REPORT PROGETTO (P.S.D) A.A 2024/2025 CLASSE: PEU-Z

Traccia 3: Sistema di pianificazione e monitoraggio delle attività di studio

STRUTTURA E REALIZZAZIONE DEI TIPI DI DATI ASTRATTI (ADT)

Durante lo sviluppo del progetto sono stati adottati e implementati due tipi principali di ADT:

- Coda a priorità
- Attività

ADT: CODA A PRIORITÀ

Per l'ADT Coda a Priorità è stata implementata una struttura, la quale contiene i seguenti elementi:

- **vet:** Un array contenente gli oggetti di tipo 'Attività'
- **num_elementi:** Un intero che tiene traccia del numero corrente degli elementi presenti nella coda

MOTIVI E SCELTA DELL'IMPLEMENTAZIONE: CODA A PRIORITÀ

L'ADT Coda a Priorità è stato gestito tramite max-heap, poiché offre numerosi vantaggi:

- **Prestazioni migliorate:**
  - L'elemento con massima priorità è sempre collocato in cima all'heap consentendo accesso immediato in tempo costante.
  - Le operazioni di inserimento ed eliminazione garantiscono efficienza anche quando il numero di attività aumenta

- **Semplicità di implementazione:**

- La scelta di un array statico è giustificata dalla natura del sistema, il quale, per scelta, gestisce un numero limitato di attività (MAX_HEAP 50). Tutto ciò evita anche complicazioni dovute ad eventuali riallocazioni dinamiche.

ADT ATTIVITÀ

Per l'ADT Attività è stata implementata una *struct,* la quale raccoglie tutte le proprietà rilevanti per ogni singola attività. Difatti i campi risultano i seguenti:

- **Descrizione:** Una stringa (max 100 caratteri) che definisce il contenuto dell'attività

- **Nome del corso:** Una stringa (max 50 caratteri) che definisce il nome del corso oppure della materia correlata

- **data_di_scadenza:** Una stringa con formato GG/MM/AAAA che specifica la data entro cui l'attività deve essere completata

- **Tempo stimato:** Un valore intero che rappresenta il tempo (in ore) previsto per portare a termine l'attività

- **Importanza:** Un campo di tipo enum (Priorita) che può assumere i valori BASSA, MEDIA, ALTA , determinando il livello di priorità dell'attività

- **Stato:** Un campo enum (Stato) che definisce se l'attività è in RITARDO, IN CORSO, COMPLETATA

- **Progresso:** Un intero che indica la percentuale di completamento da (0 a 100)

- **data_completamento:** Una stringa che memorizza la data di fine attività nel formato GG/MM/AAAA, se l'attività non è ancora conclusa questo campo viene lasciato vuoto o impostato a un valore predefinito come "Non disponibile"

MOTIVI E SCELTA DELL'IMPLEMENTAZIONE: ATTIVITÀ

Per l'ADT Attività è stato scelto di implementare una struct, in modo tale da centralizzare tutte le informazioni necessarie su una singola attività, favorendo vari aspetti come:

- **Raccolta unica dei dati:**

- Essere in possesso di tutte le caratteristiche relative ad un'attività in un unico oggetto, facilita le operazioni di modifica, visualizzazione e registrazione.

- **Chiarezza e facilità di manutenzione:**

- La struttura attività così definita offre un'interfaccia pulita e chiara, semplificando il lavoro del programmatore per eventuali aggiornamenti

PROGETTAZIONE: STRUTTURA DEL SISTEMA E FUNZIONAMENTO

FASE INIZIALE E CARICAMENTO DEGLI INPUT

All'avvio del programma, il sistema prova a recuperare i dati salvati nel file di testo denominato ("attività.txt"). Pertanto, si descrivono dettagliatamente le fasi iniziali:

- **Caricamento:**

- In questa fase viene invocata la funzione *caricamento_da_file,* la quale legge il file "attività.txt" seguendo il formato prestabilito. Successivamente se il file risulta inesistente o presenta errori durante il caricamento, il programma avvisa l'utente tramite un messaggio d'errore e provvede alla creazione di una nuova struttura dati (vuota).

