SPECIFICHE

Sommario

SPECIFICA ADT: Corso

Sintattica

- **Tipo di riferimento:** Corso
- **Tipi usati:** string, Data, Orario, int, FILE

Semantica

Corso rappresenta l'insieme di (ID, nome, dataLezione, oraLezione, numPartecipanti) dove:

- ID è una stringa identificativa del corso
- nome è una stringa che descrive il nome del corso
- dataLezione è la data in cui si tiene la lezione
- oraLezione rappresenta l'ora in cui si terrà la lezione
- numPartecipanti è un intero che rappresenta il numero di partecipanti al corso

Orario è l'insieme di (ora, minuti) dove:

- ora è un intero compreso tra 0 e 23
- minuti è un intero compreso tra 0 e 59

Specifica Operatori

Sintattica

- **creaOrario(int, int) → Orario**
  Crea un nuovo orario con l'ora e i minuti specificati e restituisce l'orario creato
- **creaCorso(string, string, Data, int, int, int) → Corso**
  Crea un nuovo corso con ID, nome, data della lezione, ora, minuti e numero di partecipanti specificati e restituisce il corso creato.
- **stampaCorso(Corso) → void**

Stampa a schermo le informazioni del corso fornito come parametro.

- **stampaCorsoCompatta(Corso) → void**
  Stampa le informazioni essenziali del corso su una singola riga
- **getIDCorso(Corso) → string**
  Restituisce l'ID del corso fornito come parametro.
- **getNomeCorso(Corso) → string**
  Restituisce il nome del corso fornito come parametro.
- **getDataCorso(Corso) → Data**
  Restituisce la data della lezione del corso fornito come parametro.
- **getOrario(Corso) → Orario**
  Restituisce l'orario della lezione del corso fornito come parametro.

- **confrontaOrario(Orario, Orario) → int**
  Confronta i due orari passati in input, controllando se sia uguale, minore o maggiore.
- **incrementaPartecipanti(Corso) → void**
  Aumenta di uno il numero di partecipanti al corso
- **decrementaPartecipanti(Corso) → void**
  Diminuisce di uno il numero di partecipanti al corso
- **scriviCorso(Corso, FILE*) → void**
  Scrive le informazioni del corso su un file.
- **Disponibilita(Corso) → int**
  Verifica se c'è disponibilità di posti nel corso e restituisce un booleano che indica se il corso ha ancora posti disponibili.

Semantica

- **creaOrario(ora, minuti) → o**
  Pre: $0 \leq ora < 24 \land 0 \leq minuti < 60$
  Side effect: Allocazione memoria per la struttura orario; se allocazione fallisce, stampa messaggio d'errore e restituisce NULL
- **creaCorso(ID, nome, dataLezione, ora, minuti, nPartecipanti) → co**
  Pre: $ID \neq NULL \land nome \neq NULL \land dataLezione \neq NULL \land (0 \leq ora < 24) \land (0 \leq minuti < 60) \land nPartecipanti \geq 0$
  Side effects: Allocazione memoria per la struttura corso e le stringhe ID e nome; copia della Data fornita; creazione dell'Orario associato; se allocazione fallisce, stampa messaggio d'errore e restituisce NULL
- **stampaCorso(co) →void**
  Pre: $co \neq NULL$
  Side effects: Se co è NULL, stampa "Il corso non esiste"; se co ha disponibilità, stampa su stdout le informazioni del corso (ID, nome, data, orario, numero partecipanti)
- **stampaCorsoCompatta(co) → void**
  Pre: $co \neq NULL$
  Side effects: Stampa informazioni essenziali su una riga
- **getIDCorso(co) → co->ID**
  Pre: $co \neq NULL$
  Side effects: Se co è NULL, stampa "Il corso non esiste" e restituisce NULL
- **getNomeCorso(co) → co->nome**
  Pre: $co \neq NULL$
  Side effects: Se co è NULL, stampa "Il corso non esiste" e restituisce NULL
- **getDataCorso(co) → co->dataLezione**
  Pre: $co \neq NULL$
  Side effects: Se co è NULL, stampa "Il corso non esiste" e restituisce NULL

- **getNumPartecipantiCorso(co) → co→numPartecipanti**
  Pre: co ≠ NULL
  Post: Restituisce numPartecipanti
- **getOrario(co) → co->oraLezione**
  Pre: co ≠ NULL
  Side effects: Se co è NULL, stampa "Il corso non esiste" e restituisce NULL
- **confrontaOrario(o1, o2) → int**
  Pre: o1 ≠ NULL ∧ o2 ≠ NULL
  Post:
  - i = -1 se o1 è prima di o2
  - i = 0 se o1 è uguale a o2
  - i = 1 se o1 è dopo o2

  Side effect: Se uno dei due orari è NULL, stampa "Errore: uno dei due orari è NULL" e restituisce 0
- **incrementaPartecipanti(co) → void**
  Pre: co ≠ NULL
  Post: Incrementa numPartecipanti di 1
- **decrementaPartecipanti(co) → void**
  Pre: co ≠ NULL
  Post: Decrementa numPartecipanti di 1
- **scriviCorso(co, fp) →void**
  Pre: co ≠ NULL ∧ fp ≠ NULL
  Side effect: Scrive su file fp le informazioni del corso co
- **Disponibilita(co) → int**
  Pre: co ≠ NULL
  Post: Restituisce:
  - 1 se numPartecipanti < maxPartecipanti
  - 0 altrimenti

SPECIFICA ADT: Data

Sintattica

- **Tipo di riferimento**: Data
- **Tipi usati**: int, Data

Semantica

Data è l'insieme delle tuple (giorno, mese, anno) dove:

- giorno è un intero compreso tra 1 e 31
- mese è un intero compreso tra 1 e 12
- anno è un intero che rappresenta l'anno

Specifica Operatori

Sintattica

- **creaData(int, int, int) → Data**
  Crea una nuova data con giorno, mese e anno specificati e restituisce la data creata.
- **calcoloDataScadenza(Data, int) → Data**
  Calcola data di scadenza e restituisce la nuova data.
- **copiaData(Data) → Data**
  Restituisce una copia indipendente della data fornita.
- **stampaData(Data) → void**
  Stampa la data fornita come parametro nel formato GG/MM/AAAA.
- **getGiorno(Data) → int**
  Restituisce il giorno della data fornita come parametro.
- **getMese(Data) → int**
  Restituisce il mese della data fornita come parametro.
- **getAnno(Data) → int**
  Restituisce l'anno della data fornita come parametro.
- **confrontaData(Data, Data) → int**
  Confronta due date.
- **dataOggi() → Data**
  Restituisce un oggetto Data con la data odierna del sistema

