze.

1.6 ADT Conferenza

Infine vi è l'ADT Conferenza che è stato realizzato per gestire in modo completo e strutturato l'organizzazione di una conferenza, che quindi include gli eventi, le sale e la loro assegnazione.
Infatti esso prevede:

- un albero binario di ricerca che contiene gli eventi della conferenza.
- un contatore per assegnare identificativi univoci agli eventi.
- un contatore per assegnare identificativi univoci alle sale.
- una lista che contiene le sale della conferenza.

Capitolo 2

Progettazione

2.1 Struttura del Sistema

Il sistema di gestione delle conferenze è strutturato in modo modulare, suddiviso in vari componenti che interagiscono tra loro per fornire le funzionalità richieste. Ogni componente è stato progettato come un ADT per garantire una chiara separazione delle responsabilità e facilitare il mantenimento del codice.

2.1.1 Moduli Principali

I principali moduli del sistema sono:

- **ADT Data**: Gestisce le date e gli orari degli eventi.
- **ADT Sala**: Rappresenta le sale dove si tengono gli eventi.
- **ADT Lista_sala**: Gestisce una lista dinamica di sale.
- **ADT Evento**: Rappresenta gli eventi della conferenza.
- **ADT EventoBst**: Organizza gli eventi in un albero binario di ricerca per una gestione efficiente.
- **ADT Conferenza**: Coordina tutti gli elementi sopra menzionati, fornendo un'interfaccia unificata per la gestione della conferenza.

2.2 Interazione tra Componenti

I componenti interagiscono tra loro attraverso chiamate a funzioni che rispettano i contratti definiti dai rispettivi ADT. Questa progettazione permette di mantenere un alto grado di incapsulamento e modularità.

2.2.1 Gestione delle Date e Orari (ADT Data)

Il modulo **ADT Data** è utilizzato da altri moduli per creare, confrontare, e validare date e orari. Ad esempio, quando viene creato un nuovo evento (ADT Evento), vengono utilizzate le funzioni di ADT Data per impostare la data e l'ora di inizio e fine.

2.2.2 Gestione delle Sale (ADT Sala e ADT Lista_sala)

Le sale sono gestite come oggetti di ADT Sala e sono organizzate in una lista dinamica tramite **ADT Lista_sala**. Questa lista permette di aggiungere, rimuovere e cercare sale in modo efficiente. Quando viene creato un evento, si può assegnare una sala disponibile cercando tra gli oggetti ADT Sala nella lista.

2.2.3 Gestione degli Eventi (ADT Evento e ADT EventoBst)

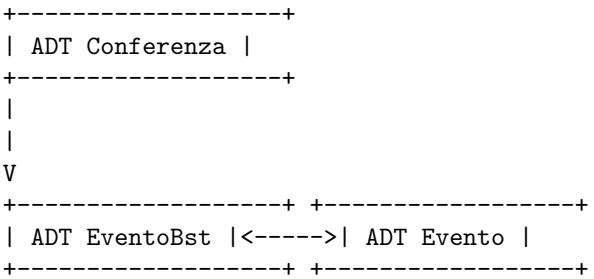
Gli eventi sono rappresentati come oggetti di **ADT Evento** e sono organizzati in un albero binario di ricerca tramite **ADT EventoBst**. L'albero permette di aggiungere, cercare e rimuovere eventi in modo relativamente efficiente. Quando viene creato un nuovo evento, viene inserito nell'albero binario di ricerca, garantendo un ordinamento basato sulle date e orari.

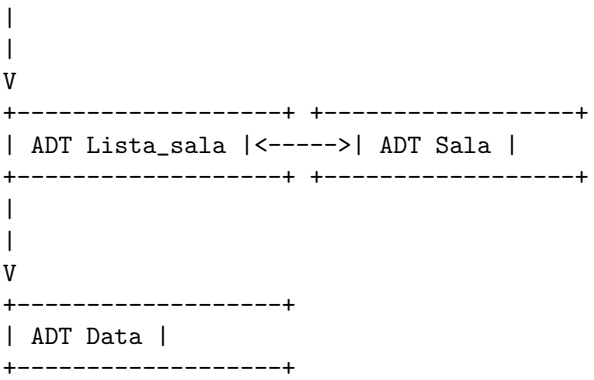
2.2.4 Coordinamento della Conferenza (ADT Conferenza)

Il modulo **ADT Conferenza** funge da coordinatore centrale. Esso mantiene un riferimento all'albero degli eventi (**ADT EventoBst**) e alla lista delle sale (**ADT Lista_sala**). Gestisce anche i contatori per assegnare identificativi univoci agli eventi e alle sale. Quando un utente interagisce con il sistema per creare, modificare o rimuovere un evento, il modulo **ADT Conferenza** invoca le appropriate funzioni degli altri moduli per eseguire le operazioni richieste.

2.3 Diagramma delle Interazioni

Il seguente diagramma rappresenta le interazioni tra i diversi componenti del sistema:





Tirando le somme, questa progettazione consente al sistema di gestione delle conferenze di essere robusto, efficiente e facile da mantenere, assicurando che possa gestire un ampio numero di eventi e sale in modo organizzato e performante.

Capitolo 3

Specifica Sintattica e Semantica

3.1 Utile.c

Specifica Sintattica degli Operatori

Tipi Usati:

- `int, char*, FILE*, unsigned long, void*`

Operatori:

- `pulisci_buffer(FILE*) -> void`
- `leggi_input_da_file(char*, int, FILE*) -> int`
- `leggi_input(char*, int) -> int`
- `leggi_intero() -> int`
- `my_alloc(unsigned long, unsigned long) -> void*`
- `my_realloc(void*, unsigned long, unsigned long) -> void*`
- `my_strdup(char*) -> char*`

Specifica Semantica degli Operatori

1. `pulisci_buffer(file) = void`
   - **Input:** `file`
   - **Precondizione:** Il file deve essere aperto in modalità lettura.

- **Output:** Nessuno.
- **Postcondizione:** Il buffer di input viene pulito fino a incontrare un carattere di nuova riga o la fine del file.

