Sistema di Pianificazione e Monitoraggio delle Attività di Studio

PROGETTO PROGRAMMAZIONE E STRUTTURE DATI

Di Sanza Salvatore mat. 0512106066

Il progetto prevede la creazione di un sistema che permetta di pianificare e monitorare le attività di studio di uno studente.

Sommario

File eseguibili

->”./a.out” file eseguibile del programma,

->”./test.out” file eseguibile casi di test.

LINK GITHUB: https://github.com/SDisanza/MonitoraggioStudio

Progettazione

L’obiettivo del progetto è quello di realizzare un programma che permetta di pianificare e monitorare lo studio di uno studente.

Lo studente, da qui in poi “utente”, una volta avviato il programma, programmerà le sue attività studio. Una volta compilati i campi, in automatico, partirà il monitoraggio dell’attività.

Ogni volta che l’utente completerà una sessione di studio, questa non verrà più visualizzata ma la troverà nel report settimanale. Il report settimanale riporterà tutte le attività inserite dall’utente che siano completate o no in base alla priorità.

Scelta ADT e Strutture

- Struct Studio:

Andiamo a definire Studio come una struttura dati che raggruppa in un'unica entità tutti i campi necessari alla descrizione di uno studio.

Questo approccio consente di centralizzare l’accesso e la modifica delle informazioni, migliorando la leggibilità e la manutenibilità del codice, oltre a semplificare la gestione dei moduli che gestiscono questi dati.

```c
typedef struct studio
{
    char nome[100];
    char corso[100];
    int priorita; //2 = alta, 1 = media, 0 = bassa
    int completata; // 1 = completata, 0 = in corso, -1 = non iniziata
    int durata; // in minuti
    int dataScadenza; // formato AAAAMMGG (es. 20231005 per il 5 ottobre 2023)
} *Studio;
```

- ADT List studio:

Ogni qualvolta l'utente crea una nuova "attività", questa viene salvata nella testa della lista. Questo permetterà di semplificare la gestione del codice e renderà più difficile la creazione di bug ed errori di logica. Inoltre, viene semplificata la creazione del report.

```c
struct list
{
    int size;
    struct node *head;
};
```

```c
struct node
{
    Item item;
    struct node *next;
};
```

Interazioni Componenti e Utilizzo del Programma

Una volta creato un nuovo Studio, inserendo il nome dello studio, il nome del corso dello studio, la priorità che di default è impostata su -1 e può avere 3 valori (1 per "completata", 0 per "in corso", -1 per "non iniziata", la durata in minuti e la data di scadenza che deve essere ad una data successiva alla data corrente.
Questo, poi, viene inserito nella lista in modalità LIFO.

Dal menù principale del programma abbiamo 4 possibili scelte:

```
Benvenuto nel monitoraggio dello studio!
Questo programma ti permette di tenere traccia dei tuoi studi e delle tue scadenze.
Puoi inserire nuovi studi, visualizzare quelli esistenti e generare report.
Per creare un attivita' premi 1.
Per visualizzare il report, premi 2.
Per aggiornare il progresso di uno studio, premi 3.
Per uscire, digita 0.
Inserisci la tua scelta:
```

- Creare un attività: permette di creare una nuovo studio e inserirla nella lista;
- Visualizzare il report: stampa la lista e controlla quali attività sono scadute o no;
- Aggiornare il progresso di uno studio: permette di modificare il campo "completata" in Studio che di default è -1 "non iniziata" in 1 "completata" e 0 "in corso".
  Dopo aver modificato tale dato viene rigenerato il report
- Uscire: libera la memoria ed esce dal programma