Semantica

- **creaData(giorno, mese, anno) → data**
  Pre: 1 ≤ giorno ≤ 31 AND 1 ≤ mese ≤ 12 AND anno > 0
  Side effects: Se i parametri non sono validi, stampa "Data non valida", richiede input dall'utente con scanf fino a quando non vengono inseriti valori validi; allocazione memoria per la struttura data; se allocazione fallisce, stampa "Errore allocazione memoria" e termina il programma con exit(1)

- **calcoloDataScadenza(data, durata) → scadenza**
  Pre: data ≠ NULL AND durata > 0
  Post: scadenza = (giorno di data, mese di data + durata, anno di data) con gestione del cambio anno se mese > 12
  Side effects: Se data è NULL o durata ≤ 0, stampa "Valori Errati" e restituisce NULL; allocazione memoria per la nuova struttura data; se allocazione fallisce, stampa "Errore allocazione memoria" e termina il programma con exit(1)
- **copiaData(dataOriginale) → d**
  Pre: dataOriginale ≠ NULL
  Side effects: Se dataOriginale è NULL, stampa "Valori inesistenti" e restituisce NULL; allocazione memoria per la nuova struttura data; se allocazione fallisce, stampa "Errore allocazione memoria" e termina il programma con exit(1)
- **stampaData(data) →void**
  Pre: data ≠ NULL
  Side effect: Se data è NULL, stampa "Valori inesistenti"; altrimenti stampa su stdout la data nel formato "giorno/mese/anno"
- **getGiorno(data) → data->giorno**
  Pre: data ≠ NULL
  Side effect: Se data è NULL, stampa "Valori inesistenti" e restituisce -1
- **getMese(data) → data->mese**
  Pre: data ≠ NULL
  Side effect: Se data è NULL, stampa "Valori inesistenti" e restituisce -1
- **getAnno(data) → data->anno**
  Pre: data ≠ NULL
  Side effect: Se data è NULL, stampa "Valori inesistenti" e restituisce -1
- **confrontaData(d1, d2) → int**
  Pre: d1 ≠ NULL ∧ d2 ≠ NULL
  Post: Restituisce:
  - r = -1 se d1 < d2
  - r = 0 se d1 = d2
  - r = 1 se d1 > d2

  Side effect: Se d1 o d2 sono NULL, stampa "Valori inesistenti" e termina il programma con exit(1)
- **dataOggi() → Data**
  Side effects: Restituisce una data costruita dalla data corrente del sistema.

SPECIFICA ADT: Hash

Sintattica

- **Tipo di riferimento**: hashtable
- **Tipi usati**: Iscritto, const char, string, int, FILE, hashtable

Semantica

Una hashtable rappresenta una struttura di dati composta da:

- size: dimensione della tabella (numero di slot)
- table: array di liste concatenate di Iscritto, una per slot, per gestire le collisioni

Specifica Operatori

Sintattica

- **newHashtable(int) → hashtable**
  Crea una nuova tabella hash con la dimensione specificata e restituisce la tabella creata.
- **insertHash(hashtable, Iscritto) → int**
  Inserisce un elemento Iscritto nella tabella hash.
- **hashSearch(hashtable, string) → Iscritto**
  Cerca un elemento nella tabella hash basandosi sulla chiave fornita e restituisce l'elemento trovato.
- **hashDelete(hashtable, string) → Iscritto**
  Elimina un elemento dalla tabella hash basandosi sulla chiave fornita e restituisce l'elemento eliminato.
- **destroyHashtable(hashtable) → void**
  Distrugge completamente la tabella hash liberando tutta la memoria allocata.
- **deleteList (Iscritto) → void**
  Elimina ricorsivamente tutti gli elementi di una lista concatenata di Iscritto.
- **ricercaGenerica(hashtable, int, string) → int**
  Effettua una ricerca generica nella tabella hash basata su diversi criteri.
- **stampaHash(hashtable) → void**
  Stampa tutti gli elementi presenti nella tabella hash.
- **stampaHashMinima(hashtable) → void**
  Stampa solo ID, nome e cognome degli iscritti.
- **scriviFileClienti(hashtable) → void**
  Scrive tutti gli elementi della tabella hash su file.

Semantica

- **newHashtable(size) → h**
  Pre: size > 0
  Post: h è una nuova tabella hash con dimensione size e tutti i bucket inizializzati a NULL
  Side effects: Allocazione memoria per la struttura hash e per l'array di puntatori; se allocazione fallisce, stampa "Errore di Allocazione" e termina il programma con exit(1)
- **insertHash(h, elem) → int**
  Pre: h ≠ NULL AND elem ≠ NULL
  Post: se elem non è già presente in h, viene inserito in testa alla lista del bucket appropriato e result = 1; altrimenti result = 0
  Side effects: Calcolo dell'indice hash; attraversamento della lista concatenata nel bucket; modifica dei puntatori per l'inserimento in testa
- **hashSearch(h, key) → curr**
  Pre: key ≠ NULL ∧ h ≠ NULL
  Post: se esiste un elemento con ID key, allora x = elemento; altrimenti x = NULL.
- **hashDelete(h, key) → curr**
  Pre: h ≠ NULL ∧ key ≠ NULL
  Post: se esiste un elemento con chiave key in h, viene rimosso e restituito; altrimenti restituisce NULL
  Side effects: Calcolo dell'indice hash; attraversamento della lista concatenata; modifica dei puntatori per la rimozione
- **destroyHashtable(h) → void**
  Pre: h ≠ NULL
  Post: tutta la memoria allocata per h viene liberata
  Side effects: Liberazione ricorsiva di tutte le liste concatenate nei bucket; liberazione dell'array table; liberazione della struttura hash
- **deleteList(p) → static void**
  Pre: p può essere NULL o puntatore valido a Iscritto
  Post: tutti gli elementi della lista concatenata che inizia con p vengono liberati dalla memoria
  Side effects: Se p è NULL, nessun effetto; altrimenti chiamata ricorsiva su tutti i nodi successivi e liberazione della memoria di ogni nodo con free(); rischio di stack overflow su liste molto lunghe

- **ricercaGenerica(h, sel, str) → trovato**
  Pre: h ≠ NULL ∧ str ≠ NULL ∧ (sel = 0 OR sel = 1 OR sel = 2)
  Post: esegue la ricerca nei bucket in base a:
    - sel = 0 → per nome
    - sel = 1 → per cognome
    - sel = 2 → per durata (convertita da string a int)

  Ritorna true se è stato trovato almeno un elemento.