2. `leggi_input_da_file(riga, dimensione, file) = int`

- **Input:** `riga`, `dimensione`, `file`
- **Precondizione:** La stringa deve avere una dimensione sufficiente per contenere la riga letta. Il file deve essere aperto in modalità lettura.
- **Output:** Intero che indica il risultato dell'operazione (`0`: successo, `-1`: errore durante la lettura, `-2`: riga troppo lunga).
- **Postcondizione:** La riga viene letta dal file e memorizzata nella stringa fornita, senza il carattere di nuova riga alla fine. Se la riga è troppo lunga, il buffer viene pulito.

3. `leggi_input(riga, dimensione) = int`

- **Input:** `riga`, `dimensione`
- **Precondizione:** La stringa deve avere una dimensione sufficiente per contenere la riga letta.
- **Output:** Intero che indica il risultato dell'operazione (`0`: successo, `-1`: errore durante la lettura, `-2`: riga troppo lunga).
- **Postcondizione:** La riga viene letta dallo standard input (`stdin`) e memorizzata nella stringa fornita, senza il carattere di nuova riga alla fine. Se la riga è troppo lunga, il buffer viene pulito.

4. `leggi_intero() = int`

- **Input:** Nessuno.
- **Precondizione:** Nessuna.
- **Output:** Intero letto dallo standard input (`stdin`). Valori di ritorno specifici per errori: `-1` (errore durante la lettura della riga), `-2` (errore nella conversione della stringa in intero), `-3` (nessun numero trovato nella stringa).
- **Postcondizione:** Viene letto un intero dallo standard input. In caso di errore, vengono restituiti valori di ritorno specifici per indicare il tipo di errore.

5. `my_alloc(num_membri, dimensione) = p`
   - **Input:** `num_membri`, `dimensione`
   - **Precondizione:** Nessuna.
   - **Output:** Puntatore alla memoria allocata.
   - **Postcondizione:** Viene allocata memoria per l'array specificato. Se l'allocazione fallisce, viene stampato un messaggio di errore e il programma termina.

6. `my_realloc(p, num_membri, dimensione) = temp`
   - **Input:** `p`, `num_membri`, `dimensione`
   - **Precondizione:** Nessuna.
   - **Output:** Puntatore alla memoria riallocata.
   - **Postcondizione:** La memoria viene riallocata per l'array specificato. Se la riallocazione fallisce, viene stampato un messaggio di errore e il programma termina.

7. `my_strdup(stringa) = temp`
   - **Input:** `stringa`
   - **Precondizione:** La stringa deve essere valida.
   - **Output:** Puntatore alla stringa duplicata.
   - **Postcondizione:** Viene allocata memoria per una copia della stringa fornita. Se l'allocazione fallisce, viene stampato un messaggio di errore e il programma termina.

3.2 Data.c

Specifica Tipi di Dato

Sintattica:

- Tipo di riferimento: Data
- Tipi usati: `int`

Semantica:

Il tipo di dato "Data" è una struttura che rappresenta una data e un'ora specifiche, composta da campi come anno, mese, giorno, ora e minuti.

Specifica Operatori

Sintattica:

- `nuova_data(int, int, int, int, int) -> Data`
- `controllo_data(Data) -> int`
- `utile_data(int, int) -> int`
- `confronta_date(Data, Data) -> int`
- `input_data() -> Data`
- `stampa_data(Data) -> void`
- `salva_data_su_file(Data, FILE*) -> void`
- `leggi_data_da_file(FILE*) -> Data`
- `libera_data(Data) -> void`

Semantica:

1. `nuova_data(anno, mese, giorno, ora, minuto) = data`
   - **Input:** Anno, mese, giorno, ora, minuti.
   - **Precondizione:** Tutti i valori di input devono essere validi.
   - **Output:** Puntatore a una nuova struttura `Data`.
   - **Postcondizione:** Viene creata una nuova data con i valori forniti e viene restituito un puntatore ad essa. Se l'allocazione di memoria fallisce o se i valori forniti non rappresentano una data valida, viene restituito `NULL`.

2. `controllo_data(data) = int`
   - **Input:** Puntatore a una struttura `Data`.
   - **Precondizione:** La data deve essere una struttura `Data` valida.
   - **Output:** Intero che indica se la data è valida (1) o meno (0).
   - **Postcondizione:** Viene verificato se la data è valida in base ai criteri specificati. Se la data è valida, restituisce 1, altrimenti restituisce 0.

3. `utile_data(d1,d2) = int`

   - **Input:** Due interi da confrontare.
   - **Precondizione:** Nessuna.
   - **Output:** Intero che indica la relazione tra i due interi (-1: primo minore, 0: uguali, 1: primo maggiore).
   - **Postcondizione:** Viene restituita la relazione tra i due interi. Se il primo intero è minore del secondo, restituisce -1; se sono uguali, restituisce 0; se il primo intero è maggiore del secondo, restituisce 1.

4. `confronta_date(data1, data2) = int`

   - **Input:** Due puntatori a strutture `Data`.
   - **Precondizione:** Entrambe le date devono essere valide.
   - **Output:** Intero che indica la relazione tra le due date.
   - **Postcondizione:** Viene restituita la relazione tra le due date in base all'anno, al mese, al giorno, all'ora e ai minuti. Se la prima data è precedente alla seconda, restituisce un valore negativo; se sono uguali, restituisce 0; se la prima data è successiva alla seconda, restituisce un valore positivo.

5. `input_data(void) = data`

   - **Input:** Nessuno.
   - **Precondizione:** Nessuna.
   - **Output:** Puntatore a una nuova struttura `Data`.
   - **Postcondizione:** Legge una data dall'input dell'utente e restituisce un puntatore ad essa. Se l'input non è valido o se la data non è valida, restituisce `NULL`.

6. `stampa_data(data) = void`

   - **Input:** Puntatore a una struttura `Data`.
   - **Precondizione:** La data deve essere valida.
   - **Output:** Nessuno.
   - **Postcondizione:** Stampa la data nel formato giorno/mese/anno, ora:minuti.