Specifica Sintattica

ADT Studio

- Tipo di riferimento: Studio
- Tipi usati: char[], int

newStudio(char, char, int, int, int) → Studio

inputStudio() → Studio

outputStudio(Studio) → void

deleteStudio(Studio) → void

aggiornaCompletamento(Studio, int, const char) →int

controlloStudio(Studio, int) → int

Stuct List

- Tipo di riferimento: List
- Tipi usati: int, struct node

newList() → List

isEmpty(List) → int

addHead(List, Item) → void

sizeList(List) → int

printList(List) → void

freeList(List) → void

getListItem(List, int) → Item

Struct node

- Tipo di riferimento: node
- Tipi usati: Item, struct node

Specifica Semantica

- File: utils.h, utils.c
    - Funzione: dataOggi
        - Descrizione: Calcola la data odierna.
        - Dati d'ingresso: Nessuno.
        - Dati d'uscita: Un intero che rappresenta la data corrente nel formato AAAAMMGG (es. 20250613 per il 13 Giugno 2025).
        - Precondizioni: Nessuna.
        - Postcondizioni: Il valore restituito è un intero che rappresenta una data valida.
    - Funzione: dataValida
        - Descrizione: Verifica se un dato intero rappresenta una data valida nel formato AAAAMMGG. Considera la validità del mese (1-12), del giorno (in base al mese) e degli anni bisestili per febbraio.
        - Dati d'ingresso:
            - int data: L'intero che rappresenta la data da validare.
        - Dati d'uscita:
            - 1: Se la data è valida.
            - 0: Se la data non è valida.
        - Precondizioni: Nessuna.
        - Postcondizioni: Nessuna modifica dello stato del sistema; restituisce un flag booleano.
- File: item.h, item.c
    - Tipo di Dato: Item
        - Descrizione: Un puntatore generico (void*) utilizzato per memorizzare dati di tipo arbitrario. Nel contesto di questo progetto, Item è concretamente utilizzato per puntare a stringhe (char*), specificamente i nomi degli studi o dei corsi.
    - Funzione: inputItem
        - Descrizione: Legge una stringa (massimo NCHAR-1 caratteri) dallo standard input (stdin) e la alloca dinamicamente.
        - Dati d'ingresso: Nessuno.
        - Dati d'uscita: Un Item (puntatore a char*) che contiene la stringa letta. Termina il programma (exit(EXIT_FAILURE)) in caso di errore di allocazione memoria.
        - Precondizioni: Nessuna.

        - Postcondizioni: La memoria per la stringa input è allocata dinamicamente. Il chiamante è responsabile della deallocazione con freeItem.
    - Funzione: outputItem
        - Descrizione: Stampa il contenuto dell' Item (interpretato come stringa char*) sullo standard output (stdout). Non aggiunge un newline.
        - Dati d'ingresso:
            - Item item: L'elemento da stampare. Si assume che punti a una stringa valida e terminata con \0.
        - Dati d'uscita: Nessuno.
        - Precondizioni: item deve puntare a una stringa valida e terminata con \0.
        - Postcondizioni: La stringa è stampata su stdout.
    - Funzione: cmpItem
        - Descrizione: Confronta due Item (interpretati come stringhe char*) lessicograficamente utilizzando strcmp.
        - Dati d'ingresso:
            - Item itemA: Il primo elemento da confrontare.
            - Item itemB: Il secondo elemento da confrontare.
        - Dati d'uscita:
            - Un valore minore di 0 se itemA è lessicograficamente "minore" di itemB.
            - 0: Se itemA è lessicograficamente "uguale" a itemB.
            - Un valore maggiore di 0 se itemA è lessicograficamente "maggiore" di itemB.
        - Precondizioni: itemA e itemB devono puntare a stringhe valide e terminare con \0.
        - Postcondizioni: Nessuna modifica dello stato.
    - Funzione: freeItem
        - Descrizione: Libera la memoria allocata per un Item utilizzando free().
        - Dati d'ingresso:
            - Item item: L'elemento la cui memoria deve essere liberata.
        - Dati d'uscita: Nessuno.
        - Precondizioni: item deve essere un puntatore valido a memoria allocata con malloc (o simile), oppure NULL.
        - Postcondizioni: La memoria puntata da item è deallocata.

