Autrice: Ruocco Daniela Lucia, matricola 0512122089

Progetto Gestione Attività Di Studio

Il progetto *Gestione Attività di Studio* nasce con l'obiettivo di fornire uno strumento semplice ed efficace per aiutare uno studente a pianificare, monitorare e organizzare le proprie attività accademiche.

L'applicativo è stato sviluppato interamente in linguaggio **C**, utilizzando tecniche di programmazione *modulare e strutturata*. Esso consente di:

- Registrare attività di studio con informazioni dettagliate come **descrizione**, **corso di appartenenza**, **data di scadenza**, **tempo stimato** e **priorità**.
- Visualizzare l'elenco completo delle attività e monitorarne il **progresso** attraverso lo stato di completamento e le ore svolte.
- Generare un **report settimanale** che raggruppa le attività in base alla settimana di scadenza, distinguendole tra **completate**, **in corso** e **in ritardo**.
- Interagire tramite un'interfaccia testuale chiara e guidata, pensata per essere intuitiva anche per utenti con poca esperienza informatica.

Il progetto pone particolare enfasi sulla **gestione dinamica dei dati**, attraverso l'uso di strutture dati appropriate (liste concatenate) e sull'**organizzazione modulare del codice**, al fine di garantire chiarezza, estendibilità e facilità di manutenzione.

Questo documento descrive nel dettaglio le scelte progettuali, l'architettura del software, la motivazione nella selezione degli ADT utilizzati, la specifica sintattica e semantica delle funzioni principali, nonché il razionale dei casi di test implementati.

Motivazione della Scelta dell'ADT

La scelta dell'ADT si è rivolta verso l'utilizzo della **lista concatenata**, implementata all'interno del progetto come segue:

```c
/*
-------------------------------------------------------
|  Struttura dati 'Nodo'
|  Descrizione: Nodo della lista concatenata, contiene:
|    - Un'attività di tipo 'Attivita'
|    - Un puntatore al prossimo nodo della lista
-------------------------------------------------------
*/
typedef struct Nodo {
    Attivita attivita;          // Informazioni sull'attività
    struct Nodo *next;          // Puntatore al prossimo nodo
} Nodo;

// Definizione di un puntatore alla lista di attività
typedef Nodo* ListaAttivita;
```

Definizione dell'ADT tratto dal file "attivita.h"

Questa scelta è stata guidata da differenti *motivazioni*, quali:

1. Inserimenti dinamici frequenti

- Le attività vengono inserite durante l'esecuzione.
- La lista concatenata permette di aggiungere elementi in testa con ***complessità O(1)***, senza riallocazioni come accadrebbe con un array.

2. Dimensione variabile

- Il numero di attività non è noto a priori.
- La lista permette una ***crescita dinamica*** senza limiti fissi di memoria.

3. Operazioni sequenziali naturali

- Visualizzazione, ricerca, aggiornamento e rimozione vengono fatte in ***modo sequenziale.***
- L'accesso lineare è perfettamente adatto a queste operazioni.

4. Memoria gestita dinamicamente

- ***Nessuno spreco di memoria*** per celle vuote.
- Ogni nodo viene ***allocato solo quando necessario***.

Inoltre, essa risulta essere *più efficiente* per questo progetto rispetto ad altre possibili strutture dati dalla cui applicazione si evincono differenti problematiche. Ad esempio:

- Gli **Array dinamici** non risultano essere efficienti poiché richiedono *riallocazione e gestione manuale* della capacità
- Gli **Alberi binari** risultano essere *eccessivi* per una gestione non ordinata e di piccole dimensioni
- Le **Tabelle Hash** risultano essere necessarie, in quanto le *attività non richiedono accesso diretto per chiavi univoche*

La **lista concatenata** rappresenta la *soluzione più flessibile*, semplice e adatta a questo tipo di progetto: supporta inserimenti rapidi, gestione dinamica della memoria, ed è facilmente estendibile per funzionalità future come il salvataggio su file, ordinamenti o filtri.

Progettazione

Il progetto Gestione Attività di Studio è stato realizzato con **un'architettura modulare**, suddivisa in **componenti indipendenti**, ciascuno con responsabilità specifiche. Questo approccio migliora la *leggibilità del codice*, la *manutenibilità* e la *scalabilità futura del sistema*.

Nella tabella seguente vengono elencati i file componenti il progetto stesso e la loro struttura.

File	Responsabilità principale
main.c	Interfaccia utente e gestione del flusso di controllo
attivita.h	Definizione delle strutture dati (Attivita, ListaAttivita) e prototipi delle funzioni su di essa definite
attivita.c	Implementazione delle funzioni che operano sulla lista delle attività
utile.h	Prototipi delle funzioni ausiliarie di gestione data/calcolo
utile.c	Funzioni per conversione delle date, calcolo settimana e intervalli
test.c	Casi di test per verificare il corretto funzionamento delle funzioni implementate sulla struttura
Makefile	Automazione della compilazione, esecuzione e testing

I moduli *interagiscono* tra di loro secondo quanto segue:

- ***main.c*** utilizza le funzioni di **attivita.c** per aggiungere, modificare o visualizzare le attività.
- ***attivita.c*** usa le funzioni di **utile.c** per elaborare le date (es. calcolo settimana, parsing).
- Le **strutture dati** sono definite in ***attivita.h*** e condivise via #include.

