UNIVERSITÀ DEGLI STUDI DI SALERNO

Progetto Programmazione e Strutture Dati

Robustelli Renato
Squitieri Andrea
Vitolo Angelo

INDICE

1 Introduzione **4**
1.1 Obiettivi del progetto 4

2 Motivazione della scelta degli ADT **4**
2.1 Motivazioni per l'uso dell'ADT RoomList 4
2.2 Motivazioni per l'uso dell'ADT EventBst 5

3 Progettazione **5**
3.1 Moduli Principali 5
3.2 Gestione delle Date e Orari (Date) 6
3.3 Gestione delle Sale (Room e RoomList) 6
3.4 Gestione degli Eventi (Event e EventBst) 6
3.5 Coordinamento della Conferenza (Conference) 6

4 Specifiche sintattiche e semantiche **7**
4.1 Funzioni e tipi di logging 7
4.1.1 Specificazione del tipo `LogLevel` 7
4.1.2 Specificazione di `void set_log_file(FILE*)` 7
4.1.3 Specificazione di `void log_message(LogLevel, char*)` 7
4.1.4 Specificazione di `void log_error(char*)` 8
4.2 Funzioni e tipi di utilità 8
4.2.1 Specificazione del tipo `ResultInt` 8
4.2.2 Funzione `void clean_file(FILE*)` 9
4.2.3 Funzione `int read_line_from_file(char*, int, FILE*)` 9
4.2.4 Funzione `int read_line(char*, int)` 9
4.2.5 Funzione `ResultInt read_int(void)` 10
4.2.6 Funzione `void my_alloc(unsigned long, unsigned long)` 10
4.2.7 Funzione `void my_realloc(void*, unsigned long, unsigned long)` 10
4.2.8 Funzione `char* my_strdup(char*)` 11

4.2.9 Funzione void trim_whitespaces(char*, char*, int) 11
4.3 Specifica dell'ADT Date 11
4.3.1 Funzione Date new_date(unsigned char, unsigned char, unsigned char, unsigned short) 12
4.3.2 Funzione Date copy_date(Date) 12
4.3.3 Funzione int cmp_date(Date, Date) 12
4.3.4 Funzione void save_date_to_file(Date, FILE*) 12
4.3.5 Funzione Date read_date_from_file(FILE*) 13
4.3.6 Funzione void print_date(Date) 13
4.3.7 Funzione: Date read_date(void) 13
4.3.8 Funzione: void free_date(Date) 14
4.4 Specifica dell'ADT Room 14
4.4.1 Funzione: Room new_room(char*, unsigned int, unsigned int) . . 14
4.4.2 Funzione: Room copy_room(Room) 15
4.4.3 Funzione: bool are_rooms_equal(Room, Room) 15
4.4.4 Funzione: char* get_room_name(Room) 15
4.4.5 Funzione: unsigned int get_room_id(Room) 15
4.4.6 Funzione: void print_room(Room) 16
4.4.7 Funzione: Room read_room(unsigned int) 16
4.4.8 Funzione: void save_room_to_file(Room, FILE*) 16
4.4.9 Funzione: Room read_room_from_file(FILE*) 16
4.4.10 Funzione: void free_room(Room) 17
4.5 Specifica dell'ADT Event 17
4.5.1 Funzione: void is_valid_event_type(int) 18
4.5.2 Funzione: Event new_event(EventType, char *, Date, Date, unsigned int) 18
4.5.3 Funzione: int are_events_equal(Event, Event) 18
4.5.4 Funzione: int cmp_event(Event, Event) 18
4.5.5 Funzione: bool do_events_overlap(Event, Event) 19
4.5.6 Funzione: EventType get_event_type(Event event) 19
4.5.7 Funzione: int set_event_type(Event event, EventType type) . . . 19
4.5.8 Funzione: Date get_event_start_date(Event event) 20
4.5.9 Funzione: int set_event_start_date(Event event, Date start_date) 20
4.5.10 Funzione: Date get_event_end_date(Event event) 20
4.5.11 Funzione: int set_event_end_date(Event event, Date end_date) 20
4.5.12 Funzione: char *get_event_name(Event event) 21
4.5.13 Funzione: int set_event_name(Event event, char *name) 21
4.5.14 Funzione: unsigned int get_event_room_id(Event event) 21
4.5.15 Funzione: int set_event_room_id(Event event, unsigned int room_id) 21
4.5.16 Funzione: unsigned int get_event_id(Event event) 22
4.5.17 Funzione: void print_event(Event event, Room assigned_room) . 22
4.5.18 Funzione: Event read_event(unsigned int event_id) 22
4.5.19 Funzione: void save_event_to_file(Event event, FILE *file) . . 22
4.5.20 Funzione: Event read_event_from_file(FILE *file) 23
4.5.21 Funzione: void free_event(Event event) 23
4.6 Specifica dell'ADT EventBst 23
4.6.1 Funzione: EventBst new_event_bst 24
4.6.2 Funzione: int bst_insert_event(EventBst bst, Event event) . . . 24
4.6.3 Funzione: Event bst_remove_event(EventBst bst, Event event) . 24

4.6.4 Funzione: `Event bst_remove_event_by_id(EventBst bst, unsigned int id)` . . . 24
4.6.5 Funzione: `Event bst_get_event_by_id(EventBst bst, unsigned int id)` . . . 25
4.6.6 Funzione: `size_t get_bst_size(EventBst bst)` . . . 25
4.6.7 Funzione: `bool print_event_bst(EventBst bst, RoomList room_list)` 25
4.6.8 Funzione: `bool event_bst_every(EventBst, EventPredicate, ...)` 26
4.6.9 Funzione: `EventBst read_event_bst_from_file(FILE* file)` . . . 26
4.6.10 Funzione: `void save_event_bst_to_file(EventBst bst, FILE* file)` 26
4.6.11 Funzione: `void save_event_bst_to_file_sorted(EventBst bst, FILE* file)` . . . 27
4.6.12 Funzione: `void free_event_bst(EventBst bst)` . . . 27
4.7 Specifica dell'ADT RoomList . . . 27
4.7.1 Funzione: `RoomList new_room_list(void)` . . . 28
4.7.2 Funzione: `bool is_room_list_empty(RoomList list)` . . . 28
4.7.3 Funzione: `int get_size_room_list(RoomList list)` . . . 28
4.7.4 Funzione: `void cons_room_list(RoomList list, Room room)` . . . 29
4.7.5 Funzione: `Room tail_room_list(RoomList list)` . . . 29
4.7.6 Funzione: `Room get_first_room_list(RoomList list)` . . . 29
4.7.7 Funzione: `Room get_at_room_list(RoomList list, int pos)` . . . 29
4.7.8 Funzione: `Room get_room_by_id(RoomList list, unsigned int room_id)` 30
4.7.9 Funzione: `Room remove_at_room_list(RoomList list, int pos)` . . 30
4.7.10 Funzione: `int get_pos_room_list(RoomList list, Room to_search)` 30
4.7.11 Funzione: `void print_room_list(RoomList list)` . . . 30
4.7.12 Funzione: `void save_room_list_to_file(RoomList list, FILE* file)` 31
4.7.13 Funzione: `RoomList read_room_list_from_file(FILE* file)` . . . 31
4.7.14 Funzione: `void free_room_list(RoomList list)` . . . 31
4.8 Specifica dell'ADT Conference . . . 31
4.8.1 Funzione: `Conference new_conference(void)` . . . 32
4.8.2 Funzione: `int add_conference_event(Conference conf)` . . . 32
4.8.3 Funzione: `int edit_conference_event(Conference conf)` . . . 32
4.8.4 Funzione: `int remove_conference_event(Conference conf)` . . . 33
4.8.5 Funzione: `void display_conference_schedule(Conference conf)` . 33
4.8.6 Funzione: `void display_conference_rooms(Conference conf)` . . . 33
4.8.7 Funzione: `int add_conference_room(Conference conf)` . . . 34
4.8.8 Funzione: `int remove_conference_room(Conference conf)` . . . 34
4.8.9 Funzione: `int conference_assign_event_to_room(Conference conf)` 34
4.8.10 Funzione: `int conference_free_event_room(Conference conf)` . . . 34
4.8.11 Funzione: `void free_conference(Conference conf)` . . . 35
4.8.12 Funzione: `void save_conference_to_file(Conference conf, FILE* file)` . . . 35
4.8.13 Funzione: `void save_conference_to_file_sorted(Conference conf, FILE* file)` . . . 35
4.8.14 Funzione: `Conference read_conference_from_file(FILE* file)` . . 35

5 Razionale dei casi di test **36**
5.1 Tipi di Test . . . 36
5.2 Funzionamento Generale . . . 36
5.3 Formato file conferenze . . . 36
5.4 Test per l'aggiunta di un evento . . . 37
5.4.1 Formato del file `input.txt` per i Test di Aggiunta Evento . . . 38

5.4.2 Casi di test relativi al nome dell'evento 38
5.4.3 Casi di test relativi alla tipologia dell'evento 39
5.4.4 Casi di test relativi alle date dell'evento 39
5.5 Test per la rimozione di un evento 40
5.5.1 Formato del file `input.txt` per i Test di Rimozione Evento 40
5.5.2 Casi di test 41
5.6 Test per la modifica di un evento 41
5.6.1 Formato del file `input.txt` per i Test di Modifica Evento 41
5.6.2 Casi di test 42
5.7 Test per l'assegnazione di una sala a un evento 43
5.7.1 Formato del file `input.txt` per i Test di assegnazione di una sala a un evento 43
5.7.2 Casi di test 44
5.8 Test per la liberazione di una sala assegnata a un evento 44
5.8.1 Formato del file `input.txt` per i Test di liberazione di una sala assegnata a un evento 44
5.8.2 Casi di test 45
5.9 Test per la visualizzazione del programma dela conferenza 45
5.9.1 Casi di test 46

6 Compilazione e esecuzione programmi 47
6.1 Compilazione e esecuzione dell'eseguibile principale 47
6.2 Compilazione e esecuzione del programma di testing 47

1 INTRODUZIONE

Il progetto si concentra sulla creazione di un sistema di gestione delle conferenze, con l'obiettivo di organizzare e gestire eventi e stanze in modo efficiente. Il sistema è progettato per facilitare la pianificazione, la modifica, la visualizzazione e l'archiviazione di eventi e stanze all'interno di una conferenza.

1.1 Obiettivi del progetto

Gli obiettivi principali del progetto includono:

- Gestione di Conferenze: Possibilità di aggiungere, modificare e rimuovere eventi e stanze di una conferenza.
- Assegnazione delle Stanze: Funzionalità per assegnare stanze a specifici eventi.
- Visualizzazione delle Informazioni: Strumenti per visualizzare gli eventi ordinati per data d'inizio.
- Persistenza dei Dati: Capacità di salvare e caricare i dati delle conferenze da file per garantire la persistenza delle informazioni.