- File: list.h, list.c
    - Tipo di Dato: List
        - Descrizione: Un puntatore alla struttura struct list che rappresenta l'intera lista concatenata. struct list contiene un contatore di dimensione (size) e un puntatore al primo nodo (head).
    - Funzione: newList
        - Descrizione: Crea una nuova lista vuota, allocando la memoria per la sua struttura di controllo (struct list). Inizializza size a 0 e head a NULL.
        - Dati d'ingresso: Nessuno.
        - Dati d'uscita: Un puntatore List alla lista appena creata.
        - Precondizioni: Nessuna.
        - Postcondizioni: Viene restituita una lista inizializzata.
    - Funzione: isEmpty
        - Descrizione: Controlla se la lista è vuota basandosi sul valore del suo campo size.
        - Dati d'ingresso:
            - List list: La lista da controllare.
        - Dati d'uscita:
            - 1: Se la lista è vuota (list->size == 0).
            - 0: Se la lista non è vuota.
        - Precondizioni: list non deve essere NULL.
        - Postcondizioni: Nessuna modifica dello stato della lista.
    - Funzione: addHead
        - Descrizione: Aggiunge un nuovo elemento (Item) all'inizio della lista. Un nuovo nodo viene allocato, il suo item è impostato, il suo next punta alla vecchia testa e il campo head della lista viene aggiornato al nuovo nodo. La dimensione della lista viene incrementata.
        - Dati d'ingresso:
            - List list: La lista a cui aggiungere l'elemento.
            - Item item: L'elemento da aggiungere.
        - Dati d'uscita: Nessuno.
        - Precondizioni: list non deve essere NULL.
        - Postcondizioni: Un nuovo nodo contenente item è aggiunto all'inizio della lista. La dimensione della lista (list->size) è incrementata di 1.
    - Funzione: sizeList

        - Descrizione: Restituisce il numero di elementi attualmente presenti nella lista.
        - Dati d’ingresso:
            - List list: La lista di cui ottenere la dimensione.
        - Dati d’uscita: Un intero che rappresenta la dimensione della lista (il valore del campo list->size).
        - Precondizioni: list non deve essere NULL.
        - Postcondizioni: Nessuna modifica dello stato della lista.
    - Funzione: printList
        - Descrizione: Itera attraverso tutti gli elementi della lista, dal primo all'ultimo, e li stampa chiamando outputStudio per ciascun Item. Questo implica che gli Item memorizzati in questa lista sono di tipo Studio.
        - Dati d’ingresso:
            - List list: La lista da stampare.
        - Dati d’uscita: Nessuno.
        - Precondizioni: list non deve essere NULL. Ogni Item nella lista deve essere un puntatore valido a uno Studio.
        - Postcondizioni: I dettagli di ogni Studio nella lista sono stampati su stdout.
    - Funzione: freeList
        - Descrizione: Dealloca ricorsivamente tutta la memoria occupata dalla lista, inclusi tutti i nodi e gli Item contenuti in essi. La deallocazione di ogni Item avviene tramite freeItem (che nel contesto del progetto chiama deleteStudio poiché Item è uno Studio).
        - Dati d’ingresso:
            - List list: La lista da deallocare.
        - Dati d’uscita: Nessuno.
        - Precondizioni: list deve essere un puntatore valido a una lista allocata con newList (o NULL).
        - Postcondizioni: Tutta la memoria associata alla lista, ai suoi nodi e agli item è liberata.
    - Funzione: getListItem
        - Descrizione: Restituisce l'elemento (Item) presente nella lista alla posizione specificata (indice base 0).
        - Dati d’ingresso:
            - List list: La lista da cui recuperare l'elemento.
            - int pos: La posizione dell'elemento desiderato (0 per la testa).
        - Dati d’uscita:

        - L' Item alla posizione pos.
        - NULL se la posizione pos è fuori dai limiti validi della lista (minore di 0 o maggiore/uguale a sizeList). Stampa "Indice fuori dai limiti" su stderr in caso di errore.
    - Precondizioni: list non deve essere NULL.
    - Postcondizioni: Nessuna modifica dello stato della lista.

