Kalk

di Giacomo Corrò
1122451

1. DESCRIZIONE GENERALE

KALK è una calcolatrice in grado di effettuale le principali operazioni su polinomi e vettori (intesi come segmenti orientati).

La gerarchia delle classi è la seguente:

- la classe base BaseClass che ha come campo dati il nome dell'oggetto;
- la classe Polynom derivata pubblicamente da BaseClass, che rappresenta un polinomio attaverso un array dinamico di QPair, un template di classe costituito una coppia di elementi, che nel caso di KALK sono un double e un int, cioè coefficiente ed esponente di un monomio;
- la classe EuclideanVector derivata pubblicamente da BaseClass, che rappresenta un vettore attaverso due punti cartesiani, l'angolo e il modulo.

Ho scelto di implementare le operazioni sugli oggetti come metodi costanti, quindi senza fare side-effect sull'oggetto di invocazione e ritornando un nuovo oggetto come risultato (tranne che per l'operatore di divisione).

La classe KalkMemory si occupa della gestione della memoria. È composta da un vettore di oggetti BaseClass*, in cui l'utente può salvare eventuali i vari oggetti per future operazioni attraverso un'apposita linea di insetimento.

La classe Kalk si occupa della gestione dei dati inseriti attraverso l'intefaccia grafica. È composta da due puntatori ad oggetti BaseClass, in cui vengono memmorizzati temporanenamente il primo e secondo operando, e da un puntatore ad un oggetto KalkMemory. Per semplificare le operazioni ho scelto di inserire gli operandi in questa classe e non nella classe KalkMemory.

La classe KalkWidow si occupa dell'interfaccia grafica della calcolatrice, invocando le oppourtune chiamate ai metodi messi a disposizione dalla classe KalkMemory. La memoria è limitata a 26 oggetti, perché ogni oggetto in memoria è identificato da un nome composto da una sola lettera dell'alfabeto, questo per semplificare le operazioni. A memoria piena verrà sollevata un'eccezione e l'utente dovrà liberarla attraverso l'apposito pulsante.

L'idea iniziale era aveve due interfaccie grafiche separate per vettori e polinomi (e due "memorie") per poter anche avere un grafico dei vettori, ma questo avrebbe causato una duplicazione inutile di codice quindi ho deciso di raggruppare tutto in un'unica interfaccia. Ho dichiarato virtuali gli operatori aritmetici in modo da poter effettuare chiamate su puntatori polimorfi. Così facendo l'utente può lavorate contemporaneamente con polinomi e vettori. Ovviamente non ha senso eseguire operazioni tra un polinomio e un vettore: nel caso l'utente faccia ciò viene sollevata un'opportuna eccezione che segnala ciò.

2. CODICE C++

2.1 GERARCHIA DI ECCEZIONI

Per poter gestire in modo semplice e uniforme la segnalazione di errori ho definito classe astratta Exception con il metodo astratto “String toQString() const”, il cui scopo è ritornare una QString contenente la spiegazione dell’errore avvenuto.
Tutte le eccezioni concrete sono specializzazioni della classe Exception e forniscono un’implementazione del metodo “toQString”; questo permette di usare il solo sovratipo Exception nelle catch per il controllo delle eccezioni e la stampa del relativo errore attraverso una chiamata polimorfa al metodo toQString().
Per semplicità ho raggruppato tutte le eccezioni in un unico file.

2.2 GERARCHIA DELLE CLASSI

2.2.1 CLASSE BaseClass

È la classe base della gerarchia in cui una stringa identifica o un vettore o un polinomio.

Campi dati:

- Qstirng name: è il nome dell'oggetto.

Il nome è obbligatorio nel caso l'oggetto venisse salvato in memoria, in caso contrario può essere anche omesso.

La classe fornisce i seguenti metodi**:**

- BaseClass(QString): costuisce un oggetto BaseClass e inizializza il campo dati name;
- BaseClass(const QString&): è il costruttore di copia;
- virtual QString print() const: è la funzione di stampa;
- virtual ~BaseClass(): distruttore virtuale della classe;
- void setP(QPointF): setta il valore del campo dati name;
- QString getName(): ritorna il nome dell'oggetto;
- virtual BaseClass* operator+(const BaseClass& b) const: overloading operatore di somma;

- virtual BaseClass* operator-(const BaseClass& b) const: overloading operatore di differenza;
- virtual BaseClass* operator*(const BaseClass& b) const: : overloading operatore di moltiplicazione;
- virtual BaseClass* operator/(const BaseClass& b) : : overloading operatore di divisione;
- virtual BaseClass* operator=(const BaseClass& b) : : overloading operatore di assegnazione;

Ho scelto di rendere polimorfa questa classe in modo che l'invocazione dei memodi sugli oggetti sia dinamica, in particolare per quanto riguarda gli operatori aritmetici.