2 MOTIVAZIONE DELLA SCELTA DEGLI ADT

2.1 Motivazioni per l'uso dell'ADT RoomList

Abbiamo deciso di utilizzare una lista per gestire le informazioni sulle sale all'interno del sistema di gestione delle conferenze per diverse ragioni. Le liste offrono flessibilità nella gestione dinamica delle sale, consentendo l'aggiunta e la rimozione efficiente di elementi in risposta

alle variazioni nel numero di sale nel tempo. Inoltre, semplificano l'accesso e la ricerca delle sale, consentendo operazioni come l'ordinamento o la ricerca di una specifica sala.
Per implementare l'ADT RoomList, abbiamo scelto di utilizzare un array dinamico anziché una lista concatenata. Questa decisione è stata motivata da diverse considerazioni. Gli array dinamici offrono un accesso casuale più efficiente rispetto alle liste concatenate, il che si traduce in un'operatività più veloce e prevedibile per le operazioni che richiedono l'accesso non sequenziale alle sale nella lista.
Inoltre, gli array dinamici consentono un accesso continuo e contiguo alla memoria, migliorando la località dei dati e riducendo il numero di cache miss durante l'accesso ai dati nella lista. Questo porta a prestazioni ottimali in termini di accesso ai dati e di utilizzo della memoria.
Infine, l'utilizzo della memoria è più efficiente con gli array dinamici rispetto alle liste concatenate. Mentre una lista concatenata avrebbe utilizzato sempre il doppio della memoria necessaria (un puntatore al valore e un puntatore al prossimo nodo), la nostra implementazione utilizza **al massimo** il doppio della memoria grazie all'utilizzo di un fattore di crescita per l'array dinamico che garantisce inoltre un numero limitato di spostamenti di dati durante il processo di reallocazione (il numero di spostamenti effettuati sarà sempre inferirore a 2n, dove n rappresenta il numero di sale).

2.2 Motivazioni per l'uso dell'ADT EventBst

Per gestire in modo ordinato la raccolta di eventi all'interno del sistema, abbiamo adottato un albero binario di ricerca. Questa scelta consente di aggiungere eventi con una complessità temporale media di $O(\log n)$. Tuttavia, per garantire un'usabilità ottimale del software, le operazioni di modifica e rimozione degli eventi non sfruttano le proprietà dell'albero e sono quindi di complessità $O(n)$, poiché precedute da una ricerca lineare per l'ID dell'evento.
Nonostante questa considerazione, riteniamo che l'utilizzo dell'albero di ricerca binario sia giustificato, dato che rappresenta un buon compromesso da un punto di vista computazionale. Altre strutture ordinate avrebbero mantenuto comunque una complessità temporale di $O(n)$ per le operazioni di modifica e rimozione, rendendo però lineare anche la complessità dell'aggiunta degli eventi. La preferenza per una struttura ordinata è derivata dalla frequenza delle operazioni di visualizzazione degli eventi nel software, evitando così la necessità di ordinare la struttura prima di ogni operazione di visualizzazione, che potrebbe causare rallentamenti.

3 PROGETTAZIONE

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.

3.1 Moduli Principali

I principali moduli del sistema sono:

- **Date** (date.h): Gestisce le date e gli orari degli eventi.
- **Room** (room.h): Rappresenta le sale dove si tengono gli eventi.
- **RoomList** (room_list.h): Gestisce una lista dinamica di sale.
- **Event** (mevent.h): Rappresenta gli eventi della conferenza.
- **EventBst** (event_bst.h): Organizza gli eventi in un albero binario di ricerca per una gestione efficiente.

- **Conference** (conference.h): Coordina tutti gli elementi sopra menzionati, fornendo un'interfaccia unificata per la gestione della conferenza.

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.

3.2 Gestione delle Date e Orari (Date)

Il modulo **Date** è impiegato dai moduli **Event** e **Conference** per la gestione delle operazioni relative alle date e agli orari, come la creazione, il confronto e la validazione. Ad esempio, quando si crea un nuovo evento tramite il modulo **Event**, le funzioni di **Date** vengono utilizzate per definire con precisione la data e l'orario di inizio e fine. Questo modulo garantisce la validità delle date assegnate agli eventi, prevenendo conflitti temporali. Inoltre, permette di verificare che la data di inizio e quella di fine di un evento siano appropriate e che non vi siano due eventi programmati nella stessa sala nello stesso periodo.

3.3 Gestione delle Sale (Room e RoomList)

Le stanze, trattate come istanze dell'ADT **Room**, sono gestite all'interno di una struttura dati dinamica denominata **RoomList**. Questo approccio consente di gestire l'aggiunta, la rimozione e la ricerca delle stanze in modo efficiente. Attraverso questi due moduli, è possibile associare una stanza a un evento; all'interno della rappresentazione di un evento, la stanza è identificata dal suo ID, che viene utilizzato per individuarla all'interno della **RoomList** e recuperarne le informazioni pertinenti.

3.4 Gestione degli Eventi (Event e EventBst)

I singoli eventi sono rappresentati come istanze dell'ADT **Event** e vengono organizzati all'interno di una struttura gerarchica nota come **EventBst**, un albero binario di ricerca specializzato. Questo albero consente di gestire l'aggiunta, la ricerca e la rimozione degli eventi in modo relativamente efficiente.
All'atto della creazione di un nuovo evento, quest'ultimo viene inserito nell'albero binario di ricerca, garantendo un ordinamento basato sulla data di inizio dell'evento. In caso di eventi con la stessa data di inizio, l'ordinamento viene effettuato in base al nome dell'evento.
Le operazioni di modifica e rimozione degli eventi vengono eseguite attraverso una ricerca lineare dell'ID dell'evento all'interno dell'albero binario di ricerca. Gli eventi contengono al loro interno un ID che indica la sala a loro assegnata. Nel caso in cui non ci sia una sala assegnata all'evento, questo ID viene impostato al valore **"NULL_ROOM_ID"** definito nel modulo **Room**.
Ogni evento possiede una data di inizio e fine, rappresentate attraverso istanze dell'ADT **Date**.
La tipologia dell'evento è rappresentata nel modulo attraverso l'enumerazione **EventType**.

3.5 Coordinamento della Conferenza (Conference)

Il modulo **Conference** funge da coordinatore centrale. Esso mantiene un riferimento all'albero degli eventi (**EventBst**) e alla lista delle sale (**RoomList**). Gestisce anche gli inizializzatori degli ID per assegnare identificativi univoci agli eventi e alle sale. Quando un utente interagisce con il sistema per creare, modificare o rimuovere un evento o una sala, il modulo **Conference** invoca le appropriate funzioni degli altri moduli per eseguire le operazioni richieste. Garantisce quindi che tutti gli eventi e le sale abbiano ID univoci e che non ci siano sovrapposizioni temporali tra due eventi assegnati alla stessa sala. Permette anche di salvare la sessione e leggere da file quelle precedenti. In questo modulo abbiamo quindi implementato le seguenti operazioni:

- **Aggiunta di un evento**: Vengono chiesti i dati dell'evento all'utente, viene creato l'oggetto Event, assegnandogli un nuovo ID, e viene aggiunto all'albero degli eventi.
- **Rimozione di un evento**: Viene mostrata la lista ordinata degli eventi e si chiede all'utente di inserire l'ID dell'evento che si desidera rimuovere. Viene poi rimosso l'evento dall'albero degli eventi.
- **Modifica di un evento**: Viene mostrata la lista ordinata degli eventi e si chiede all'utente di inserire l'ID dell'evento che si desidera modificare. Viene poi rimosso l'evento selezionato, apportate le modifiche richieste dall'utente e reinserito nell'albero degli eventi.
- **Mostra eventi**: Viene mostrata la lista ordinata degli eventi.
- **Aggiunta di una sala**: Vengono chiesti i dati della sala all'utente, viene creato l'oggetto Room, assegnandogli un nuovo ID, e viene aggiunto alla lista delle sale.
- **Mostra sala**: Viene mostrata la lista delle sale
- **Assegnazione di una sala ad un evento**: Viene mostrata la lista ordinata degli eventi e si chiede all'utente di inserire l'ID dell'evento a cui si desidera assegnare la sala. Viene poi mostrata la lista delle sale e si chiede all'utente di inserire l'ID della sala da assegnare. Viene verifacto poi che non ci siano conflitti con gli orari di altri eventi assegnati alla stessa sala e in caso positivo viene modificato il campo room_id dell'evento selezionato.
- **Liberare evento da sala**: Viene mostrata la lista ordinata degli eventi e si chiede all'utente di inserire l'ID dell'evento che si desidera liberare. Viene quindi impostato l'ID della sala contenuto nell'evento a "**NULL_ROOM_ID**".

4 Specifiche sintattiche e semantiche

4.1 Funzioni e tipi di logging

4.1.1 Specificazione del tipo `LogLevel`

Definisce un tipo enumerato chiamato `LogLevel` con i seguenti valori:

- `LOG_INFO`: Messaggi informativi.
- `LOG_WARN`: Messaggi di avviso.
- `LOG_ERROR`: Messaggi di errore.

4.1.2 Specificazione di `void set_log_file(FILE*)`

Specificazione Sintattica:

- `set_log_file(FILE*)`
- tipi: `FILE*`
- tipi interni: Nessuno

Specificazione Semantica:

- `set_log_file(file)`
- Imposta il file di log al file specificato.
- Precondizioni: `file` è un puntatore `FILE` valido aperto in modalità scrittura.
- Postcondizioni: Le funzioni di log scriveranno nel file specificato. Se `file` è `NULL`, le funzioni di log non scriveranno in nessun file.

4.1.3 Specificazione di `void log_message(LogLevel, char*)`

Specificazione Sintattica:

- `log_message(LogLevel, char*)`
- tipi: `LogLevel, char*`
- tipi interni: `time_t`, `struct tm`, `char[]`, `FILE*`

Specificazione Semantica:

- `log_message(level, message)`
- Registra un messaggio con il livello di log specificato.
- Precondizioni:
  - `level` è uno tra `LOG_INFO`, `LOG_WARN`, `LOG_ERROR`.
  - `message` è un puntatore non nullo a una stringa terminata da null.
  - Il file di log è stato impostato usando `set_log_file`.
- Postcondizioni:
  - Il messaggio è scritto nel file di log con un prefisso che include timestamp e livello di log.
  - Se il file di log non è impostato, non sarà generato alcun output.

4.1.4 Specificazione di `void log_error(char*)`

Specificazione Sintattica:

- `log_error(char*)`
- tipi: `char*`
- tipi interni: Nessuno

Specificazione Semantica:

- `log_error(message)`
- Registra un messaggio di errore.
- Precondizioni:
  - `message` è un puntatore non nullo a una stringa.
  - Il file di log è stato impostato usando `set_log_file`.
- Postcondizioni:
  - Il messaggio di errore è scritto nel file di log con un prefisso che include timestamp e la dicitura "ERROR".
  - Se il file di log non è impostato, non sarà generato alcun output.

4.2 Funzioni e tipi di utilità

4.2.1 Specificazione del tipo `ResultInt`

Specificazione Sintattica:

- Tipo di riferimento: ResultInt
- Tipi utilizzati: int
- Campi:
  - error_code: Codice di errore: 0 per successo, valori negativi per gli errori.
  - value: Valore intero risultante.

Specificazione Semantica:
Rappresenta il valore intero di ritorno di un'operazione che potrebbe fallire.

4.2.2 Funzione `void clean_file(FILE*)`

Specificazione Sintattica:

- `clean_file(FILE*)`
- Tipi: `FILE*`
- Tipi interni: Nessuno

Specificazione Semantica:

- `clean_file(file)`
- Scarta i caratteri dal flusso del file fino a raggiungere una nuova riga o la fine del file.
- Precondizioni: 'file' è un puntatore FILE valido aperto in modalità lettura.
- Postcondizioni: Il puntatore del file viene avanzato oltre i caratteri scartati.

4.2.3 Funzione `int read_line_from_file(char*, int, FILE*)`

Specificazione Sintattica:

- `read_line_from_file(char*, int, FILE*) -> int`
- Tipi: `char*, int, FILE*`
- Tipi interni: Nessuno

Specificazione Semantica:

- `read_line_from_file(line, size, file) -> res`
- Legge una riga dal file specificato nel buffer, assicurandosi che ci sia spazio per essa e gestendo eventuali errori.
- Precondizioni: 'line' è un buffer valido di dimensione 'size'. 'file' è un puntatore FILE valido aperto in modalità lettura.
- Postcondizioni: In caso di successo, la riga viene memorizzata in 'line' senza il carattere di nuova riga. In caso di fallimento, restituisce -1 se viene raggiunta la fine del file, -2 se la riga è troppo lunga e -3 se si verifica un errore di lettura.