  Side effect: stampa su stdout gli elementi trovati.
- **stampaHash(h) → void**
  Pre: h ≠ NULL
  Side effects: Se h è NULL, stampa "Tabella vuota"; altrimenti stampa intestazione formattata e tutti gli elementi della tabella tramite stampaCliente(); attraversamento completo della tabella e delle liste concatenate
- **stampaHashMinima(h) → void**
  Pre: h ≠ NULL
  Side effects: Se h è NULL, stampa "Tabella vuota"; altrimenti stampa intestazione con formato tabellare (ID, Cognome, Nome) e tutti gli elementi tramite stampaMinimaCliente(); attraversamento completo della tabella e delle liste concatenate
- **scriviFileClienti(h) → void**
  Pre: h ≠ NULL
  Side effects: Apertura del file "iscritti.txt" in modalità scrittura; se apertura fallisce, stampa "Errore apertura file iscritti" e termina con exit(0); scrittura di tutti gli elementi su file tramite scriviCliente(); chiusura del file; attraversamento completo della tabella e delle liste concatenate

SPECIFICA ADT: Iscritto

Sintattica

- **Tipo di riferimento**: Iscritto
- **Tipi usati**: string, Data, int, FILE, Iscritto

Semantica

Iscritto è una struttura composta da (ID, nome, cognome, dataIscrizione, dataScadenza, durata, next) dove:

- ID: identificativo univoco dell'iscritto
- nome: nome dell'iscritto
- cognome: cognome dell'iscritto
- dataIscrizione: data di registrazione dell'iscritto
- dataScadenza: data di scadenza dell'abbonamento
- durata: durata dell'abbonamento espressa in mesi
- next è un puntatore a un altro oggetto Iscritto (lista concatenata)

Specifica Operatori

Sintattica

- **CreaIscritto(string, string, Data, int, string) → Iscritto**
  Crea un nuovo iscritto con i dati specificati e restituisce l'iscritto creato.
- **getNext(Iscritto) → Iscritto**
  Restituisce il puntatore al prossimo iscritto nella sequenza.
- **setNext(Iscritto, Iscritto) → void**
  Imposta il puntatore al prossimo iscritto nella sequenza.
- **getID(Iscritto) → string**
  Restituisce l'ID dell'iscritto.
- **getNome(Iscritto) → string**
  Restituisce il nome dell'iscritto.
- **getCognome(Iscritto) → string**
  Restituisce il cognome dell'iscritto.
- **getDurata(Iscritto) → int**
  Restituisce la durata dell'abbonamento in mesi.
- **getDataScadenza(Iscritto is) → Data**
  Restituisce la data di scadenza dell'abbonamento
- **eliminaIscritto(Iscritto) → void**
  Dealloca tutta la memoria associata all'iscritto.
- **rinnovaAbbonamento(Iscritto, int) → void**
  Aumenta la durata dell'abbonamento e aggiorna la data di scadenza.
- **stampaMinimaCliente(Iscritto) → void**
  Stampa le informazioni essenziali dell'iscritto (ID, cognome, nome).

- **stampaCliente(Iscritto) → void**
  Stampa tutte le informazioni dell'iscritto.
- **scriviCliente(Iscritto, FILE*) → void**
  Scrive le informazioni su file.

Semantica

- **CreaIscritto(nome, cognome, dataIscrizione, durata, ID) → Is**
  Pre: nome ≠ nil ∧ cognome ≠ nil ∧ dataIscr ≠ nil ∧ dur > 0 ∧ id ≠ nil
  Side Effect: alloca memoria dinamica per i, i.nome, i.cognome, i.ID; in caso di errore di allocazione termina il programma
- **getNext(is) → is→next**
  Pre: is ≠ nil
  Side effect: restituisce il campo next della struttura Iscritto
- **setNext(is, next) → void**
  Pre: is ≠ nil
  Side effect: modifica il collegamento della lista
- **getID(is) → is→ID**
  Pre: is ≠ nil
  Side effect: restituisce il campo ID della struttura Iscritto
- **getNome(is) → is→nome**
  Pre: is ≠ nil
  Side effect: restituisce il campo nome della struttura Iscritto
- **getCognome(is) → is→cognome**
  Pre: is ≠ nil
  Side effect: restituisce il campo cognome della struttura Iscritto
- **getDurata(is) → is→durata**
  Pre: is ≠ nil
  Side effect: restituisce il campo durata della struttura Iscritto
- **getDataScadenza(is) → is→ dataScadenza**
  Pre: is ≠ NULL
  Side effect: restituisce il campo dataScadenza
- **eliminaIscritto(is) → void**
  Pre: is ≠ nil
  Side effect: chiama free() su tutti i campi dinamici e su is stesso
- **rinnovaAbbonamento(is, durata) → void**
  Pre: is ≠ NULL ∧ durata > 0
  Side effect: stampa errore se parametri invalidi
- **stampaMinimaCliente(is) → void**
  Pre: is ≠ nil
  Side effect: stampa a schermo ID,cognome e nome
- **controlloAbbonamento(is) → intPre**: is ≠ NULL
  Post: Restituisce 1 se scaduto, 0 se valido

- **stampaCliente(is) → void**
  Pre: is ≠ nil
  Side effect: stampa a schermo tutte le informazioni dell'iscritto
- **scriviCliente(is, fp) → void**
  Pre: is ≠ nil ∧ fp ≠ NULL
  Side effect: modifica il contenuto del file puntato da fp

SPECIFICA ADT: Liste

Sintattica

- **Tipo di riferimento**: list
- **Tipi usati**: Corso, string, Data, Orario, int, list

Semantica

List rappresenta una lista dinamica di oggetti Corso, implementata come lista concatenata. Ogni nodo contiene un corso e un puntatore al nodo successivo. Il modulo supporta operazioni di inserimento, rimozione, ricerca, ordinamento e stampa.

