20/10/2025

CPP02

1. Controllare flag di compilazione
   -Wall, -Werror, -Wextra – std=c++98
2. Implementazione della OCF
   Ogni classe deve implementare costruttore di default, costruttore di copia, operatore di assegnazione e distruttore.
3. Assenza Memory Leak

EX00

Obiettivo → creare una classe *Fixed* che rappresenti un numero a virgola fissa (fixed -point)

Cosa controllare: Definizione della classe

Domande:

- ✗ Cos'è un numero fixed-point?
  - ✗ Un numero fixed-point è una rappresentazione numerica che utilizza una quantità fissa di bit per la parte intera e una quantità fissa di bit per la parte frazionaria. A differenza dei floating-point (virgola mobile) in cui la posizione della virgola può variare, qui è sempre fissa. Un valore intero viene scalato moltiplicando o dividendo per la potenza di due, così da simulare valori decimali pur lavorando con interi
- ✗ Perché si usano 8 bit frazionari?
  - ✗ L'uso di 8 bit frazionari è una scelta, significa che ogni numero viene moltiplicato per 2^8 = 256, permettendo di rappresentare i valori con una precisione fino a 1/256 = ≈ 0.00390625.
- ✗ Perché _fractionalBits è static const?

Perché tutti gli oggetti condividono lo stesso valore, che non cambia mai.

- ✗ Cosa rappresenta il valore 8? il numero di bit riservati alla parte frazionaria

Nell'output bisogna verificare:
- ordine di chiamata dei costruttori, valore iniziale di _fractionalBits deve essere 0

EX01

Obiettivo → aggiungere costruttori per *int* e *float*, metodi di conversione e l'operatore **<<**

Cosa controllare: nuovi costruttori

Domande:

- ✗ Come si converte un intero in fixed-point?
  - ✗ Valore >> 8
- ✗ Come si converte un float in fixed-point?
  - ✗ roundf(valore * (1 << 8))
- ✗ Come funziona **toFloat()?**
  - ✗ Divide *_fixedPointValue* per *(1 << _fractionalBits)*
- ✗ Perché usare roundf()?
  - ✗ Per evitare errori di arrotondamento nei float.

EX02

Obiettivo: implementare gli operatori di confronto, aritmetici e di incremento/decremento

- ✗ Come si implementa la moltiplicazione?
  - (a * b) >> _fractionalBits
- ✗ Come si implementa la divisione?
  - (a << _fractionalBits) / b
- ✗ Qual è la differenza tra ***++a*** e ***a++***?
  - Pre → incrementa e poi restituisce; post → restituisce e poi incrementa
- ✗ Perché bisogna implementare le versioni sia const che non const?
  - Per supportare oggetti const e non const

EX03
Obiettivo – Usare la classe Fixed per determinare se un punto si trova all'interno di un triangolo
Domande:

- ✗ Perché _x e _y sono const?
  - ✗ Perché le coordinate di un punto sono immutabili dopo la creazione
- ✗ Come funziona l'operatore di assegnazione con membri const?
  - ✗ Non può modificarli, restituisce semplicemente *this

Logica da verificare:
Calcolare i prodotti vettoriali tra il punto e i lati, se un prodotto è 0 significa che il punto è sul bordo (**false)** quindi fuori. Se tutti hanno lo stesso segno = true e quindi è dentro, altrimenti FALSE ed è fuori.

- ✗ A cosa serve il prodotto vettoriale?
  - ✗ Indica da che lato della retta su trova il punto
- ✗ Perché i punti sui bordi devono restituire false?
  - ✗ Il subject lo richiede.
- ✗ Perché gli operatori di confronto restituiscono **bool**, mentre quelli aritmetici restituiscono **Fixed?**
  - ✗ Gli operatori di confronto (**>**, **<**, **==**, ecc.) servono per verificare una condizione logica tra due oggetti **Fixed**, quindi devono restituire un valore booleano (**true** o **false**).
    Gli operatori aritmetici (**+**, **-**, *****, **/**) invece generano un nuovo valore numerico risultante da un calcolo, che deve anch'esso essere un oggetto **Fixed**.
    Perciò la differenza nel tipo di ritorno è legata alla natura dell'operazione: logica per i confronti, aritmetica per i calcoli.
- ✗ Perché il pre-incremento restituisce un riferimento e il post-incremento una copia?
  - ✗ Nel pre-incremento si vuole restituire l'oggetto stesso dopo averlo modificato, restituire una referenza evita copie inutili e rispetta il comportamento del linguaggio. Nel post-incremento il valore restituito deve essere quello precedente all'incremento, quindi serve creare una copia temporanea del valore prima di modificarlo.
- ✗ Come il prodotto vettoriale determina la posizione del punto rispetto al triangolo?
  - ✗ se il segno del prodotto vettoriale tra i lati e i segmenti che connettono il punto con i vertici è sempre lo stesso, significa che il punto è "dallo stesso lato" di ogni lato, e quindi **dentro** il triangolo. Se anche solo uno dei segni cambia, il punto è fuori.
- ✗ Perché la classe `Point` non alloca memoria dinamica?
  - ✗ La classe Point è volutamente semplice, contiene solo due membri Fixed _x e _y che rappresentano le coordinate del punto.
- ✗ Perché esistono versioni `const` e non `const` di `min()` e `max()`?
  - ✗ Le funzioni min() e max() sono implementate in doppia versione per motivi di compatibilità: La versione non const serve quando vogliamo poter modificare il risultato (es. Fixed::min(a, b) = nuovo_valore;). La versione const serve quando lavoriamo con oggetti costanti, cioè che non devono essere modificati. In questo modo, la funzione può essere chiamata su oggetti const senza violare la loro immutabilità, mantenendo la massima flessibilità d'uso.