Relazione Progetto C++ - Matrice3D

Introduzione

Il Progetto Matrice3D tratta la realizzazione di una classe **templata Matrice3D** e di un programma relativo per testarne le funzionalità (Nello specifico un programma a riga di comando). La classe Matrice3D implementa quindi una matrice a 3 dimensioni: z (piano), y (riga), x(colonna) le quali rappresentano delle coordinate in un piano e alcuni metodi e funzionalità per il suo utilizzo:

- I **Metodi fondamentali** di una classe: il Costruttore (default e secondario in questo caso), Il Costruttore di copia, L'operatore di assegnamento e il Distruttore.
- Possibilità di **convertire** una Matrice3D definita su tipo U in una Matrice3D di tipo T se il tipo U è convertibile nel tipo T.
- Leggere e scrivere attraverso **l'operatore()** specificando le coordinate: Es. mat(1,2,3) = mat(2,2,3).
- Possibilità di utilizzare gli opportuni **iteratori** di **lettura** e **scrittura** per accedere ai valori della Matrice3D, secondo l'ordine riportato.
- Metodo **slice**, il quale ritorna una sotto-matrice contenente valori nell'intervallo z1 a z2, y1 a y2 e x1 a x2.
- Possibilità di comparare con **l'operatore ==** due Matrici3D.
- Metodo **fill** per riempire una Matrice3D con valori presi da una sequenza di dati identificata da iteratori generici. I vecchi valori verranno sovrascritti e i nuovi inseriti nell'ordine di iterazione dei dati della matrice.
- Metodo globale **trasform**, che data una Matrice3D A su tipo T e un funtore F, ritorna una nuova matrice B convertita a tipo Q con elementi ottenuti dall'applicazione del funtore F sugli elementi di A.

Struttura del Progetto

Il progetto è strutturato nel seguente modo:

- *main.cpp* (file main con test effettuati sulla matrice).
- *matrice3d.h* (la classe templata Matrice3D).
- *Makefile* (per compilazione veloce).
- *Doxyfile* (e relativa cartella html con la generazione della documentazione).

Scelte implementative

Per la semplicità di gestione dei dati, la Matrice3D è stata rappresentata attraverso un **singolo puntatore** (un array di tipo T) di dimensione equivalente al prodotto delle tre dimensioni. Inoltre è stato introdotto un ulteriore dato membro "**size**" per contenere la dimensione totale della matrice e un funtore di tipo "**Cmp**" utilizzato successivamente per la comparazione nell'operatore ==.

I metodi fondamentali gestiscono la creazione corretta della matrice, con eventuali **try-catch** ed eccezioni se necessario. Nello specifico, essi sono stati utilizzati in caso di fallimento di un **assegnamento** di due tipi T della Matrice3D, in quanto non conoscendo il tipo T, esso può essere primitivo ma anche custom, e in quest'ultimo caso potrebbe richiedere operazioni più complesse di una copia bit a bit (nel caso di un int per esempio). Per cui attraverso try-catch, riporto l'oggetto allo stato **coerente** attraverso la funzione **clear()**, la quale cancella l'array, lo fa puntare a nullptr e resetta le dimensioni della matrice. La gestione delle new viene "ignorata" in quanto, in caso di fallimento, la initialization list mantiene lo stato coerente dell'oggetto. Oltre ai quattro metodi fondamentali è stato definito un costruttore secondario per la creazione di una Matrice3D con dimensioni impostate dall'utilizzatore. Il tipo utilizzato dalle dimensioni è **unsigned int** per scelta implementativa, in quanto la matrice viene rappresentata molto spesso come una tabella con varie celle. Unsigned è specificato in quanto non possiamo avere una matrice di dimensioni negative.

La **possibilità di conversione** della Matrice3D di tipo T a un tipo U è stata gestita tramite un costruttore di conversione implicita, attraverso uno static cast al tipo T da un tipo U se possibile.

Per **l'operatore()** è stata utilizzata una formula matematica per convertire il sistema di coordinate in un unico indice (la nostra matrice è un array "lungo") ovvero: **z* dimensioneX * dimensione Y + y * dimensione X + x.** Le prime due moltiplicazioni ci spostano sul piano z corretto mentre il resto si occupa di selezionare la cella corretta.

Gli **iteratori** che sono stati implementati, ovvero l'iterator e il const_iterator per matrici costanti, sono di tipo **random-access.** Il random-access iterator è stato scelto in quanto è quello più coerente con il metodo di accesso dei dati interni alla nostra classe (ovvero l'array) e ci permette di iterare i dati nella direzione indicata dalla consegna. Inoltre la sua realizzazione è stata effettuata tramite la versione "**trucco**", ovvero utilizzando dei **puntatori**, che sono a

tutti gli effetti dei random-access iterator. I traits e tutte le operazioni necessarie per l'iteratore sono definiti dal puntatore.

Il **funtore** precedentemente menzionato viene utilizzato con **l'operator==** per effettuare il confronto tra i dati delle due matrici da confrontare. E' stato definito inoltre un **funtore di default** all'interno della classe per semplificare ulteriormente il suo utilizzo. Il funtore di default utilizza come mezzo di comparazione **l'operatore== del tipo T** della matrice e viene utilizzato nel caso non venga specificato un comparatore specifico (può risultare comodo se si utilizzano tipi primitivi o tipi custom che sicuramente hanno l'operatore== ridefinito). Altrimenti possiamo specificare durante la creazione della matrice, oltre al tipo, il funtore da utilizzare per la **comparazione**. La motivazione della presenza del funtore è simile a quella per cui sono stati utilizzati i try-catch con assegnamenti importanti (che alterano la matrice), ovvero che **non possiamo** sapere il tipo di dato che verrà utilizzato dall'utente, se quest'ultimo utilizza un dato custom, esso potrebbe **non** aver implementato **l'operatore==**. Per cui diamo la possibilità all'utente di definire un funtore di comparazione in modo rendere la nostra classe più **generica**.

Sono state definite inoltre alcune **eccezioni custom** all'interno della classe, per individuare in caso di errore (eccezioni), il problema in maniera più rapida.

Gli altri metodi sono stati realizzati in funzione del testo dell'esame, nello specifico nel metodo **slice** e **trasform** non vi è la necessità di riportare, tramite try-catch, lo stato coerente della matrice in quanto vengono creati nuovi oggetti, non alterando la matrice vera e propria. Per tanto in caso fallimento **verrà** lanciato un **throw** automatico dall'errore riscontrato, ma non ho bisogno di reimpostare uno stato **coerente**.

Nel caso in cui vengano **inseriti** dei dati ritenuti **non validi** (Es. una matrice di dimensioni negative), l'input verrà **gestito** e lanciata una **eccezione** custom.

Nel file "*main.cpp*" sono stati effettuati vari **test** su ogni funzionalità richiesta, quindi per i metodi fondamentali, conversione e tutti gli altri metodi necessari. Sono state inoltre testate alcune **eccezioni** e comportamenti utilizzando **dati custom** (*Coordinates* definita nel main) e funtori differenti.

Il progetto è stato **testato** in un ambiente **Windows 11** e **Ubuntu Linux** e controllato attraverso **valgrind** per evitare eventuali **memory leaks**.