Specifica Operatori

Sintattica

- **newList() → list**
  Crea una nuova lista vuota e restituisce la lista creata.
- **isEmpty(list) → int**
  Verifica se la lista passata come argomento è vuota e restituisce un booleano che indica se la lista è vuota o meno.
- **insertList(list, int, Corso) → int**
  Inserisce un nuovo corso nella posizione specificata della lista e restituisce true se l'operazione ha successo.
- **insertNode(struct node*, int, Corso) → static struct node***
  Questa è una funzione ausiliaria (interna) che gestisce l'inserimento di un nodo nella lista, usata da insertList
- **removeList(list, int) → int**
  Rimuove l'elemento nella posizione specificata dalla lista e restituisce true se l'operazione ha successo.
- **removeNode(struct node*, int) → static struct node***
  Questa funzione interna rimuove il nodo in posizione pos dalla lista puntata da l e restituisce la (eventuale nuova) testa della lista.
- **reverseList(list) → list**
  Restituisce una nuova lista con gli elementi in ordine inverso rispetto alla lista originale.
- **ricercaGenericaLista(list, int, string) → list**
  Cerca i corsi nella lista in base al criterio specificato (ID o nome) e restituisce una lista con i risultati.
- **ricercaData(list, Data) → list**
  Restituisce tutti i corsi con la data uguale a quella passata.
- **ricercaMese(list, int) → list**
  Restituisce una lista dei corsi che si tengono nel mese specificato

- **lezioniInEvidenza(list) → list**
  Restituisce una lista con i 3 corsi con più partecipanti
- **ricercaOrario(list, int, int) → list**
  Restituisce tutti i corsi che si tengono all'orario specificato.
- **stampaLista(list) → void**
  Stampa tutti gli elementi della lista.
- **stampaListaEssenziale(list) → void**
  Stampa i corsi in formato compatto tabellare (ID, nome, data, orario)
- **scriviFileCorso(list) → void**
  Scrive tutti i corsi della lista nel file "corsi.txt".
- **getFirstCorso (list) → corso**
  Restituisce il primo corso della lista.
- **cancellaCorso(list, string) → int**
  Rimuove il corso con ID uguale a IDCorso passato, se presente
- **scriviLezioniInEvidenza(list, FILE*) → void**
  Scrive su file i 3 corsi con più partecipanti (se presenti)

Semantica

- **newList() → l**
  Side effect: Alloca memoria dinamica per la struttura lista; in caso di errore di allocazione restituisce nil
- **isEmpty(l) → int**
  Post: 1 se lista è vuota o NULL, 0 altrimenti
- **insertList(l, pos, val) → int**
  Pre: l ≠ nil ∧ pos ≥ 0 ∧ c ≠ nil
  Post: se l'inserimento ha successo allora b = true AND l.size = l.size + 1 AND c'è inserito in posizione pos, altrimenti b = false
  Side effect: Alloca memoria dinamica per il nuovo nodo; modifica la struttura della lista; in caso di errore di allocazione restituisce false
- **insertNode(l, pos, val) → l**
  Pre:0 ≤ pos ≤ size(l)
  Post: Se pos == 0:
  la funzione restituisce il nuovo nodo come nuova testa della lista:
  l' = <val, c_1, c_2, ..., c_n>
  Se pos > 0:
  inserisce val nella posizione pos, restituendo la testa invariata:
  l' = <c_1, ..., c_{p-1}, val, c_p, ..., c_n>
  Se pos è oltre i limiti della lista, restituisce NULL.
  Side effect: Alloca dinamicamente memoria per un nuovo nodo (struct node)
  Se malloc fallisce, stampa errore e restituisce NULL
  Se pos è invalida (oltre la lunghezza della lista), libera la memoria allocata e restituisce NULL**,** modifica l'ordine dei puntatori nella lista

- **removeList(l, pos) → int**
  Pre: l ≠ nil ∧ 0 ≤ pos < size(l)
  Post: se la rimozione ha successo allora b = true AND l.size = l.size - 1 AND l'elemento in posizione pos è rimosso, altrimenti b = false
  Side Effect: dealloca memoria del nodo rimosso; modifica la struttura della lista
- **removeNode(l, pos) → l**
  Pre: 0 ≤ pos < size(l)
  Post:
  Se pos == 0, rimuove il primo nodo e restituisce l1 = tail(l)
  Se pos > 0, rimuove il nodo alla posizione pos e restituisce la testa invariata
  Se la posizione è non valida, la lista non viene modificata e viene restituito l'originale
  Side effect: Libera (con free) la memoria associata al nodo rimosso, se l == NULL o pos è fuori dai limiti, non esegue alcuna rimozione e modifica i puntatori di collegamento tra nodi nella lista
- **reverseList(l) → rev**
  Pre: l ≠ nil
  Post: Restituisce nuova lista con gli elementi in ordine inverso
  l = <c1, c2, ..., cn> AND l' = <cn, ..., c2, c1>
  Side effect: Alloca memoria per una nuova lista e tutti i suoi nodi; in caso di errore di inserimento termina il programma con messaggio di errore
- **ricercaGenericaLista(l, sel, str) → result**
  Pre: l ≠ nil ∧ sel ∈ {0,1} ∧ str ≠ nil
  Post: l' contiene tutti i corsi di l che soddisfano il criterio di ricerca
  Side effect: Alloca memoria per una nuova lista; se l = nil stampa messaggio "Lista vuota"
- **ricercaData(l, data) → result**
  Pre: l ≠ nil ∧ data ≠ nil
  Post: l' contiene tutti i corsi con data uguale a data
  Side effect: Alloca memoria per una nuova lista; se l = nil stampa messaggio "Lista vuota"
- **ricercaMese(l, mm) → result**
  Pre: 1 ≤ mm ≤ 12
  Post: Restituisce lista con corsi del mese mm
- **lezioniInEvidenza(l) → classifica**
  Pre: l ≠ nil
  Post: Restituisce i 3 corsi con più partecipanti, ordinati

