UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Matematica
CORSO DI LAUREA IN INFORMATICA

RELAZIONE SUL PROGETTO DI PROGRAMMAZIONE AD OGGETTI “MATRIX CALC”

Studente: Marco Masiero, 1124621

ANNO ACCADEMICO 2017-2018

1. ABSTRACT

Il progetto si prefigge lo scopo di implementare una calcolatrice in grado di effettuare calcoli di natura numerica tra matrici oltre alle operazioni specifiche su di esse matrici. I tipi numerici che possono essere utilizzati saranno interi, razionali e complessi. La calcolatrice offrirà funzioni specifiche delle matrici come operazioni algebriche del calcolo del determinante, della matrice inversa, della traccia, del rango, della trasposta, moltiplicazione per uno scalare e utilizzo del metodo di gauss. Le operazioni messe a disposizioni saranno disponibili agli utenti attraverso un'interfaccia intuitiva.

2. COMPILAZIONE ED ESECUZIONE

Parte C++

La compilazione del progetto necessita di un file project (.pro) per qmake diverso da quello ottenibile tramite l'invocazione di qmake -project.

Viene quindi consegnato un file MatrixCalc.pro che permette la generazione automatica tramite qmake del Makefile tramite la sequenza di comandi qmake ⇒ make.

3. MODELLO LOGICO

3.1 TIPI NUMERICI

1) La classe **Rational** è una classe i cui oggetti rappresentano numeri razionali nel formato *num/den*.

Ogni istanza di Rational descrive un numero razionale che è caratterizzato da un numeratore(**num**) e denominatore(**den**) con valori **long long int**, la scelta è stata fatta per ridurre al minimo gli errori di overflow nel caso di calcoli con valori elevati. Vengono offerti dei metodi costanti di estrapolazione dei campi dati privati come **Numerator()**, **Denominator().** Viene fornito un costruttore **Rational(n,d)** con valori di default a (0/1) per rappresentare lo zero. Nel caso di input errato nella costruzione del razionale, ossia un razionale che ha 0 al denominatore, viene lanciata un'eccezione a **RationalError** con l'opportuna spiegazione: "Non è possibili dividere per 0". Ogni tipo razionale istanziato sarà ridotto ai minimi termini grazie al metodo **Simplify()** il quale a sua volta utilizza un metodo statico **MCD** che reperisce il minimo comune denominatore del razionale e lo restituisce a Simplify per effettuare la riduzione. Per comodità della visualizzazione a schermo del numero razionale è stato introdotto un metodo **Normalize**() che cambia il segno del numero razionale, se il denominatore risulta essere negativo. Vengono inoltre gestite le operazioni quali addizione, sottrazione, moltiplicazione e divisione; sia tra razionali che tra semplici interi.

2) La classe **Complex** è una classe i cui oggetti rappresentano numeri razionali nel formato *a+ib,* dove "*a"* rappresenta la parte reale, e "*b"* la parte immaginaria.

Complex è una classe **templetizzata,** in cui ogni istanza descrive un numero complesso caratterizzato da una parte reale e una immaginaria instanziabili al tipo T scelto, con valore di default **double**. Offre i metodi costanti **getReal()** e **getImg()** per ottenere il valore reale ed immaginario rispettivamente. Come per la classe Rational, Complex implementa le operazioni aritmetiche addizione, sottrazione, moltiplicazione e

divisione. Queste operazioni sono usabili sia tra tipi Complessi che tra tipi primitivi T. Viene gestita l'eccezione per la la divisione per zero tramite una throw a **ComplexError:** "Non è possibili dividere per 0". Grazie ai template è possibile utilizzare oggetti complessi istanziati con valori razionali.

3.2 GERARCHIA

La classe che permette le operazioni alla base dell'idea della calcolatrice è **Matrix**, una classe **templetizzata** che rappresenta matrici quadrate e rettangolari istanziabili con tipi numerici primitivi, razionali o complessi. Si tratta di una **classe astratta polimorfa** da cui si deriveranno le classi concrete **SquareMatrix** e **NonSquareMatrix.**

Matrix è caratterizzata dal numero di colonne e di righe che definiscono la dimensione della matrice e un vettore che viene utilizzato per contenerne i valori.

Vengono definiti i metodi costanti **getRows()** e **getColumns()** per ottenere il numero di righe e di colonne di cui è composta la matrice. Le dimensioni della matrice sono **unsigned int** per evitare che si possa costruire una matrice di dimensioni negative, in quanto queste non possono esistere. La scelta di utilizzare il vector della libreria STL è dovuto al voler mantenere separati la parte di modellazione logica del programma dal framework di Qt. I campi dati che la compongono sono stati definiti protected per renderli accessibili all'interno delle classi derivate.