2.2.2 CLASSE EuclideanVector

Rappresenta un vettore, cioè un segmento orientato. Si può costruire a partire da angolo e modulo oppure con uno o due punti; il vettore sarà inserito in un sistema di assi cartesiani e avrà come punto di applicazione l'origine o il primo punto inserito. L'angolo ha come unità di misura i gradi.

Campi dati:

- p1 : punto di applicazione;
- p2 : secondo punto che identifica il vettore;
- lenght : lunghezza, o modulo, del vettore;
- angle : angolo tra il vettore e l'asse delle ascisse.

La classe fornisce i seguenti metodi:

- EuclideanVector(double l, double a, QpointF p, QString n): costruisce un vettore a partire da modulo, angolo e punto di applicazione;
- EuclideanVector(double l, double a, QString n): costruisce un vettore a partire da modulo e angolo;
- EuclideanVector(QString n, QpointF p2, QpointF p1): costruisce un vettore a partire da due punti;
- QString print() const: stampa su una stringa i campi dati dell'oggetto;
- double getL(): restituisce il modulo di un vettore;
- double getA(): restituisce l'angolo di un vettore;
- static double Angle(const EuclideanVector&,const EuclideanVector&) : calcola l'algolo compreso tra due vettori;
- EuclideanVector operator/(const BaseClass&): prodotto vettoriale tra due vettori, ritorna un vettore il cui angolo è 0 per semplicità (l'angolo dovrebbe essere perpendicolare al piano formato dai due vettori moltiplicati);
- EuclideanVector operator*(const BaseClass&) const : prodotto scalare tra due vettori;

- EuclideanVector operator+(const BaseClass&) const : overloading operatore di somma tra vettori;
- EuclideanVector operator-(const BaseClass&) const : overloading operatore di sottrazione tra vettori.

A causa dei problemi con le funzioni seno e coseno che, in presenza di particolari angoli in cui tali funzioni si annullano, restituivano valori errati dovuti all'aritmetica del calcolatore, i costruttori sono implementati in modo tale da eliminare parzialmente questi errori.

L'operatore di divisone tra vettori è stato implementato in modo che operi come il prodotto vettoriale, mentre l'operatore di moltiplicazione è il prodotto scalare che ritorna un vettore al posto di uno scalare per coerenza con il metodo ereditato dalla classe base.

2.2.3 CLASSE Polynom

Rappresenta un polinomio di qualunque grado nella variabile x. Viene costruito a partire da una stringa che viene poi trasformata in un vector di QPair attraverso il metodo statico QPair<double,int> converti(QString). Un oggetto QPair è formato da due valori che sono il coefficiente e l'esponente di un monomio; una volta creato il vector di QPair, viene ordinato in modo descrescente dal metodo void sort().

Campi dati:

- list : vector di Pair, i monomi che formano un polinomio.

La classe fornisce i seguenti metodi:

- void sort() : metodo privato che ordina "list" seconod potenze di x descrescenti;
- static QPair<double,int> converti(QString) : metodo privato, converte da QString a Qvector<QPair<double,int> >;
- Polynom(QString poly,QString name) : costruttore della classe Polynom, costruisce un oggetto Polynom a partire da una QString;
- Polynom(const Polynom&) : costruttore di copia;
- Polynom(QVector< QPair<double,int> > p,QString name): costruisce un oggetto Polynom attraverso un QVector
- Polynom() : costruttore di defaul;
- Polynom operator+(const BaseClass&&) const : overloading operatore di somma;
- Polynom operator-(const BaseClass&&) const : overloading operatore di differenza;
- Polynom operator*(const BaseClass&&) const : overloading operatore di moltiplicazione;
- Polynom operator/(const BaseClass&&) : overloading operatore di differenza;
- Polynom operator*(const QPair<double,int>& term) const : overloading operatore di moltiplicazione, utilizzato per semplifiacare le operazioni di differenza, moltiplicazione e divisione.
- QString print() const : stampa su una stringa i campi dati dell'oggetto.

Il costrutture Polynom(QString poly,QString name), prima di costruire il sottooggetto verifica, attraverso un'espressione regolare, che la stringa "poly" si valida, ovvero che ogni monomio sia scritto nella maniera corretta; in caso contrario distrugge l'oggetto e solleva un'eccezione.
Il metodo Polynom operator/(const Polynom&) è l'unico overloading di operatore non costante in quando fa side-effect sull'oggetto di invocazione: questo metodo ritorna un nuovo oggetto Polynom che rappresenta il quoziente, mentre l'oggetto di invocazione diventerà il resto della divisione.

2.3 LA CLASSE KALKMEMORY

Com già detto in precedenza, la classe KalkMemory si occupa della gestione della memoria.

Campi dati:

- memoria: un vector di BaseClass* in cui vengono salvati gli oggetti.

La classe fornisce i seguenti metodi:

- KalkMemory(): costruttore di default;
- ~KalkMemory(): distruttore;
- BaseClass* getLast() const: ritorna l'ultimo elemento in memoria;
- void insert(BaseClass *): inserisce un oggetto in memoria;
- BaseClass* search(QString b) const: cerca un oggetto con nome "b" in memoria, se non presente solleva un'eccezione.