- **Allocazione dei dati:**

- La coda a priorità viene inizializzata dinamicamente, pertanto lo stato iniziale prevede un numero di attività pari a 0. Successivamente gli elementi inseriti saranno gestiti dal max-heap, garantendo un inserimento sempre ordinato, basato sul grado di importanza (Priorità).

INTERFACCIA UTENTE E NAVIGAZIONE NEL SISTEMA

La componente interattiva del sistema viene gestita da un menù principale, il quale offre diverse opzioni per la manipolazione delle attività inserite. Dunque, una volta avuto accesso al programma l'utente potrà scegliere tra le seguenti opzioni:

Scelta 1. (Aggiungi attività):
Il programma richiede all'utente di fornire tutte le informazioni essenziali relative all'attività che si desidera aggiungere, di conseguenza saranno richiesti i seguenti dati:

- **Descrizione (max 100 caratteri)**
- **Nome del corso (max 50 caratteri)**
- **Data di scadenza: (Inserita rigorosamente nel formato GG/MM/AAAA)**
- **Tempo stimato: (in ore da 1 a 10.000)**
- **Priorità: (da 1 a 3, che si riferisce rispettivamente a BASSA, MEDIA, ALTA)**

A questo punto la funzione di inserimento *aggiungi_attivita*, raccoglie e valida i dati forniti, assicurandosi che ogni campo rispetti il formato atteso e che i valori obbligatori siano tutti presenti.

Se i dati risultano validi, viene creato un nuovo oggetto di tipo 'Attivita', inoltre sarà inserito nella coda a priorità (max-heap) tramite la funzione *inserisci.*

Infine, il sistema esegue un salvataggio su file, tramite la funzione *salvataggio_su_file,* in modo tale da garantire la permanenza dei dati.

Scelta 2. (Modifica attività):
Quando l'utente seleziona questa opzione, il programma verificherà l'esistenza di almeno un'attività da modificare. A questo punto l'utente viene invitato a specificare l'indice (compreso tra 1 e il numero totale delle attività presenti) dell'attività che desidera modificare.

Una volta selezionato l'indice, viene recuperato un puntatore alla struttura dell'attività tramite la funzione *ottieni_attivita_puntatore.*

Dopo tale operazione, la funzione *modifica_attivita* permette di aggiornare le informazioni dell'attività scelta, a partire da (Descrizione, Nome del corso, data di scadenza, tempo stimato, priorità). Pertanto, tale funzione gestisce anche il ricalcolo dello stato (come verificare se l'attività è in RITARDO).

Infine, se la modifica viene effettuata con successo, i dati aggiornati vengono aggiornati sul file "attività.txt" sempre tramite la funzione *salvataggio_su_file.*

Scelta 3. (Elimina attività):
Se l'utente sceglie questa opzione, gli viene richiesto di fornire la descrizione dell'attività che desidera eliminare. Di conseguenza, il sistema ricerca nella coda a priorità l'attività corrispondente alla descrizione fornita.

A questo punto, se l'attività viene trovata, verrà stampato su terminale un messaggio di conferma e l'attività verrà eliminata dalla struttura dati.

In particolar modo, la procedura di eliminazione include lo "scambio" dell'elemento da cancellare con l'ultimo elemento dell'array e anche una riorganizzazione per preservare la proprietà del max-heap.

Infine, il cambiamento viene salvato su file sempre tramite la funzione *salvataggio_su_file.*

Scelta 4. (Aggiorna progresso dell'attività):
Se l'utente sceglie questa opzione, il sistema gli chiederà di fornire l'indice dell'attività per la quale desidera aggiornare il progresso. Successivamente, viene richiesta la nuova percentuale di completamento (da 0 a 100).

Di conseguenza, viene invocata la funzione *aggiorna_progresso_attivita,* che controlla la validità del valore inserito (sempre compreso tra 0 e 100).