4.2.4 Funzione `int read_line(char*, int)`

Specificazione Sintattica:

- `read_line(char*, int) -> int`
- Tipi: `char*, int`
- Tipi interni: Nessuno

Specificazione Semantica:

- `read_line(line, size) -> res`
- Legge una riga dallo standard input nel buffer.
- Precondizioni: 'line' è un buffer valido di dimensione 'size'.
- Postcondizioni: In caso di successo, la riga viene memorizzata in 'line' senza il carattere di nuova riga. In caso di errore, restituisce -1 se si raggiunge la fine del file, -2 se la riga è troppo lunga e -3 se si verifica un errore di lettura.

4.2.5 Funzione `ResultInt read_int(void)`

Specificazione Sintattica:

- `read_int(void) -> ResultInt`
- Tipi: `ResultInt`
- Tipi interni: `int`, `char[]`, `char*`, `long`, `errno_t`

Specificazione Semantica:

- `read_int() -> res`
- Legge un intero dallo standard input e gestisce gli errori.
- Precondizioni: Nessuna.
- Postcondizioni: In caso di successo, restituisce un `ResultInt` con il valore dell'intero e `error_code` pari a 0. In caso di errore, `error_code` viene impostato a -1 per errori di lettura, -2 per errori di intervallo e -3 per input non valido.

4.2.6 Funzione `void my_alloc(unsigned long, unsigned long)`

Specificazione Sintattica:

- `my_alloc(unsigned long, unsigned long)`
- Tipi: `unsigned long`
- Tipi interni: `void*`

Specificazione Semantica:

- `my_alloc(nmemb, size)`
- Alloca memoria per un array di 'nmemb' elementi, ciascuno di dimensione 'size' byte e inizializza tutti i byte a zero.
- Precondizioni: 'nmemb' e 'size' devono essere diversi da zero.
- Postcondizioni: In caso di successo, restituisce un puntatore alla memoria allocata. In caso di errore, registra un errore e termina il programma.

4.2.7 Funzione `void my_realloc(void*, unsigned long, unsigned long)`

Specificazione Sintattica:

- `my_realloc(void*, unsigned long, unsigned long)`
- Tipi: `void*`, `unsigned long`
- Tipi interni: `void*`

Specificazione Semantica:

- `my_realloc(p, nmemb, size)`
- Rialloca la memoria per un array di 'nmemb' elementi, ciascuno di dimensione 'size' byte.
- Precondizioni: 'p' è un puntatore a un blocco di memoria precedentemente allocato dinamicamente o NULL. 'nmemb' e 'size' devono essere diversi da zero.
- Postcondizioni: In caso di successo, restituisce un puntatore alla memoria riallocata. In caso di errore, registra un errore e termina il programma.

4.2.8 Funzione `char* my_strdup(char*)`

Specificazione Sintattica:

- `my_strdup(char*) -> char*`
- Tipi: `char*`
- Tipi interni: `char*`

Specificazione Semantica:

- `my_strdup(string) -> res_string`
- Duplica la stringa data.
- Precondizioni: 'string' è una stringa valida terminata da '\0'.
- Postcondizioni: In caso di successo, restituisce un puntatore alla stringa duplicata. In caso di errore, registra un errore e termina il programma.

4.2.9 Funzione `void trim_whitespaces(char*, char*, int)`

Specificazione Sintattica:

- `trim_whitespaces(char*, char*, int)`
- Tipi: `char*`, `char*`, `int`
- Tipi interni: Nessuno

Specificazione Semantica:

- `trim_whitespaces(dest, src, max_size)`
- Rimuove gli spazi vuoti iniziali e finali dalla stringa di origine 'src' e la copia nel buffer di destinazione 'dest'.
- Precondizioni: 'dest' e 'src' sono puntatori validi, 'max_size' è la dimensione di 'dest'.
- Postcondizioni: 'dest' contiene la stringa senza spazi vuoti. Se 'max_size' è 0, la funzione non fa nulla.

4.3 Specifica dell'ADT Date

Specificazione Sintattica:

- Tipo di riferimento: Date
- Tipi utilizzati: unsigned char, unsigned short

Specificazione Semantica:
Date rappresenta una data valida con precisione al minuto.

Operatori:

- `new_date(unsigned int, unsigned int, unsigned int) -> Date`
- `copy_date(Date) -> Date`
- `cmp_date(Date, Date) -> int`
- `save_date_to_file(Date, FILE*) -> void`
- `read_date_from_file(FILE*) -> Date`
- `print_date(Date) -> void`
- `read_date(unsigned int) -> Date`
- `free_date(Date) -> void`

4.3.1 Funzione `Date new_date(unsigned char, unsigned char, unsigned char, unsigned char, unsigned short)`

Specificazione Sintattica:

- `new_date(unsigned char, unsigned char, unsigned char, unsigned char, unsigned short) -> Date`
- Tipi: `unsigned char`, `unsigned short`
- Tipi interni: `Date`

Specificazione Semantica:

- `new_date(minutes, hour, day, month, year) -> return_date`
- Crea un nuovo oggetto `Date` con i componenti specificati.
- Precondizioni: Nessuna.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Date` creato o NULL se la data non è valida. Il chiamante è responsabile di liberare la memoria allocata utilizzando `free_date()`.

4.3.2 Funzione `Date copy_date(Date)`

Specificazione Sintattica:

- `copy_date(Date) -> Date`
- Tipi: `Date`
- Tipi interni: Nessuno

Specificazione Semantica:

- `copy_date(date) -> return_date`
- Crea una copia dell'oggetto `Date` fornito.
- Precondizioni: 'date' è un oggetto `Date` valido.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Date` creato, che è una copia di 'date'. Il chiamante è responsabile di liberare la memoria allocata utilizzando `free_date()`.

4.3.3 Funzione `int cmp_date(Date, Date)`

Specificazione Sintattica:

- `cmp_date(Date, Date) -> int`
- Tipi: `Date`
- Tipi interni: `int`

Specificazione Semantica:

- Confronta due oggetti `Date`.
- Precondizioni: 'date_a' e 'date_b' sono oggetti `Date` validi.
- Postcondizioni: Restituisce un intero minore di zero, uguale a zero o maggiore di zero se 'date_a' è rispettivamente inferiore, uguale o superiore a 'date_b'.

4.3.4 Funzione `void save_date_to_file(Date, FILE*)`

Specificazione Sintattica:

- `save_date_to_file(Date, FILE*)`
- Tipi: `Date`, `FILE*`

- Tipi interni: Nessuno

Specificazione Semantica:

- `save_date_to_file(date, file)`
- Salva l'oggetto `Date` fornito in 'file'.
- Precondizioni: 'date' è un oggetto `Date` valido. 'file' è un file valido aperto in scrittura.
- Postcondizioni: I componenti di 'date' vengono scritti nel file 'file'.

4.3.5 Funzione `Date read_date_from_file(FILE*)`

Specificazione Sintattica:

- `read_date_from_file(FILE*) -> Date`
- Tipi: `FILE*`
- Tipi interni: `Date`, `int`

Specificazione Semantica:

- `read_date_from_file(file) -> return_date`
- Legge un oggetto `Date` da un file.
- Precondizioni: 'file' è un file valido.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Date` letto dal file 'file', o NULL se la lettura fallisce. Il chiamante è responsabile di liberare la memoria allocata utilizzando `free_date()`.

4.3.6 Funzione `void print_date(Date)`

Specificazione Sintattica:

- `print_date(Date)`
- Tipi: `Date`
- Tipi interni: Nessuno

Specificazione Semantica:

- `print_date(date)`
- Stampa i componenti dell'oggetto `Date` su stdout in un formato leggibile dall'utente.
- Precondizioni: 'date' è un oggetto `Date` valido.
- Postcondizioni: I componenti di 'date' vengono stampati su stdout in un formato comprensibile.

4.3.7 Funzione: `Date read_date(void)`

Specificazione Sintattica:

- `read_date(void) -> Date`
- Tipi: Nessuno
- Tipi interni: `Date`, `int`

Specificazione Semantica:

- `read_date() -> return_date`
- Legge un oggetto `Date` dall'input standard.
- Precondizioni: Nessuna.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Date` letto dall'input standard (stdin), o NULL se la lettura fallisce. Il chiamante è responsabile di liberare la memoria allocata utilizzando `free_date()`.

4.3.8 Funzione: `void free_date(Date)`

Specificazione Sintattica:

- `free_date(Date)`
- Tipi: `Date`
- Tipi interni: Nessuno

Specificazione Semantica:

- `free_date(date)`
- Libera la memoria allocata per un oggetto `Date`.
- Precondizioni: 'date' è un oggetto `Date` valido.
- Postcondizioni: La memoria allocata per 'date' viene deallocata.

4.4 Specifica dell'ADT Room

Specificazione Sintattica:

- Tipo di riferimento: Room
- Tipi utilizzati: unsigned int, char*

Specificazione Semantica:
Room rappresenta una sala della conferenza.

Operatori:

- `new_room(char*, unsigned int, unsigned int) -> Room`
- `copy_room(Room) -> Room`
- `are_rooms_equal(Room, Room) -> bool`
- `get_room_name(Room) -> char*`
- `get_room_id(Room) -> unsigned int`
- `print_room(Room) -> void`
- `read_room(unsigned int) -> Room`
- `save_room_to_file(Room, FILE*) -> void`
- `read_room_from_file(FILE*) -> Room`
- `free_room(Room) -> void`

4.4.1 Funzione: `Room new_room(char*, unsigned int, unsigned int)`

Specificazione Sintattica:

- `new_room(char* name, unsigned int id, unsigned int capacity) -> Room`
- Tipi: `char *`, `unsigned int`
- Tipi interni: `Room`

Specificazione Semantica:

- `new_room(name, id, capacity) -> return_room`
- Crea un nuovo oggetto `Room` con il nome, l'ID e la capacità specificati.
- Precondizioni: 'name' è una stringa valida che rappresenta il nome della stanza. 'id' è un intero non negativo che rappresenta l'ID della stanza. 'capacity' è un intero non negativo che rappresenta la capacità della stanza.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Room` creato, o NULL se si verifica un errore. Il chiamante è responsabile di liberare la memoria allocata utilizzando la funzione `free_room()`.

4.4.2 Funzione: `Room copy_room(Room)`

Specificazione Sintattica:

- `copy_room(Room room) -> Room`
- Tipi: `Room`
- Tipi interni: `Room`

Specificazione Semantica:

- `copy_room(room) -> return_room`
- Crea una copia dell'oggetto `Room` fornito.
- Precondizioni: Il parametro di input `room` è un oggetto `Room` valido.
- Postcondizioni: Restituisce un puntatore alla copia appena creata di `room`. Il chiamante è responsabile di liberare la memoria allocata utilizzando la funzione `free_room()`.

4.4.3 Funzione: `bool are_rooms_equal(Room, Room)`

Specificazione Sintattica:

- `are_rooms_equal(Room room_a, Room room_b) -> bool`
- Tipi: `Room`, `bool`
- Tipi interni: Nessuno

Specificazione Semantica:

- `are_rooms_equal(room_a, room_b) -> result`
- Verifica se due oggetti `Room` sono uguali in base ai loro ID.
- Precondizioni: I parametri di input `room_a` e `room_b` sono oggetti `Room` validi.
- Postcondizioni: Restituisce true se `room_a` e `room_b` hanno lo stesso ID, altrimenti false.