Sono poi definiti i costruttori ad uno e due parametri, utilizzati dalle classi derivate, il costruttore di copia avente lo stesso comportamento di quello standard e il distruttore reso virtuale in modo tale da richiamare il giusto distruttore della gerarchia in caso di distruzione.

La classe Matrix definisce i prototipi dei metodi virtuali di **clone()**, **addizione**, **sottrazione**, **moltiplicazione**, **trasposta**, **rango** e **moltiplicazione per scalare** che saranno effettivamente implementati nelle classi derivate concrete. Ridefinisce **l'operatore()**,come metodo costante e non (dovute alle operazioni che le utilizzano che fanno o meno side-effect), per ottenere il valore all'interno della matrice agli indici (*i,j)*; l'inserimento di valori che non coincidono con le dimensioni della matrice vengono gestiti dall'errore **OutOfRangeError:** "L'elemento in posizione richiesta non esiste".

Viene implementato il metodo **gauss**, che ha come parametro di default un puntatore a Matrix nullo.
La funzione permette di ottenere una matrice triangolare superiore dalla matrice d'invocazione, se s'incontra una riga di soli zeri viene effettuato uno scambio di riga per mantenere la definizione di matrice triangolare superiore, ossia avete tutti zeri al di sotto della diagonale principale.

NonSquareMatrix è una classe derivata che concretizza la classe base **Matrix**.

Viene utilizzato il costruttore a due parametri richiamando quello di Matrix e ne implementa i metodi astratti.

Clone: restituisce un nuovo puntatore a matrice che è la copia di quella di invocazione.

Addizione: restituisce una matrice, con lo stesso numero di righe e colonne delle due matrici, che rappresenta la somma degli elementi che occupano la stessa posizione.

Sottrazione: restituisce una matrice, con lo stesso numero di righe e colonne delle due matrici, che rappresenta la differenza degli elementi che occupano la stessa posizione.

Questi due metodi possono lanciare rispettivamente gli errori **AdditionError()** e **SubtractionError()** che indicano che le dimensioni delle matrici non sono uguali rendendo impossibile effettuare l'operazione

Moltiplicazione: restituisce una matrice risultato che ha come dimensione il numero di righe della prima matrice e il numero delle colonne della seconda. La matrice ottenuta avrà come valore nella posizione (i,j) il risultato della "moltiplicazione riga per colonna". Questo metodo può lanciare l'eccezione **MultiplicationError()** che indica la condizione necessaria indicata sopra per effettuare l'operazione.

Moltiplicazione per uno scalare: Ogni elemento della matrice viene moltiplicato per un intero

Rango: restituisce il valore che rappresenta il massimo numero di righe linearmente indipendenti tra loro, per farlo utilizza il metodo di gauss ereditato dalla classe base.

Trasposta: data una matrice (n,m) restituisce una matrice (m,n) invertendo anche gli elementi in posizione (i,j) in (j,i).

SquareMatrix è la seconda classe derivata che concretizza la classe base **Matrix**.

Viene utilizzato il costruttore ad un parametro richiamando quello di Matrix ed implementa i metodi astratti.

I metodi astratti effettuano le stesse operazioni descritte per la NonSquareMatrix restituendo però delle matrici quadrate.

Questa classe implementa dei metodi propri, utilizzabili per le proprietà della matrice quadrata, quali:

Trace(): restituisce la somma degli elementi della diagonali principale

Determinant(): calcola il determinante della matrice sfruttando il metodo di Gauss; effettua prima l'eliminazione di Gauss della matrice e poi ne somma gli elementi diagonali. Il numero di scambi tra righe nell'eliminazione di Gauss deve essere in numero pari altrimenti il determinante cambierebbe di segno; per ovviare a questo dettaglio su cui fare attenzione è stato messo un flag booleano nel metodo di Gauss che ad ogni scambio muta da true a false o viceversa in modo da poter verificare tramite il valore intero restituito se gli scambi sono stati in numero pari (valore di ritorno positivo) o in numero dispari (valore di ritorno negativo).

Se la matrice tramite il metodo di Gauss risulta singolare allora il determinante sarà uguale a 0 altrimenti restituisce la somma degli elementi diagonali facendo attenzione al segno del valore ritornato da Gauss che ne determina il segno.