A questo punto, se il progresso raggiunge il 100% , la funzione imposta direttamente lo stato dell'attività a COMPLETATA, e registra la data di completamento facendo utilizzo della libreria *time.h* in modo tale da ottenere la data corrente.

Infine, il sistema aggiorna il file di salvataggio inserendo la nuova percentuale e la data di completamento, utilizzando sempre la funzione *salvataggio_su_file.*

Scelta 5. (Visualizza progresso):
Se l'utente sceglie questa opzione, il sistema stampa su terminale un riepilogo dettagliato di tutte le attività gestite.

In particolar modo, la funzione *mostra_progresso,* effettua una scansione della coda a priorità, e per ogni singola attività stampa a schermo:

- **La descrizione**
- **Il nome del corso**
- **La data di scadenza (formato GG/MM/AAAA)**
- **Il tempo stimato (In ore)**
- **Il livello di importanza (1 = BASSA, 2 = MEDIA, 3 = ALTA)**
- **La percentuale di progresso**
- **La data di completamento (solo se disponibile)**

Dunque questa funzionalità permette all'utente di avere una panoramica dello stato delle attività inserite.

Scelta 6. (Genera report settimanale):
Se l'utente sceglie questa opzione, visualizzerà su terminale un report settimanale delle attività (IN RITARDO, IN CORSO, COMPLETATE).

Precisamente, la funzione *genera_report_settimanale,* determina l'intervallo temporale della settimana corrente (da Lunedì dalle 00:00:00 a Domenica alle 23:59:59).

Il sistema andrà ad analizzare ogni singola attività, verificando se la data di scadenza (o completamento se presente) cade all'interno della settimana corrente.

ATTENZIONE:
Il counter del report settimanale riguardante le attività (IN RITARDO, IN CORSO, IN RITARDO), viene aggiornato solo se l'attività ricade nella settimana corrente.

Scelta 7. (Ottieni l'attività con massima priorità):
Se l'utente sceglie questa opzione, il sistema farà utilizzo della funzione *ottieni_max,* la quale estrae dall'heap l'elemento posto in cima, il quale corrisponde all'attività con massima priorità.

Successivamente, la funzione *mostra_attivita*, stampa a schermo tutte le informazioni relative a quella attività, consentendo all'utente di identificare facilmente l'attività più critica.

Scelta 8. (Mostra notifiche):
Se l'utente sceglie questa opzione, viene invocata la funzione *mostra_notifiche* , la quale scorre l'intera coda a priorità e filtra solo le attività che, in base alla data di scadenza e allo stato attuale, risultano in ritardo.

Quindi, verrà stampato a schermo un elenco di notifiche che evidenziano le attività che risultano in ritardo.

Invece, se non viene individuata alcuna attività in ritardo, viene stampato a schermo un messaggio informativo, che ne comunica l'assenza.

Scelta 9. (Uscita dal programma):
L'utente, selezionando questa opzione, decide di terminare l'esecuzione del programma.

In particolar modo, il sistema esce dal ciclo interattivo, stampando a schermo un messaggio di conferma. Successivamente procede a liberare le risorse allocate dinamicamente (ovvero la coda a priorità), tramite la funzione *free(),* prima di terminare l'esecuzione dell'applicazione.

SPECIFICA SINTATTICA E SEMANTICA DELLE FUNZIONI UTILIZZATE

FUNZIONI RELATIVE A 'coda_priorita.h'

- *Funzione*: **nuova_PC**

- Crea e inizializza una nuova coda di priorità

- *Specifica sintattica*: PCoda nuova_PC(void) -> PCoda

- *Parametri*: (Nessuno parametro)

- *Specifica semantica*: nuova_PC() -> PCoda

- *Pre-Condizioni*: (Nessuna)

- *Post-Condizioni*: Restituisce un oggetto di tipo PCoda
Altrimenti NULL in caso di fallimento dell'allocazione

- *Ritorna*: Un oggetto di tipo PCoda
Altrimenti NULL

- *Effetti collaterali*: Viene allocata dinamicamente della memoria per la 'Coda a priorità'
Deve essere liberata tramite free()