7. `salva_data_su_file(data,file) = void`
   - **Input:** Puntatore a una struttura `Data` e puntatore a un file.
   - **Precondizione:** La data e il file devono essere validi.
   - **Output:** Nessuno.
   - **Postcondizione:** Scrive la data nel file nel formato giorno mese anno ora minuti.

8. `leggi_data_da_file(file) = data`
   - **Input:** Puntatore a un file.
   - **Precondizione:** Il file deve essere valido.
   - **Output:** Puntatore a una nuova struttura `Data`.
   - **Postcondizione:** Legge una data dal file e restituisce un puntatore ad essa. Se il file non contiene una data valida o se la lettura fallisce, restituisce `NULL`.

9. `libera_data()` = **void**
   - **Input:** Puntatore a una struttura `Data`.
   - **Precondizione:** Nessuna.
   - **Output:** Nessuno.
   - **Postcondizione:** Libera la memoria allocata per la data.

3.3 Sala.c

Specifica Tipi di Dato

Sintattica:

- Tipo di riferimento: Sala
- Tipi usati: `int`, `char*`

Semantica:

Il tipo di dato `Sala` è una struttura che rappresenta una sala. È composta da un ID e un nome.

Specifica Operatori

Sintattica:

- `nuova_sala(char*, int) -> Sala`
- `sale_uguali(Sala, Sala) -> bool`
- `input_sala(int) -> Sala`
- `prendi_sala_nome(Sala) -> char*`
- `prendi_sala_id(Sala) -> int`
- `stampa_sala(Sala) -> void`
- `salva_sala_su_file(Sala, FILE*) -> void`
- `leggi_sala_da_file(FILE*) -> Sala`
- `libera_sala(Sala) -> void`

Semantica:

1. `nuova_sala(nome, id) = s`
   - **Input:** Nome della sala (char*), ID della sala (int).
   - **Precondizione:** Tutti i valori di input devono essere validi.
   - **Output:** Puntatore a una nuova struttura `Sala`.
   - **Postcondizione:** Viene creata una nuova sala con il nome e l'ID forniti, e viene restituito un puntatore ad essa. Se l'allocazione di memoria fallisce o se i valori forniti non sono validi, viene restituito `NULL`.

2. `sale_uguali(s1, s2) = bool`
   - **Input:** Due sale di tipo `Sala`.
   - **Precondizione:** Nessuna.
   - **Output:** Valore booleano.
   - **Postcondizione:** Restituisce `true` se le due sale hanno lo stesso ID, altrimenti restituisce `false`.

3. `input_sala(id) = s`
   - **Input:** ID della sala (int).
   - **Precondizione:** Nessuna.
   - **Output:** Puntatore a una nuova struttura `Sala`.
   - **Postcondizione:** Legge il nome della sala dall'input dell'utente e crea una nuova sala con l'ID fornito, restituendo un puntatore ad essa.

4. `prendi_sala_nome(s) = s -> nome`
   - **Input:** Una sala di tipo `Sala`.
   - **Precondizione:** Nessuna.
   - **Output:** Nome della sala (char*).
   - **Postcondizione:** Restituisce il nome della sala.

5. `prendi_sala_id(s) = s -> id`
   - **Input:** Una sala di tipo `Sala`.
   - **Precondizione:** Nessuna.
   - **Output:** id della sala (int).
   - **Postcondizione:** Restituisce l'id della sala.

6. `stampa_sala(s) = void`
   - **Input:** Una sala di tipo `Sala`.
   - **Precondizione:** Nessuna.
   - **Output:** Nessuno.
   - **Postcondizione:** Stampa il nome della sala.

7. `salva_sala_su_file(s, file) = void`
   - **Input:** Una sala di tipo `Sala`, un puntatore a un file di tipo `FILE*`.
   - **Precondizione:** Nessuna.
   - **Output:** Nessuno.
   - **Postcondizione:** Salva l'ID e il nome della sala nel file specificato.

8. `leggi_sala_da_file(file) = sala`
   - **Input:** Un puntatore a un file di tipo `FILE*`.
   - **Precondizione:** Nessuna.
   - **Output:** Puntatore a una nuova struttura `Sala`.
   - **Postcondizione:** Legge l'ID e il nome della sala dal file specificato e restituisce una nuova sala.

9. `libera_sala(sala) = void`
   - **Input:** Una sala di tipo `Sala`.
   - **Precondizione:** Nessuna.
   - **Output:** Nessuno.
   - **Postcondizione:** Libera la memoria allocata per la sala specificata.

3.4 Lista_Sala.c

Specifica Tipi di Dato:

Sintattica:

- Tipo di Riferimento: `ListaSala`
- Tipi Usati: `Sala`, `int`

Semantica:

Il tipo di dato `ListaSala` è una struttura che rappresenta una lista dinamica di sale. Include campi per l'array di sale, la capacità dell'array e la dimensione attuale della lista.

Specifica Operatori

Sintattica:

- `nuova_lista_sala() -> ListaSala`
- `aumenta_capacita_lista_sala(ListaSala) -> void`
- `lista_sala_vuota(ListaSala) -> bool`
- `aggiungi_sala_lista(ListaSala, Sala) -> void`
- `sala_fine_lista(ListaSala) -> Sala`

- `prendi_prima_sala_lista(ListaSala) -> Sala`
- `prendi_dimensione_lista_sala(ListaSala) -> int`
- `prendi_posizione_sala_lista(ListaSala, Sala) -> int`
- `sala_in_posizione_lista(ListaSala, int) -> Sala`
- `sala_per_id_lista(ListaSala, int) -> Sala`
- `stampa_lista_sala(ListaSala) -> void`
- `salva_lista_sala_su_file(ListaSala, FILE*) -> void`
- `leggi_lista_sala_da_file(FILE*) -> ListaSala`
- `libera_lista_sala(ListaSala) -> void`

Semantica:

1. `nuova_lista_sala() = nuova_lista`
   - **Input:** Nessuno.
   - **Precondizione:** Nessuna.
   - **Output:** ListaSala.
   - **Postcondizione:** Crea e restituisce una nuova lista di sale vuota con una capacità iniziale predefinita.

2. `aumenta_capacita_lista_sala(lista) = void`
   - **Input:** ListaSala.
   - **Precondizione:** ListaSala deve essere valida.
   - **Output:** Nessuno.
   - **Postcondizione:** Aumenta la capacità dell'array di sale nella lista raddoppiandola.