- **ricercaOrario(l, h, m) → result**
  Pre: l ≠ nil AND 0 ≤ h ≤ 23 AND 0 ≤ m ≤ 59
  Post: l' contiene tutti i corsi che si tengono alle h:m
  Side effect: Alloca memoria per una nuova lista e per l'orario di ricerca; se l = nil stampa messaggio "Lista vuota"; in caso di errore nella creazione dell'orario termina il programma
- **stampaLista(l) → void**
  Side effect: stampa tutti gli elementi di l, se l = nil o l.first = nil stampa messaggio "La lista è vuota o non inizializzata"
- **stampaListaEssenziale(l) → void**
  Pre: l ≠ nil ∧ l.first ≠ nil
  Side effects: Stampa a video i corsi in formato compatto (ID, Nome, Data, Orario)
- **scriviFileCorso(l) → void**
  Pre: l ≠ nil
  Post: tutti i corsi di l vengono scritti nel file "corsi.txt"
  Side effect: Apre un file in scrittura, scrive il contenuto su disco e stampa messaggio e termina (exit) se il file non è accessibile
- **getFirstCorso(l) → l->first->c**
  Pre: l ≠ nil ∧ l.first ≠ nil
  Side Effect: se l = nil o l.first = nil stampa messaggio di errore e restituisce nil
- **cancellaCorso(l, IDCorso) → int**
  Pre: l ≠ nil ∧ l.first ≠ nil
  Post: Rimuove il corso con ID IDCorso, se presente, restituisce:
  1 se rimosso, 0 altrimenti
- **scriviLezioniInEvidenza(l, fp) → void**
  Pre: l ≠ nil ∧ l.first ≠ nil
  fp aperto in scrittura
  Post: Scrive i corsi in evidenza (ID, nome, data, partecipanti); se lista vuota, scrive messaggio

SPECIFICA ADT: ListaPrenotazione

Sintattica

- **Tipo di riferimento:** listP
- **Tipi usati:** Prenotazione, Data, int, FILE, list, listP

Semantica

listP è l'insieme delle sequenze ordinate: L = p_1, p_2, ..., p_n, dove ogni p_i è un elemento di tipo Prenotazione.

L'insieme include anche un valore nil, che rappresenta la lista vuota.
Ogni lista è rappresentata da:

- first: puntatore al primo nodo
- size: numero di elementi nella lista

Ogni nodo contiene un elemento Prenotazione e un puntatore al nodo successivo.

Specifica Operatori

Sintattica

- **newListPrenotati() → listP**
  Crea una nuova lista di prenotazioni vuota e restituisce la lista creata.
- **insertListPrenotati(listP, int, Prenotazione) → int**
  Inserisce una prenotazione nella posizione specificata della lista e restituisce 1 se l'operazione ha successo, 0 altrimenti.
- **insertNode(struct node*, int, Prenotazione) → static struct node***
  Questa funzione interna inserisce un nuovo nodo contenente una Prenotazione nella posizione pos all'interno della lista collegata puntata da l, e restituisce la (possibilmente nuova) testa della lista.
- **removeListPrenotati(listP, int) → int**
  Rimuove la prenotazione nella posizione specificata dalla lista e restituisce 1 se l'operazione ha successo, 0 altrimenti.
- **removeNode(struct node*, int) → static struct node***
  Questa funzione interna rimuove dalla lista il nodo in posizione pos e restituisce il puntatore alla (possibilmente nuova) testa della lista
- **reverseListPrenotazioni(list) → listP**
  Restituisce una nuova lista con l'ordine delle prenotazioni invertito.
- **ricercaListaPrenotati (listP, int, string) → listP**
  Ricerca le prenotazioni in base al criterio specificato (0=ID Cliente, 1=ID Corso) e restituisce una nuova lista contenente le prenotazioni trovate.
- **cancellaPrenotazione(listP, string, string) → int**
  Cancella una specifica prenotazione
- **cancellaPrenotazioneDi(listP, int, string) → int**
  Cancella tutte le prenotazioni di un cliente o corso

- **getFirstPrenotazione(listP) → Prenotazione**
  Restituisce la prima prenotazione senza rimuoverla
- **isEmptyPrenotazione (listP) → int**
  Verifica se la lista di prenotazioni è vuota e restituisce 1 se vuota, 0 altrimenti.
- **ricercaMesePrenotazione(listP, int) → listP**
  Ricerca prenotazioni per mese
- **stampaListaPrenotazioni(listP) → void**
  Stampa tutte le prenotazioni presenti nella lista.
- **scriviFilePrenotazione(listP) → void**
  Scrive tutte le prenotazioni della lista su file.
- **getSize(listP) → int**
  Restituisce la dimensione della lista

Semantica

- **newListPrenotati() → l**
  Post: l ≠ NULL ∧ l.size = 0 ∧ l.first = NULL
  Side effect: alloca memoria per la struttura della lista, se allocazione fallisce, restituisce NULL
- **insertListPrenotati(l, pos, val) → int**
  Pre: l ≠ NULL ∧ 0 ≤ pos ≤ size(l)
  Side effect: Allocazione memoria per nuovo nodo; se allocazione fallisce, restituisce 0; aggiorna puntatori lista
- **insertNode(l, pos, val) → l**
  Post: restituisce il nuovo puntatore alla testa della lista con val inserita in posizione pos
  Side effect: Alloca memoria per un nodo, stampa errore e restituisce NULL se malloc fallisce e libera new se la posizione è invalida
- **removeListPrenotati(l, pos) → int**
  Pre: l ≠ nil ∧ 0 ≤ pos < size(l)
  Side effect: Modifica della lista l, deallocazione di memoria del nodo rimosso, decremento della dimensione della lista
- **removeNode(l, pos) → l**
  Post: restituisce la testa aggiornata dopo la rimozione del nodo in posizione pos
  Side effect: Libera memoria del nodo rimosso, non modifica se posizione non valida
- **reverseListPrenotazioni(l) → rev**
  Pre: l ≠ NULL
  Post: l = <p1, p2, ..., pn> AND l' = <pn, ..., p2, p1>
  Side effect: Creazione di una nuova lista, allocazione di memoria per i nuovi nodi