==========

- *Funzione:* **vuota_PC**

- Verifica se la coda di priorità è vuota oppure non è stata inizializzata

- *Specifica sintattica*: int vuota_PC(PCoda) -> int

- *Parametri:*
c: PCoda

- *Specifica semantica:* vuota_PC(c) -> int

- *Pre-Condizioni:* Il parametro inserito deve essere di tipo PCoda (oppure NULL)

- *Post-Condizioni*: Restituisce 1 se la coda è vuota o non inizializzata
Altrimenti 0

- *Ritorna*: 1 se la coda è vuota o non valida
0 se contiene almeno un elemento

- *Effetti collaterali*: (Nessuno)

==========

- *Funzione*: **inserisci**

- Inserisce un nuovo elemento di tipo 'Attivita' in una coda organizzata con max-heap

- *Specifica sintattica*: int inserisci(PCoda, Attivita) -> int

- *Parametri:*
    c: PCoda
    nuovaAttivita: Attivita

- *Specifica semantica*: inserisci(c, nuovaAttivita) -> int

- *Pre-Condizioni*: La coda deve essere inizializzata e avere spazio sufficiente, ovvero
    (num_elementi < MAX_HEAP)

- *Post-Condizioni*: La coda mantiene sempre la proprietà del max-heap e contiene l'elemento
    inserito

- *Ritorna*: 1 se l'elemento è stato inserito con successo
    0 altrimenti

- *Effetti collaterali*: La struttura di tipo PCoda viene modificata

==========

-*Funzione*: **ottieni_max**

- Restituisce l'elemento con il massimo valore di importanza dalla coda di priorità

- *Specifica sintattica*: Attivita ottieni_max(PCoda) -> Attivita

- *Parametri*:
    c: PCoda

- *Specifica semantica*: ottieni_max(c) -> Attivita

- *Pre-Condizioni*: (Nessuna)

- *Post-Condizioni*: Se la coda è vuota viene restituita una 'Attivita' predefinita
Altrimenti, l'elemento in cima.

- *Ritorna:* l'oggetto di tipo 'Attivita' con il maggior grado di importanza

- *Effetti collaterali*: (Nessuno)

==========

- *Funzione*: **mostra_progresso**

- Permette di visualizzare i dettagli e i progressi degli elementi presenti nella coda

- *Specifica sintattica*: void mostra_progresso(PCoda) -> void

- *Parametri*:
c: PCoda

- *Specifica semantica*: mostra_progresso(c) -> void

- *Pre-Condizioni*: La coda deve essere inizializzata

- *Post-Condizioni*: I dettagli degli elementi vengono stampati a schermo

- *Ritorna:* (Nessun valore)

- *Effetti collaterali*: Stampa l'output a schermo

==========

- *Funzione:* **genera_report_settimanale**

- Genera un report dei dati temporali di tutti gli elementi della coda a priorità
  che rientrano nella settimana corrente

- *Specifica sintattica*: void genera_report_settimanale(PCoda) -> void

- *Parametri:*
  c: PCoda

- *Specifica semantica*: genera_report_settimanale(c) -> void

- *Pre-Condizioni*: La coda deve essere inizializzata

- *Post-Condizioni*: Viene visualizzato un report di riepilogo degli elementi filtrati per intervallo

- *Ritorna*: (Nessun valore)

- *Effetti collaterali*: Stampa l'output a schermo

==========

- *Funzione*: **elimina_attivita**

- Rimuove un elemento di tipo 'Attivita' basandosi sulla descrizione inserita e aggiorna l'ordinamento per mantenere il max-heap

- *Specifica sintattica*: void elimina_attivita(PCoda) -> void

- *Parametri*:

c: PCoda

- *Specifica semantica*: elimina_attivita(c) -> void

- *Pre-Condizioni*: La coda deve essere inizializzata e contenere almeno un elemento

- *Post-Condizioni*: L'elemento corrispondente alla ricerca viene eliminato e la

struttura mantiene il max-heap

- *Ritorna*: (Nessun valore)