- File: studio.h, studio.c
    - Tipo di Dato: Studio
        - Descrizione: Un puntatore alla struttura struct studio che rappresenta un singolo studio/attività, contenente nome, corso, priorità, stato di completamento, durata e data di scadenza.
    - Funzione: newStudio
        - Descrizione: Alloca dinamicamente memoria per una nuova istanza di Studio e la inizializza copiando le stringhe nome e corso, e assegnando i valori per priorita, durata e dataScadenza. Il campo completata è impostato inizialmente a -1 (non iniziata).
        - Dati d'ingresso:
            - char *nome: Nome dello studio.
            - char *corso: Corso relativo allo studio.
            - int priorita: Livello di priorità (0, 1 o 2).
            - int durata: Durata stimata dello studio in minuti.
            - int dataScadenza: Data di scadenza nel formato AAAAMMGG.
        - Dati d'uscita: Un puntatore Studio alla nuova istanza creata. NULL in caso di errore di allocazione.
        - Precondizioni: I Dati d'ingresso stringa (nome, corso) devono essere validi e terminati con \0.
        - Postcondizioni: Un nuovo Studio è allocato e inizializzato.
    - Funzione: inputStudio
        - Descrizione: Guida l'utente attraverso l'input interattivo dei dati per un nuovo studio. Richiede e valida il nome, il corso, la priorità (0-2), la durata (>0) e la data di scadenza (valida e successiva alla data odierna). Utilizza inputItem e dataValida.
        - Dati d'ingresso: Nessuno.
        - Dati d'uscita: Un puntatore Studio all'istanza creata con i dati input dall'utente.
        - Precondizioni: Nessuna, gestisce la validazione dell'input dell'utente.
        - Postcondizioni: Un nuovo Studio è allocato e inizializzato con i dati validati forniti dall'utente.

- Funzione: outputStudio
  - Descrizione: Stampa i dettagli completi di un determinato Studio (nome, corso, priorità, stato di completamento, durata e data di scadenza) sullo standard output in un formato leggibile.
  - Dati d'ingresso:
    - Studio studio: Lo studio da stampare.
  - Dati d'uscita: Nessuno.
  - Precondizioni: studio deve essere un puntatore valido a un Studio.
  - Postcondizioni: I dettagli dello studio sono stampati su stdout.
- Funzione: deleteStudio
  - Descrizione: Libera tutta la memoria allocata per la struttura Studio.
  - Dati d'ingresso:
    - Studio studio: Lo studio da deallocare.
  - Dati d'uscita: Nessuno.
  - Precondizioni: studio deve essere un puntatore valido a memoria allocata per uno Studio.
  - Postcondizioni: La memoria associata a studio è liberata.
- Funzione: aggiornaCompletamento
  - Descrizione: Aggiorna lo stato di completamento (completata) di uno specifico Studio. Verifica la validità del nuovoStato (-1, 0, 1), la validità del puntatore studio e che il nomeStudio fornito corrisponda (insensibile al caso) al nome dello studio. Stampa messaggi di errore o successo.
  - Dati d'ingresso:
    - Studio studio: Il puntatore allo studio da aggiornare.
    - int nuovoStato: Il nuovo stato di completamento desiderato.
    - const char *nomeStudio: Il nome dello studio utilizzato per la verifica di corrispondenza.
  - Dati d'uscita:
    - 1: Se lo stato è stato aggiornato con successo e non era già il nuovoStato.
    - 0: Se lo stato dello studio era già uguale al nuovoStato.
    - -1: In caso di errore (stato non valido, studio NULL, o nomeStudio non corrispondente).
  - Precondizioni: studio deve essere un puntatore valido a uno Studio. nomeStudio deve essere una stringa valida e terminata con \0.

        - Postcondizioni: Lo stato di completamento dello studio viene modificato se le condizioni sono soddisfatte. Messaggi informativi o di errore sono stampati su stdout.
    - Funzione: controlloStudio
        - Descrizione: Valuta lo stato di uno studio in relazione alla sua data di scadenza e al suo stato di completamento, stampando un messaggio appropriato su stdout. Gestisce i casi: scaduto e non iniziato, scaduto ma in corso, completato, e ancora in tempo.
        - Dati d'ingresso:
            - Studio studio: Lo studio da controllare.
            - int dataOggi: La data corrente nel formato AAAAMMGG.
        - Dati d'uscita: 0 (il Dati d'uscita è sempre 0 a meno che studio non sia NULL, nel qual caso stampa un errore e restituisce -1. La funzione è orientata alla stampa di output).
        - Precondizioni: studio deve essere un puntatore valido a uno Studio. dataOggi deve essere una data valida.
        - Postcondizioni: Stampa un messaggio su stdout che descrive lo stato attuale dello studio.