4.4.4 Funzione: `char* get_room_name(Room)`

Specificazione Sintattica:

- `get_room_name(Room room) -> char*`
- Tipi: `Room`, `char *`
- Tipi interni: Nessuno

Specificazione Semantica:

- `get_room_name(room) -> name`
- Ottiene il nome dell'oggetto `Room` fornito.
- Precondizioni: Il parametro di input `room` è un oggetto `Room` valido.
- Postcondizioni: Restituisce un puntatore al nome di `room`.

4.4.5 Funzione: `unsigned int get_room_id(Room)`

Specificazione Sintattica:

- `get_room_id(Room room) -> unsigned int`
- Tipi: `Room`, `unsigned int`
- Tipo interno: Nessuno

Specificazione Semantica:

- `get_room_id(room) -> id`
- Ottiene l'ID dell'oggetto `Room` fornito.
- Precondizioni: Il parametro di input `room` è un oggetto `Room` valido.
- Postcondizioni: Restituisce l'ID di `room`.

4.4.6 Funzione: `void print_room(Room)`

Specificazione Sintattica:

- `print_room(Room room) -> void`
- Tipi: `Room`
- Tipo interno: Nessuno

Specificazione Semantica:

- `print_room(room)`
- Stampa i dettagli dell'oggetto `Room` fornito su stdout.
- Precondizioni: Il parametro di input `room` è un oggetto `Room` valido.
- Postcondizioni: I dettagli di `room` sono stampati su stdout.

4.4.7 Funzione: `Room read_room(unsigned int)`

Specificazione Sintattica:

- `read_room(unsigned int id) -> Room`
- Tipi: `unsigned int`, `Room`
- Tipo interno: `char[]`, `ResultInt`, `Room`

Specificazione Semantica:

- `read_room(id) -> room`
- Legge i dettagli di un oggetto `Room` da stdin e crea un nuovo oggetto `Room` con id 'id'.
- Precondizioni: Nessuna.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Room` creato, o NULL se si verifica un errore. Il chiamante è responsabile di liberare la memoria allocata utilizzando la funzione `free_room()`.

4.4.8 Funzione: `void save_room_to_file(Room, FILE*)`

Specificazione Sintattica:

- `save_room_to_file(Room room, FILE* file)`
- Tipi: `Room` , `FILE*`
- Tipo interno: Nessuno

Specificazione Semantica:

- `save_room_to_file(room, file)`
- Salva i dettagli dell'oggetto `Room` fornito su un file.
- Precondizioni: 'room' è un oggetto `Room` valido. 'file' è un file valido aperto in scrittura.
- Postcondizioni: I dettagli di `room` sono scritti sul file 'file'.

4.4.9 Funzione: `Room read_room_from_file(FILE*)`

Specificazione Sintattica:

- `read_room_from_file(FILE* file) -> Room`
- Tipi: `FILE*`, `Room`
- Tipo interno: `char[]`, `unsigned int`

Specificazione Semantica:

- `read_room_from_file(file) -> room`
- Legge i dettagli di un oggetto `Room` da un file e crea un nuovo oggetto `Room`.
- Precondizioni: 'file' è un file valido aperto in lettura.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Room` creato, o NULL se si verifica un errore. Il chiamante è responsabile di liberare la memoria allocata utilizzando la funzione `free_room()`.

4.4.10 Funzione: `void free_room(Room)`

Specificazione Sintattica:

- `free_room(Room room)`
- Tipi: `Room`
- Tipo interno: Nessuno

Specificazione Semantica:

- `free_room(room)`
- Libera la memoria allocata per un oggetto `Room`.
- Precondizioni: 'room' è un oggetto `Room` valido.
- Postcondizioni: La memoria allocata per 'room' viene deallocata.

4.5 Specifica dell'ADT Event

Specificazione Sintattica:

- Tipo di riferimento: Event
- Tipi utilizzati: unsigned int, char*, Date, EventType (intero da 0 a 2)

Specificazione Semantica:
Event rappresenta un evento della conferenza.

Operatori::

- `new_event(EventType, char *, Date, Date, unsigned int) -> Event`
- `are_events_equal(Event, Event) -> int`
- `cmp_event(Event, Event) -> int`
- `do_events_overlap(Event, Event) -> bool`
- `get_event_type(Event) -> EventType`
- `set_event_type(Event, EventType) -> int`
- `get_event_start_date(Event) -> Date`
- `set_event_start_date(Event, Date) -> int`
- `get_event_end_date(Event) -> Date`
- `set_event_end_date(Event, Date) -> int`
- `get_event_name(Event) -> char *`
- `set_event_name(Event, char *) -> int`
- `get_event_room_id(Event) -> unsigned int`
- `set_event_room_id(Event, unsigned int) -> int`
- `get_event_id(Event) -> unsigned int`
- `print_event(Event, Room) -> void`
- `read_event(unsigned int) -> Event`
- `save_event_to_file(Event, FILE *) -> void`
- `read_event_from_file(FILE *) -> Event`
- `free_event(Event) -> void`

4.5.1 Funzione: `void is_valid_event_type(int)`

Specificazione Sintattica:

- `is_valid_event_type(int type) -> int`
- Tipi: `int`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `is_valid_event_type(type) -> result`
- Verifica se il tipo evento fornito è valido.
- Precondizioni: Nessuna.
- Postcondizioni: Restituisce 1 se il tipo evento è valido, 0 altrimenti.

4.5.2 Funzione: `Event new_event(EventType, char *, Date, Date, unsigned int)`

Specificazione Sintattica:

- `new_event(EventType type, char *name, Date start_date, Date end_date, unsigned int id) -> Event`
- Tipi: `EventType`, `char *`, `Date`, `unsigned int`
- Tipo interno: `Event`

Specificazione Semantica:

- Funzione: `new_event(type, name, start_date, end_date, id) -> event`
- Crea un nuovo evento con il tipo, nome, data di inizio, data di fine e ID specificati.
- Precondizioni: Nessuna.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Event` creato.

4.5.3 Funzione: `int are_events_equal(Event, Event)`

Specificazione Sintattica:

- `are_events_equal(Event event_a, Event event_b) -> int`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `are_events_equal(event_a, event_b) -> result`
- Verifica se due eventi sono uguali in base ai loro ID.
- Precondizioni: event_a e event_b sono due oggeti Event validi..
- Postcondizioni: Restituisce 1 se gli eventi sono uguali, 0 altrimenti.

4.5.4 Funzione: `int cmp_event(Event, Event)`

Specificazione Sintattica:

- `cmp_event(Event event_a, Event event_b) -> int`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `cmp_event(event_a, event_b) -> result`

- Confronta due eventi per l'ordinamento basato sulla data di inizio e sul nome.
- Precondizioni: event_a e event_b sono due oggeti Event validi.
- Postcondizioni: Restituisce un valore negativo se `event_a` viene prima di `event_b`, un valore positivo se `event_a` viene dopo `event_b`, e 0 se sono uguali.

4.5.5 Funzione: `bool do_events_overlap(Event, Event)`

Specificazione Sintattica:

- `do_events_overlap(Event event_a, Event event_b) -> bool`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `do_events_overlap(event_a, event_b) -> result`
- Verifica se due eventi si sovrappongono nel tempo.
- Precondizioni: event_a e event_b sono due oggeti Event validi.
- Postcondizioni: Restituisce vero se gli eventi si sovrappongono, falso altrimenti.

4.5.6 Funzione: `EventType get_event_type(Event event)`

Specificazione Sintattica:

- `get_event_type(Event event) -> EventType`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `get_event_type(event) -> type`
- Recupera il tipo dell'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce il tipo di evento (`EventType`).

4.5.7 Funzione: `int set_event_type(Event event, EventType type)`

Specificazione Sintattica:

- `set_event_type(Event event, EventType type) -> int`
- Tipi: `Event`, `EventType`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `set_event_type(event, type) -> result`
- Imposta il tipo dell'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce 0 se l'operazione ha avuto successo, -1 altrimenti.

4.5.8 Funzione: `Date get_event_start_date(Event event)`

Specificazione Sintattica:

- `get_event_start_date(Event event) -> Date`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `get_event_start_date(event) -> start_date`
- Recupera la data di inizio dell'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce un puntatore all'oggetto Data di inizio dell'evento.

4.5.9 Funzione: `int set_event_start_date(Event event, Date start_date)`

Specificazione Sintattica:

- `set_event_start_date(Event event, Date start_date) -> int`
- Tipi: `Event`, `Date`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `set_event_start_date(event, start_date) -> result`
- Imposta la data di inizio dell'evento.
- Precondizioni: event è un oggetto Event valido, start date è un oggetto date valido.
- Postcondizioni: Restituisce 0 se l'operazione ha avuto successo, -1 altrimenti.

4.5.10 Funzione: `Date get_event_end_date(Event event)`

Specificazione Sintattica:

- `get_event_end_date(Event event) -> Date`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `get_event_end_date(event) -> end_date`
- Recupera la data di fine dell'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce un puntatore all'oggetto Data di fine dell'evento.

4.5.11 Funzione: `int set_event_end_date(Event event, Date end_date)`

Specificazione Sintattica:

- `set_event_end_date(Event event, Date end_date) -> int`
- Tipi: `Event`, `Date`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `set_event_end_date(event, end_date) -> result`
- Imposta la data di fine dell'evento.
- Precondizioni: event è un oggetto Event valido, end date è un oggetto date valido.
- Postcondizioni: Restituisce 0 se l'operazione ha avuto successo, -1 altrimenti.

4.5.12 Funzione: `char *get_event_name(Event event)`

Specificazione Sintattica:

- `get_event_name(Event event) -> char *`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `get_event_name(event) -> name`
- Restituisce il nome dell'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce un puntatore alla stringa del nome dell'evento.

4.5.13 Funzione: `int set_event_name(Event event, char *name)`

Specificazione Sintattica:

- `set_event_name(Event event, char *name) -> int`
- Tipi: `Event, char *`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `set_event_name(event, name) -> result`
- Imposta il nome dell'evento.
- Precondizioni: event è un oggetto Event valido, name è una stringa valida.
- Postcondizioni: Restituisce 0 se l'operazione ha avuto successo, -1 altrimenti.

4.5.14 Funzione: `unsigned int get_event_room_id(Event event)`

Specificazione Sintattica:

- `get_event_room_id(Event event) -> unsigned int`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `get_event_room_id(event) -> room_id`
- Restituisce l'ID della stanza assegnata all'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce l'ID della stanza dell'evento.

4.5.15 Funzione: `int set_event_room_id(Event event, unsigned int room_id)`

Specificazione Sintattica:

- `set_event_room_id(Event event, unsigned int room_id) -> int`
- Tipi: `Event`, `unsigned int`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `set_event_room_id(event, room_id) -> result`
- Imposta l'ID della stanza assegnata all'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce 0 se l'operazione ha avuto successo, -1 altrimenti.

4.5.16 Funzione: `unsigned int get_event_id(Event event)`

Specificazione Sintattica:

- `get_event_id(Event event) -> unsigned int`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `get_event_id(event) -> id`
- Restituisci l'ID dell'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce l'ID dell'evento.

4.5.17 Funzione: `void print_event(Event event, Room assigned_room)`

Specificazione Sintattica:

- `print_event(Event event, Room assigned_room) -> void`
- Tipi: `Event`, `Room`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `print_event(event, assigned_room)`
- Stampa i dettagli dell'evento su stdout, inclusa l'informazione sulla stanza assegnata se disponibile.
- Precondizioni: Nessuna.
- Postcondizioni: Nessuna.

4.5.18 Funzione: `Event read_event(unsigned int event_id)`

Specificazione Sintattica:

- `read_event(unsigned int event_id) -> Event`
- Tipi: `unsigned int`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `read_event(event_id) -> event`
- Legge i dettagli dell'evento dall'input dell'utente.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto Event creato.