- *Effetti collaterali*: Modifica la struttura di tipo CodaPriorita

==========

- *Funzione*: **mostra_notifiche**

- Permette di visualizzare in output le notifiche relative agli elementi in ritardo

- *Specifica sintattica*: void mostra_notifiche(PCoda) -> void

- *Parametri*:

c: PCoda

- *Specifica semantica*: mostra_notifiche(c) -> void

- *Pre-Condizioni*: Il parametro deve essere di tipo PCoda e rappresentare una collezione valida

- *Post-Condizioni*: Le notifiche, se presenti, vengono mostrate su schermo altrimenti viene visualizzato un messaggio indicante l'assenza di elementi in ritardo

- *Ritorna*: (Nessun valore)

- *Effetti collaterali*: Stampa l'output a schermo

==========

- *Funzione*: **ottieni_numero_attivita**

- Restituisce il numero di attività presenti nella coda

- *Specifica sintattica*: int ottieni_numero_attivita(PCoda) -> int

- *Parametri*:

c: PCoda

- *Specifica semantica*: ottieni_numero_attivita(c) -> int

- *Pre-Condizioni*: La coda deve essere inizializzata (anche se vuota)

- *Post-Condizioni*: Viene restituito il numero di attività contenute nella coda
Altrimenti, se la coda è NULL 0

- *Ritorna*: Un intero che rappresenta il numero di elementi nella coda

- *Effetti collaterali*: (Nessuno)

==========

- *Funzione*: **ottieni_attivita**

- Restituisce una copia dell'attività presente nella coda in corrispondenza dell'indice che è stato fornito

- *Specifica sintattica*: Attivita ottieni_attivita(PCoda, int) -> Attivita

- *Parametri*:
c: PCoda
indice: int

- *Specifica semantica*: ottieni_attivita(c, indice) -> Attivita

- *Pre-Condizioni*: La coda deve essere inizializzata e l'indice deve essere compreso tra 1 e il numero di elementi

- *Post-Condizioni*: Viene restituita una copia corretta dell'attività se l'indice è valido, oppure un'attività vuota se l'indice non è valido

- *Ritorna*: Un oggetto di tipo Attivita contenente i dati dell'attività richiesta oppure un oggetto "vuoto"

- *Effetti collaterali*: (Nessuno)

==========

- *Funzione*: **ottieni_attivita_puntatore**

- Restituisce il puntatore all'attività presente nella coda in corrispondenza dell'indice fornito, consentendo di modificarla direttamente

- *Specifica sintattica*: Attivita* ottieni_attivita_puntatore(PCoda c, int indice)

- *Parametri*: c: PCoda
indice: int

- *Specifica semantica*: ottieni_attivita_puntatore(c, indice) -> Attivita*

- *Pre-Condizioni*: La coda deve essere inizializzata e l'indice deve essere compreso tra 1 e il numero totale di elementi

- *Post-Condizioni*: Viene restituito un puntatore valido all'attività se l'indice è corretto
Altrimenti NULL se l'indice non è valido

- *Ritorna*: Un puntatore all'elemento di tipo 'Attivita' richiesto oppure NULL in caso di indice non valido

- *Effetti collaterali*: (Nessuno)

==========

FUNZIONI RELATIVE A ‘attivita.h’

- *Funzione:* **aggiungi_attivita**

- Raccoglie in input i dati necessari per creare un nuovo elemento di tipo 'Attivita', inizializzando il progresso a 0 e determinando lo stato tramite un confronto temporale

- *Specifica sintattica*: int aggiungi_attivita(Attivita*) -> int

- *Parametri*:
  a: Attivita*

- *Specifica semantica*: aggiungi_attivita(a) -> int

- *Pre-Condizioni*: Il puntatore deve riferirsi a una struttura di tipo 'Attivita' valida

- *Post-Condizioni*: L'elemento di tipo 'Attivita' viene popolato e il suo progresso viene inizializzato a 0

- *Ritorna*: 1 se l'inserimento ha avuto successo
  0 altrimenti

- *Effetti collaterali*: Modifica il contenuto dell'oggetto 'Attivita'

