UNIVERSITA’ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA TRIENNALE IN
INFORMATICA

GESTIONE STUDIO

Sistema di pianificazione e monitoraggio delle attività di studio

Partecipanti:
Riccio Vincenzo
Matricola:
0512122158

Anno accademico:
2024/2025

Indice

1. Introduzione

Il progetto **gestione_studio** ha lo scopo di realizzare un programma in C che aiuti uno studente a gestire e monitorare la sua attività di studio. Il sistema permette di inserire attività di studio, assegnare priorità, visualizzare il progresso e generare report settimanali.

Gestione Studio – Sistema di pianificazione e monitoraggio delle attività di studio

2. Motivazione della scelta dell'ADT – Lista concatenata semplice

Nel progetto *Gestione Studio* è stato necessario individuare una struttura dati in grado di rappresentare un insieme dinamico di attività, ciascuna con attributi multipli (descrizione, corso, data di scadenza, tempo stimato, stato di avanzamento e priorità).

Dopo aver valutato diverse opzioni la scelta è ricaduta sulla **lista concatenata semplice** per la gestione interna delle attività.

Le motivazione principali di questa scelta sono le seguenti:

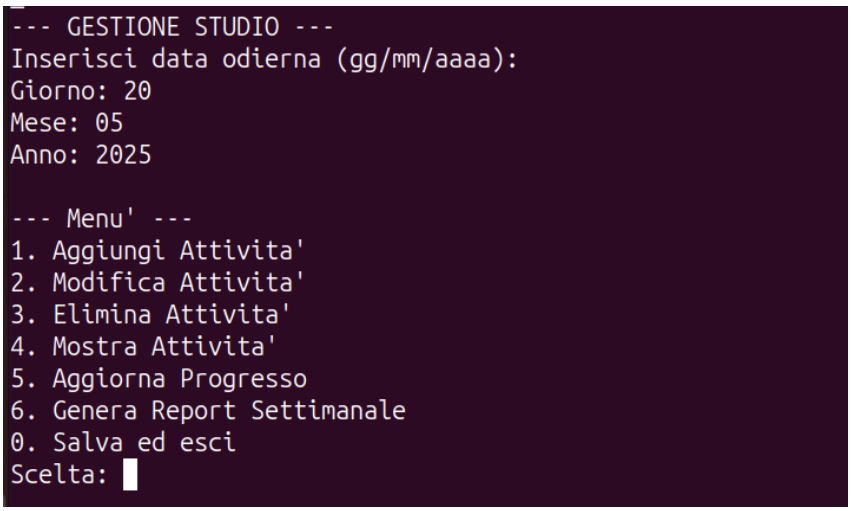
1.1 Flessibilità e gestione dinamica della memoria. Una lista concatenata permette di gestire una collezione di elementi di dimensione variabile senza dover stabilire un numero massimo di attività a priori. Questo è ideale per un'applicazione in cui l'utente può aggiungere o rimuovere diversi tipi di attività in qualunque momento. In particolare è stata scelta la modalità di inserimento in coda, cioè ogni nuova attività viene aggiunta alla fine della lista. Questa scelta è stata fatta poiché rispecchia il comportamento naturale di un utente, cioè registrare le attività in ordine cronologico. Questa scelta garantisce che le attività vengano visualizzate nell'ordine in cui sono state aggiunte, semplificando la consultazione e l'interazione. Favorisce la leggibilità e la prevedibilità durante la visualizzazione e nel report settimanale che può elencare le attività dalla più vecchia alla più recente.

1.2 Costo accettabile per l'inserimento. L'inserimento in coda, sebbene abbia una complessità lineare O(n), resta comunque efficiente e adatto alla frequenza d'uso prevista nel sistema, poiché non richiede inserimenti massivi o simultanei, per cui una scansione fino all'ultimo nodo è accettabile.

1.3 Semplicità di implementazione e manutenzione. La lista concatenata semplice con inserimento in coda è stata scelta anche per la sua facilità di implementazione:

- Solo un campo *prossimoNodo* per ogni attività, senza doppia referenza come nelle liste doppiamente concatenate.
- Le operazioni di aggiunta, rimozione e scorrimento sono facili da scrivere, testare e mantenere, minimizzando la possibilità di errori.
- Le funzioni di gestione della lista sono modulari e possono essere facilmente estese (ad esempio, per ordinamento o filtraggio).