I **vantaggi** della produzione modulare risultano essere molteplici, tra cui:

- *Separazione delle responsabilità* → Ogni file ha una funzione specifica
- *Facilità di test* → Moduli come utils.c possono essere testati isolatamente

Specifica sintattica e semantica

Nel *progetto Gestione Attività di Studio*, le **operazioni** sono distribuite in tre moduli principali: ***attivita.c*** (gestione della lista e delle attività), ***utile.c*** (gestione delle date), e ***main.c*** (interfaccia utente). Di seguito si fornisce una descrizione dettagliata, con input, output, precondizioni, postcondizioni ed effetti collaterali.

1. Funzioni di "attivita.c"

- `creaLista()`
  - Crea una lista vuota.
  - **Input:** nessuno
  - **Output:** puntatore a lista (NULL)
  - **Precondizioni:** nessuna
  - **Postcondizioni:** viene creata una lista vuota
  - **Effetti collaterali:** nessuno

- **`aggiungiAttivita(ListaAttivita *lista, Attivita a)`**
  - Aggiunge una nuova attività all'inizio della lista.
  - **Input:** puntatore alla lista e struttura `Attivita`
  - **Output:** nessuno
  - **Precondizioni:** l'attività deve essere valida
  - **Postcondizioni:** la nuova attività è aggiunta in testa alla lista
  - **Effetti collaterali:** viene allocata memoria dinamica per il nuovo nodo
- **`visualizzaAttivita(ListaAttivita lista)`**
  - Visualizza tutte le attività memorizzate nella lista.
  - **Input:** puntatore alla lista
  - **Output:** stampa a terminale
  - **Precondizioni:** lista inizializzata (può essere vuota)
  - **Postcondizioni:** nessuna modifica alla lista
  - **Effetti collaterali:** stampa a schermo
- **`ricercaAttivita(ListaAttivita lista, const char *descrizione)`**
  - Cerca un'attività nella lista tramite la descrizione.
  - **Input:** lista e stringa descrizione
  - **Output:** puntatore al nodo trovato o NULL
  - **Precondizioni:** lista inizializzata
  - **Postcondizioni:** nessuna modifica
  - **Effetti collaterali:** nessuno
- **`aggiornaAttivita(ListaAttivita lista, const char *descrizione, int oreAggiunte)`**
  - Aggiorna le ore svolte per una specifica attività.
  - **Input:** lista, descrizione e ore da sommare
  - **Output:** nessuno
  - **Precondizioni:** l'attività esiste
  - **Postcondizioni:** l'attributo `oreSvolte` viene incrementato; `completato` può diventare 1
  - **Effetti collaterali:** modifica diretta dei dati nel nodo
- **`monitoraggioProgresso(ListaAttivita lista)`**
  - Mostra per ogni attività se è completata, in corso o in ritardo.
  - **Input:** lista
  - **Output:** stampa
  - **Precondizioni:** lista inizializzata
  - **Postcondizioni:** nessuna modifica
  - **Effetti collaterali:** stampa a terminale
- **`generaReportSettimanale(ListaAttivita lista)`**
  - Raggruppa e stampa le attività per settimana, mostrando il loro stato.
  - **Input:** lista
  - **Output:** stampa a schermo
  - **Precondizioni:** lista valida
  - **Postcondizioni:** nessuna modifica
  - **Effetti collaterali:** output su stdout
- **`rimuoviAttivita(ListaAttivita *lista, const char *descrizione)`**
  - Rimuove un'attività dalla lista.
  - **Input:** puntatore alla lista e stringa descrizione
  - **Output:** nessuno
  - **Precondizioni:** l'attività deve esistere (o essere ignorata se non trovata)
  - **Postcondizioni:** il nodo corrispondente viene eliminato

- **Effetti collaterali:** deallocazione della memoria

- **`liberaMemoria(ListaAttivita *lista)`**
  - Libera tutta la memoria occupata dalla lista.
  - **Input:** puntatore alla lista
  - **Output:** nessuno
  - **Precondizioni:** lista valida
  - **Postcondizioni:** tutte le celle sono liberate e `*lista` diventa `NULL`
  - **Effetti collaterali:** memoria liberata
- **`creaAttivita()`**
  - Interagisce con l'utente per acquisire i dati di una nuova attività.
  - **Input:** inserimento utente da tastiera
  - **Output:** struttura `Attivita` compilata
  - **Precondizioni:** input ben formati
  - **Postcondizioni:** l'attività è pronta per essere inserita
  - **Effetti collaterali:** input/output con l'utente

2. Funzioni di "utile.c"

- **`convertiData(const char *data)`**
  - Converte una stringa `gg/mm/aaaa` in una `struct tm`.
  - **Input:** stringa di data
  - **Output:** struttura `tm` valida
  - **Precondizioni:** formato corretto della stringa
  - **Postcondizioni:** `tm` inizializzato correttamente
  - **Effetti collaterali:** nessuno