4.5.19 Funzione: `void save_event_to_file(Event event, FILE *file)`

Specificazione Sintattica:

- `save_event_to_file(Event event, FILE *file) -> void`
- Tipi: `Event`, `FILE`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `save_event_to_file(event, file)`
- Salva i dettagli dell'evento su un file.
- Precondizioni: event è un oggetto Event valido, il file è aperto in scrittura.
- Postcondizioni: Nessuna.

4.5.20 Funzione: `Event read_event_from_file(FILE *file)`

Specificazione Sintattica:

- `read_event_from_file(FILE *file) -> Event`
- Tipi: `FILE`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `read_event_from_file(file) -> event`
- Legge i dettagli dell'evento da un file.
- Precondizioni: event è un oggetto Event valido, il file è aperto in lettura.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto Event creato.

4.5.21 Funzione: `void free_event(Event event)`

Specificazione Sintattica:

- `free_event(Event event) -> void`
- Tipi: `Event`
- Tipo interno: Nessuno

Specificazione Semantica:

- Funzione: `free_event(event)`
- Libera la memoria allocata per l'evento.
- Precondizioni: event è un oggetto Event valido.
- Postcondizioni: La memoria allocate per event viene deallocata.

4.6 Specifica dell'ADT EventBst

Specificazione Sintattica:

- Tipo di riferimento: EventBst
- Tipi utilizzati: Event

Specificazione Semantica:
EventBst rappresenta un albero binario di ricerca di oggetti di tipo Event

Operatori:

- `new_event_bst(void) -> EventBst`
- `bst_insert_event(EventBst bst, Event event) -> int`
- `bst_remove_event(EventBst bst, Event event) -> Event`
- `bst_remove_event_by_id(EventBst bst, unsigned int id) -> Event`
- `bst_get_event_by_id(EventBst bst, unsigned int id) -> Event`
- `get_bst_size(EventBst bst) -> size_t`
- `print_event_bst(EventBst bst, RoomList room_list) -> void`
- `event_bst_every(EventBst bst, EventPredicate predicate, ...)  -> bool`
- `read_event_bst_from_file(FILE* file) -> EventBst`
- `save_event_bst_to_file(EventBst bst, FILE* file) -> void`
- `save_event_bst_to_file_sorted(EventBst bst, FILE* file) -> void`
- `free_event_bst(EventBst bst) -> void`

4.6.1 Funzione: `EventBst new_event_bst`

Specificazione Sintattica:

- `new_event_bst(void) -> EventBst`
- Tipi: Nessuno
- Tipi interni: `EventBst`

Specificazione Semantica:

- Funzione: `new_event_bst() -> bst`
- Crea un nuovo albero di ricerca binario (BST) vuoto per memorizzare eventi.
- Precondizioni: Nessuna.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `EventBst` vuoto.

4.6.2 Funzione: `int bst_insert_event(EventBst bst, Event event)`

Specificazione Sintattica:

- `bst_insert_event(EventBst, Event) -> int`
- Tipi: `EventBst, Event`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `bst_insert_event(bst, event) -> result`
- Inserisce un evento nell'albero di ricerca binario (BST).
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido. `event` deve puntare a un oggetto `Event` valido.
- Postcondizioni: Se l'operazione ha successo, l'evento viene inserito nel BST. Restituisce 0 in caso di successo, -1 in caso di errore.

4.6.3 Funzione: `Event bst_remove_event(EventBst bst, Event event)`

Specificazione Sintattica:

- `bst_remove_event(EventBst, Event) -> Event`
- Tipi: `EventBst, Event`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `bst_remove_event(bst, event) -> removed_event`
- Rimuove l'evento specificato dall'albero di ricerca binario (BST).
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido. `event` deve puntare a un oggetto `Event` valido.
- Postcondizioni: Se l'operazione ha successo, l'evento viene rimosso dal BST. Restituisce l'oggetto Event rimosso, o NULL_EVENT se l'evento non è stato trovato.

4.6.4 Funzione: `Event bst_remove_event_by_id(EventBst bst, unsigned int id)`

Specificazione Sintattica:

- `bst_remove_event_by_id(EventBst, unsigned int) -> Event`
- Tipi: `EventBst, unsigned int`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `bst_remove_event_by_id(bst, id) -> removed_event`
- Rimuove l'evento con l'ID specificato dall'albero di ricerca binario (BST).
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido.
- Postcondizioni: Se l'operazione ha successo, l'evento con l'ID specificato viene rimosso dal BST. Restituisce l'oggetto Event rimosso, o NULL_EVENT se l'evento non è stato trovato.

4.6.5 Funzione: `Event bst_get_event_by_id(EventBst bst, unsigned int id)`

Specificazione Sintattica:

- `bst_get_event_by_id(EventBst, unsigned int) -> Event`
- Tipi: `EventBst, unsigned int`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `bst_get_event_by_id(bst, id) -> event`
- Recupera l'evento con l'ID specificato dall'albero di ricerca binario (BST).
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido.
- Postcondizioni: Restituisce l'oggetto Event con l'ID specificato, o NULL_EVENT se l'evento non è stato trovato.

4.6.6 Funzione: `size_t get_bst_size(EventBst bst)`

Specificazione Sintattica:

- `get_bst_size(EventBst) -> size_t`
- Tipi: `EventBst`
- Tipi interni: `Nessuna`

Specificazione Semantica:

- Funzione: `get_bst_size(bst) -> size`
- Restituisce il numero di eventi memorizzati nell'albero di ricerca binario (BST).
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido.
- Postcondizioni: Restituisce il numero di eventi memorizzati nel BST.

4.6.7 Funzione: `bool print_event_bst(EventBst bst, RoomList room_list)`

Specificazione Sintattica:

- `print_event_bst(EventBst, RoomList) -> void`
- Tipi: `EventBst, RoomList`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `print_event_bst(bst, room_list)`
- Stampa in ordine gli eventi memorizzati nell'albero di ricerca binario (BST) insieme alle stanze assegnate.
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido. `room_list` deve puntare a un oggetto `RoomList` valido.
- Postcondizioni: Stampa gli eventi e le stanze assegnate all'output standard.

4.6.8 Funzione: `bool event_bst_every(EventBst, EventPredicate, ...)`

Specificazione Sintattica

- `event_bst_every(EventBst, EventPredicate, ...) -> bool`
- Tipi: `EventBst, EventPredicate`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `event_bst_every(bst, predicate, ...)`
- Verifica se un dato predicato è vero per ogni evento nell'albero di ricerca binario (BST).
- Precondizione : `bst` deve puntare a un oggetto valido di tipo `EventBst`. `predicate` deve essere un puntatore valido a una funzione predicato che prende un Evento e argomenti variabili (copie degli ulteriori argomenti passati alla funzione `event_bst_every`).
- Postcondizioni: Restituisce `true` se il predicato è vero per ogni evento nel BST, `false` altrimenti.

4.6.9 Funzione: `EventBst read_event_bst_from_file(FILE* file)`

Specificazione Sintattica:

- `read_event_bst_from_file(FILE*) -> EventBst`
- Tipi: `FILE*`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `read_event_bst_from_file(file) -> bst`
- Legge i dati degli eventi da un file e costruisce un albero di ricerca binario (BST) per memorizzare gli eventi.
- Precondizioni: `file` deve puntare a un oggetto `FILE` valido aperto in lettura.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `EventBst` contenente gli eventi letti dal file.

4.6.10 Funzione: `void save_event_bst_to_file(EventBst bst, FILE* file)`

Specificazione Sintattica:

- `save_event_bst_to_file(EventBst, FILE*) -> void`
- Tipi: `EventBst, FILE*`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `save_event_bst_to_file(bst, file)`
- Salva gli eventi memorizzati nell'albero di ricerca binario (BST) su un file.
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido. `file` deve puntare a un oggetto `FILE` valido aperto in modalità scrittura.
- Postcondizioni: Scrive gli eventi memorizzati nel BST sul file specificato.

4.6.11 Funzione: `void save_event_bst_to_file_sorted(EventBst bst, FILE* file)`

Specificazione Sintattica:

- `save_event_bst_to_file_sorted(EventBst, FILE*) -> void`
- Tipi: `EventBst, FILE*`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `save_event_bst_to_file_sorted(bst, file)`
- Salva gli eventi memorizzati nell'albero di ricerca binario (BST) su un file in ordine ordinato.
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido. `file` deve puntare a un oggetto `FILE` valido aperto in modalità scrittura.
- Postcondizioni: Scrive gli eventi memorizzati nel BST sul file specificato in ordine ordinato.

4.6.12 Funzione: `void free_event_bst(EventBst bst)`

Specificazione Sintattica:

- `free_event_bst(EventBst) -> void`
- Tipi: `EventBst`
- Tipi interni: `EventBstNode`

Specificazione Semantica:

- Funzione: `free_event_bst(bst)`
- Libera la memoria allocata per l'albero di ricerca binario (BST) e i suoi nodi.
- Precondizioni: `bst` deve puntare a un oggetto `EventBst` valido.
- Postcondizioni: Libera la memoria allocata per il BST e i suoi nodi.

4.7 Specifica dell'ADT RoomList

Specificazione Sintattica:

- Tipo di riferimento: RoomList
- Tipi utilizzati: int, Room

Specificazione Semantica:
RoomList rappresenta una sequenza di oggetti di tipo Room

Operazioni:

- `new_room_list(void) -> RoomList`
- `is_room_list_empty(RoomList list) -> bool`
- `get_size_room_list(RoomList list) -> int`
- `cons_room_list(RoomList list, Room room) -> void`
- `tail_room_list(RoomList list) -> Room`
- `get_first_room_list(RoomList list) -> Room`
- `get_at_room_list(RoomList list, int pos) -> Room`
- `get_room_by_id(RoomList list, unsigned int room_id) -> Room`
- `remove_at_room_list(RoomList list, int pos) -> Room`
- `get_pos_room_list(RoomList list, Room to_search) -> int`

- `print_room_list(RoomList list) -> void`
- `save_room_list_to_file(RoomList list, FILE* file) -> void`
- `read_room_list_from_file(FILE* file) -> RoomList`
- `free_room_list(RoomList list) -> void`

4.7.1 Funzione: `RoomList new_room_list(void)`

Specificazione Sintattica:

- `new_room_list(void) -> RoomList`
- Tipi: Nessuno
- Tipi interni: `RoomList`

Specificazione Semantica:

- Funzione: `new_room_list() -> list`
- Crea una nuova lista vuota di stanze.
- Precondizioni: Nessuna.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `RoomList` vuoto creato.

4.7.2 Funzione: `bool is_room_list_empty(RoomList list)`

Specificazione Sintattica:

- `is_room_list_empty(RoomList) -> bool`
- Tipi: `RoomList`
- Tipi interni: `bool`

Specificazione Semantica:

- Funzione: `is_room_list_empty(list) -> empty`
- Verifica se la lista delle stanze è vuota.
- Precondizioni: `list` è un oggetto `RoomList` valido.
- Postcondizioni: Restituisce true se la lista è vuota, false altrimenti.

4.7.3 Funzione: `int get_size_room_list(RoomList list)`

Specificazione Sintattica:

- `get_size_room_list(RoomList) -> int`
- Tipi: `RoomList`
- Tipi interni: `int`

Specificazione Semantica:

- Funzione: `get_size_room_list(list) -> size`
- Ottiene il numero di stanze nella lista.
- Precondizioni: `list` è un oggetto `RoomList` valido.
- Postcondizioni: Restituisce il numero di stanze nella lista.