2. Modalità di utilizzo del sistema



Il sistema è pensato per essere utilizzato tramite un'interfaccia testuale interattiva. Una volta avviato il programma, l'utente si trova davanti a un menù principale, da cui può selezionare diverse opzioni, tra cui:

1. **Aggiungere una nuova attività di studio**, fornendo descrizione, corso, data di scadenza (*gg/mm/aaaa*), tempo stimato (*in ore*), livello di priorità (*0=bassa, 1=media, 2=alta*), stato di avanzamento (*0=in corso, 1=completata, 2=in ritardo*).

2. **Modificare un'attività esistente** (utile per correggere eventuali errori).

```
--- Menu' ---
1. Aggiungi Attivita'
2. Modifica Attivita'
3. Elimina Attivita'
4. Mostra Attivita'
5. Aggiorna Progresso
6. Genera Report Settimanale
0. Salva ed esci
Scelta: 2
Posizione dell'attivita' da modificare: 1
Descrizione: Progetto gestione studio
Corso: PSD
Inserisci la data di scadenza (gg/mm/aaaa):
Giorno scadenza: 30
Mese scadenza: 05
Anno scadenza: 2025
Tempo stimato (ore): 120
Priorita' (0=bassa,1=media,2=alta): 2
Stato (0=in corso,1=completata,2=in ritardo): 0
Attivita' modificata con successo.
```

3. **Eliminare un'attività**, selezionandola tramite indice.

```
--- Menu' ---
1. Aggiungi Attivita'
2. Modifica Attivita'
3. Elimina Attivita'
4. Mostra Attivita'
5. Aggiorna Progresso
6. Genera Report Settimanale
0. Salva ed esci
Scelta: 3
Posizione dell'attivita' da eliminare: 1
Attivita' rimossa con successo.
```

4. **Visualizzare tutte le attività registrate**, in base all'ordine in cui sono state inserite (dalla meno recente alla più recente).

```
------------------
Attivita' 4:

Descrizione: Progetto
Corso: PSD
Scadenza: 30/05/2025
Tempo stimato: 120 ore
Priorita': Alta
Stato: In corso
------------------
Attivita' 5:

Descrizione: Progetto
Corso: Analisi 1
Scadenza: 20/05/2025
Tempo stimato: 80 ore
Priorita': Alta
Stato: In corso
```

5. **Aggiornare lo stato di avanzamento di un'attività** es. segnarla come completata o aggiornarne il progresso. (se la data attuale è successiva a quella di scadenza viene automaticamente inserito lo stato "in ritardo".

```
--- Menu' ---
1. Aggiungi Attivita'
2. Modifica Attivita'
3. Elimina Attivita'
4. Mostra Attivita'
5. Aggiorna Progresso
6. Genera Report Settimanale
0. Salva ed esci
Scelta: 5
Posizione dell'attivita' da aggiornare: 2
Nuovo stato (0=in corso,1=completata,2=in ritardo): 1
Stato aggiornato con successo.
```

6. **Generare un report settimanale**, che mostra le attività previste in scadenza in un determinato periodo e le statistiche generali.

```
--- Menu' ---
1. Aggiungi Attivita'
2. Modifica Attivita'
3. Elimina Attivita'
4. Mostra Attivita'
5. Aggiorna Progresso
6. Genera Report Settimanale
0. Salva ed esci
Scelta: 6
Data inizio (gg/mm/aaaa):
Giorno: 13
Mese: 05
Anno: 2025
Data fine (gg/mm/aaaa):
Giorno: 20
Mese: 05
Anno: 2025
Report settimanale generato con successo: report.txt
```

All’avvio viene caricata la lista delle attività da un file (attivita.txt). L’inserimento di una nuova attività o eventuali modifiche di attività esistenti vengono salvate su file all’uscita dal programma, per garantire la persistenza.