3. `lista_sala_vuota(lista) = bool`
   - **Input:** ListaSala.
   - **Precondizione:** Nessuna.
   - **Output:** bool.
   - **Postcondizione:** Restituisce `true` se la lista di sale è vuota, altrimenti `false`.

4. `aggiungi_sala_lista(lista, s) = void`
   - **Input:** ListaSala, Sala.
   - **Precondizione:** ListaSala e Sala devono essere validi.
   - **Output:** Nessuno.
   - **Postcondizione:** Aggiunge una sala alla lista, aumentando la capacità se necessario.

5. `sala_fine_lista(lista) = sala`
   - **Input:** ListaSala.
   - **Precondizione:** ListaSala deve essere valida.
   - **Output:** Sala.
   - **Postcondizione:** Restituisce l'ultima sala nella lista e decrementa la dimensione della lista di uno.

6. `prendi_prima_sala_lista(lista) = sala`
   - **Input:** ListaSala.
   - **Precondizione:** ListaSala deve essere valida.
   - **Output:** Sala.
   - **Postcondizione:** Restituisce la prima sala nella lista.

7. `prendi_dimensione_lista_sala(lista) = int`
   - **Input:** ListaSala.
   - **Precondizione:** ListaSala deve essere valida.
   - **Output:** int.
   - **Postcondizione:** Restituisce il numero di sale nella lista.

8. `prendi_posizione_sala_lista(lista, da_cercare) = int`
   - **Input:** ListaSala, Sala.
   - **Precondizione:** ListaSala e Sala devono essere validi.
   - **Output:** int.
   - **Postcondizione:** Restituisce la posizione della sala specificata nella lista, o -1 se non trovata.

9. `sala_in_posizione_lista(lista, posizione) = sala`
   - **Input:** ListaSala, int.
   - **Precondizione:** La posizione deve essere valida.
   - **Output:** Sala.
   - **Postcondizione:** Restituisce la sala alla posizione specificata nella lista.

10. `sala_per_id_lista(lista, id_sala) = sala`
    - **Input:** ListaSala, int.
    - **Precondizione:** ListaSala deve essere valida.
    - **Output:** Sala.
    - **Postcondizione:** Restituisce la sala con l'ID specificato nella lista, o `NULL` se non trovata.

11. `stampa_lista_sala(lista) = void`
    - **Input:** ListaSala.
    - **Precondizione:** ListaSala deve essere valida.
    - **Output:** Nessuno.
    - **Postcondizione:** Stampa tutte le sale nella lista.

12. `salva_lista_sala_su_file(lista, file) = void`
    - **Input:** ListaSala, FILE*.
    - **Precondizione:** ListaSala e FILE* devono essere validi.
    - **Output:** Nessuno.
    - **Postcondizione:** Salva la lista di sale nel file specificato.

13. `leggi_lista_sala_da_file(file) = lista`
    - **Input:** FILE*.
    - **Precondizione:** FILE* deve essere valido.
    - **Output:** ListaSala.
    - **Postcondizione:** Legge una lista di sale dal file specificato e la restituisce.

14. `libera_lista_sala(lista) = void`

- **Input:** ListaSala.
- **Precondizione:** ListaSala deve essere valida.
- **Output:** Nessuno.
- **Postcondizione:** Libera la memoria allocata per la lista di sale.

3.5 Evento.c

Specifica Tipi di Dato:

Sintattica:

- Tipo di Riferimento: `Evento`
- Tipi Usati: `Data`, `char*`, `int`,

Semantica:

Il tipo di dato "Evento" è una struttura che rappresenta un evento.
Include campi per l'identificativo dell'evento, il nome, il tipo, le date di inizio e fine, e l'identificativo della sala.

Specifica Operatori

Sintattica:

- `crea_evento(char*, int, Data, Data, int) -> Evento`
- `prendi_id(Evento) -> int`
- `prendi_nome(Evento) -> char*`
- `prendi_evento_sala_id(Evento) -> int`
- `prendi_tipo(Evento) -> int`
- `prendi_data_inizio(Evento) -> Data`
- `prendi_data_fine(Evento) -> Data`
- `imposta_data_inizio(Evento, Data) -> int`
- `imposta_data_fine(Evento, Data) -> int`
- `imposta_nome(Evento, char*) -> int`

- `imposta_tipo(Evento, int) -> int`
- `imposta_evento_sala_id(Evento, int) -> int`
- `id_uguale(Evento, Evento) -> int`
- `input_evento(int) -> Evento`
- `stampa_evento(Evento, Sala) -> void`
- `confronta_eventi(Evento, Evento) -> int`
- `salva_evento_su_file(Evento, FILE*) -> void`
- `leggi_evento_da_file(FILE*) -> Evento`
- `libera_evento(Evento) -> void`

Semantica

1. `crea_evento(nome, tipo, inizio, fine, id) = evento`
   - **Input**: `char* nome`, `int tipo`, `Data inizio`, `Data fine`, `int id`
   - **Precondizione**: `nome` deve essere non NULL; `inizio` e `fine` devono essere date valide.
   - **Output**: `Evento`
   - **Postcondizione**: Crea e restituisce un nuovo evento con i parametri specificati.

2. `prendi_id(evento) = evento -> id`
   - **Input**: `Evento evento`
   - **Precondizione**: `evento` deve essere valido.
   - **Output**: `int`
   - **Postcondizione**: Restituisce l'identificativo dell'evento.

3. `prendi_nome(evento) = evento -> nome`
   - **Input**: `Evento evento`
   - **Precondizione**: `evento` deve essere valido.
   - **Output**: `char*`
   - **Postcondizione**: Restituisce una copia del nome dell'evento.

4. `prendi_evento_sala_id(evento) = evento -> sala_id`
   - **Input**: `Evento evento`
   - **Precondizione**: `evento` deve essere valido.
   - **Output**: `int`
   - **Postcondizione**: Restituisce l'identificativo della sala dell'evento.

