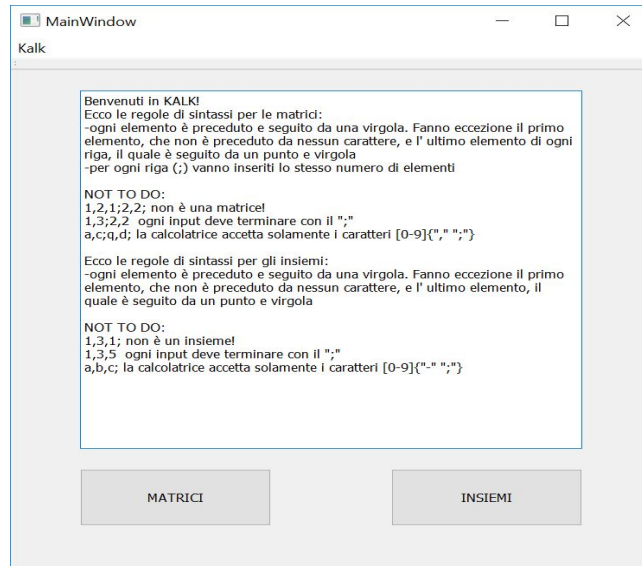


KALK

Relazione di Progetto

Anna Poletti 1123587



Sommario

1. Scopo del progetto
2. Descrizione delle gerarchie utilizzate
 - 2.1. Gerarchia della Collezione Dati
 - 2.1.1. MatriceR
 - 2.1.2. MatriceQ
 - 2.1.3. Insieme
 - 2.2. Gerarchia delle eccezioni
3. Descrizione dell'uso del codice polimorfo
4. Descrizione della GUI
5. Manuale utente
6. Ore di lavoro
7. Ambiente di sviluppo

1 Scopo del progetto

Il progetto si prefigge lo scopo di creare una calcolatrice che permette l'uso di operazioni e calcoli sulle matrici e sugli insiemi. Le matrici inserite sono di due tipi: quadrate e rettangolari.

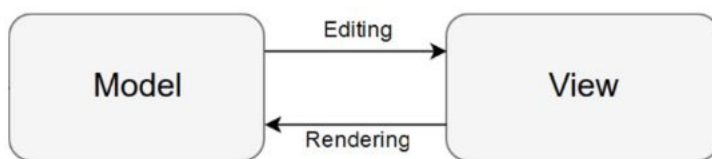
La calcolatrice permette di eseguire sia operazioni che coinvolgono due matrici, sia operazioni che coinvolgono una singola matrice. Per quanto riguarda gli insiemi è possibile eseguire sia operazioni che coinvolgono due insiemi, sia operazioni che coinvolgono un singolo insieme.

Sulle matrici la prima categoria di operazioni è possibile sommare, sottrarre e moltiplicare due matrici, mentre nella seconda è possibile calcolare matrice trasposta, traccia della matrice, matrice elevata alla n , matrice moltiplicata per uno scalare. Inoltre è possibile verificare se la matrice è simmetrica.

Le operazioni permesse dipendono dal tipo di matrici inserite (ad esempio non sarà possibile calcolare la traccia di una matrice rettangolare) e dalla compatibilità delle matrici inserite (ad esempio non sarà possibile sommare due matrici di diverse dimensioni). Negli insiemi invece questo problema non sussiste.

Le matrici e gli insiemi contengono elementi di tipo *int*.

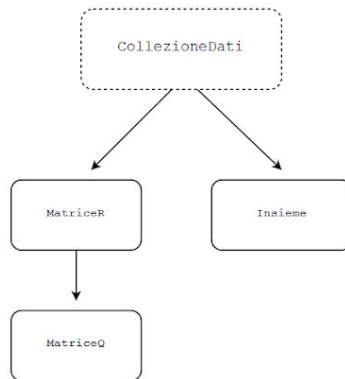
Per la realizzazione del progetto ho ritenuto opportuno utilizzare il pattern architetturale *Model View* che permette l'astrazione tra model e view. Questa separazione agevola la localizzazione di possibili errori e semplifica implementazioni e modifiche future.



2 Descrizione delle gerarchie utilizzate

Il progetto presenta due gerarchie: una che rappresenta la collezione dati e una che rappresenta le eccezioni che possono essere lanciate dal programma.