3. Progettazione del sistema e interazione tra i componenti

Il sistema è suddiviso in moduli, ciascuno con responsabilità specifiche, per migliorare la manutenibilità e la chiarezza del progetto:

- attività.c/h
  Contiene la definizione della struttura Attività, le funzioni di creazione, modifica, stampa e confronto tra attività.

- lista.c/h
  Gestisce la lista concatenata di attività, con operazioni di inserimento e stampa.

- gestione.c/h
  Implementa le funzionalità del sistema: aggiunta e modifica di nuove attività, aggiornamento del progresso, caricamento e salvataggio su file.

- menu.c/h
  Gestisce il menu testuale e le scelte dell’utente, richiamando le funzioni appropriate in base alla scelta effettuata.

- main.c
  è il punto di ingresso del programma. Inizializza la struttura dati, mostra il menù e gestisce il ciclo principale dell’applicazione.

Tutti i moduli sono dipendenti tra loro, la comunicazione avviene tramite puntatori ad Attività e a Nodo, rendendo facile l’eventuale modifica.

4. Specifiche sintattiche e semantiche

SPECIFICA SINTATTICA E SEMANTICA - Attivita.h/.c

creaAttivita

Sintattica:
Attivita* creaAttivita(*const char* descrizione, const char* corso, Data scadenza, int tempo_stimato, int priorita, int stato*);

Semantica:
Precondizioni:
- descrizione e corso sono puntatori validi a stringhe terminate da '\0'.
- scadenza rappresenta una data valida.
- tempo_stimato è un intero >= 0 (tempo stimato in ore).
- priorita è uno dei valori definiti: BASSA (0), MEDIA (1), ALTA (2).
- stato è uno dei valori definiti: IN_CORSO (0), COMPLETATA (1), IN_RITARDO (2).

Postcondizioni:

- Alloca dinamicamente un nuovo Attivita e ne inizializza i campi con i valori forniti (stringhe copiate fino a lunghezza massima).
- Restituisce un puntatore a questa nuova Attivita.

Side effects:
- Se l'allocazione fallisce, stampa un messaggio di errore e termina il programma con exit(1).

==========

stampaAttivita

Sintattica:
void stampaAttivita(Attivita* att, Data oggi);

Semantica:
Precondizioni:
- att è un puntatore valido a una struttura Attivita oppure NULL.
- oggi è una data valida (ad esempio passata come data corrente).

Postcondizioni:
- Se att == NULL, stampa "Attivita' non valida.\n" e termina.

Side effects:
- Se l'attività non è completata (stato != COMPLETATA), aggiorna lo stato a IN_RITARDO se la scadenza è antecedente a oggi, altrimenti a IN_CORSO.
- Stampa i dettagli dell'attività in formato leggibile, includendo: descrizione, corso, scadenza, tempo stimato, priorità (stringa), stato (stringa).

==========

dataValida

Sintattica:
int dataValida(Data d);

Semantica:
Precondizioni:
- d è una struttura Data con campi interi giorno, mese, anno.

Postcondizioni:
- Restituisce 1 (vero) se d rappresenta una data valida considerando:
    - Giorno compreso tra 1 e massimo giorni del mese (con gestione anni bisestili).
    - Mese compreso tra 1 e 12.
    - Anno >= 1900 (limite arbitrario).
- Restituisce 0 (falso) in caso contrario.

Side effects: Nessuno

==========

liberaAttivita

Sintattica:
void liberaAttivita(Attivita* att);

Semantica:
Precondizioni:
- att è un puntatore a un'area di memoria precedentemente allocata dinamicamente tramite creaAttivita, oppure NULL.

Postcondizioni:
- Se att != NULL, libera la memoria allocata per l'attività.
- Se att == NULL, non fa nulla.

Side-Effects: Nessuno

==========

SPECIFICA SINTATTICA E SEMANTICA - ADT Lista.h/.c

creaLista

Sintattica:
Lista creaLista();

Semantica:
Descrizione: Crea e restituisce una lista vuota (rappresentata da NULL).

Precondizioni:
- Nessuna.