==========

- *Funzione*: **modifica_attivita**

- Aggiorna i dati di un elemento di tipo 'Attivita' con nuovi valori forniti in input

- *Specifica sintattica*: void modifica_attivita(Attivita*) -> void

- *Parametri*:
    a: Attivita*

- *Specifica semantica*: modifica_attivita(a) -> void

- *Pre-Condizioni*: L'oggetto di tipo Attivita deve essere inizializzato

- *Post-Condizioni*: I valori dell'elemento vengono aggiornati in base alle definizioni dei tipi

- *Ritorna*: (Nessun valore)

- *Effetti collaterali*: Modifica l'oggetto di tipo 'Attivita'

==========

- *Funzione*: **verifica_ritardo**

- Confronta una data la quale viene espressa come stringa nel formato "GG/MM/AAAA", con la data corrente per determinare se l'elemento è in ritardo

- *Specifica sintattica*: int verifica_ritardo(const char*) -> int

- *Parametri*:
    data_di_scadenza: const char*

-*Specifica semantica*: verifica_ritardo(data_di_scadenza) -> int

- *Pre-Condizioni*: La stringa deve avere il seguente formato "GG/MM/AAAA"

- *Post-Condizioni*: Il confronto viene effettuato utilizzando esclusivamente il formato e i tipi

- *Ritorna*: 1 se la data inserita è antecedente la data corrente
0 altrimenti

- *Effetti collaterali*: (Nessuno)

==========

- *Funzione*: **mostra_attivita**

- Visualizza in modo formattato i dati di un elemento di tipo 'Attivita'
seguendo lo schema definito dai tipi

- *Specifica sintattica*: void mostra_attivita(const Attivita*) -> void

- *Parametri*:
a: const Attivita*

- *Specifica semantica*: mostra_attivita(a) -> void

- *Pre-Condizioni*: Il puntatore deve fare riferimento a un oggetto di tipo 'Attivita'

- *Post-Condizioni*: I dati dell'elemento sono visualizzati secondo lo schema dei tipi

- *Ritorna:* (Nessun valore)

- *Effetti collaterali*: Stampa l'output a schermo

==========

- *Funzione*: **aggiorna_progresso_attivita**

- Aggiorna il valore di avanzamento di un elemento di tipo Attivita, se raggiunge 100 imposta lo stato a COMPLETATA e registra la data corrente

- *Specifica sintattica*: void aggiorna_progresso_attivita(Attivita*) -> void

- *Parametri*:
  a: Attivita*

- *Specifica semantica*: aggiorna_progresso_attivita(a) -> void

- *Pre-Condizioni*: Il puntatore deve essere valido e il valore per il progresso deve essere compreso tra 0 e 100

- *Post-Condizioni*: Il campo del progresso viene aggiornato, se pari a 100 anche lo stato e la data di completamento vengono modificati

- *Ritorna*: (Nessun valore)

- *Effetti collaterali*: Modifica l'oggetto di tipo 'Attivita'

==========

FUNZIONI RELATIVE A ‘funzioni_file.h’

- *Funzione*: **salvataggio_su_file**

- Salva gli elementi di tipo 'Attivita' in un file di testo usando un formato fisso, comprensivo di intestazione e separatori

- *Specifica sintattica*: int salvataggio_su_file(const char*, PCoda) -> int

- *Parametri*:
  nome_file: attività.txt
  c: PCoda

- *Specifica semantica*: salvataggio_su_file(nome_file, c) -> int

- *Pre-Condizioni*: Il file, se è presente, deve corrispondere al formato stabilito

- *Post-Condizioni*: I dati vengono scritti su disco in base al formato stabilito

- *Ritorna*: 1 se il salvataggio è avvenuto con successo
  0 altrimenti

- *Effetti collaterali*: Crea oppure sovrascrive il file sul disco

==========

- *Funzione:* **caricamento_da_file**

- Carica gli elementi di tipo Attivita da un file di testo e li inserisce in una struttura di tipo PCoda, basandosi su un formato stabilito

