
Progetto Programmazione e Strutture Dati

Robustelli Renato
Squitieri Andrea
Vitolo Angelo

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 Date, dell'ADT Room e dell'ADT Event

Gli eventi devono essere pianificati in specifici giorni e orari. Per garantire una gestione precisa e coerente di queste informazioni, è stato sviluppato l'ADT Date. L'uso di questo ADT permette di:

- **Assicurare la validità delle date:** Le date inserite nel sistema sono sempre valide (es. nessuna data con un mese o giorno inesistente).
- **Effettuare confronti corretti tra date:** Il confronto tra due date avviene in maniera corretta e semplice.

In particolare, l'ADT Date viene utilizzato per rappresentare le date di inizio e fine di ogni evento, con una precisione che arriva al minuto.

2.2 Motivazioni per l'uso dell'ADT Room

L'ADT Room gestisce le informazioni relative alle sale all'interno del sistema di gestione delle conferenze. Esso rappresenta una sala con attributi quali il nome della sala, il numero di posti e l'eventuale disponibilità.

2.3 Motivazioni per l'uso dell'ADT RoomList

L'ADT RoomList è stato introdotto per gestire in modo efficiente un insieme di sale conferenze. Esso rappresenta una collezione di oggetti Room, permettendo di mantenere organizzate e facilmente accessibili tutte le informazioni relative alle diverse sale.

2.4 Motivazioni per l'uso dell'ADT Event

L'ADT Event è stato creato per gestire in modo efficiente le informazioni relative agli eventi all'interno del sistema. Esso rappresenta un singolo evento, includendo dettagli cruciali come il nome dell'evento, la data e l'ora di inizio e fine, e la sala assegnata.

2.5 Motivazioni per l'uso dell'ADT EventBst

L'ADT EventBst è stato introdotto per gestire in modo ordinato la raccolta di eventi all'interno del sistema. Esso rappresenta un albero binario di ricerca in cui ogni nodo contiene un singolo evento, e le relazioni di ordinamento sono definite sulla base delle proprietà degli eventi, come la data e l'ora di inizio e, se condivise da altri eventi, il nome.

L'uso di un albero binario di ricerca permette di aggiungere eventi con una complessità temporale media di $O(\log n)$. Tuttavia, per favorire l'usabilità del software, le operazioni di modifica e rimozione non possono sfruttare le proprietà dell'albero e sono quindi di complessità $O(n)$ (precedute da una ricerca lineare per ID dell'evento).

Nonostante ciò, riteniamo che l'uso di questo ADT sia giustificato, dato che le possibili alternative avrebbero comunque mantenuto una complessità temporale di $O(n)$ nella modifica e nella rimozione, ma avrebbero reso lineare anche la complessità dell'aggiunta degli eventi.

2.6 Motivazioni per l'uso dell'ADT Conference

L'ADT Conference è stato implementato per gestire in modo completo e strutturato l'organizzazione di una conferenza, inclusi gli eventi, le sale e le loro assegnazioni. Esso è progettato come un'astrazione completa della conferenza, comprendendo una collezione di eventi, una lista delle sale disponibili e le assegnazioni degli eventi alle sale.

Grazie a questa astrazione, ci assicuriamo che ciascun evento e sala siano identificati da ID univoci e che non vi siano sovrapposizioni temporali tra gli eventi assegnati alla stessa sala.

3 PROGETTAZIONE

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 *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 *log_message(LogLevel, const char*)*

Specificazione Sintattica:

- `log_message(LogLevel, const char*)`
- tipi: `LogLevel, const 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 *log_error(const char*)*

Specificazione Sintattica:

- `log_error(const char*)`
- tipi: `const 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 `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 *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 *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(const char*)*

Specificazione Sintattica:

- `my_strdup(const 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 *trim_whitespace(char*, char*, int)*

Specificazione Sintattica:

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

Specificazione Semantica:

- `trim_whitespace(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

4.3.1 Specificazione del tipo Date

Specificazione Sintattica:

- Tipo di riferimento: Date
- Tipi utilizzati: unsigned char, unsigned short
- Campi:
 - minutes: Campo rappresentante i minuti.
 - hour: Campo rappresentante le ore.
 - day: Campo rappresentante i giorni.
 - months: Campo rappresentante i mesi.
 - year: Campo rappresentante gli anni.

Specificazione Semantica:

Rappresenta una data valida con precisione al minuto.

4.3.2 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()`.

4.3.3 Funzione Date *copy_date(ConstDate)*

Specificazione Sintattica:

- `copy_date(ConstDate) -> 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()`.

4.3.4 Funzione *int cmp_date(ConstDate, ConstDate)*

Specificazione Sintattica:

- `cmp_date(ConstDate, ConstDate) -> int`
- Tipi: `ConstDate`
- 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'`.