Postcondizioni:
- Viene restituito un puntatore NULL che rappresenta una lista vuota.

Side effects: Nessuno

==========

inserisciInCoda

Sintattica:
Lista inserisciInCoda(Lista l, Attivita* att);

Semantica:
Descrizione: Inserisce un'attività in coda alla lista l.

Precondizioni:
- att è un puntatore valido a una struttura Attività.
- l può essere NULL (lista vuota) o non vuota.

Postcondizioni:
- Viene restituita la lista aggiornata con il nuovo nodo in coda contenente att.

- Se l'allocazione di memoria fallisce, la lista originale l viene restituita invariata.
- Se la lista è vuota, il nuovo nodo diventa la testa della lista.

Side effects:

- Alloca nuova memoria dinamica.

==========

stampaLista

Sintattica:

void stampaLista(const Lista l, Data oggi);

Semantica:

Descrizione: Stampa tutte le attività contenute nella lista l, indicando il loro ordine e passando la data corrente oggi per la formattazione.

Precondizioni:

- l può essere NULL o non vuota.
- oggi è una data valida (struttura Data).

Postcondizioni: Nessuna

Side effects:

- Vengono stampate a video tutte le attività presenti nella lista, con le relative informazioni.

==========

lunghezzaLista

Sintattica:

int lunghezzaLista(Nodo* testa);

Specifica semantica

Descrizione: Restituisce il numero di nodi presenti a partire da testa.

Precondizioni:

- testa può essere NULL (lista vuota) o puntare al primo nodo di una lista.

Postcondizioni:

- Viene restituito un intero ≥ 0 pari al numero di nodi nella lista.

Side effects: Nessuno

==========

liberaLista

Sintattica:

void liberaLista(Lista l);

Semantica:
Descrizione: Libera tutta la memoria occupata dalla lista l e dalle attività contenute in essa.

Precondizioni:
- l può essere NULL (lista vuota) o non vuota.

Postcondizioni:
- Dopo l'esecuzione, l non deve essere più usata senza essere reinizializzata.

Side effects:
- Tutta la memoria allocata per i nodi della lista e per le attività è liberata correttamente.

==========

SPECIFICA SINTATTICA E SEMANTICA - gestione.h / gestione.c

aggiungiAttivita

Sintattica:
Lista aggiungiAttivita(Lista l, Attivita* att, int inTesta);

Semantica:
Descrizione: Inserisce un'attività nella lista l. Se inTesta == 1, l'attività viene inserita in testa; altrimenti in coda.

Precondizioni:
- Il puntatore att deve essere diverso da NULL.
- La lista l può essere vuota (NULL) o non vuota.
- Il parametro inTesta deve essere 0 o 1.

Postcondizioni:
- La nuova attività viene aggiunta in testa o in coda alla lista.
- La lista restituita contiene tutti gli elementi precedenti più il nuovo.

Side effects: Alloca memoria dinamica per il nuovo nodo.

==========

modificaAttivita

Sintattica:
Lista modificaAttivita(Lista l, int posizione, Attivita* nuovaAtt);

Semantica:
Descrizione: Sostituisce l'attività alla posizione posizione con nuovaAtt.

Precondizioni:
- La lista l deve contenere almeno posizione + 1 elementi.
- nuovaAtt è un puntatore valido ad una struttura Attivita.

Postcondizioni:

- L'attività alla posizione specificata viene sostituita con nuovaAtt.

Side effects:
- La vecchia attività viene liberata.

==========

rimuoviAttivita

Sintattica:
Lista rimuoviAttivita(Lista l, int posizione);

Semantica:
Descrizione: Rimuove l'attività alla posizione posizione e libera la memoria.

Precondizioni:
- La lista l deve contenere almeno posizione + 1 elementi.

Postcondizioni:
- L'attività alla posizione specificata viene rimossa.
- La lista risultante ha un elemento in meno.

Side effects:
- La memoria relativa all'attività viene liberata

==========

aggiornaProgresso

Sintattica:
Lista aggiornaProgresso(Lista l, int posizione, int nuovoStato);

Semantica:
Descrizione: Modifica il campo stato dell'attività alla posizione indicata con il valore nuovoStato.