5. `prendi_tipo(evento) = evento -> tipo`
   - **Input**: `Evento evento`
   - **Precondizione**: `evento` deve essere valido.
   - **Output**: `int`
   - **Postcondizione**: Restituisce il tipo dell'evento.

6. `prendi_data_inizio(evento) = evento -> inizio`
   - **Input**: `Evento evento`
   - **Precondizione**: `evento` deve essere valido.
   - **Output**: `Data`
   - **Postcondizione**: Restituisce la data di inizio dell'evento.

7. `prendi_data_fine(evento) = evento -> fine`
   - **Input**: `Evento evento`
   - **Precondizione**: `evento` deve essere valido.
   - **Output**: `Data`
   - **Postcondizione**: Restituisce la data di fine dell'evento.

8. `imposta_data_inizio(evento, inizio) = int`
   - **Input**: `Evento evento`, `Data inizio`
   - **Precondizione**: `evento` deve essere valido; `inizio` deve essere una data valida.
   - **Output**: `int`
   - **Postcondizione**: Imposta la data di inizio dell'evento e restituisce 0 se l'operazione è riuscita, -1 se l'evento è NULL, -2 se la nuova data di inizio è successiva alla data di fine.

9. `imposta_data_fine(evento, fine) = int`

- **Input**: `Evento evento`, `Data fine`
- **Precondizione**: `evento` deve essere valido; `fine` deve essere una data valida.
- **Output**: `int`
- **Postcondizione**: Imposta la data di fine dell'evento e restituisce 0 se l'operazione è riuscita, -1 se l'evento è NULL, -2 se la nuova data di fine è precedente alla data di inizio.

10. `imposta_nome(evento, nome) = int`

- **Input**: `Evento evento`, `char* nome`
- **Precondizione**: `evento` deve essere valido; `nome` deve essere non NULL.
- **Output**: `int`
- **Postcondizione**: Imposta il nome dell'evento e restituisce 0 se l'operazione è riuscita, -1 se l'evento è NULL.

11. `imposta_tipo(evento, tipo) = int`

- **Input**: `Evento evento`, `int tipo`
- **Precondizione**: `evento` deve essere valido.
- **Output**: `int`
- **Postcondizione**: Imposta il tipo dell'evento e restituisce 0 se l'operazione è riuscita, -1 se l'evento è NULL.

12. `imposta_evento_sala_id(evento, id_sala) = int`

- **Input**: `Evento evento`, `int id_sala`
- **Precondizione**: `evento` deve essere valido.
- **Output**: `int`
- **Postcondizione**: Imposta l'ID della sala dell'evento e restituisce 0 se l'operazione è riuscita, -1 se l'evento è NULL.

13. `id_uguale(evento1, evento2) = int`

- **Input**: `Evento evento1`, `Evento evento2`
- **Precondizione**: `evento1` e `evento2` devono essere validi.
- **Output**: `int`
- **Postcondizione**: Restituisce 1 se gli ID degli eventi sono uguali, 0 altrimenti.

14. `input_evento(id_evento) = evento`

- **Input**: `int id_evento`
- **Precondizione**: Nessuna
- **Output**: `Evento`
- **Postcondizione**: Crea e restituisce un nuovo evento con i dati inseriti dall'utente.

15. `stampa_evento(evento, sala_assegnata) = void`

- **Input**: `Evento evento`, `Sala sala_assegnata`
- **Precondizione**: `evento` deve essere valido.
- **Output**: `void`
- **Postcondizione**: Stampa le informazioni dell'evento.

16. `confronta_eventi(evento1, evento2) = int`

- **Input**: `Evento evento1`, `Evento evento2`
- **Precondizione**: `evento1` e `evento2` devono essere validi.
- **Output**: `int`
- **Postcondizione**: Confronta due eventi e restituisce -1 se la data di inizio del primo evento è precedente alla data di fine del secondo, 1 se è successiva, e il risultato del confronto dei nomi degli eventi se le date coincidono.

17. `salva_evento_su_file(evento, file) = void`

- **Input**: `Evento evento`, `FILE* file`
- **Precondizione**: `evento` e `file` devono essere validi.
- **Output**: `void`
- **Postcondizione**: Salva i dati dell'evento sul file.

18. `leggi_evento_da_file(file) = evento`

- **Input**: `FILE* file`
- **Precondizione**: `file` deve essere valido.
- **Output**: `Evento`
- **Postcondizione**: Legge e crea un nuovo evento dai dati nel file.

19. `libera_evento(evento) = void`

- **Input**: `Evento evento`
- **Precondizione**: `evento` deve essere valido.
- **Output**: `void`
- **Postcondizione**: Libera la memoria allocata per l'evento.

3.6 Bst_evento.c

Specifica Tipi di Dato

Sintattica:

- Tipo di riferimento: `eventoBst`
- Tipo di riferimento Nodo: `eventoBstNodo`
- Tipi usati: `Evento`, `int`

Semantica:

Il tipo di dato `eventoBst` rappresenta un albero binario di ricerca per gli eventi. Include un puntatore alla radice dell'albero e la dimensione corrente dell'albero. Il tipo di dato `eventoBstNodo` rappresenta un nodo dell'albero binario di ricerca per gli eventi. Ogni nodo contiene un evento, puntatori al nodo genitore, al figlio sinistro e al figlio destro.

Specifica Operatori

Sintattica:

- `nuovo_evento_bst() -> eventoBst`
- `inserisci_evento_bst(eventoBst, Evento) -> int`
- `bst_cerca_evento(eventoBst, Evento) -> eventoBstNodo`

- `bst_cerca_evento_by_id(eventoBst, int) -> eventoBstNodo`
- `bst_rimuovi_nodo(eventoBst, eventoBstNodo) -> Evento`
- `bst_rimuovi_evento(eventoBst, Evento) -> Evento`
- `bst_rimuovi_evento_by_id(eventoBst, int) -> Evento`
- `stampa_evento_bst(eventoBst, ListaSala) -> void`
- `prendi_bst_dimensione(eventoBst) -> int`
- `bst_prendi_evento_by_id(eventoBst, int) -> Evento`
- `salva_evento_bst_su_file(eventoBst, FILE*) -> void`
- `leggi_evento_bst_da_file(FILE*) -> eventoBst`
- `libera_evento_bst(eventoBst) -> void`

Semantica:

1. `nuovo_evento_bst() = nuovo`
   - **Input:** Nessuno.
   - **Precondizione:** Nessuna.
   - **Output:** `eventoBst`
   - **Postcondizione:** Crea e restituisce un nuovo albero binario di ricerca degli eventi vuoto.