4.3.5 Funzione *save_date_to_file(ConstDate, FILE*)*

Specificazione Sintattica:

- `save_date_to_file(ConstDate, FILE*)`
- Tipi: `ConstDate, 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'`.

4.3.6 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()`.

4.3.7 Funzione *print_date(ConstDate)*

Specificazione Sintattica:

- `print_date(ConstDate)`
- Tipi: `ConstDate`
- 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.

4.3.8 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()`.

4.3.9 Funzione: *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

4.4.1 Specificazione del tipo *Room*

Specificazione Sintattica:

- Tipo di riferimento: `Room`
- Tipi utilizzati: `unsigned int`, `char*`
- Campi:
 - `id`: Campo rappresentante l'ID dell'oggetto `Room`.
 - `name`: Campo rappresentante il nome della sala.
 - `capacity`: Campo rappresentante i posti all'interno della sala.

Specificazione Semantica:

Rappresenta una sala della conferenza.

4.4.2 Funzione: *Room new_room(const char*, unsigned int, unsigned int)*

Specificazione Sintattica:

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

4.4.3 Funzione: *Room copy_room(ConstRoom)*

Specificazione Sintattica:

- `copy_room(ConstRoom room) -> Room`
- Tipi: `ConstRoom`
- 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()`.

4.4.4 Funzione: *bool are_rooms_equal(ConstRoom, ConstRoom)*

Specificazione Sintattica:

- `are_rooms_equal(ConstRoom room_a, ConstRoom room_b) -> bool`
- Tipi: `ConstRoom`, `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`.

4.4.5 Funzione: *const char* get_room_name(ConstRoom)*

Specificazione Sintattica:

- `get_room_name(ConstRoom room) -> const char*`
- Tipi: `ConstRoom`, `const 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.

4.4.6 Funzione: `unsigned int get_room_id(ConstRoom)`

Specificazione Sintattica:

- `get_room_id(ConstRoom room) -> unsigned int`
- Tipi: `ConstRoom`, `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.

4.4.7 Funzione: `print_room(ConstRoom)`

Specificazione Sintattica:

- `print_room(ConstRoom room) -> void`
- Tipi: `ConstRoom`
- 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.

4.4.8 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()`.

4.4.9 Funzione: *save_room_to_file(ConstRoom, FILE*)*

Specificazione Sintattica:

- `save_room_to_file(ConstRoom room, FILE* file)`
- Tipi: `ConstRoom`, `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'.

4.4.10 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()`.

4.4.11 Funzione: *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.

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, nel nostro caso "3".
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:

```
1 4 4
2 3
3 1 0 1 Evento test 1
4 0 12 1 1 1970
5 0 13 1 1 1970
6 2 1 0 Evento test 2
7 0 12 1 1 1970
8 0 13 1 1 1970
9 3 2 3 Evento test 3
10 0 12 1 1 1970
11 0 13 1 1 1970
12 3
13 1
14 Sala A
15 50
16 2
17 Sala B
18 50
19 3
20 Sala C
21 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`:

```
1 NomeEvento1
2 1
3 15/06/2024 09:00
4 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:

```
1 NomeEvento1
2 1
3 15/06/2024 09:00
4 15/06/2024 08:00
5 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 di aggiunta evento, 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:

```
1 4 2
2 3
3 1 0 0 Evento test 1
4 0 12 1 1 1970
5 0 13 1 1 1970
6 2 1 0 Evento test 2
7 0 12 1 1 1970
8 0 13 1 1 1970
9 3 2 0 Evento test 3
10 0 12 1 1 1970
11 0 13 1 1 1970
12 1
13 1
14 Sala A
15 50
```

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

```
1 1
```

In questo esempio 1 indica l'ID di "Evento test 1"

Il file output.txt sarà quindi:

```
1 4 2
2 2
3 2 1 0 Evento test 2
4 0 12 1 1 1970
5 0 13 1 1 1970
6 3 2 0 Evento test 3
7 0 12 1 1 1970
8 0 13 1 1 1970
9 1
10 1
11 Sala A
12 50
```

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

1 4

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

```
1 4 2
2 3
3 1 0 0 Evento test 1
4 0 12 1 1 1970
5 0 13 1 1 1970
6 2 1 0 Evento test 2
7 0 12 1 1 1970
8 0 13 1 1 1970
9 3 2 0 Evento test 3
10 0 12 1 1 1970
11 0 13 1 1 1970
12 1
13 1
14 Sala A
15 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 3. 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

- **TC060** : testiamo il caso in cui venga modificato un evento già presente con id 0.
- **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.
- **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

- **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

- **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 della conferenza

- **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.