Precondizioni:
- La lista l deve contenere almeno posizione + 1 elementi.

Postcondizioni: Nessuna

Side effects:
- Lo stato dell'attività alla posizione specificata viene aggiornato con il nuovo valore.

==========

caricaAttivitaDaFile

Sintattica:
Lista caricaAttivitaDaFile(const char* nomeFile);

Semantica:
Descrizione: Carica le attività da un file di testo e costruisce la lista.

Precondizioni:
- Il file specificato da nomeFile deve esistere e avere un formato valido.

Postcondizioni:
- La lista restituita contiene tutte le attività lette correttamente dal file.

Side effects:
- Legge da file.
- Alloca memoria dinamica per ogni attività e nodo.

==========

salvaAttivitaSuFile

Sintattica:
void salvaAttivitaSuFile(Lista l, const char* nomeFile);

Semantica:
Descrizione: Salva la lista su un file di testo formattato.

Precondizioni:
- Il nome del file deve essere un puntatore valido.
- La lista può essere vuota (in tal caso si salva un file vuoto).

Postcondizioni:
- Il file di testo specificato viene sovrascritto con i dati delle attività presenti nella lista.

Side effects:
- Scrive su file

==========

liberaListaAttivita

Sintattica:
void liberaListaAttivita(Lista l);

Semantica:
Descrizione: Libera tutta la memoria allocata dalla lista delle attività.

Precondizioni:
- La lista può essere NULL o contenere uno o più nodi.

Postcondizioni: Nessuna

Side effects:
- Tutta la memoria dinamica della lista e delle attività viene correttamente liberata.

==========

SPECIFICA SINTATTICA E SEMANTICA - Report.h/.c

dataInIntervallo

Sintattica:
int dataInIntervallo(Data data, Data inizio, Data fine);

Semantica:
Descrizione: Verifica se la data data è compresa (inclusi gli estremi) tra le date inizio e fine.

Precondizioni:
- Le date data, inizio e fine devono essere valide (cioè rispettare il formato del calendario gregoriano).
- La data inizio deve essere minore o uguale a fine.

Postcondizioni:
- Restituisce 1 se data ∈ [inizio, fine]; 0 altrimenti.

Side effects: Nessuno

==========

statisticheStatoAttivita

Sintattica:
void statisticheStatoAttivita(Lista lista, FILE* fp);

Semantica:
Descrizione: Scrive nel file fp un riepilogo statistico delle attività presenti nella lista, suddivise per stato (IN_CORSO, COMPLETATA, IN_RITARDO).

Precondizioni:
- lista è una lista correttamente inizializzata (può essere anche vuota).
- fp è un puntatore a file aperto in scrittura.

Postcondizioni:
- Vengono scritte nel file le statistiche relative al numero di attività per ciascun stato.

Side effects: Nessuno

==========

generaReportSettimanale

Sintattica:
void generaReportSettimanale(Lista lista, Data inizio, Data fine, const char* nome_file);

Semantica:
Descrizione: Genera un file di report contenente le attività la cui scadenza è compresa tra le date inizio e fine. Il report include anche statistiche sullo stato delle attività.

Precondizioni:

- lista è una lista correttamente inizializzata (può essere anche vuota).
- inizio e fine sono date valide, e inizio ≤ fine.
- nome_file è una stringa valida che rappresenta il nome del file di output.

Postcondizioni:

- Viene creato (o sovrascritto) un file con nome nome_file, contenente:
    - Le attività nel periodo specificato, con i relativi dettagli (descrizione, corso, scadenza, tempo stimato, priorità, stato).
    - Le statistiche di tutte le attività presenti nella lista (lista), indipendentemente dal periodo.

Side effects: Nessuno

==========

SPECIFICA SINTATTICA E SEMANTICA - Menu.h/.c

mostraMenu

Sintattica:

void mostraMenu(Lista *lista, Data oggi);

Semantica:

Descrizione: La funzione mostraMenu presenta un menu testuale all'utente e consente l'interazione con il sistema di gestione delle attività di studio. L'utente può inserire, modificare, eliminare e visualizzare attività, aggiornare il loro stato, generare un report settimanale su file, e infine salvare l'intera lista su file prima dell'uscita.