- **ricercaListaPrenotati(l, sel, str) → result**
  Pre: l ≠ NULL ∧ sel ∈ {0, 1, 2} ∧ str ≠ NULL
  Post: result contiene tutte le prenotazioni di l che soddisfano il criterio di ricerca specificato
  Side Effect: Creazione di una nuova lista result, allocazione di memoria per i nodi trovati, output su console se lista vuota
- **cancellaPrenotazione(l, IDPrenotazione, IDCliente) → result**
  Pre: l ≠ NULL ∧ IDPrenotazione ≠ NULL ∧ IDCliente ≠ NULL
  Post: result = 1 se prenotazione trovata e cancellata, 0 altrimenti
  Side effects: Rimozione elemento dalla lista se trovato; aggiorna dimensione lista
- **cancellaPrenotazioneDi(l, sel, ID) → elementiCancellati**
  Pre: l ≠ NULL ∧ sel ∈ {0, 1} ∧ ID ≠ NULL
  Side effects: Se l = NULL, stampa "Lista vuota" e restituisce 0; rimozione multipla elementi; aggiorna dimensione lista
- **getFirstPrenotazione(l) → l→first→p**
  Pre: l ≠ NULL ∧ l.size > 0
  Side effects: Se l = NULL, stampa "Lista vuota" e restituisce 0
- **isEmptyPrenotazione(l) → int**
  Post: se l = nil OR l->first = nil, ritorna 1, 0 altrimenti
- **ricercaMesePrenotazione(l, mm) → result**
  Pre: l ≠ NULL ∧ mm ∈ {1, 2, ..., 12}
  Post: result contiene tutte le prenotazioni di l del mese mm ∧ l rimane invariata
  Side effects: Se l = NULL, stampa "Lista vuota" e restituisce 0; allocazione memoria per lista risultato
- **stampaListaPrenotazioni(l) → void**
  Post: stampa su stdout tutte le prenotazioni della lista
  Side effect: Se la lista è vuota o NULL, stampa messaggio "Non ci sono prenotazioni".
- **scriviFilePrenotazione(l) → void**
  Pre: l ≠ NULL
  Post: scrive su "prenotazioni.txt" le informazioni di ogni prenotazione
  Side effect: Apre file "prenotazioni.txt" in scrittura; se apertura fallisce, stampa "Errore apertura file iscritti" e termina programma; scrittura su file; chiusura file
- **getSize(l) → l→size**
  Pre: l ≠ NULL
  Side effects: Restituisce la dimensione della lista

SPECIFICA ADT: Prenotazione

Sintattica

- **Tipo di riferimento:** Prenotazione
- **Tipi usati:** string, Data, FILE

Semantica

Una prenotazione rappresenta un'associazione tra un cliente e un corso in una specifica data.
È una struttura composta da (IDPrenotazione, IDCorso, IDCliente, dataPrenotazione)

dove:

- IDPrenotazione – identificativo univoco della prenotazione (es. "PRT001")
- IDCorso – identificativo del corso prenotato
- IDCliente – identificativo del cliente che ha prenotato
- dataPrenotazione – data della prenotazione

Specifica degli Operatori

Sintattica

- **creaPrenotazione(string, string, string, Data) → Prenotazione**
  Crea e restituisce una nuova prenotazione con ID, cliente, corso e data associati.
- **eliminaPrenotazione(Prenotazione) → void**
  Libera tutta la memoria associata a una prenotazione.
- **getDataPrenotazione(Prenotazione) → Data**
  Restituisce la data della prenotazione.
- **getIDCorsoPrenotazione (Prenotazione) → string**
  Restituisce l'identificatore del corso associato alla prenotazione.
- **getIDClientePrenotazione (Prenotazione) → string**
  Restituisce l'identificatore del cliente associato alla prenotazione.
- **getIDPrenotazione (Prenotazione) → string**
  Restituisce l'identificatore univoco della prenotazione.
- **stampaPrenotazione(Prenotazione) → void**
  Stampa i dettagli della prenotazione a video.
- **scriviPrenotazione(Prenotazione, FILE*) → void**
  Scrive i dati della prenotazione su file.

Semantica

- **creaPrenotazione(IDPrenotazione, IDCorso, IDCliente, dataPrenotazione) → pr**
  Pre: IDPrenotazione ≠ NULL ∧ IDCorso ≠ NULL ∧ IDCliente ≠ NULL ∧ dataPrenotazione ≠ NULL ∧ dataPrenotazione è data valida
  Post: Crea un oggetto Prenotazione con i campi assegnati
  Side effect: Allocazione memoria per struttura prenotazione e tutti i campi stringa; se parametri non validi, stampa "Valori sbagliati o inesistenti" e restituisce NULL; se allocazione memoria fallisce, stampa "Errore allocazione memoria" e restituisce NULL; se copia data fallisce, stampa "Errore nella data" e restituisce NULL; utilizza strcpy per copiare stringhe e copiaData per copiare la data
- **eliminaPrenotazione(pr) → void**
  Post: Se pr ≠ NULL, tutta la memoria associata a pr viene liberata ∧ pr non è più accessibile
  Side effect: Se pr = NULL, stampa "La prenotazione non esiste"; libera memoria di IDPrenotazione, IDCorso, IDCliente, dataPrenotazione e struttura principale
- **getDataPrenotazione(pr) → pr→dataPrenotazione**
  Pre: pr ≠ NULL
  Side effect: Se pr = NULL, stampa "La prenotazione non esiste" e termina programma con exit(1), restituisce il campo pr->dataPrenotazione
- **getIDCorsoPrenotazione (pr) → pr→IDCorso**
  Pre: pr ≠ NULL
  Side effect: Se pr = NULL, stampa "La prenotazione non esiste" e termina programma con exit(1), restituisce il campo pr->IDCorso
- **getIDClientePrenotazione (pr) → pr→IDCliente**
  Pre: pr ≠ NULL
  Side effect: Se pr = NULL, stampa "La prenotazione non esiste" e termina programma con exit(1),restituisce il campo pr->IDCliente
- **getIDPrenotazione (pr) → pr→IDPrenotazione**
  Pre: pr ≠ NULL
  Side effect: Se pr = NULL, stampa "La prenotazione non esiste" e termina programma con exit(1), restituisce il campo pr->IDPrenotazione
- **stampaPrenotazione(p) → void**
  Pre: pr deve essere una prenotazione valida
  Post: Se pr ≠ NULL, tutti i dettagli di pr vengono stampati a video
  Side effect: Se p = NULL, stampa "La prenotazione non esiste", stampa "ID Prenotazione:", "ID Corso:", "ID Cliente:", "Data Prenotazione:" seguiti dai rispettivi valori; utilizza stampaData() per stampare la data

- **scriviPrenotazione(p, fp) → void**
  Pre: pr ≠ NULL**,** fp è un puntatore a file aperto in scrittura
  Side effect: I dati della prenotazione vengono scritti nel file puntato da fp nel formato "IDPrenotazione IDCorso IDCliente giorno mese anno"

SPECIFICA ADT: Test

Sintattica

- **Tipo di riferimento:** Test
- **Tipi usati:** int, string, hashtable, list, listP, FILE*

Semantica

Test è l'insieme delle operazioni per l'esecuzione automatica di test su un sistema di prenotazioni corsi. Il modulo gestisce l'esecuzione di test suite definite in file esterni e la verifica dei risultati tramite confronto con oracle.