Inverse() restituisce la matrice inversa utilizzando il metodo di Gauss-Jordan. Grazie al metodo **inverseExt()** che ha il compito di procedere all'eliminazione di Gauss della matrice estesa con la matrice identità; effettuando i calcoli sugli elementi di pivot si arrivare a restituire l'inversa. Se la matrice tramite il metodo di Gauss risulta singolare allora il determinante sarà uguale a 0, altrimenti restituisce la somma degli elementi diagonali facendo attenzione al segno del valore ritornato da Gauss che ne determina il segno.

4. DESCRIZIONE GUI

Per la progettazione dell'interfaccia grafica abbiamo adottato il pattern Model-View-Controller utilizzando Qt Designer. MVC è un patter che adotta una suddivisione dei compiti in cui la parte logica è gestita interamente dal Model, la View gestisce l'interfaccia per l'utente e il Controller si occupa di tradurre le iterazioni dell'utente dalla View in esecuzione dei dati sul Model.

View: La view rappresenta ciò che l'utente vede e interagisce. E' caratterizzata da un campo dati puntatore a Controller che viene creato alla costruzione della View, e fa puntare il controller al model istanziato in quel

momento. Utilizza tre TableWidget per rappresentare le matrici. Sono stati inseriti due SpinBox per settare il numero di righe e colonne della matrice nella TableWidget sottostante e i PushBotton necessari per rappresentare le operazioni tra matrici e quelle sulla matrice. La View contiene dei metodi propri che fanno utilizzo di un parser da noi definito che utilizza espressioni regolari. **setZero(bool)** (bool indica quale matrice, true per la prima, false la seconda) per settare a zero le matrici alla creazione di tali, sia per rendere esteticamente migliore la vista all'utente che per l'uso pratico del parser per convertire l'input al numero digitato. **printResult()** è invece utilizzato per stampare nella TableWidget dei risultati i valori delle matrici convertiti a stringhe grazie al parser. Viene ridefinito il distruttore che richiama la distruzione del controller puntato.

Controller: Il controller funge da tramite tra View e Model. E' caratterizzato da un puntatore a Model. Il controller elabora le interazioni dell'utente nell'interfaccia e passa i comandi al Model per effettuare le operazioni. Gli errori che possono verificarsi vengono gestiti nel controller che attraverso dei MessageBox a pop up segnalano l'eventuale errore. Viene ridefinita la distruzione del controller che richiama la distruzione del Model puntato.

Model: Il Model è caratterizzato da tre puntatori alla classe base Matrix, i quali utilizzano tipi complessi con campi dati razionali. In questo modo viene sfruttato il polimorfismo a run-time invocando il metodo esatto nelle classi derivate. Abbiamo deciso di rendere possibile tutti i calcoli su interi, razionali e complessi disponibili utilizzando solo tre puntatori, che rappresentano rispettivamente le due matrici operandi e il risultato di queste. Viene ridefinito il distruttore che distrugge le matrici puntate dai suoi campi dati.

5. SUDDIVISIONE DEL LAVORO

Compagno: Stefano Nordio, 1122976.

L'intero progetto è stato pensato e svolto in ogni sua fase contemporaneamente da entrambi, sia in fase di progettazione che in fase di codifica. La parte del lavoro che ho svolto singolarmente riguarda la definizione delle classi Rational e Complex, h e cpp, e il lancio delle loro eccezioni. Lo studio e la documentazione sono stati effettuati individualmente. Per tutto il resto del progetto, abbiamo collaborato alla creazione e all'implementazione della riguardante la GUI.

6. MANUALE UTENTE GUI

L'interfaccia come già descritto presenta due tabelle, per poter scegliere le dimensioni bisogna utilizzare gli spinBox sopra le tabelle.

Le operazioni +, *, - tra le due matrici indicano le operazioni disponibili tra di loro.

Le operazioni sotto le tabelle indicano le funzionalità applicabili alla singola matrice.

Regole per l'input di:

- numeri razionali complessi:
  segno(+ o -)[cifre]/[cifre] segno(+ o -)[cifre]/[cifre] es. +1/56-72i
  senza spaziature tra reali e razionale
- per numeri razionali: [cifre]/[cifre] es. 7/5
- interi [cifre] es. 72

7. ORE IMPIEGATE

analisi preliminare del problema: 4h

progettazione modello e GUI: 15h

apprendimento libreria Qt: 9h

codifica modello e GUI:15h

debugging: 3h

testing: 4h

8. AMBIENTE DI SVILUPPO

Versione di Qt: Qt Creator 4.5.0

Libreria Qt: 5.10.0

Compilatore: MinGW 32 Bit

Sistema operativo: Windows 10, home