- *Specifica sintattica*: int caricamento_da_file(const char*, PCoda) -> int

- *Parametri*:

  nome_file: attività.txt

  c: PCoda

- *Specifica semantica*: caricamento_da_file(nome_file, c) -> int

- *Pre-Condizioni:* Il file esiste ed è formattato secondo lo schema previsto

- *Post-Condizioni*: La struttura PCoda viene popolata con gli elementi letti dal file

- *Ritorna*: 1 se il caricamento ha avuto successo

  0 altrimenti

- *Effetti collaterali*: Modifica la struttura di tipo CodaPriorita

==========

FUNZIONI RELATIVE A ‘test_progetto.h’

- *Funzione*: **test_aggiungi_attivita**

- Simula la creazione di un nuovo elemento di tipo Attivita
e verifica che i valori inizializzati corrispondano all'oracolo atteso

- *Specifica sintattica*: void test_aggiungi_attivita(void)

- *Parametri*: (Nessuno)

- *Specifica semantica*: test_aggiungi_attivita() -> void

- *Pre-Condizioni*: I puntatori fp_input, fp_oracle e fp_output devono essere già aperti e validi
La struttura Attivita è considerata allocata e valida

- *Post-Condizioni*: Il test verifica che la struttura Attivita contenga i parametri previsti
L'input, l'oracolo atteso e l'esito del test vengono scritti nei file

- *Ritorna*: (Nessun valore)

- *Effetti collaterali*: Scrive su file (fp_input, fp_oracle, fp_output) e su terminale
Crea una variabile locale 'Attivita' e ne modifica i campi

==========

- *Funzione*: **test_aggiorna_progresso**

- Simula l'aggiornamento del progresso di un'attività esistente, passando da un valore iniziale pari a 0 a un aggiornamento al 100%
  Imposta il relativo stato a COMPLETATA e registra la data di completamento

- *Specifica sintattica*: void test_aggiorna_progresso(void)

- *Parametri*: (Nessuno)

- *Specifica semantica*: test_aggiorna_progresso() -> void

- *Pre-Condizioni*: I puntatori fp_input, fp_oracle e fp_output devono essere già aperti e validi
  La struttura Attivita è considerata allocata e valida

- *Post-Condizioni*: La struttura 'Attivita' viene aggiornata con i parametri previsti
  L'input, l'oracolo atteso e l'esito del test vengono scritti nei file

- *Ritorna*: (Nessun valore)

- *Effetti collaterali*: Modifica la variabile 'Attivita';
  Scrive su file (fp_input, fp_oracle, fp_output)

==========

- *Funzione*: **test_report_settimanale**

- Simula la generazione di un report settimanale, che raggruppa le attività per stato in base alla data corrente
  Verifica che il report legga correttamente i conteggi per le attività completate, in corso, o in Ritardo

- *Specifica sintattica*: void test_report_settimanale(void)

- *Parametri*: (Nessuno)

- *Specifica semantica*: test_report_settimanale(void) -> void

- *Pre-Condizioni*: I puntatori per i file devono essere aperti
La funzione nuova_PC() deve restituire un puntatore valido a una coda
Le funzioni inserisci(), ottieni_numero_attivita() e ottieni_attivita() devono essere definite correttamente

- *Post-Condizioni*: La coda contenente le attività viene analizzata
Il report viene confrontato con i valori attesi
L'input, l'oracolo atteso e l'esito del test vengono scritti nei file

- *Ritorna*: (Nessun valore)

- *Effetti collaterali*: La funzione va a modificare i contenuti della coda creata
Esegue scritture sui file (fp_input, fp_oracle, fp_output)

==========

RAZIONALE CASI DI TEST

TEST CASE 1 – AGGIUNGI ATTIVITÀ

Il primo test effettuato riguarda la verifica della corretta creazione di una nuova attività e la corretta inizializzazione dei seguenti campi:

- **Descrizione**
- **Nome del Corso**
- **Data di scadenza (Formato GG/MM/AAAA)**
- **Tempo stimato (In ore)**
- **Priorità**

Inoltre il test riguarda anche lo stato dell'attività, il quale deve essere determinato in base alla data di scadenza.