2. `inserisci_evento_bst(bst, evento) = int`
   - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), un evento (`Evento`).
   - **Precondizione:** L'albero e l'evento devono essere validi.
   - **Output:** `int`
   - **Postcondizione:** Inserisce un nuovo evento nell'albero binario di ricerca e restituisce 0 se l'inserimento ha avuto successo, altrimenti -1.

3. `bst_cerca_evento(bst, evento) = result`

   - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), un evento (`Evento`).
   - **Precondizione:** L'albero e l'evento devono essere validi.
   - **Output:** `eventoBstNodo`
   - **Postcondizione:** Cerca un nodo contenente l'evento specificato nell'albero e restituisce il nodo se trovato, altrimenti restituisce `NULL`.

4. `bst_cerca_evento_by_id(bst, id) = eventoBstNodo`

   - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), un ID di tipo `int`.
   - **Precondizione:** L'albero deve essere valido.
   - **Output:** `eventoBstNodo`
   - **Postcondizione:** Cerca un nodo contenente l'evento con l'ID specificato nell'albero e restituisce il nodo se trovato, altrimenti restituisce `NULL`.

5. `bst_rimuovi_nodo(bst, nodo) = valore`

   - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), un nodo (`eventoBstNodo`).
   - **Precondizione:** L'albero e il nodo devono essere validi.
   - **Output:** `Evento`
   - **Postcondizione:** Rimuove il nodo dall'albero binario di ricerca degli eventi e restituisce l'evento contenuto nel nodo rimosso.

6. `bst_rimuovi_evento(bst, evento) = evento`

   - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), un evento (`Evento`).
   - **Precondizione:** L'albero e l'evento devono essere validi.
   - **Output:** `Evento`
   - **Postcondizione:** Rimuove l'evento specificato dall'albero binario di ricerca degli eventi e restituisce l'evento rimosso.

7. `bst_rimuovi_evento_by_id(bst, id) = evento`

   - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), un ID di tipo `int`.
   - **Precondizione:** L'albero deve essere valido.
   - **Output:** `Evento`
   - **Postcondizione:** Rimuove l'evento con l'ID specificato dall'albero binario di ricerca degli eventi e restituisce l'evento rimosso.

8. `stampa_evento_bst(bst, lista) = void`

   - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), una lista di sale (`ListaSala`).
   - **Precondizione:** L'albero e la lista di sale devono essere validi.
   - **Output:** `void`
   - **Postcondizione:** Stampa tutti gli eventi nell'albero binario di ricerca degli eventi.

9. `prendi_bst_dimensione(bst) = bst -> dimensione`

   - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`).
   - **Precondizione:** L'albero deve essere valido.
   - **Output:** `int`
   - **Postcondizione:** Restituisce la dimensione (numero di nodi) dell'albero.

10. `bst_prendi_evento_by_id(bst, id) = e -> valore`

    - **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), un ID di tipo `int`.
    - **Precondizione:** L'albero deve essere valido.
    - **Output:** `Evento`
    - **Postcondizione:** Restituisce l'evento con l'ID specificato dall'albero binario di ricerca degli eventi, se presente, altrimenti `NULL`.

11. `salva_evento_bst_su_file(bst, file) = void`

- **Input:** Un albero binario di ricerca degli eventi (`eventoBst`), un puntatore a un file di tipo `FILE*`.
- **Precondizione:** L'albero e il puntatore al file devono essere validi.
- **Output:** `void`
- **Postcondizione:** Salva l'intero albero binario di ricerca degli eventi sul file specificato.

12. `leggi_evento_bst_da_file(file) = bst`

- **Input:** Un puntatore a un file di tipo `FILE*`.
- **Precondizione:** Il puntatore al file deve essere valido.
- **Output:** `eventoBst`
- **Postcondizione:** Legge un albero binario di ricerca degli eventi dal file specificato e lo restituisce.

13. `libera_evento_bst(bst) = void`

- **Input:** Un albero binario di ricerca degli eventi (`eventoBst`).
- **Precondizione:** L'albero deve essere valido.
- **Output:** `void`
- **Postcondizione:** Libera la memoria allocata per l'albero binario di ricerca degli eventi.

3.7 Conferenza.c

Specifica Tipi di Dato

Sintattica:

- Tipo di riferimento: `Conferenza`
- Tipi usati: `eventoBst`, `int`, `ListaSala`

Semantica:

Il tipo di dato `Conferenza` rappresenta una conferenza e include un albero binario di ricerca degli eventi, contatori per gli ID degli eventi e delle sale, e una lista delle sale disponibili.

Specifica Operatori

Sintattica:

- `nuova_conferenza() -> Conferenza`
- `aggiungi_conferenza_evento(Conferenza) -> int`
- `aggiungi_conferenza_sala(Conferenza) -> int`
- `rimuovi_conferenza_evento(Conferenza) -> int`
- `modifica_nome_conferenza_evento(Conferenza, Evento) -> int`
- `modifica_data_inizio_conferenza_evento(Conferenza, Evento) -> int`
- `modifica_data_fine_conferenza_evento(Conferenza, Evento) -> int`
- `modifica_tipo_conferenza_evento(Conferenza, Evento) -> int`
- `modifica_conferenza_evento(Conferenza) -> int`
- `stampa_conferenza(Conferenza) -> int`
- `stampa_conferenza_sale(Conferenza) -> int`
- `conferenza_assegna_evento_a_sala(Conferenza) -> int`
- `salva_conferenza_su_file(Conferenza, FILE*) -> void`
- `leggi_conferenza_da_file(FILE*) -> Conferenza`
- `libera_conferenza(Conferenza) -> void`

Semantica:

1. `nuova_conferenza() = conf`
   - **Input:** Nessuno.
   - **Precondizione:** Nessuna.
   - **Output:** `Conferenza`
   - **Postcondizione:** Crea e restituisce una nuova conferenza con un albero di eventi vuoto e una lista di sale vuota.