- File: report.h, report.c
    - Funzione: generaReport
        - Descrizione: Genera un report riassuntivo degli studi nella lista. Stampa un'intestazione, il numero totale di studi, e poi i dettagli di ogni studio utilizzando printList (che a sua volta chiama outputStudio).
        - Dati d'ingresso:
            - List lista: La lista di studi per cui generare il report.
        - Dati d'uscita: Nessuno.
        - Precondizioni: lista deve essere una List valida (non NULL).
        - Postcondizioni: Il report è stampato su stdout. Se la lista è vuota, stampa "La lista degli studi è vuota."
    - Funzione: aggiornaCompletamentoNome
        - Descrizione: Itera attraverso la List per trovare uno studio il cui nome corrisponde (insensibile al caso, usando strcasecmp) a nomeStudio. Se uno studio corrisponde, tenta di aggiornarne lo stato di completamento chiamando aggiornaCompletamento. Dopo il tentativo di aggiornamento (indipendentemente dall'esito), rigenera e stampa il report e il monitoraggio dei dati chiamando generaReport e monitoraggioData.
        - Dati d'ingresso:
            - List lista: La lista di studi da percorrere.
            - char *nomeStudio: Il nome dello studio da cercare e aggiornare.

        - int nuovoStato: Il nuovo stato di completamento da impostare.
    - Dati d’uscita: Nessuno.
    - Precondizioni: lista deve essere una List valida. nomeStudio deve essere una stringa valida e terminata con \0.
    - Postcondizioni: Lo stato di uno studio potrebbe essere aggiornato. Vengono sempre stampati i report e i messaggi di monitoraggio.
  - Funzione: monitoraggioData
    - Descrizione: Esegue un monitoraggio degli studi nella List. Per ogni studio, chiama controlloStudio per valutare e stampare il suo stato rispetto alla data odierna.
    - Dati d’ingresso:
      - List lista: La lista di studi da monitorare.
    - Dati d’uscita: Nessuno.
    - Precondizioni: lista deve essere una List valida (non NULL).
    - Postcondizioni: Vengono stampati messaggi su stdout che indicano lo stato di ciascuno studio (es. scaduto, completato, in tempo), come determinato da controlloStudio. Se la lista è vuota, stampa "La lista degli studi è vuota."
- File: main.c
  - Funzione: main
    - Descrizione: Punto di ingresso principale del programma. Inizializza un sistema di monitoraggio degli studi basato su una lista concatenata. Presenta un menu interattivo all'utente per gestire le attività: aggiungere nuovi studi, visualizzare report completi, aggiornare il progresso di uno studio e uscire.
    - Dati d’ingresso: Nessuno.
    - Dati d’uscita: 0 (EXIT_SUCCESS) in caso di terminazione normale.
    - Precondizioni: Nessuna.
    - Postcondizioni: Il programma esegue un ciclo interattivo fino a quando l'utente non sceglie di uscire. Tutta la memoria allocata per la lista e i suoi studi viene liberata all'uscita.
    - Flusso Operativo Dettagliato:
      1. Inizializza una List vuota.
      2. Stampa messaggi di benvenuto e istruzioni iniziali.
      3. Entra in un ciclo do-while per il menu principale:
         - Presenta opzioni: 1 (Creare attività), 2 (Visualizzare report), 3 (Aggiornare progresso), 0 (Uscire).
         - Legge la scelta dell'utente.

- case 1 (Creare attività):
    - Richiede input interattivo per un nuovo studio tramite inputStudio().
    - Aggiunge il nuovo Studio alla lista usando addHead().
    - Stampa un messaggio di conferma e il contenuto della lista aggiornata con printList().
- case 2 (Visualizzare report):
    - Controlla se la lista è vuota usando isEmpty().
    - Se non vuota, chiama generaReport(lista) e monitoraggioData(lista).
    - Stampa "Report generato con successo!".
    - Se vuota, stampa "La lista degli studi è vuota. Non è possibile generare un report."
- case 3 (Aggiornare progresso):
    - Richiede all'utente il nome dello studio da aggiornare (stringa nomestudio).
    - Richiede all'utente il nuovoStato (-1, 0, 1).
    - Chiama aggiornaCompletamentoNome(lista, nomestudio, stato).
- case 0 (Uscire):
    - Stampa "CIAO.".
    - Dealloca tutta la memoria della lista e dei suoi elementi chiamando freeList(lista).
    - Termina il programma.
- default:
    - Stampa "Scelta non valida. Riprova."
    - Pulisce lo schermo (system("clear")).