Input

La funzione *test_aggiungi_attivita* simula l'inserimento dei seguenti dati predefiniti :

Descrizione: "Progetto di Analisi"
Nome del corso: "Matematica"
Data di scadenza: "15/06/2025"
Tempo stimato: 20 ore
Priorità: ALTA

Oracolo

I campi dell'attività aggiunta si aspettano correttamente inizializzati, identici all'input, con l'aggiunta del parametro progresso, il quale dovrebbe essere uguale a 0%. Dunque l'attività risulta con 'Stato' = IN CORSO

Output

I campi dell'attività aggiunta sono correttamente inizializzati, identici all'input, con l'aggiunta del parametro progresso, il quale è uguale allo 0%. Dunque l'attività risulta con 'Stato' = IN CORSO

Esito

Il programma stamperà su terminale e 'test_output.txt' "SUPERATO" se i campi sono stati letti, inizializzati e aggiunti correttamente. Mentre il progresso risulta uguale allo 0% e lo 'Stato' risulta IN CORSO.
Altrimenti stamperà TEST 1 "FALLITO".

TEST CASE 2 – AGGIORNA PROGRESSO

Il secondo test effettuato riguarda la verifica del corretto aggiornamento del progresso. Ma in particolar modo esso verifica che il progresso passato da 0% a 100%, comporti il corretto cambiamento dello stato in COMPLETATA con la conseguente registrazione della data di completamento.

Input

La funzione *test_aggiorna_progresso* imposta il progresso dell'attività già aggiunta nel 'Test Case 1' dallo 0% al 100%, aggiornando anche il relativo 'Stato'.

Oracolo

Il campo 'Progresso' dell'attività scelta si aspetta correttamente aggiornato, così come lo 'Stato' che deve risultare come 'COMPLETATA' e il campo 'Data di completamento' deve essere non vuoto.

Output

I campi ‘Progresso’, ‘Stato’ e ‘Data di completamento’ risultano correttamente aggiornati, identici a come previsti dall’oracolo.

Esito

Il programma stamperà su terminale e ‘test_output.txt’ “SUPERATO” se i campi ‘Progresso’, ‘Stato’, ‘Data di completamento risultano aggiornati correttamente.
Altrimenti stamperà TEST 2 “FALLITO”.

TEST CASE 3 – GENERAZIONE DEL REPORT SETTIMANALE

Il terzo test effettuato verifica la corretta generazione del report settimanale. In particolar modo verifica che le attività che rientrano nell’intervallo della settimana corrente e il loro conteggio in (COMPLETATE, IN CORSO, IN RITARDO) sia corretto.

Input

La funzione *test_report_settimanale* crea tre attività di prova (A, B, C) le quali risultano rispettivamente:

- A = COMPLETATA
- B = IN CORSO CON DATA DI SCADENZA [Oggi]
- C = IN CORSO MA CON DATA DI SCADENZA AL DI FUORI DELLA SETTIMANA CORRENTE

Inoltre, utilizza i dati forniti per generare il report settimanale.

Oracolo

I conteggi attesi risultano essere i seguenti:

- 1 ATTIVITÀ COMPLETATA
- 1 ATTIVITÀ IN CORSO
- 0 ATTIVITÀ IN RITARDO

Output

I conteggi risultano identici a quelli attesi

Esito

Il test ha evidenziato che il meccanismo di filtraggio basato sulle date e lo ‘Stato’ delle attività funziona correttamente, generando un report coerente con l’oracolo stabilito.
Di conseguenza il sistema stamperà su terminale e su ‘test_output.txt’ “SUPERATO”, altrimenti stamperà TEST 3 “FALLITO”.

NOTA BENE:

I file ‘test_input.txt’, ‘test_oracle.txt’ e ‘test_output.txt’ verranno riempiti solo dopo aver chiuso il menù di testing, scegliendo l’opzione 0.

AUTORE

Saggese Carmine