2.1 Gerarchia della CollezioneDati



La classe base astratta ***CollezioneDati*** fornisce l'interfaccia comune ai tipi derivati. La scelta di segnare *protected* i campi dati è stata adottata in modo che i campi dati fossero accessibili direttamente dalle classi derivate, ma inaccessibili all'esterno.

La classe *CollezioneDati* dispone dei seguenti costruttori :

- `CollezioneDati(unsigned int);` costruttore a un parametro utilizzato dalle classi derivate.
- `CollezioneDati(unsigned int, std::vector<int>);` costruttore a due parametri utilizzato dalle classi derivate.
- `CollezioneDati(const CollezioneDati&);` costruttore di copia che si comporta come quello standard.
- `virtual ~CollezioneDati();` distruttore virtuale che in caso di distruzione richiama il giusto distruttore della gerarchia

La classe *CollezioneDati* dispone dei metodi get dei campi dati e i metodi `int getMin() const` e `int getMax() const` implementati direttamente nella classe base.

Sono presenti i metodi virtuali puri di assegnazione, clonazione, uguaglianza, differenza e conversione in stringa che sono implementati nelle classi derivate. La scelta di renderli virtuali puri dipende dal fatto che hanno bisogno di lavorare con oggetti concreti e quindi implementarli nella classe base sarebbe stato sbagliato.

La matrice presenta i seguenti metodi virtuali puri:

- `virtual CollezioneDati& operator=(const CollezioneDati&)=0;`
- `virtual CollezioneDati* clone() const=0;`
- `virtual bool operator==(const CollezioneDati&) const = 0;`
- `virtual CollezioneDati* operator-(const CollezioneDati&) const=0;`
- `virtual std::string toString() const = 0;`

2.1.1 MatriceR

La classe derivata concreta **MatriceR** possiede due campi dati protetti (righe,colonne) ed implementa il costruttore a due e tre parametri e il costruttore di copia utilizzando i relativi costruttori della classe padre. Possiede inoltre un costruttore **MatriceR(QString)** che funge da parser. Nel caso in cui la stringa non sia corretta per il parsing viene lanciata un'eccezione. Il metodo statico **static bool checkparserR(QString all)**; effettua un tentativo di parsing per verificare la correttezza della stringa.

MatriceR possiede i metodi get di righe e colonne e inoltre offre i seguenti metodi propri:

Operazioni:

- **MatriceR*** operator*(**const int& s**) **const**;
- **MatriceR*** operator+(**const MatriceR&**) **const**;
- **MatriceR*** operator*(**const MatriceR&**) **const**;
- **MatriceR*** trasposta() **const**;

Metodi di accesso agli elementi:

- **const int&** getElemento(**const unsigned int&**, **const unsigned int&**) **const**;
- **int&** goElemento(**const unsigned int &**, **const unsigned int &**);

La classe **MatriceR** implementa inoltre il distruttore con comportamento standard e i metodi virtuali puri definiti in **CollezioneDati**.

2.1.2 MatriceQ

La classe concreta **MatriceQ**, derivata da **MatriceR**, implementa il costruttore a uno e due parametri e il costruttore di copia utilizzando i costruttori della classe padre. Viene inoltre dichiarato un costruttore a due parametri **MatriceQ(unsigned int, std::vector<int>)** che viene implementato con il costruttore a tre parametri della classe padre. Vengono implementati il distruttore virtuale e i metodi **MatriceQ(QString)**; e **static bool checkparserQ(QString all)** che si comportano in modo simile alle rispettive funzioni di **MatriceR**.