Casi di Test

Il file main_test.c contiene una suite di test funzionali progettata per validare il comportamento dei moduli che compongono il sistema di "Monitoraggio Studio". L'implementazione dei test si focalizza sulla verifica delle funzionalità delle singole unità e delle loro interazioni.

Date le molteplici funzioni da testare, è stata fatta la scelta progettuale di automatizzare il test.

Un aspetto chiave adottato per i test che generano output testuale è la cattura dell'output standard. Questa tecnica reindirizza l'output a un buffer o file temporaneo, a seconda del caso, permettendo l'analisi automatizzata e il confronto con l'output atteso.

1. test_dataOggi():
    - Obiettivo: Verificare che la funzione dataOggi() restituisca un valore numerico rappresentante la data corrente nel formato AAAAMMGG e che tale valore sia valido semanticamente.
    - Casi Testati: La validità del valore restituito è accertata tramite:
        - Controllo che il valore sia positivo.
        - Verifica che rientri in un intervallo di anni plausibile (es., tra 20000101 e 21000101).
        - Convalida del valore da parte della funzione dataValida(), assicurando la coerenza interna del modulo.
    - Risultato Atteso: La funzione deve produrre un intero che sia una data valida nel formato specificato.
    - Risultato Ottenuto: SI
2. test_dataValida():
    - Obiettivo: Valutare l'accuratezza della funzione dataValida() nella determinazione della validità di una data nel formato AAAAMMGG, includendo la gestione delle regole per gli anni bisestili e i giorni dei mesi.
    - Casi Testati:
        - Validità Standard: Un intero rappresentante una data valida tipica (es., 20231005).
        - Anno Bisestile: Un intero rappresentante il 29 febbraio di un anno bisestile (es., 20240229).
        - Giorno Inesistente: Un intero con un giorno non esistente per il mese specificato (es., 20230230).

        - Mese Inesistente: Un intero con un mese al di fuori dell'intervallo [1, 12] (es., 20231301).
        - Giorno Eccessivo: Un intero con un giorno superiore al massimo consentito per il mese (es., 20230631).
    - Risultato Atteso: La funzione deve ritornare 1 per le date valide e 0 per quelle non valide.
    - Risultato Ottenuto: SI

3. test_inputItem_outputItem_freeItem_cmpItem():
    - Obiettivo: Convalidare le operazioni fondamentali di I/O, confronto e gestione della memoria per gli oggetti di tipo Item.
    - Casi Testati:
        - outputItem(): Viene creato un Item (stringa) e l'output generato da outputItem() viene catturato e confrontato byte per byte con la stringa originale, verificandone l'esatta riproduzione.
        - cmpItem(): Vengono eseguite comparazioni lessicografiche su diverse coppie di Item (stringhe). I valori di ritorno (<0, 0, >0) sono validati rispetto all'ordine alfabetico atteso, in analogia con la funzione strcmp().
        - freeItem(): La funzione freeItem() viene invocata su Item allocati dinamicamente. Il test si concentra sull'assenza di violazioni di accesso alla memoria o crash durante l'esecuzione, sebbene una verifica approfondita dei memory leak richieda strumenti esterni.
        - inputItem(): Dato che questa funzione richiede interazione con stdin, il suo testing automatizzato è complesso. La sua validazione è prevalentemente affidata ai test di integrazione che la invocano (es., nel main), dove il comportamento dell'input interattivo può essere osservato o simulato tramite redirezione di stdin.
    - Risultato Atteso: outputItem deve riprodurre fedelmente la stringa; cmpItem deve eseguire confronti lessicografici corretti; freeItem non deve causare instabilità o errori di runtime.
    - Risultato Ottenuto: SI

4. test_newList_isEmpty():
    - Obiettivo: Verificare la corretta inizializzazione di una nuova lista e l'accuratezza della funzione isEmpty() nel riflettere lo stato iniziale di vuoto.
    - Casi Testati: Creazione di una nuova lista (newList()) seguita dalla verifica che il puntatore restituito non sia NULL e che isEmpty() ritorni 1 (vero).