4.7.4 Funzione: *void cons_room_list(RoomList list, Room room)*

Specificazione Sintattica:

- `cons_room_list(RoomList, Room) -> void`
- Tipi: `RoomList, Room`
- Tipi interni: Nessuno

Specificazione Semantica:

- Funzione: `cons_room_list(list, room)`
- Aggiunge una stanza all'inizio della lista.
- Precondizioni: `list` è un oggetto `RoomList` valido, `room` è un oggetto `Room` valido.
- Postcondizioni: La stanza viene aggiunta all'inizio della lista.

4.7.5 Funzione: *Room tail_room_list(RoomList list)*

Specificazione Sintattica:

- `tail_room_list(RoomList) -> Room`
- Tipi: `RoomList`
- Tipi interni: `Room`

Specificazione Semantica:

- Funzione: `tail_room_list(list) -> room`
- Rimuove e restituisce l'ultima stanza dalla lista.
- Precondizioni: `list` è un oggetto `RoomList` valido.
- Postcondizioni: Restituisce l'ultima stanza dalla lista. La dimensione della lista viene ridotta di uno.

4.7.6 Funzione: *Room get_first_room_list(RoomList list)*

Specificazione Sintattica:

- `get_first_room_list(RoomList) -> Room`
- Tipi: `RoomList`
- Tipi interni: `Room`

Specificazione Semantica:

- Funzione: `get_first_room_list(list) -> room`
- Ottiene la prima stanza nella lista.
- Precondizioni: `list` è un oggetto `RoomList` valido.
- Postcondizioni: Restituisce la prima stanza nella lista.

4.7.7 Funzione: *Room get_at_room_list(RoomList list, int pos)*

Specificazione Sintattica:

- `get_at_room_list(RoomList, int) -> Room`
- Tipi: `RoomList, int`
- Tipi interni: `Room`

Specificazione Semantica:

- Funzione: `get_at_room_list(list, pos) -> room`
- Ottiene la stanza alla posizione specificata nella lista.
- Precondizioni: `list` è un oggetto `RoomList` valido, `pos` è un indice valido.
- Postcondizioni: Restituisce la stanza alla posizione specificata nella lista.

4.7.8 Funzione: `Room get_room_by_id(RoomList list, unsigned int room_id)`

Specificazione Sintattica:

- `get_room_by_id(RoomList, unsigned int) -> Room`
- Tipi: `RoomList, unsigned int`
- Tipi interni: `Room`

Specificazione Semantica:

- Funzione: `get_room_by_id(list, room_id) -> room`
- Ottiene la stanza con l'ID specificato dalla lista.
- Precondizioni: `list` è un oggetto `RoomList` valido.
- Postcondizioni: Restituisce la stanza con l'ID specificato dalla lista, o NULL se non trovata.

4.7.9 Funzione: `Room remove_at_room_list(RoomList list, int pos)`

Specificazione Sintattica:

- `remove_at_room_list(RoomList, int) -> Room`
- Tipi: `RoomList, int`
- Tipi interni: `Room`

Specificazione Semantica:

- Funzione: `remove_at_room_list(list, pos) -> room`
- Rimuove e restituisce la stanza alla posizione specificata nella lista.
- Precondizioni: `list` è un oggetto `RoomList` valido, `pos` è un indice valido.
- Postcondizioni: Restituisce la stanza rimossa dalla lista.

4.7.10 Funzione: `int get_pos_room_list(RoomList list, Room to_search)`

Specificazione Sintattica:

- `get_pos_room_list(RoomList, Room) -> int`
- Tipi: `RoomList, Room`
- Tipi interni: `int`

Specificazione Semantica:

- Funzione: `get_pos_room_list(list, to_search) -> pos`
- Ottiene la posizione della stanza specificata nella lista.
- Precondizioni: `list` è un oggetto `RoomList` valido, `to_search` è un oggetto `Room` valido.
- Postcondizioni: Restituisce la posizione della stanza nella lista, o -1 se non trovata.

4.7.11 Funzione: `void print_room_list(RoomList list)`

Specificazione Sintattica:

- `print_room_list(RoomList) -> void`
- Tipi: `RoomList`
- Tipi interni: int

Specificazione Semantica:

- Funzione: `print_room_list(list)`
- Stampa tutte le stanze nella lista.
- Precondizioni: `list` è un oggetto `RoomList` valido.
- Postcondizioni: Stampa i dettagli di ogni stanza nella lista su stdout.

4.7.12 Funzione: `void save_room_list_to_file(RoomList list, FILE* file)`

Specificazione Sintattica:

- `save_room_list_to_file(RoomList, FILE*) -> void`
- Tipi: `RoomList, FILE*`
- Tipi interni: int

Specificazione Semantica:

- Funzione: `save_room_list_to_file(list, file)`
- Salva l'elenco delle stanze su un file.
- Precondizioni: `list` è un oggetto `RoomList` valido, `file` è un file valido aperto in scrittura.
- Postcondizioni: Scrive i dettagli di ogni stanza nella lista sullo stream del file.

4.7.13 Funzione: `RoomList read_room_list_from_file(FILE* file)`

Specificazione Sintattica:

- `read_room_list_from_file(FILE*) -> RoomList`
- Tipi: `FILE*`
- Tipi interni: `RoomList`, `int`

Specificazione Semantica:

- Funzione: `read_room_list_from_file(file) -> list`
- Legge un elenco di stanze da un file.
- Precondizioni: `file` è un file valido aperto in lettura.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `RoomList` contenente le stanze lette dal file.

4.7.14 Funzione: `void free_room_list(RoomList list)`

Specificazione Sintattica:

- `free_room_list(RoomList) -> void`
- Tipi: `RoomList`
- Tipi interni: Nessuno

Specificazione Semantica:

- Funzione: `free_room_list(list)`
- Libera la memoria allocata per l'elenco delle stanze.
- Precondizioni: `list` è un oggetto `RoomList` valido.
- Postcondizioni: Rilascia la memoria allocata per l'oggetto `RoomList` e tutte le sue stanze contenute.

4.8 Specifica dell'ADT Conference

Specificazione Sintattica:

- Tipo di riferimento: Conference
- Tipi utilizzati: EventBst, RoomList, unsigned int

Specificazione Semantica:
Conference rappresenta una conferenza

Operatori:

- `new_conference(void) -> Conference`
- `add_conference_event(Conference conf) -> int`
- `edit_conference_event(Conference conf) -> int`
- `remove_conference_event(Conference conf) -> int`
- `display_conference_schedule(Conference conf) -> voic`
- `display_conference_rooms(Conference conf) -> voic`
- `add_conference_room(Conference conf) -> int`
- `remove_conference_room(Conference conf) -> int`
- `conference_assign_event_to_room(Conference conf) -> int`
- `conference_free_event_room(Conference conf) -> int`
- `free_conference(Conference conf) -> void`
- `save_conference_to_file(Conference conf, FILE* file) -> void`
- `save_conference_to_file_sorted(Conference conf, FILE* file) -> Conference`
- `read_conference_from_file(FILE* file)`

4.8.1 Funzione: `Conference new_conference(void)`

Specificazione Sintattica:

- `new_conference(void) -> Conference`
- Tipi: Nessuno
- Tipo interno: `Conference`, `EventBst`, `RoomList`, `unsigned int`

Specificazione Semantica:

- Funzione: `new_conference() -> conferenza`
- Crea una nuova conferenza vuota per memorizzare eventi e stanze.
- Precondizioni: Nessuna.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Conference` vuoto.

4.8.2 Funzione: `int add_conference_event(Conference conf)`

Specificazione Sintattica:

- `add_conference_event(Conference conf) -> int`
- Tipi: `Conference`
- Tipo interno: `Event`

Specificazione Semantica:

- Funzione: `add_conference_event(conf) -> risultato`
- Aggiunge un nuovo evento alla conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Restituisce 0 in caso di successo, -1 in caso di errore.

4.8.3 Funzione: `int edit_conference_event(Conference conf)`

Specificazione Sintattica:

- `edit_conference_event(Conference conf) -> int`

- Tipi: `Conference`
- Tipo interno: `Event`

Specificazione Semantica:

- Funzione: `edit_conference_event(conf) -> risultato`
- Modifica un evento esistente nella conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Restituisce 0 in caso di successo, -1 in caso di errore.

4.8.4 Funzione: `int remove_conference_event(Conference conf)`

Specificazione Sintattica:

- `remove_conference_event(Conference conf) -> int`
- Tipi: `Conference`
- Tipo interno: `Event`

Specificazione Semantica:

- Funzione: `remove_conference_event(conf) -> risultato`
- Rimuove un evento dalla conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Restituisce 0 in caso di successo, -1 in caso di errore.

4.8.5 Funzione: `void display_conference_schedule(Conference conf)`

Specificazione Sintattica:

- `display_conference_schedule(Conference conf) -> void`
- Tipi: `Conference`
- Tipo interno: `Event`, `Room`

Specificazione Semantica:

- Funzione: `display_conference_schedule(conf)`
- Mostra l'orario degli eventi nella conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Stampa l'orario degli eventi.

4.8.6 Funzione: `void display_conference_rooms(Conference conf)`

Specificazione Sintattica:

- `display_conference_rooms(Conference conf) -> void`
- Tipi: `Conference`
- Tipo interno: `Event`, `Room`

Specificazione Semantica:

- Funzione: `display_conference_rooms(conf)`
- Mostra l'elenco delle stanze disponibili nella conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Stampa l'elenco delle stanze.

4.8.7 Funzione: `int add_conference_room(Conference conf)`

Specificazione Sintattica:

- `add_conference_room(Conference conf) -> int`
- Tipi: `Conference`
- Tipo interno: `Room`

Specificazione Semantica:

- Funzione: `add_conference_room(conf) -> risultato`
- Aggiunge una nuova stanza alla conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Restituisce 0 in caso di successo, -1 in caso di errore.

4.8.8 Funzione: `int remove_conference_room(Conference conf)`

Specificazione Sintattica:

- `remove_conference_room(Conference conf) -> int`
- Tipi: `Conference`
- Tipo interno: `Room`

Specificazione Semantica:

- Funzione: `remove_conference_room(conf) -> risultato`
- Rimuove una stanza dalla conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Restituisce 0 in caso di successo, -1 in caso di errore.

4.8.9 Funzione: `int conference_assign_event_to_room(Conference conf)`

Specificazione Sintattica:

- `conference_assign_event_to_room(Conference conf) -> int`
- Tipi: `Conference`
- Tipo interno: `Event`, `Room`

Specificazione Semantica:

- Funzione: `conference_assign_event_to_room(conf) -> risultato`
- Assegna un evento a una stanza nella conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Restituisce 0 in caso di successo, -1 in caso di errore.

4.8.10 Funzione: `int conference_free_event_room(Conference conf)`

Specificazione Sintattica:

- `conference_free_event_room(Conference conf) -> int`
- Tipi: `Conference`
- Tipo interno: `Event`

Specificazione Semantica:

- Funzione: `conference_free_event_room(conf) -> risultato`
- Libera la stanza assegnata per un evento nella conferenza.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Restituisce 0 in caso di successo, -1 in caso di errore.

4.8.11 Funzione: `void free_conference(Conference conf)`

Specificazione Sintattica:

- `free_conference(Conference)`
- Tipi: `Conference`
- Tipi interni: `Event, Room`

Specificazione Semantica:

- Funzione: `free_conference(conf)`
- Libera la memoria allocata per la conferenza e i suoi eventi e sale.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido.
- Postcondizioni: Libera la memoria allocata per la conferenza.

4.8.12 Funzione: `void save_conference_to_file(Conference conf, FILE* file)`

Specificazione Sintattica:

- `save_conference_to_file(Conference, FILE*) -> void`
- Tipi: `Conference, FILE*`
- Tipi interni: `Event, Room`

Specificazione Semantica:

- Funzione: `save_conference_to_file(conf, file)`
- Salva i dati della conferenza su un file.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido. `file` deve puntare a un oggetto `FILE` valido aperto in modalità scrittura.
- Postcondizioni: Scrive i dati della conferenza sul file specificato.