2.4 LA CLASSE KALKMEMORY

Il modello, già descritto nell'introduzione, è rappresentato dalla classe Kalk.
La classe è implementata con un puntatore a un oggetto KalkMemory, una classe creata appositamente che si occupa di gestire la memoria, e si occupa di scatenare opportune eccezioni se si cerca di eseguire operazioni tra oggetti non presenti in memoria, oppure se il nome non è valido. Invece la classe Kalk mette a disposizione i metodi offerti dalla gerarchia Matrix e gestisce la condivisione di memoria.
Ho scelto di gestire la memoria con condivisione: il salvataggio in memoria non provoca una copia profonda, ma salva un puntatore allo stesso oggetto;

Questa classe gestisce l'interazione tra i dati inseriti dall'utente e la memoria, esegue le operazioni aritmetiche invocando i metodi delle classi KalkMemory, Polynom ed EuclideanVector. Inoltre questa classe mantiene salvati temporaneamente i due operandi delle operazioni che l'utente sta effettuando.

Campi dati:

- memoria: puntatore ad un oggetto KalkMemory in cui salvare i dati
- first: puntatore ad un oggetto BaseClass, è il primo operando delle operazioni
- second: puntatore ad un oggetto BaseClass, è il secondo operando delle operazioni;

2.5 CODICE GUI

La classe KalkWindow gestisce l'interfaccia grafica e contiene un puntatore a un oggetto Kalk che viene inizializzato a puntare a un nuovo oggetto alla creazione della classe “MainWindow". Siccome avevo la necessità si salvarmi temporanenamente il risultato delle operazioni, i metodi on_uguale_released() e on_esegui_released() contengono un puntatore ad un oggetto BaseClass che punta al risultato. Dato che ogni operazione crea un nuovo oggetto avevo la necessita di tenerne traccia e salvarlo in memoria il risultato mi è sembrato inutile. Al termine di ogni operazione l'oggetto "risultato" viene eliminato.

3 CODICE POLIMORFO

Il cuore del polimorfismo nel progetto risiede nella classe BaseClass: essa definisce i metodi virtuali print(), per la stampa degli oggetti, e gli operatori funzione e poi sfrutta invocazioni polimorfe a tutti questi metodi per implementare le altre funzionalità offerte.

Inoltre, il polimorfismo viene sfruttato nelle catch delle eccezioni: infatti viene catturato un riferimento alla classe base Exception e poi si ottiene la QString contenente la descizione dell’errore attraverso l’invocazione polimorfa del metodo “toQString()” sul riferimento.

4 DIFFERENZE IN JAVA

L'unica differenza con la parte C++ risiede negli operatori: non potendo fare l'overloading delle funzioni matematiche, la classe base dichiara dei metodi astratti che fungono da operatori; il medoto "dotP" (dot product, cioè il prodotto scalare tra vettori) è implementato solo nella classe EuclideanVector, mentre la classe Polynom ritorna il valore 0, che segnala l'impossibilità di eseguire tale operazione tra polinomi;

Il metodo public abstract BaseClass div(BaseClass b), invocato su oggetti Polynom, ritorna solamente il quoziente della divisione e non il resto, in quando non ho trovato un modo semplice ed efficace per ritornare anche il resto.

5 ORE IMPIEGATE

- Analisi preliminare del problema: 4
- Progettazione e Codifica della Gerarchia dei tipi in C++: 18
- Codifica della Gerarchia dei tipi in Java: 10
- Progettazione e Codifica del Modello (Kalk.h): 5
- Apprendimento della libreria Qt: 5
- Sviluppo GUI: 8
- Debugging: 10 ore

TOTALE: 60

Lo sforamento di 10 ore nelle 50 previste, è dovuto a problemi della codifica del codice Java (non disponendo delle librerie di Qt) e nella risoluzioni di alcune bug che ho riscontrato nell'utilizzo della calcolatrice. Alcuni bug (le parentesi non sono supportate, la priorita' delle operazioni non viene rispettata e l'esponente dei polinomi deve essere un numeno intero positivo) sono ancora irrisolti, in quanto non volevo sforare ultiriormente nelle ore previste.
La struttura principale della GUI è stata generata con QtDesign quindi la compilazione deve essere effettuata tramite il file “Progetto2.pro”.

6 AMBIENTE DI SVILUPPO

Sistema operativo di sviluppo: Ubuntu 16.04 LTS

Versione Qt: Qt Creator 4.7.0 Based on Qt 5.11.1
Compilatore: gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609

Per la parte in Java:
IDE: IntelliJ IDEA 2018.2.2 (Community Edition)
Build #IC-182.4129.33, built on August 21, 2018
JRE: 1.8.0_152-release-1248-b8 amd64
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o