Precondizioni:

- lista deve essere un puntatore valido a una lista eventualmente già allocata (anche vuota).
- oggi deve rappresentare una data valida.
- Devono essere disponibili in memoria tutte le funzioni invocate (creaAttivita, aggiungiAttivita, modificaAttivita, rimuoviAttivita, stampaLista, aggiornaProgresso, generaReportSettimanale, salvaAttivitaSuFile, dataValida, lunghezzaLista).
- Devono essere definite le costanti DESCRIZIONE_LEN e CORSO_LEN.
- Il programma deve avere accesso in scrittura al file "report.txt" e "attivita.txt".

Postcondizioni:

- In base alla scelta dell'utente:
    - Può essere **aggiunta** una nuova attività in coda alla lista.
    - Può essere **modificata** un'attività esistente in una posizione specifica.
    - Può essere **rimossa** un'attività in una posizione specifica.
    - Può essere **visualizzata** la lista delle attività.
    - Può essere **aggiornato** lo stato di una singola attività.
    - Può essere **generato** un report settimanale su file.
    - Può essere **salvata** la lista su file "attivita.txt" prima dell'uscita.
- L'interazione termina solo quando l'utente seleziona l'opzione di uscita (0).
- Se la lista è modificata, *lista conterrà il nuovo puntatore alla lista aggiornato.

- In caso di inserimenti/modifiche/eliminazioni, il contenuto della lista è coerente con i dati forniti dall'utente.
- Il file attivita.txt conterrà una rappresentazione persistente delle attività al momento dell'uscita.

Side effects:

- Stampa su schermo (output testuale del menu e messaggi).
- Scrittura su file (report.txt, attivita.txt).
- Possibile allocazione dinamica di memoria per nuove attività.
- Possibile deallocazione di memoria per attività rimosse.
- Modifica indiretta di *lista.

==========

5. Razionale dei casi di test

I casi di test sono stati progettati per coprire tutte le funzionalità fondamentali del sistema, con particolare attenzione a:

- **Inserimento corretto e errato** di attività, testando validazione della data e dei campi.
- **Aggiornamento dello stato** con valori validi e non validi.
- **Cancellazione** di attività in testa, in mezzo o in coda alla lista.
- **Generazione del report settimanale** in presenza e assenza di attività rilevanti.
- **Persistenza**: caricamento e salvataggio su file, anche con file parzialmente corrotti.

Ogni test è stato commentato nel file test_suite.c, con input attesi caricati da file.txt presenti nella cartella file_di_test/ e risultati verificabili manualmente o tramite stampa su console.

```
TC1 - Inserimento valido: PASS
TC2 - Inserimento con data errata: PASS
TC3 - Inserimento con priorità errata: PASS
TC4 - Aggiornamento stato corretto: PASS
TC5 - Aggiornamento stato non valido: PASS
TC6 - Cancellazione in testa: PASS
TC7 - Cancellazione in mezzo: PASS
TC8 - Cancellazione in coda: PASS
Report settimanale generato con successo: TC9_output.txt
TC9 - Report con attività: PASS
Report settimanale generato con successo: TC10_output.txt
TC10 - Report vuoto: PASS
TC11 - Salva e carica: PASS
TC12 - File corrotto: PASS
```

6. Conclusione

Il progetto **gestione studio** ha permesso di creare un programma per gestire attività di studio usando una lista collegata in C. rispetta tutte le indicazioni presenti nella traccia,

permette di inserire, modificare e cancellare elementi dalla lista, e mostrare un report delle attività.

La scelta di usare una lista collegata ha reso possibile gestire un numero variabile di attività in modo flessibile. Inoltre, scrivere la test suite ha aiutato a verificare che tutte le funzioni funzionino correttamente.

Questo lavoro è stato utile per capire meglio come progettare un programma modulare e come organizzare il codice in più file. In futuro, il progetto potrà migliorare ulteriormente aggiungendo nuove funzionalità o ottimizzando il codice.