- Risultato Atteso: newList() deve produrre una lista allocata e inizializzata correttamente, e isEmpty() deve riflettere tale stato.
- Risultato Ottenuto: SI

5. test_addHead_sizeList_getListItem():
    - Obiettivo: Convalidare l'operazione di aggiunta di elementi in testa (addHead), la precisione del conteggio degli elementi (sizeList), e l'affidabilità del recupero degli elementi tramite indice (getListItem).
    - Casi Testati:
        - Aggiunta sequenziale di più oggetti Studio alla lista.
        - Chiamate a sizeList() dopo ogni aggiunta per confermare l'incremento corretto della dimensione.
        - Verifica che isEmpty() ritorni 0 (falso) dopo le aggiunte.
        - Recupero degli elementi tramite getListItem() a posizioni specifiche, per confermare l'ordine di inserimento LIFO.
        - Test di getListItem() con indici fuori limite (negativi o maggiori/uguali alla dimensione della lista), verificando che ritorni NULL e che stampi un messaggio di errore su stderr.
    - Risultato Atteso: Gli elementi devono essere aggiunti correttamente in testa, sizeList() deve riflettere la dimensione esatta, e getListItem() deve fornire accesso corretto o segnalare errori per indici non validi.
    - Risultato Ottenuto: SI
6. test_printList_freeList():
    - Obiettivo: Assicurare che la funzione printList() sia in grado di iterare e stampare correttamente tutti gli elementi della lista, e che freeList() deallochi in modo sicuro tutta la memoria associata alla lista.
    - Casi Testati:
        - Popolamento della lista con due istanze di Studio.
        - Cattura dell'output di printList() e confronto con i dettagli attesi di entrambi gli studi, verificando la loro presenza e il formato.
        - Invocazione di freeList() per deallocare la lista e i suoi Item. Il test verifica l'assenza di crash o violazioni di memoria.
    - Risultato Atteso: printList() deve generare un output completo e formattato degli studi e freeList() deve completare la deallocazione senza errori di runtime.
    - Risultato Ottenuto: SI

7. test_newStudio_deleteStudio():
    - Obiettivo: Convalidare la corretta allocazione, inizializzazione dei campi, e deallocazione di un'istanza di Studio.
    - Casi Testati: Creazione di un Studio con parametri predefiniti tramite newStudio(). Verifica dell'assegnazione corretta di nome, corso, priorità, durata, dataScadenza, e che completata sia inizializzato a -1. Successiva invocazione di deleteStudio() per garantire una deallocazione pulita.
    - Risultato Atteso: newStudio() deve produrre un'istanza Studio con campi correttamente inizializzati, e deleteStudio() deve liberare le risorse senza errori.
    - Risultato Ottenuto: SI
8. test_inputStudio():
    - Obiettivo: Verificare la capacità della funzione di acquisire interattivamente i dati di uno studio dall'utente, applicando le regole di validazione per ciascun campo.
    - Metodologia: Il test, sebbene non completamente automatizzato qui, implicherebbe la simulazione dell'input utente per esplorare percorsi di input validi e non validi e osservarne il comportamento.
    - Risultato Atteso: La funzione dovrebbe guidare l'utente attraverso l'input, accettando dati validi e rifiutando quelli non conformi con messaggi appropriati.
    - Risultato Ottenuto: SI
9. test_aggiornaCompletamento():
    - Obiettivo: Verificare la logica di aggiornamento dello stato di completamento di un Studio, inclusa la gestione delle precondizioni e delle condizioni di errore.
    - Casi Testati:
        - Aggiornamento Valido: Modifica dello stato di uno studio a un valore valido (es., da -1 a 0).
        - Stato Invariato: Tentativo di aggiornare uno studio a uno stato già esistente. Si attende un valore di ritorno che indichi nessun cambiamento.
        - Stato Non Valido: Tentativo di aggiornare con un nuovoStato al di fuori dell'intervallo [-1, 1].
        - Nome Non Corrispondente: Tentativo di aggiornare uno studio fornendo un nomeStudio che non corrisponde al nome dello studio passato alla funzione.

