# Relazione Progetto programmazione C++

Tombolini Simone

Matricola: 869564

Università degli studi di Milano-Bicocca

15/01/2023

## Introduzione

Il progetto ha lo scopo di realizzare una classe Templata chiamata Matrix3D, rappresentante una matrice tridimensionale, per la semantica e le funzionalità richieste faccio riferimento al file “Esame-230125.pdf” fornito come traccia d’esame, in questa relazione procederò a esporre le principali scelte progettuali, i dettagli sono rimandati alla documentazione doxygen fornita all’interno della consegna d’esame e generata dal codice.

## Struttura dati utilizzata

La Matrice è disposta su una memoria contigua allocata sullo heap tramite un array monodimensionale, il funzionamento della matrice è definito solo a livello logico, e la matrice non può cambiare dimensione una volta istanziata.

Ho ritenuto questa scelta la migliore per semplicità di progettazione della struttura efficienza dell’accesso ai dati e risparmio di memoria.

I dati sono disposti seguendo le frecce indicate per gli iteratori come mostrato nell’immagine del testo d’esame, quindi la posizione (0, 0, 0) è piazzata all’inizio dell’array e viene incrementata esattamente seguendo la figura.

Per l’iteratore, dato la tipologia della struttura dati ho deciso di implementare un random access iterator, che permette la maggiore libertà possibile nel manipolare la struttura dati, essendo essa un array gli operatori sono modellati sui puntatori nativi di C++.

## L’operatore ==

Oltre alla classe templata richiesta dal progetto è aggiunto, all’inizio del file un funtore templato su tipi T chiamato eql_default, esso è dichiarato come scelta di default all’interno della dichiarazione dei tipi nella matrice, riutilizza l’operatore == del tipo T permettendo così di implementare un funtore per tutti i tipi che hanno definito l’operatore ==, come ad esempio i tipi primitivi.

Ovviamente la possibilità di dichiarare un funtore personalizzato rimane comunque valida, sia per ridefinire la logica dell’operatore == di un singolo tipo, dichiarare un funtore è necessario quando i tipi custom non implementano ==, e quindi non hanno accesso al funtore di default.

## L’operatore [ ]

Questo è un operatore non richiesto dal testo, ho deciso di implementarlo comunque visto che mi è di supporto per alcuni metodi e rende estremamente più facile leggere e scrivere una matrice.

Ci tengo a far notare che il tipo di ritorno non espone nulla della struttura interna, visto ritorna solo il valore T e non il puntatore all’array.

## Metodi di stampa di debug

Ho creato due metodi di debug, resi comunque pubblici, che mi sono stati utili in fase di test e di debug, print_message e print_matrix.

Il primo stampa un messaggio su cout, mentre il secondo stampa l'intera matrice con l'aggiunta di alcune parentesi che aiutano a visualizzarla.

Una volta passati in modalità release le stampe non saranno più eseguite e i metodi non avranno più alcuna utilità, inoltre questi metodi possono essere tranquillamente impostati a false e le stampe non vengono eseguite, in una release pubblica sarebbe stato questo il caso, ma essendo che questo è un progetto didattico e la classe viene usata solo in fase di test ho deciso di eseguire tutte le stampe.

## Eccezione custom

Ho realizzato un eccezione custom chiamata out_of_bound_exception, derivata dalla std::logic_error, questa eccezione è utilizzata quando l'utente richiede una posizione fuori dalla matrice, indicando anche i punti richiesti erroneamente con i metodi get_.

Il motivo di questa eccezione è principalmente didattico, l'utilizzo di questa eccezione è limitata all'operatore (), in caso contrario avrei dovuto creare eccezioni non strettamente richieste per qualsiasi casistica, essendo un lavoro troppo oneroso per il tempo limitato del progetto ho deciso di implementarla in un solo punto e rilanciare eccezioni generiche o logic error quando necessario.

## Struttura dei Test

I test non sono strutturati in un modo particolare, vengono inseriti in dei metodi di comodo, che vengono eseguiti nel main.

Tutti i metodi, gli operatori e i casi limite che ho individuato sono testati all'interno di questi metodi, divisi da delle stampe come titolo su cout e con parecchi messaggi di debug per mostrare l'evoluzione dei test.

Sono presenti anche due classi custom, user e point, la prima rappresenta un utente in una rubrica e la seconda rappresenta un punto su un piano cartesiano, sono classi meramente di comodo per testare le funzionalità della matrice anche con diversi funtori realizzati appositamente per queste classi e per alcuni primitivi.

Inoltre, è presente un altro test totalmente commentato, che ho eseguito per simulare il fallimento della new, per poterlo riprodurre è necessario seguire alla lettera i commenti in italiano all'interno del test, ricordandosi di rimettere tutto com'era prima, visto che necessita di generare delle eccezioni artificiali con stampe di throw nel momento della new.

I test della serializzazione creano un file chiamato output.txt, in cui viene scritta una matrice di prova.

## Informazioni sullo sviluppo

Il progetto è stato realizzato sulla mia macchina personale tramite l'IDE visual studio 2022, compilato con il compilatore Microsoft integrato nella piattaforma, inoltre come richiesto è stato impiegato Valgrind per il controllo di memory leak, WLS come ulteriore piattaforma di controllo, la versione è la seguente:

```
C:\Users\PC>wsl -l -v
  NAME                   STATE           VERSION
```

* Ubuntu-20.04  Running  2

Con istallata la versione di gcc:

gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04.1)

Inoltre, è stato testato sul compilatore della macchina virtuale fornita appositamente per il corso.

Così come la parte di QT, creata nella mia macchina locale ma testata su quella virtuale.