4.8.13 Funzione: `void save_conference_to_file_sorted(Conference conf, FILE* file)`

Specificazione Sintattica:

- `save_conference_to_file_sorted(Conference, FILE*) -> void`
- Tipi: `Conference, FILE*`
- Tipi interni: `Event, Room`

Specificazione Semantica:

- Funzione: `save_conference_to_file_sorted(conf, file)`
- Salva i dati della conferenza su un file in ordine ordinato.
- Precondizioni: `conf` deve puntare a un oggetto `Conference` valido. `file` deve puntare a un oggetto `FILE` valido aperto in modalità scrittura.
- Postcondizioni: Scrive i dati della conferenza sul file specificato in ordine ordinato.

4.8.14 Funzione: `Conference read_conference_from_file(FILE* file)`

Specificazione Sintattica:

- `Conference read_conference_from_file(FILE*)`
- Tipi: `FILE*`
- Tipi interni: `Event, Room`

Specificazione Semantica:

- Funzione: `read_conference_from_file(file) -> conf`
- Legge i dati della conferenza da un file e costruisce un nuovo oggetto conferenza.
- Precondizioni: `file` deve puntare a un oggetto `FILE` valido aperto in modalità lettura.
- Postcondizioni: Restituisce un puntatore al nuovo oggetto `Conference` contenente i dati della conferenza letti dal file.

5 Razionale dei casi di test

Il file `test.c` implementa un sistema di testing per il progetto di gestione delle conferenze. Questo sistema è progettato per eseguire una serie di test automatici che verificano il corretto funzionamento delle principali funzionalità del software. Di seguito viene fornita una panoramica del funzionamento del file `test.c`.

5.1 Tipi di Test

Il sistema di testing supporta vari tipi di test, definiti tramite una `enum` denominata `TestType`. I tipi di test implementati sono:

- `TEST_ADD_EVENT`: Test per l'aggiunta di un evento.
- `TEST_REMOVE_EVENT`: Test per la rimozione di un evento.
- `TEST_EDIT_EVENT`: Test per la modifica di un evento.
- `TEST_ASSIGN_ROOM_EVENT`: Test per l'assegnazione di una sala a un evento.
- `TEST_FREE_ROOM_EVENT`: Test per la liberazione di una sala assegnata a un evento.
- `TEST_DISPLAY_EVENTS`: Test per la visualizzazione del programma della conferenza.

5.2 Funzionamento Generale

Il file `test.c` segue questi passaggi per eseguire i test:

1. Legge il file di suite di test, il quale contiene le specifiche di ogni test da eseguire.
2. Per ogni test, carica i file necessari (ex. Per il test TC001 i file saranno contenuti nell'omonima cartella), inclusi i file di input (input.txt), di output atteso (oracle.txt) ed un opzionale file di conferenza iniziale (conference.txt)
3. Reindirizza l'input e l'output standard per eseguire le operazioni specificate dal tipo di test.
4. Confronta l'output generato dal test con l'output atteso (oracle) per determinare se il test è passato o fallito.
5. Registra i risultati di ogni test in un file di risultati.

5.3 Formato file conferenze

I file `conference.txt` e quelli di oracle (tranne nei test di visualizzazione del programma della conferenza) contengono le informazioni relative alla conferenza. Sono strutturati nel seguente modo:

1. Le prime due righe rappresentano gli inizializzatori degli ID per eventi e stanze.
2. Il numero di eventi presenti nella conferenza è indicato dalla riga successiva.
3. Per ciascun evento, vengono forniti i seguenti dettagli:
    - Tipo dell'evento (indicato con un numero tra 1 e 3), ID dell'evento, 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, indicato dalla riga successiva.
5. Per ciascuna sala, vengono forniti i seguenti dettagli:
    - ID della sala
    - Nome della sala
    - Posti presenti nella sala

ESEMPIO: Di seguito è riportato un esempio di come deve essere formattato un file contenente una conferenza:

```
4 4
3
1 0 1 Evento test 1
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 3 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
3
1
Sala A
50
2
Sala B
50
3
Sala C
50
```

1. il primo 4 è l'inizializzatore degli ID evento, il secondo è l' inizializzatore degli ID sala
2. 3 indica il numero di eventi nella conferenza
3. Il primo evento assume i seguenti valori:
    - 1 è l'ID dell'evento, 0 è il tipo dell'evento, 1 è l'ID della sala a cui è assegnato e "Evento test 1" è il nome dell'evento
    - `01/01/1970 12:00` è la data e l'ora di inizio dell'evento.
    - `01/01/1970 13:00` è la data e l'ora di fine dell'evento.
4. Seguono altri due eventi formattati come il precedente
5. 3 rappresenta il numero di sale nella conferenza.
6. La prima sala assume i seguenti valori:
    - 1 è l'ID della sala
    - "Sala A" è il nome della sala
    - 50 sono i posti della sala
7. Seguono altre due sale formattate in maniera similare

5.4 Test per l'aggiunta di un evento

I test di aggiunta di un evento non usano il file opzionale conference.txt. I file output.txt e oracle.txt sono conferenze formattate come precedentemente descritto. Il formato del file input.txt è descritto nella sottosezione che segue.

5.4.1 Formato del file `input.txt` per i Test di Aggiunta Evento

Per eseguire correttamente i test di aggiunta evento, il file `input.txt` deve essere formattato in modo specifico per rappresentare l'aggiunta di un singolo evento. Questo file contiene le informazioni necessarie per aggiungere un nuovo evento alla conferenza, con ogni campo separato su righe diverse:

- **Nome evento**: Una stringa che rappresenta il nome dell'evento.
- **Tipologia evento**: Un intero che indica la tipologia dell'evento (valore 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`.

Ogni riga non valida viene ignorata

ESEMPIO 1: Di seguito è riportato un esempio di come deve essere formattato il file `input.txt`:

```
NomeEvento1
1
15/06/2024 09:00
15/06/2024 11:00
```

In questo esempio:

- `NomeEvento1` è il nome dell'evento.
- `1` è la tipologia dell'evento.
- `15/06/2024 09:00` è la data e l'ora di inizio dell'evento.
- `15/06/2024 11:00` è la data e l'ora di fine dell'evento.

ESEMPIO 2: Di seguito è riportato un esempio di come vengono interpretati dei file con righe non valide:

```
NomeEvento1
1
15/06/2024 09:00
15/06/2024 08:00
15/06/2024 11:00
```

In questo esempio:

- `NomeEvento1` è il nome dell'evento.
- `1` è la tipologia dell'evento.
- `15/06/2024 09:00` è la data e l'ora di inizio dell'evento.
- `15/06/2024 11:00` è la data e l'ora di fine dell'evento, dato che la riga "15/06/2024 08:00" viene ignorata

5.4.2 Casi di test relativi al nome dell'evento

- **TC001** : testiamo il caso in cui vengano inseriti regolarmente il nome dell'evento, la tipologia dell'evento, data di inizio e di fine.
- **TC002** : testiamo il caso in cui vengano inseriti 101 caratteri nel nome dell'evento. Il programma chiederà nuovamente di inserire il nome perchè il limite (100) è stato superato.
- **TC003** : testiamo il caso in cui vengano inseriti 100 caratteri nel nome dell'evento. Il programma accetterà il nome perchè il 100esimo carattere è compreso nel limite.
- **TC004** : testiamo il caso in cui venga dato un'invio all'inserimento del nome dell'evento. Il programma chiederà nuovamente di inserire il nome dell'evento.

- **TC005** : testiamo il caso in cui vengano dati degli spazi all'inserimento del nome dell'evento. Il programma non considererà gli spazi.
- **TC006** : testiamo il caso in cui vengano dati invii e spazi all'inserimento del nome dell'evento. Il programma continuerà a chiedere il nome dopo ogni invio e ignorerà gli spazi.

5.4.3 Casi di test relativi alla tipologia dell'evento

- **TC007** : testiamo il caso in cui venga inserita la tipologia "2" dell'evento.
- **TC008** : testiamo il caso in cui venga inserita la tipologia "3" dell'evento.
- **TC009** : testiamo il caso in cui venga inserito 0 alla richiesta della tipologia dell'evento. Il programma chiederà nuovamente la tipologia.
- **TC010** : testiamo il caso in cui venga inserito 4 alla richiesta della tipologia dell'evento. Il programma chiederà nuovamente la tipologia.
- **TC011**: testiamo il caso in cui venga inserito -1 alla richiesta della tipologia dell'evento. Il programma chiederà nuovamente la tipologia.
- **TC012**: testiamo il caso in cui venga inserita una stringa alla richiesta della tipologia dell'evento. Il programma chiederà nuovamente la tipologia.

5.4.4 Casi di test relativi alle date dell'evento

- **TC013** : testiamo il caso in cui venga inserita una data con il giorno 0. Il programma chiederà nuovamente la data.
- **TC014** : testiamo il caso in cui venga inserita una data con il giorno -1. Il programma chiederà nuovamente la data.
- **TC015** : testiamo il caso in cui venga inserito 31 come giorno a gennaio.
- **TC016** : testiamo il caso in cui venga inserito 32 come giorno. Il programma chiederà nuovamente la data.
- **TC017** : testiamo il caso in cui venga inserito 28 come giorno a febbraio in un anno non bisestile.
- **TC018** : testiamo il caso in cui venga inserito 29 come giorno a febbraio in un anno non bisestile. Il programma chiederà nuovamente la data.
- **TC019** : testiamo il caso in cui venga inserito 29 come giorno a febbraio in un anno bisestile.
- **TC020** : testiamo il caso in cui venga inserito 30 come giorno a febbraio in un anno bisestile. Il programma chiederà nuovamente la data.
- **TC021** : testiamo il caso in cui venga inserito 28 come giorno a febbraio in un anno non bisestile.
- **TC022** : testiamo il caso in cui venga inserito 29 come giorno a febbraio in un anno non bisestile. Il programma chiederà nuovamente la data.
- **TC023** : testiamo il caso in cui venga inserito 29 come giorno a febbraio in un anno bisestile.
- **TC024** : testiamo il caso in cui venga inserito 30 come giorno a febbraio in un anno bisestile. Il programma chiederà nuovamente la data.
- **TC025** fino al **TC044** : testiamo tutti i casi in cui il giorno superi il limite di giorni in quel determinato mese. Il programma chiederà nuovamente di inserire la data.
- **TC045** : testiamo il caso in cui venga inserito "13" come mese. Il programma chiederà nuovamente la data.
- **TC046** : testiamo il caso in cui venga inserito "00" come mese. Il programma chiederà nuovamente la data.

- **TC047** : testiamo il caso in cui venga inserito “-01” come mese. Il programma chiederà nuovamente la data.
- **TC048** : testiamo il caso in cui venga inserito “-01” come anno. Il programma chiederà nuovamente la data.
- **TC049** : testiamo il caso in cui venga inserito “24:00” come orario. Il programma chiederà nuovamente la data.
- **TC050** : testiamo il caso in cui venga inserito “-1:00” come orario. Il programma chiederà nuovamente la data.
- **TC051** : testiamo il caso in cui venga inserito “60” come minuti all’interno dell’orario. Il programma chiederà nuovamente la data.
- **TC052** : testiamo il caso in cui venga inserito “-1” come minuti all’interno dell’orario. Il programma chiederà nuovamente la data.
- **TC053** : testiamo il caso in cui venga inserita come seconda data una data minore rispetto alla prima. Il programma chiederà nuovamente la data.
- **TC054** : testiamo il caso in cui venga inserita una data di inizio e di fine uguale. Andrea vaffanculo (spiegherai tu questa cosa).
- **TC054** : testiamo il caso in cui vengano inserite una data di inizio e una di fine uguale, abbiamo deciso di considerare corretto questo caso.