Sono poi implementate operazioni di **MatriceQ** che riguardano soltanto le matrici quadrate:

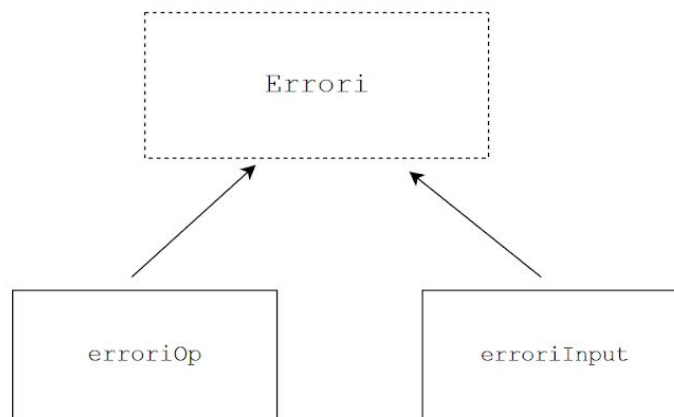
- **int** traccia() **const**; somma gli elementi che si trovano sulla diagonale principale della matrice.
- **bool** isSimmetrica() **const**; verifica che la matrice di invocazione sia uguale alla sua trasposta
- **matriceQ*** esponenziale(**const unsigned int&**) **const**; eleva la matrice ad un certo numero

2.1.2 Insieme

La classe concreta **Insieme** derivata dalla classe base astratta **CollezioneDati** implementa il costruttore a uno e due parametri e il costruttore di copia utilizzando i costruttori della classe base. Possiede inoltre un costruttore **Insieme(QString)** che funge da parser. Nel caso in cui la stringa non sia corretta per il parsing viene lanciata un'eccezione. Il metodo statico **static bool checkparser(QString all)**; effettua un tentativo di parsing per verificare la correttezza della stringa.

La classe Insieme possiede i metodi propri **Insieme*** **unione(const Insieme&) const**; ed **Insieme*** **intersezione(const Insieme&) const**; e implementa il distruttore e i metodi virtuali della classe base astratta.

2.2 Gerarchia delle eccezioni



La gerarchia della classe che gestisce le eccezioni è composta dalla classe base astratta **Errori** (con campo dati la stringa **exc** e il relativo costruttore a un parametro) e dalle classi derivate **erroriOp** ed **erroriInput**.

La classe **erroriOp** gestisce gli errori relativi alle operazioni e possiede il costruttore a un parametro **erroriOp(unsigned int)**; implementato utilizzando il costruttore della classe base. La classe **erroriInput** gestisce gli errori relativi all'inserimento dei dati e implementa il costruttore a un parametro.

3 Descrizione dell'uso del codice polimorfo

Sono presenti i seguenti metodi virtuali puri che si estendono in tutta la gerarchia tramite l'overriding:

-
- `CollezioneDati& operator=(const CollezioneDati&);` operatore di assegnazione
 - `CollezioneDati* clone() const;` tramite delle funzioni covarianti viene restituito un puntatore a una classe derivata
 - `bool operator==(const CollezioneDati&) const;` operatore di uguaglianza
 - `~CollezioneDati();` distruttore
 - `CollezioneDati* operator-(const CollezioneDati&) const;` operatore di differenza
 - `std::string toString() const;` viene restituita la stringa che traduce l'oggetto di invocazione

Nella gerarchia che gestisce le eccezioni il distruttore è marcato virtuale.

4 Descrizione della GUI

L'interfaccia è stata realizzata sfruttando il tool *QtDesigner*.

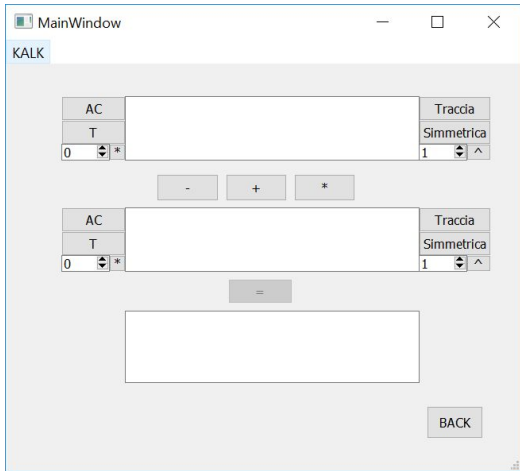
La classe *MainWindow* funge da schermata introduttiva della calcolatrice. Nella schermata è presente una breve guida all'utilizzo e due *QPushButton* **MATRICI** ed **INSIEMI** che permettono di accedere alle rispettive aree di calcolo vere e proprie, implementate dalle classi *MatrixWin* e *setWin*.