Specifica Operatori

Sintattica

- **mainTest(hashtable, list, listP) → void**
  Esegue la test suite principale leggendo i test da file
- **runTest(string, int, hashtable, list, listP) → int**
  Esegue un singolo test specifico
- **testCorrettaRegistrazionePrenotazione(string, string, string, hashtable, list) → int**
  Testa la registrazione di prenotazioni
- **testValiditàAbbonamento(string, string, string, hashtable) → int**
  Testa la validità degli abbonamenti
- **testReport(string, string, string, hashtable) → int**
  Testa la generazione di report
- **confrontaFile(string, string) → int**
  Confronta due file per verificarne l'uguaglianza

Semantica

- **mainTest(hClienti, lCorsi, lPrenotazioni) → void**
  Pre: hClienti ≠ NULL ∧ lCorsi ≠ NULL ∧ lPrenotazioni ≠ NULL ∧ file "testSuite.txt" esiste
  Post: Vengono eseguiti tutti i test specificati nel file testSuite.txt e i risultati vengono scritti in "testResult.txt"
  Side effects: Apre e legge "testSuite.txt"; crea e scrive "testResult.txt"; se apertura file fallisce, stampa "Errore apertura file"; allocazione memoria per nomeTest; se allocazione fallisce, stampa "Errore di Allocazione"; chiude i file e libera la memoria allocata
- **runTest(nomeTest, sel, hClienti, lCorsi, lPrenotazioni) → value**
  Pre: nomeTest ≠ NULL ∧ sel ∈ {1, 2, 3} ∧ hClienti ≠ NULL ∧ lCorsi ≠ NULL ∧ lPrenotazioni ≠ NULL
  Post: risultato = 1 se il test è passato, 0 altrimenti

Side effects: Allocazione memoria per stringhe input, output, oracle; costruisce nomi file basati su nomeTest; esegue il test specifico in base a sel; libera memoria allocata

- **testCorrettaRegistrazionePrenotazione(input, output, oracle, hClienti, lCorsi) → int**
  Pre: input ≠ NULL ∧ output ≠ NULL ∧ oracle ≠ NULL ∧ hClienti ≠ NULL ∧ lCorsi ≠ NULL ∧ file input e oracle esistono
  Post: risultato = 1 se il test output corrisponde all'oracle, 0 altrimenti **Side effects:** Apre file input e output; se apertura fallisce, stampa "Errore apertura file" e restituisce 0; allocazione memoria per IDCliente e IDCorso; se allocazione fallisce, stampa "Errore nell'allocazione delle stringhe" e restituisce 0; legge coppie IDCorso-IDCliente; verifica esistenza cliente e corso; crea prenotazioni di test; scrive risultati nel file output; chiude file e libera memoria
- **testValiditàAbbonamento(input, output, oracle, hClienti) → int**
  Pre: input ≠ NULL ∧ output ≠ NULL ∧ oracle ≠ NULL ∧ hClienti ≠ NULL ∧ file input e oracle esistono
  Post: risultato = 1 se il test output corrisponde all'oracle, 0 altrimenti.
  Side effects: Apre file input e output; se apertura fallisce, stampa "Errore apertura file" e restituisce 0; allocazione memoria per IDCliente; se allocazione fallisce, stampa "Errore nell'allocazione delle stringhe" e restituisce 0; legge ID clienti; verifica esistenza e validità abbonamento; scrive stato abbonamento nel file output; chiude file e libera memoria
- **testReport(input, output, oracle, hClienti) → int**
  Pre: input ≠ NULL ∧ output ≠ NULL ∧ oracle ≠ NULL ∧ hClienti ≠ NULL ∧ file input e oracle esistono ∧ file corsi e prenotazioni disponibili
  Post: risultato = 1 se il report generato corrisponde all'oracle, 0 altrimenti
  Side effects: Apre file input e output; se apertura fallisce, stampa "Errore apertura file" e restituisce 0; crea nuove strutture dati temporanee; carica corsi e prenotazioni da file; allocazione memoria per IDCliente e IDCorso; se allocazione fallisce, stampa "Errore nell'allocazione delle stringhe" e restituisce 0; legge prenotazioni di test; verifica esistenza cliente e corso; aggiunge prenotazioni valide; genera report mensile; scrive report nel file output; chiude file e libera memoria
- **confrontaFile(file1, file2) → uguali**
  Pre: file1 ≠ NULL ∧ file2 ≠ NULL ∧ entrambi i file esistono
  Post: uguali = 1 se i file sono identici, 0 altrimenti
  Side effects: Apre entrambi i file in lettura; se apertura fallisce, stampa "Errore apertura file" e restituisce 0; allocazione memoria per stringhe di confronto; se allocazione fallisce, stampa "Errore apertura file" e restituisce 0; legge e confronta linea per linea; chiude file e libera memoria allocata.

SPECIFICA ADT: Utils

Utils è l'insieme delle operazioni di utilità per la gestione del sistema di prenotazioni corsi, l'ADT Utils fornisce funzioni ausiliarie per:

- la generazione di ID univoci per clienti, prenotazioni e corsi(CLT###, CRS###, PRT###)
- Caricamento di dati da file
- Operazioni di sistema per la gestione dell'interfaccia utente

Specifica Operatori

Sintattica

- **generaIDCliente() → string**
  Genera e restituisce un nuovo ID cliente univoco nel formato "CLT###".
- **generaIDPrenotazione() → string**
  Genera e restituisce un nuovo ID prenotazione nel formato "PRT###".
- **generaIDCorso() → string**
  Genera e restituisce un nuovo ID corso nel formato "CRS###".
- **caricaFileClienti(hashtable) → void**
  Carica da file (iscritti.txt) i dati dei clienti e li inserisce nella hashtable fornita.
- **caricaFileCorso(list) → void**
  Carica da file (corsi.txt) i dati dei corsi e li inserisce nella lista fornita.
- **pulisciSchermo() → void**
  Pulisce lo schermo della console
- **caricaFilePrenotazioni (listP) → void**
  Carica i dati delle prenotazioni dal file "prenotazioni.txt" nella lista di prenotazioni fornita.