- Risultato Atteso: La funzione deve aggiornare lo stato solo in presenza di input validi e corrispondenza del nome, ritornando codici di successo/errore appropriati e stampando messaggi diagnostici.
- Risultato Ottenuto: SI

10. test_controlloStudio():
    - Obiettivo: Verificare che controlloStudio() valuti accuratamente lo stato di un Studio in relazione alla sua data di scadenza e al suo stato di completamento rispetto a una data odierna fissa, producendo il messaggio diagnostico appropriato.
    - Casi Testati: Vengono instanziati quattro Studio, ciascuno configurato per rappresentare uno specifico scenario:
        - Uno studio scaduto e non iniziato.
        - Uno studio scaduto ma in corso.
        - Uno studio completato.
        - Uno studio non completato e con scadenza futura.
    - L'output di controlloStudio per ciascuno scenario viene catturato e confrontato con i messaggi predefiniti attesi.
    - Risultato Atteso: La funzione deve stampare un messaggio diagnostico che rifletta correttamente lo stato temporale e di completamento dello studio.
    - Risultato Ottenuto: SI
11. test_generaReport():
    - Obiettivo: Accertare che generaReport() produca un report riassuntivo completo e correttamente formattato della lista di studi, gestendo il caso di una lista vuota.
    - Casi Testati:
        - Invoca generaReport() su una List vuota. L'output catturato deve indicare l'assenza di studi.
        - Popola la List con due Studio e invoca nuovamente generaReport(). L'output viene catturato e verificato per la presenza del conteggio totale degli studi e dei dettagli specifici di ogni studio.
    - Risultato Atteso: La funzione deve generare un report chiaro e completo degli studi presenti, o un messaggio che indichi l'assenza di dati se la lista è vuota.
    - Risultato Ottenuto: SI

12. test_aggiornaCompletamentoNome():

- Obiettivo: Verificare la capacità della funzione di localizzare e aggiornare uno studio per nome, e di innescare la rigenerazione dei report dopo l'aggiornamento.
- Casi Testati:
  - Aggiornamento Riuscito: Si simula l'aggiornamento dello stato di uno studio esistente tramite il suo nome. L'output catturato viene analizzato per verificare che contenga il messaggio di successo dell'aggiornamento dello stato, seguito dagli output generati da generaReport e monitoraggioData, a conferma della cascata di chiamate.
  - Nome Non Trovato: Si tenta di aggiornare uno studio specificando un nome non esistente nella lista. L'output deve contenere il messaggio "Nessuno studio trovato con il nome specificato.".
- Risultato Atteso: La funzione deve aggiornare con successo lo studio specificato e provvedere alla rigenerazione dei report, o comunicare chiaramente l'assenza dello studio.
- **Risultato Ottenuto:** SI

13. test_monitoraggioData():
    - Obiettivo: Convalidare che monitoraggioData() esegua una scansione completa di tutti gli studi nella lista, invocando controlloStudio per ciascuno al fine di determinare e stampare il suo stato.
    - Casi Testati:
      - Invoca monitoraggioData() su una List vuota. L'output catturato deve indicare che la lista è vuota.
      - Popola la List con un set diversificato di Studio (scaduti, in corso, completati, in tempo). L'output di monitoraggioData() viene catturato e verificato per la presenza di tutti i messaggi diagnostici previsti da controlloStudio per ogni studio.
    - Risultato Atteso: La funzione deve produrre un riepilogo dello stato di monitoraggio per ogni studio nella lista, o un messaggio che indichi l'assenza di studi da monitorare. Si sottolinea che per test completamente deterministici, l'input della data odierna a controlloStudio dovrebbe essere controllato.
    - Risultato Ottenuto: SI

Implementazioni Future

Il team di sviluppo è a piena conoscenza che ci sono delle funzioni che non vengono utilizzate. Ma sono state create e testate per una futura implementazione delle funzionalità del programma

- Modifica dello studio: Permettere all'utente di modificare i dati di uno studio inserito
- Ordinamento delle attività: Al posto di visualizzare le attività cosi come inserite dall'utente, queste vengono ordinate secondo la priorità e la scadenza assegnata.

La collaborazione è benvenuta: qualsiasi suggerimento per ottimizzare funzionalità o processi sarà preso in considerazione.