5.5 Test per la rimozione di un evento

I file conference.txt, output.txt e oracle.txt sono conferenze formattate come precedentemente descritto. Il formato del file input.txt è descritto nella sottosezione che segue.

5.5.1 Formato del file `input.txt` per i Test di Rimozione Evento

Per eseguire correttamente i test, il file `input.txt` deve essere formattato in modo specifico per rappresentare la rimozione di un singolo evento. Questo file contiene l’ID dell’evento da rimuovere. Gli ID non validi vengono ignorati e la conferenza non viene cambiata

ESEMPIO 1: Dato il seguente file conference.txt:

```
4 2
3
1 0 0 Evento test 1
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

Di seguito è riportato un esempio di come deve essere formattato il file `input.txt`:

```
1
```

In questo esempio 1 indica l’ID di “Evento test 1”
Il file output.txt sarà quindi:

```
4 2
2
```

```
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

ESEMPIO 2: Considerando la conferenza dell'esempio precedente di seguito si riporta il caso in cui venga inserito un ID non valido:

```
4
```

Il valore 4 nella precedente conferenza non è ID di alcun evento.
Il file output.txt sarà quindi:

```
4 2
3
1 0 0 Evento test 1
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

5.5.2 Casi di test

- **TC055** : testiamo il caso in cui venga rimosso l'evento con ID 1.
- **TC056** : testiamo il caso in cui venga rimosso l'evento con ID 2.
- **TC057** : testiamo il caso in cui venga rimosso l'evento con ID 3.
- **TC058** : testiamo il caso in cui venga rimosso l'evento con ID 4. Il programma non considererà la rimozione perché non è presente un evento con ID 4.
- **TC059** : testiamo il caso in cui venga rimosso l'evento con ID -1. Il programma non considererà la rimozione perché non è presente un evento con ID -1.

5.6 Test per la modifica di un evento

I file conference.txt, output.txt e oracle.txt sono conferenze formattate come precedentemente descritto. Il formato del file input.txt è descritto nella sottosezione che segue.

5.6.1 Formato del file `input.txt` per i Test di Modifica Evento

Per eseguire correttamente i test, il file `input.txt` deve essere formattato in modo specifico per rappresentare la modifica di un singolo evento. Questo file contiene l'ID dell'evento che si desidera modificare, la tipologia di modifica (un numero da 1 a 4 che indica il campo che si desidera modificare: 1 - Nome, 2 - Tipo Evento, 3 - Data inizio, 4 - Data fine) , la modifica e il comando di uscita. Gli ID non validi vengono ignorati e la conferenza non viene modificata. Qualsiasi altro valore non valido richiede il reinserimento del dato.

ESEMPIO: Dato il seguente file conference.txt:

```
4 2
3
1 0 0 Evento test 1
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

Di seguito è riportato un esempio di come deve essere formattato il file `input.txt`:

```
1
1
Evento test 4
5
```

In questo esempio:

- `1` è l'ID dell'evento "Evento test 1"
- `1` indica che si desidera modificare il nome dell'evento.
- `Evento test 4` è il nuovo nome dell'evento.
- `5` indica l'uscita e il salvataggio della modifica.

Il file output.txt sarà quindi:

```
4 2
3
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1 0 0 Evento test 4
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

5.6.2 Casi di test

- **TC060** : testiamo il caso in cui venga modificato un evento già presente con ID 1.
- **TC061** : testiamo il caso in cui nella modifica di un evento già presente, nell'inserimento del nome, diamo un invio. Il programma chiederà di reinserire il nome dell'evento.
- **TC062** : testiamo il caso in cui nella modifica di un evento già presente, nell'inserimento del nome, diamo degli spazi. Il programma non considererà gli spazi.
- **TC063** : testiamo il caso in cui nella modifica di un evento già presente, nell'inserimento del nome, diamo un invio e degli spazi. Il programma ignorerà gli spazi e l'invio chiedendo di reinserire il nome dell'evento.

- **TC064** : testiamo il caso in cui nella modifica di un evento già presente, nell'inserimento del nome, venga inserito un nome che superi il limite di 100 caratteri. Il programma chiederà di inserire nuovamente il nome dell'evento.
- **TC065** : testiamo il caso in cui nella modifica di un evento già presente, nell'inserimento del nome, venga inserito un nome di esattamente 100 caratteri.
- **TC066** : testiamo il caso in cui venga scelta una tipologia di evento corretta.
- **TC067** : testiamo il caso in cui venga scelta una tipologia di evento corretta.
- **TC068** : testiamo il caso in cui venga scelta una tipologia di evento corretta.
- **TC069** : testiamo il caso in cui venga inserito "0" come tipologia di evento. Il programma ciederà nuovamente la tipologia.
- **TC070** : testiamo il caso in cui venga inserito "-1" come tipologia di evento. Il programma ciederà nuovamente la tipologia.
- **TC071** : testiamo il caso in cui venga inserito "4" come tipologia di evento. Il programma ciederà nuovamente la tipologia.
- **TC072** : testiamo il caso in cui venga modificata correttamente la data di inizio.
- **TC073** : testiamo il caso in cui si tenti di inserire una data di inizio successiva rispetto a quella di fine. Il programma chiederà nuovamente di inserire la data.
- **TC074** : testiamo il caso in cui si tenti di inserire una data di inizio che si sovrapponga alla data di un altro evento nella stessa sala. Il programma chiederà nuovamente di inserire la data.
- **TC075** : testiamo il caso in cui venga modificata correttamente la data di fine.
- **TC076** : testiamo il caso in cui si tenti di inserire una data di fine precedente rispetto a quella di inizio. Il programma chiederà nuovamente di inserire la data.
- **TC077** : testiamo il caso in cui si tenti di inserire una data di fine che si sovrapponga alla data di un altro evento nella stessa sala. Il programma chiederà nuovamente di inserire la data.

5.7 Test per l'assegnazione di una sala a un evento

I file conference.txt, output.txt e oracle.txt sono conferenze formattate come precedentemente descritto. Il formato del file input.txt è descritto nella sottosezione che segue.

5.7.1 Formato del file `input.txt` per i Test di assegnazione di una sala a un evento

Per eseguire correttamente i test, il file `input.txt` deve essere formattato in modo specifico per rappresentare l'assegnazione di una sala a un evento. Questo file contiene l'ID dell'evento a cui si vuole assegnare la sala e l'ID della sala che si desidera assegnare. Gli ID non validi vengono ignorati e la conferenza non viene modificata.

ESEMPIO: Dato il seguente file conference.txt:

```
4 2
3
1 0 0 Evento test 1
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
```

```
Sala A
50
```

Di seguito è riportato un esempio di come deve essere formattato il file `input.txt`:

```
1
1
```

In questo esempio:

- 1 è l'ID dell'evento "Evento test 1"
- 1 è l'ID della sala "Sala A".

Il file output.txt sarà quindi:

```
4 2
3
1 0 1 Evento test 1
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

5.7.2 Casi di test

- **TC078** : testiamo il caso in cui venga assegnata una sala ad un evento.
- **TC079** : testiamo il caso in cui venga assegnato ad un evento una sala già occupata nel periodo di tempo in cui si verifica.
- **TC080** : testiamo il caso in cui venga assegnata una sala ad un evento che ha la data di fine uguale alla data di inizio dell'evento successivo.
- **TC081** : testiamo il caso in cui venga assegnata una sala ad un evento che ha la data di inizio uguale alla data di fine dell'evento precedente.
- **TC082** : testiamo il caso in cui due eventi abbiano la stessa data di inizio e di fine, ma siano assegnati a sale differenti.

5.8 Test per la liberazione di una sala assegnata a un evento

I file conference.txt, output.txt e oracle.txt sono conferenze formattate come precedentemente descritto. Il formato del file input.txt è descritto nella sottosezione che segue.

5.8.1 Formato del file `input.txt` per i Test di liberazione di una sala assegnata a un evento

Per eseguire correttamente i test, il file `input.txt` deve essere formattato in modo specifico per rappresentare la liberazione di una sala a un evento. Questo file contiene l'ID dell'evento da cui si vuole rimuovere la sala. Gli ID non validi vengono ignorati e la conferenza non viene modificata.

ESEMPIO: Dato il seguente file conference.txt:

```
4 2
3
1 0 0 Evento test 1
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

Di seguito è riportato un esempio di come deve essere formattato il file `input.txt`:

```
1
```

In questo esempio:

- 1 è l'ID dell'evento "Evento test 1"

Il file output.txt sarà quindi:

```
4 2
3
1 0 0 Evento test 1
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

5.8.2 Casi di test

- **TC083** : testiamo la rimozione di una sala da un evento.
- **TC084** : testiamo la rimozione di una sala inesistente.
- **TC085** : testiamo che un numero negativo annulli la selezione di una sala.

5.9 Test per la visualizzazione del programma dela conferenza

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

ESEMPIO: Dato il seguente file conference.txt:

```
4 2
3
1 0 0 Evento test 1
```

```
0 12 1 1 1970
0 13 1 1 1970
2 1 0 Evento test 2
0 12 1 1 1970
0 13 1 1 1970
3 2 0 Evento test 3
0 12 1 1 1970
0 13 1 1 1970
1
1
Sala A
50
```

Il file output.txt sarà quindi:

```
Id: 1
Evento: "Evento test 1"
Tipo: Workshop
Data inizio: 1 Gennaio 1970, 12:00
Data fine: 1 Gennaio 1970, 13:00

Id: 2
Evento: "Evento test 2"
Tipo: Sessione di keynote
Data inizio: 1 Gennaio 1970, 12:00
Data fine: 1 Gennaio 1970, 13:00

Id: 3
Evento: "Evento test 3"
Tipo: Panel di discussione
Data inizio: 1 Gennaio 1970, 12:00
Data fine: 1 Gennaio 1970, 13:00
```

5.9.1 Casi di test

- **TC086** : testiamo la corretta visualizzazione degli eventi in ordine alfabetico, perché le date sono tutte uguali.
- **TC087** : testiamo la corretta visualizzazione degli eventi con ID non ordinati.
- **TC088** : testiamo la corretta visualizzazione degli eventi con tutte le sale assegnate.
- **TC089** : testiamo la corretta visualizzazione degli eventi, con solo due sale assegnate a due eventi.
- **TC090** : testiamo la corretta visualizzazione degli eventi seguendo l'ordine delle date.
- **TC091** : visualizzazione di eventi multipli che combinano i precedenti test case.
- **TC092** : visualizzazione di lista di eventi vuota.

6 COMPILAZIONE E ESECUZIONE PROGRAMMI

6.1 Compilazione e esecuzione dell'eseguibile principale

Per generare il file eseguibile principale, eseguire il comando:

```
make bin/progetto
```

Il file risultante si troverà nella cartella `bin` e avrà il nome `progetto`. Il programma non richiede argomenti aggiuntivi da riga di comando per essere eseguito, ma può essere fornito il nome di un file di conferenza da cui leggere o in cui scrivere.

ESEMPIO:

```
./progetto {conference.txt}
```

6.2 Compilazione e esecuzione del programma di testing

Per generare il file eseguibile per i test, eseguire il comando:

```
make tests/test
```

Il file risultante si troverà nella cartella `tests` e avrà il nome `test`. L'eseguibile deve essere eseguito nella cartella contenente i test case (`tests`). Per l'esecuzione, il programma richiede un file di test suite e un file dove scrivere i risultati.

ESEMPIO:

```
./test test_suite.txt result.txt
```