La classe *MatrixWin* è caratterizzata da tre puntatori di tipo *MatriceR* che rappresentano le due matrici operando e la matrice risultato, mentre la classe *setWin* possiede un puntatore di tipo *Insieme*.

Tramite l'uso dei *QPushButton* è possibile chiamare i metodi e interagire con le funzionalità fornite dalla calcolatrice.

5 Manuale utente

Dopo aver premuto il pulsante **MATRICI** presente nella prima schermata l'utente visualizza la seguente schermata:



Operazioni possibili:

AC: clear all

T: trasposta

: moltiplicazione per scalare

Traccia;

Simmetrica: verifica se la matrice è simmetrica

: elevamento a potenza

- sottrazione tra due matrici

+ somma tra due matrici

* moltiplicazione tra due matrici

Le regole di sintassi per le matrici sono le seguenti:

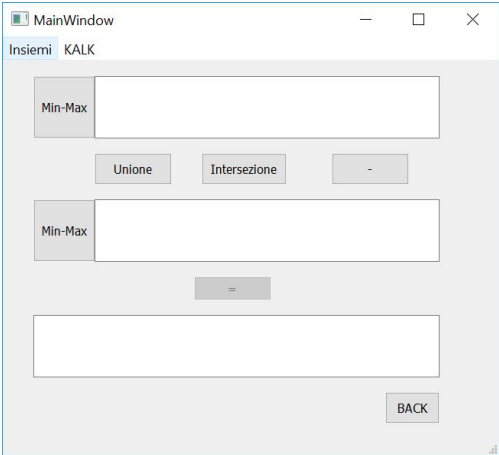
- ogni elemento è preceduto e seguito da una virgola. Fanno eccezione il primo elemento, che non è preceduto da nessun carattere, e l'ultimo elemento di ogni riga, il quale è seguito da un punto e virgola.
- per ogni riga (;) vanno inseriti lo stesso numero di elementi.
- non va inserito nessuno spazio tra un elemento e il successivo

Esempio input corretto (matrice 3x2): 1,2;3,4;5,6;

Esempi di input errati:

- 1,2,1;2,2; non è una matrice
- 1,3;2,2 ogni input deve terminare con il ";"
- a,c;q,d; la calcolatrice accetta solamente i caratteri [0-9]{"," " ";"}

Dopo aver premuto il pulsante **INSIEMI** presente nella prima schermata l'utente visualizza la seguente schermata:



Operazioni possibili:

Min-Max: calcola il valore massimo e il valore minimo

Unione: unione tra due insiemi

Intersezione: intersezione tra due insiemi

-: differenza tra due insiemi

Le regole di sintassi per gli insiemi sono le seguenti:

- ogni elemento è preceduto e seguito da una virgola. Fanno eccezione il primo elemento, che non è preceduto da nessun carattere, e l'ultimo elemento, il quale è seguito da un punto e virgola.
- non va inserito nessuno spazio tra un elemento e il successivo

Esempio input corretto : 1,2,3;

Esempi di input errati:

- 1,2,3,2; non è un insieme
- 1,3 ogni input deve terminare con il ";"
- a,c,d; la calcolatrice accetta solamente i caratteri [0-9]{"", " ";"}

In entrambi i casi, per eseguire un'operazione dopo aver scritto l'input correttamente, si deve premere il tasto riferito all'operazione desiderata e il risultato comparirà nel primo *QPlainTextEdit* in caso di operazioni che coinvolgono una sola matrice o un solo insieme, mentre quelle che ne coinvolgono due nel terzo.

6 Ore di lavoro

• Analisi preliminare del problema:	3h		Prima consegna(rejected)
• Progettazione MODEL:	6h	+ 1h	Seconda consegna
• Progettazione VIEW:	2h	+ 1h	
• Apprendimento libreria Qt:	6h		
• Codifica MODEL:	20h	+ 11h	
• Codifica VIEW:	8h	+ 1h	
• Debugging e testing:	4h	+ 2h	
TOTALE	49h		

7 Ambiente di sviluppo

Sviluppato su:

- Qt Creator 4.6.2
- Based on Qt 5.10.1
- Windows 10 (10.0)

Testato su:

- Qt Creator 3.5.1
- Based on Qt 5.5.1
- Ubuntu 16 (16.04.2 LTS)
- dip154.studenti.math.unipd.it