2. `aggiungi_conferenza_evento(conf) = int`
   - **Input:** Una conferenza (`Conferenza`).
   - **Precondizione:** La conferenza deve essere valida.
   - **Output:** `int`
   - **Postcondizione:** Aggiunge un nuovo evento alla conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

3. `aggiungi_conferenza_sala(conf) = int`

- **Input:** Una conferenza (`Conferenza`).
- **Precondizione:** La conferenza deve essere valida.
- **Output:** `int`
- **Postcondizione:** Aggiunge una nuova sala alla conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

4. `rimuovi_conferenza_evento(conf) = int`

- **Input:** Una conferenza (`Conferenza`).
- **Precondizione:** La conferenza deve essere valida.
- **Output:** `int`
- **Postcondizione:** Rimuove un evento dalla conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

5. `modifica_nome_conferenza_evento(conf, evento) = int`

- **Input:** Una conferenza (`Conferenza`), un evento (`Evento`).
- **Precondizione:** La conferenza e l'evento devono essere validi.
- **Output:** `int`
- **Postcondizione:** Modifica il nome di un evento nella conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

6. `modifica_data_inizio_conferenza_evento(conf, evento) = int`

- **Input:** Una conferenza (`Conferenza`), un evento (`Evento`).
- **Precondizione:** La conferenza e l'evento devono essere validi.
- **Output:** `int`
- **Postcondizione:** Modifica la data di inizio di un evento nella conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

7. `modifica_data_fine_conferenza_evento(conf, evento) = int`
   - **Input:** Una conferenza (`Conferenza`), un evento (`Evento`).
   - **Precondizione:** La conferenza e l'evento devono essere validi.
   - **Output:** `int`
   - **Postcondizione:** Modifica la data di fine di un evento nella conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

8. `modifica_tipo_conferenza_evento(conf, evento) = int`
   - **Input:** Una conferenza (`Conferenza`), un evento (`Evento`).
   - **Precondizione:** La conferenza e l'evento devono essere validi.
   - **Output:** `int`
   - **Postcondizione:** Modifica il tipo di un evento nella conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

9. `modifica_conferenza_evento(conf) = int`
   - **Input:** Una conferenza (`Conferenza`).
   - **Precondizione:** La conferenza deve essere valida.
   - **Output:** `int`
   - **Postcondizione:** Consente di modificare un evento nella conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

10. `stampa_conferenza(conf) = int`
    - **Input:** Una conferenza (`Conferenza`).
    - **Precondizione:** La conferenza deve essere valida.
    - **Output:** `int`
    - **Postcondizione:** Stampa tutti gli eventi della conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

11. `stampa_conferenza_sale(conf) = int`
    - **Input:** Una conferenza (`Conferenza`).
    - **Precondizione:** La conferenza deve essere valida.
    - **Output:** `int`
    - **Postcondizione:** Stampa tutte le sale della conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

12. `conferenza_assegna_evento_a_sala(conf) = int`

- **Input:** Una conferenza (`Conferenza`).
- **Precondizione:** La conferenza deve essere valida.
- **Output:** `int`
- **Postcondizione:** Assegna un evento a una sala nella conferenza e restituisce 0 se l'operazione ha successo, -1 altrimenti.

13. `salva_conferenza_su_file(conf, file) = void`

- **Input:** Una conferenza (`Conferenza`), un puntatore a un file di tipo `FILE*`.
- **Precondizione:** La conferenza e il file devono essere validi.
- **Output:** `void`
- **Postcondizione:** Salva la conferenza su un file.

14. `leggi_conferenza_da_file(file) = conf`

- **Input:** Un puntatore a un file di tipo `FILE*`.
- **Precondizione:** Il file deve essere valido.
- **Output:** `Conferenza`
- **Postcondizione:** Legge la conferenza da un file e la restituisce.

15. `libera_conferenza(conf) = void`

- **Input:** Una conferenza (`Conferenza`).
- **Precondizione:** La conferenza deve essere valida.
- **Output:** `void`
- **Postcondizione:** Libera la memoria allocata per la conferenza.

Capitolo 4

Razionale dei Casi di Test

4.1 Utilizzo dei file

La test suite prevede diversi testcase per provare diverse condizioni di utilizzo della aggiunta, rimozione, modifica e visualizzazione di un evento.
Per ogni caso di test sull'aggiunta e sulla visualizzazione usiamo tre file, per i casi di test restanti ne usiamo, invece, quattro:

1. Un file di input `input.txt` contenente i valori o le operazioni da effettuare riguardante una conferenza
2. Un file di input `oracle.txt` contenente ciò che ci si aspetta di ottenere
3. un file di output `output.txt` risultante dall'esecuzione del programma (output effettivo)
4. un file `conferenza.txt` contente le informazioni relative ad una conferenza.

Inoltre, usiamo due file aggiuntivi:

1. un file di input `test_suite.txt` che indica per ogni test case, l'identificativo del test case e il numero dell'operazione da effettuare
   - TC1_add 1 - aggiunta
   - TC1_remove 2 - rimozione
   - TC1_modify 3 - modifica
   - TC1_show 4 - visualizzazione
2. un file di output `result.txt` che memorizza l'esito di ogni test case (`PASS` / `FAIL`)

4.1.1 Struttura dei file per l'aggiunta di un evento

Il file `input.txt` prevede la segeunte formattazione:

- **Nome evento**: Una stringa che rappresenta il nome dell'evento.
- **Tipologia evento**: Un intero che indica la tipologia dell'evento (valori tra 1 e 3).
- **Data inizio evento**: La data e l'ora di inizio dell'evento, formattata come `DD/MM/YYYY HH:MM`.
- **Data fine evento**: La data e l'ora di fine dell'evento, formattata come `DD/MM/YYYY HH:MM`.