- **`calcolaSettimana(struct tm data)`**
  - Calcola il numero della settimana dell'anno per una data.
  - **Input:** `struct tm`
  - **Output:** numero di settimana (1-52)
  - **Precondizioni:** `tm` valido
  - **Postcondizioni:** restituisce il numero corretto
  - **Effetti collaterali:** nessuno

- **`calcolaIntervalloSettimana(struct tm base, struct tm *inizio, struct tm *fine)`**
  - Calcola il lunedì e la domenica della settimana corrispondente a una data.
  - **Input:** `tm` base e puntatori a `inizio` e `fine`
  - **Output:** `inizio` e `fine` aggiornati
  - **Precondizioni:** `base` valido
  - **Postcondizioni:** intervallo della settimana calcolato
  - **Effetti collaterali:** modifica delle strutture `tm` passate

3. Funzioni di “main.c”

- `menu()`
  - Mostra all’utente le opzioni disponibili.
  - **Input:** nessuno
  - **Output:** testo stampato a terminale
  - **Precondizioni:** nessuna
  - **Postcondizioni:** nessuna
  - **Effetti collaterali:** stampa a video

Razionale dei Casi di Test

Per garantire la correttezza e l’affidabilità del sistema sviluppato, sono stati implementati tre casi di test automatici, eseguiti tramite file di input/output e confrontati con oracoli predefiniti.

Obiettivo dei Test

Verificare la correttezza delle funzionalità principali del sistema di gestione delle attività di studio, assicurando che:

- Le attività siano correttamente inserite.
- Il progresso delle attività venga aggiornato in modo accurato.
- Il report settimanale venga generato con dati corretti.

📋 Dettagli dei Casi di Test

Test 1: Inserimento e Visualizzazione delle Attività

- **ID Test**: TST-001
- **Descrizione**: Verifica che le attività di studio siano correttamente inserite e visualizzate.
- **Input**: File `test1.in` contenente due attività.
- **Output Atteso**: File `test1.out` con la corretta visualizzazione delle attività.
- **Output Effettivo**: File `test1.actual`.
- **Risultato**: Pass

Test 2: Aggiornamento del Progresso delle Attività

- **ID Test**: TST-002
- **Descrizione**: Verifica il corretto aggiornamento del progresso delle attività.
- **Input**: File `test2.in` con attività e progresso aggiornato.
- **Output Atteso**: File `test2.out` con stato aggiornato.
- **Output Effettivo**: File `test2.actual`.
- **Risultato**: Pass

Test 3: Generazione del Report Settimanale

- **ID Test**: TST-003
- **Descrizione**: Verifica la generazione del report settimanale con i dati corretti.

- **Input**: File `test3.in` con attività distribuite su più settimane.
- **Output Atteso**: File `test3.out` con report settimanale corretto.
- **Output Effettivo**: File `test3.actual`.
- **Risultato**: Pass

Sommario dei Risultati

ID Test	Descrizione	Risultato
TST-001	Inserimento e Visualizzazione delle Attività	Pass
TST-002	Aggiornamento del Progresso delle Attività	Pass
TST-003	Generazione del Report Settimanale	Pass

Automazione

I test sono stati eseguiti tramite un Makefile personalizzato, con comandi del tipo: `make test`

Inoltre, ogni test (`test1.c`, `test2.c`, `test3.c`) è progettato per leggere input, scrivere l'output in un file `.actual` e confrontarlo automaticamente con un file `.out` che contiene l'output previsto.

🔍 Conclusioni

Tutti i test sono stati eseguiti con successo, confermando che le funzionalità principali del sistema operano come previsto. I risultati attesi corrispondono agli output effettivi, indicando un comportamento corretto del sistema nelle condizioni testate.

Conclusione

Il progetto **Gestione Attività di Studio** rappresenta non solo un'applicazione concreta delle strutture dati dinamiche in C, ma anche un esempio di come l'informatica possa rispondere a un'esigenza reale: organizzare il tempo, pianificare gli impegni, dare struttura allo studio.

Attraverso un'interfaccia semplice e intuitiva, e una progettazione modulare ed estensibile, il sistema consente allo studente di mantenere sotto controllo le proprie attività, di monitorare i progressi in modo oggettivo e di affrontare le scadenze con consapevolezza. La scelta dell'ADT lista concatenata, unita a funzioni di supporto ben incapsulate, ha garantito flessibilità, efficienza e facilità di manutenzione.

Ogni riga di codice scritta, ogni test effettuato, ogni output stampato sono stati un passo verso un risultato tangibile: **trasformare un'idea in uno strumento concreto**, utile e migliorabile. Questo progetto è una base solida su cui costruire ulteriori funzionalità – come il salvataggio su file, la visualizzazione grafica o l'integrazione con un calendario – e rappresenta un traguardo, ma anche un punto di partenza.

“Il miglior modo per prevedere il futuro è inventarlo.” – Alan Kay