Semantica

- **generaIDCliente() → IDCliente**
  Post: ritorna un nuovo ID cliente del tipo "CLT###", dove ### è un numero progressivo a 3 cifre
  Side effect: Incrementa contatore globale IDCounterCliente; allocazione memoria per stringa; se allocazione fallisce, stampa "Errore allocazione memoria" e termina programma
- **generaIDPrenotazione() → IDPrenotazione**
  Post: ritorna un nuovo ID prenotazione del tipo "PRT###", dove ### è un numero progressivo a 3 cifre
  Side effect: Incrementa contatore globale IDCounterPrenotazione; allocazione memoria per stringa; se allocazione fallisce, stampa "Errore allocazione memoria" e termina programma

- **generaIDCorso() → IDCorso**
  Post: ritorna un nuovo ID corso del tipo "CRS###", dove ### è un numero progressivo a 3 cifre
  Side effect: Incrementa contatore globale IDCounterCorso; allocazione memoria per stringa; se allocazione fallisce, stampa "Errore allocazione memoria" e termina programma
- **caricaFileClienti(h) → void**
  Pre: h ≠ NULL ∧ h è hashtable inizializzata ∧ file "iscritti.txt" esiste e deve essere accessibile in lettura
  Post: tutti i clienti presenti nel file vengono inseriti nella hashtable h ∧ IDCounterCliente è aggiornato al massimo ID presente nel file
  Side effect: apertura e chiusura del file "iscritti.txt", modifica del contenuto della hashtable h attraverso inserimenti, Aggiornamento IDCounterCliente, in caso di errore (file non trovato, allocazione memoria, inserimento fallito) il programma termina
- **caricaFileCorso(l) → void**
  Pre: l ≠ NULL ∧ l è lista inizializzata ∧ file "corsi.txt" esiste e deve essere accessibile in lettura
  Post: Tutti i corsi letti vengono inseriti in testa nella lista l ∧ IDCounterCorso è aggiornato al massimo ID presente nel file
  Side effect: apertura e chiusura del file "corsi.txt", modifica del contenuto della lista l attraverso inserimenti, aggiornamento IDCounterCorso, alloca e libera memoria dinamica, in caso di errore (file non trovato, allocazione memoria, inserimento fallito) il programma termina
- **pulisciSchermo() → void**
  Side effect: Lo schermo della console viene pulito
- **caricaFilePrenotazioni(l) → void**
  Pre: l ≠ NULL ∧ l è lista prenotazioni inizializzata ∧ file "prenotazioni.txt" esiste e deve essere accessibile in lettura
  Post: tutte le prenotazioni presenti nel file vengono inserite nella lista l ∧ IDCounterPrenotazione è aggiornato al massimo ID presente nel file
  Side Effect: apertura e chiusura del file "prenotazioni.txt", modifica del contenuto della lista l attraverso inserimenti, aggiornamento IDCounterPrenotazione, allocazione e deallocazione di memoria temporanea per stringhe di lavoro, in caso di errore (file non trovato, allocazione memoria, inserimento fallito) il programma termina

SPECIFICA ADT: Main

Main è l'insieme delle operazioni per la gestione di un sistema di prenotazioni corsi di una palestra. Il modulo coordina l'interazione utente attraverso menu strutturati e gestisce la persistenza dei dati su file.

Specifica Operatori

Sintattica

- **menuPrenotazione(list, hashtable, listP) → void**
  Gestisce il menu delle operazioni sulle prenotazioni
- **menuCliente(hashtable, listP, list) → void**
  Gestisce il menu delle operazioni sui clienti
- **menuCorso(list, listP) → void**
  Gestisce il menu delle operazioni sui corsi
- **report(list, hashtable, listP) → void**
  Genera un report mensile delle attività

Semantica

- **menuPrenotazione(lCorsi, hClienti, lPrenotati) → void**
  Pre: lCorsi ≠ NULL ∧ hClienti ≠ NULL ∧ lPrenotati ≠ NULL ∧ strutture dati inizializzate
  Post: Operazioni di prenotazione completate secondo selezione utente
  Side effects: Presenta menu con opzioni (prenota, ricerca, elenco, elimina, esci); per prenotazione: valida esistenza cliente e corso, controlla validità abbonamento, verifica data corso futura, genera ID prenotazione, aggiorna contatori partecipanti; per ricerca: permette ricerca per cliente o corso; per eliminazione: valida proprietà prenotazione, decrementa contatori; aggiorna file prenotazioni e corsi dopo modifiche; gestisce allocazione/deallocazione memoria per stringhe input
- **menuCliente(h, lPrenotati, lCorsi) → void**
  Pre: h ≠ NULL ∧ lPrenotati ≠ NULL ∧ lCorsi ≠ NULL ∧ strutture dati inizializzate
  Post: Operazioni sui clienti completate secondo selezione utente
  Side effects: Presenta menu con opzioni (crea, rinnova, ricerca, elenco, elimina, esci); per creazione: acquisisce dati cliente, genera ID univoco, crea abbonamento con data e durata specificate; per rinnovo: localizza cliente, estende durata abbonamento; per ricerca: permette ricerca per ID, nome, cognome, durata; per eliminazione: rimuove cliente e cancella tutte prenotazioni associate; aggiorna file clienti dopo ogni modifica; gestisce allocazione memoria per stringhe input; controlla validità date e durate

- **menuCorso(lCorsi, lPrenotati) → void**
  Pre: lCorsi ≠ NULL ∧ lPrenotati ≠ NULL ∧ strutture dati inizializzate
  Post: Operazioni sui corsi completate secondo selezione utente
  Side effects: Presenta menu con opzioni (aggiungi, ricerca, elenco, elimina, esci); per aggiunta: acquisisce nome, data, orario corso, valida data futura, genera ID univoco; per ricerca: permette ricerca per ID, nome, data, orario; per eliminazione: rimuove corso e cancella prenotazioni associate; aggiorna file corsi dopo modifiche; gestisce allocazione memoria per stringhe; valida formato data e orario
- **report(lCorsi, hClienti, lPrenotati) → void**
  Pre: lCorsi ≠ NULL ∧ hClienti ≠ NULL ∧ lPrenotati ≠ NULL ∧ strutture dati inizializzate
  Post: Report mensile generato in file "report_<nome_mese>.txt"
  Side effects: Determina mese corrente dal sistema; crea nome file report con formato "report_<mese>.txt"; apre file in scrittura, se apertura fallisce stampa "Errore nell'apertura del file" e termina; filtra prenotazioni e corsi del mese corrente; calcola classifica corsi con più partecipanti (lezioni in evidenza); scrive intestazione report con nome mese; scrive numero totale prenotazioni del mese; scrive lista corsi del mese ordinata per partecipanti; chiude file; alloca memoria per nome file, se allocazione fallisce stampa "Errore di allocazione" e termina; libera memoria allocata