`-Esempio:`

```
Evento 1
1
21/05/2024 13:45
21/05/2024 15:00
```

Mentre i file `conferenza.txt` e quelli di output e oracle (tranne nei test di visualizzazione del programma della conferenza) sono strutturati nel seguente modo:

1. Le prime due righe rappresentano gli inizializzatori degli id per eventi e sale
2. Successivamente vi è il numero di eventi presenti nella conferenza.
3. Per ciascun evento, vengono forniti i seguenti dettagli:
   - Id dell'evento, tipo dell'evento (indicato con un numero tra 1 e 3), id della sala a cui è assegnato l'evento e nome dell'evento
   - Data di inizio dell'evento (espressa come "mm hh GG MM AAAA")
   - Data di fine dell'evento (espressa come "mm hh GG MM AAAA")
4. Dopo la lista degli eventi, viene fornito il numero di sale presenti nella conferenza.
5. Infine per ciascuna sala viene fornito l'id e il nome.

`-Esempio:`

```
4 4
4
0 1 1 Evento 1
0 15 31 5 2024
0 17 31 5 2024
```

```
1 3 2 Evento 2
0 18 25 6 2025
0 19 25 6 2025
2 1 0 Evento 3
0 19 26 9 2024
0 20 26 9 2024
3 3 0 Evento 4
0 18 25 4 2026
0 20 25 4 2026
3
1
Sala 1
2
Sala 2
3
Sala 3
```

4.1.2 Struttura dei file per la rimozione di un evento

I file conferenza.txt, output.txt e oracle.txt sono formattati come precedentemente descritto.

-Esempio:

Dato il seguente file conference.txt:

```
3 4
3
0 1 1 Evento 1
0 15 31 5 2024
0 17 31 5 2024
1 2 3 Evento 2
45 13 1 6 2024
0 14 1 6 2024
2 3 2 Evento 3
0 18 25 6 2025
0 19 25 6 2025
3
1
Sala 1
2
Sala 2
3
Sala 3
```

Il formato del file input.txt contiene l'id dell'evento da rimuovere ed è descritto come segue:

```
1
```

In questo esempio 1 indica quindi l'id dell'Evento 2.
Il file output.txt sarà quindi:

```
3 4
2
0 1 1 Evento 1
0 15 31 5 2024
0 17 31 5 2024
2 3 2 Evento 3
0 18 25 6 2025
0 19 25 6 2025
3
1
Sala 1
2
Sala 2
3
Sala 3
```

Gli id non validi vengono ignorati e la conferenza non viene cambiata

4.1.3 Struttura dei file per la modifica di un evento

I file conferenza.txt, output.txt e oracle.txt sono formattati come precedentemente descritto.

-Esempio:

Dato il seguente file conference.txt:

```
2 1
2
0 1 0 Evento 1
0 14 21 5 2024
0 15 21 5 2024
1 2 0 Evento 2
0 19 21 5 2024
0 20 21 5 2024
0
```

Il formato del file input.txt contiene invece:

- id dell'evento che si desidera modificare

- intero che permette di scegliere cosa modificare
  - 1 - nome
  - 2 - data inizio
  - 3 - data fine
  - 4 - tipo
  - 5 - per uscire dal menù
- la nuova scelta
- intero per uscire dal menù di scelta

Si avrà perciò:

```
1
1
Evento modificato
5
```

Il file output.txt sarà quindi:

```
2 1
2
0 1 0 Evento 1
0 14 21 5 2024
0 15 21 5 2024
1 2 0 Evento modificato
0 19 21 5 2024
0 20 21 5 2024
0
```

4.1.4 Struttura dei file per la visualizzazione di un evento

I file conferenza.txt è una conferenza formattata come precedentemente descritto. I file output.txt e oracle.txt sono elenchi di eventi mentre il file input.txt non viene utilizzato.
I test stampano gli eventi presenti nel file conferenza.txt in ordine cronologico.

`-Esempio:`

Dato il seguente file conference.txt:

```
3 4
3
0 1 1 Evento 1
0 15 31 5 2024
0 17 31 5 2024
1 2 3 Evento 2
```

```
45 13 1 6 2024
0 14 1 6 2024
2 3 2 Evento 3
0 18 25 6 2025
0 19 25 6 2025
3
1
Sala 1
2
Sala 2
3
Sala 3
```

Il file output.txt sarà quindi:

```
ID evento:> 0
Nome evento:> Evento 1
Tipologia evento:> Workshop
Data inizio:> 31/5/2024, 15:00
Data fine:> 31/5/2024, 17:00
Sala:> Sala 1


ID evento:> 1
Nome evento:> Evento 2
Tipologia evento:> Sessione di keynote
Data inizio:> 1/6/2024, 13:45
Data fine:> 1/6/2024, 14:00
Sala:> Sala 3


ID evento:> 2
Nome evento:> Evento 3
Tipologia evento:> Panel di discussione
Data inizio:> 25/6/2025, 18:00
Data fine:> 25/6/2025, 19:00
Sala:> Sala 2
```

4.2 Test-case

A tal proposito si riportano dei test-case realizzati:

Casi di test relativi l'aggiunta:

- **TC1_add:** aggiunta regolare di un evento
- **TC2_add:** campo nome vuoto

- **TC3_add:** nome inserito più lungo di 100 caratteri
- **TC4_add:** nome inserito uguale a 100 caratteri
- **TC5_add:** tipo inserito uguale a -1 (minore di 0)
- **TC6_add:** tipo inserito uguale a 4 (maggiore di 3)
- **TC7_add:** giorno inserito 00 (minore di 1)
- **TC8_add:** giorno inserito 32 (maggiore di 31)
- **TC9_add:** mese inserito uguale a 13
- **TC10_add:** mese inserito uguale a 00

Casi di test relativi la modifica:

- **TC11_modify:** modifica nome di un evento
- **TC12_modify:** modifica tipo di un evento
- **TC13_modify:** modifica data fine di un evento
- **TC14_modify:** modifica data inizio di un evento

Casi di test relativi la rimozione:

- **T15_remove:** rimozione evento con id 1
- **T16_remove:** input -1 (annulla l'operazione di rimozione)

Casi di test relativi la visualizzazione:

- **T17_show:** visualizzazione agenda conferenze
- **T18_show:** visualizzazione degli eventi seguendo l'ordine delle date.