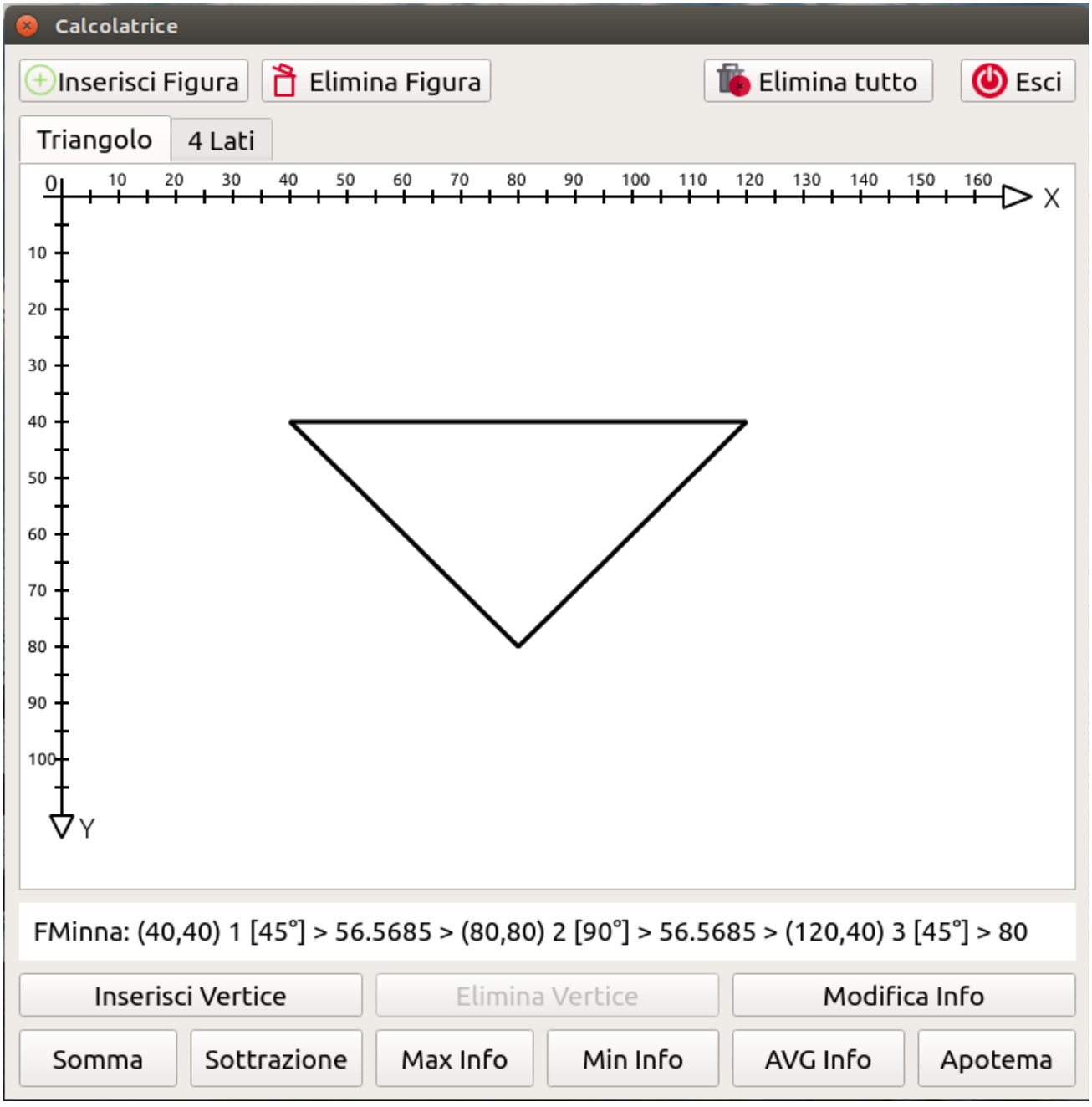
Università degli Studi di Padova ~ Dipartimento di Matematica

Progetto di Programmazione ad Oggetti

A.A. 2017/2018

Francesco Minna
1103073



Indice:

1) Scopo del progetto

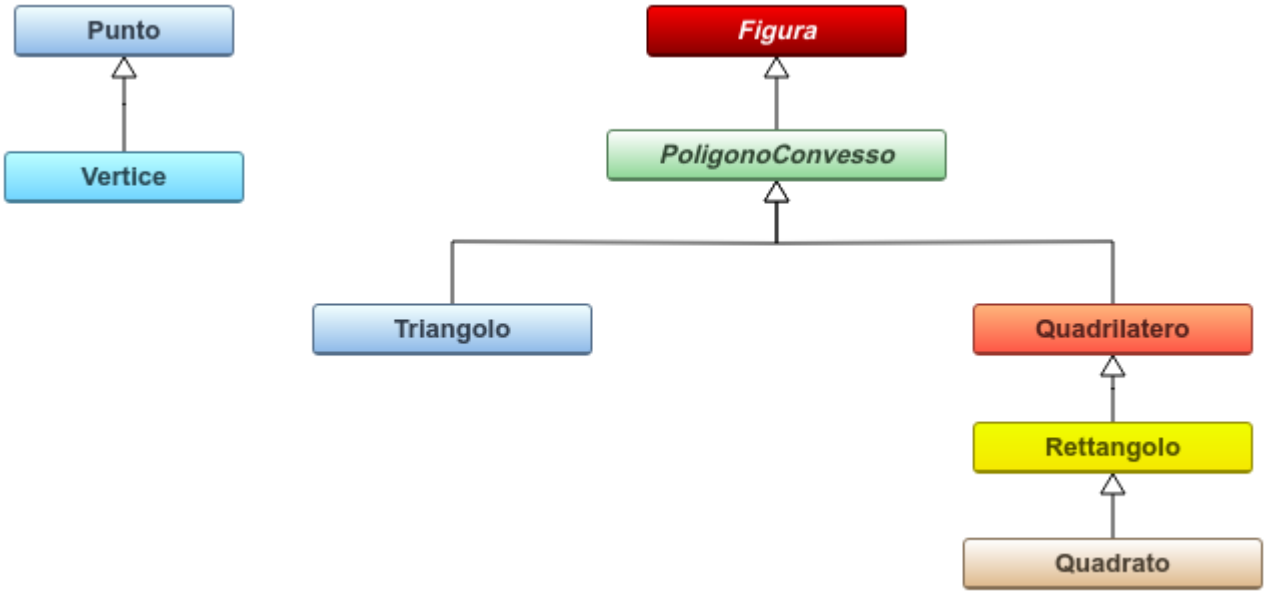
Questo progetto nasce dall'idea di creare una calcolatrice in C++/Qt che permetta di effettuare varie operazioni sulle figure geometriche.
Per ora la calcolatrice supporta solo i poligoni convessi, ed oltre alle classiche operazioni come il calcolo dell'area, del perimetro o dei gradi di un vertice, viene introdotta un informazione aggiuntiva: ogni vertice infatti contiene un campo dati informazione di natura numerica, ovvero un numero qualsiasi che l'utente sceglie di associare ad un determinato vertice, e che e' significativo per lui.

L'applicazione consente di:

- inserire una figura a scelta tra: triangolo, quadrilatero, rettangolo e quadrato, che verrà disegnata sul piano cartesiano della calcolatrice in base ai punti inseriti dall'utente.
- inserire o eliminare un vertice, in base al tipo di poligono, creando una nuova figura risultante.
- modificare il campo info associato ad un determinato vertice di un poligono.
- effettuare le operazioni di somma, sottrazione o restituire il massimo, il minimo o la media tra i campi info di un poligono.
- calcolare le diagonali o l'apotema per i poligoni per cui sono definiti.
- eliminare una o tutte le figure disegnate.

2) Descrizione della gerarchia e dei metodi virtuali

Di seguito vengono elencate le classi presenti nel progetto e le relative caratteristiche.



Punto: classe concreta che rappresenta un generico punto in un piano cartesiano a due dimensioni.
La classe Punto e' rappresentata da:

- Due campi dati privati double che rappresentano la coordinata x e la coordinata y.

La classe Punto dichiara il distruttore virtual e = default, e fornisce i seguenti metodi virtuali:

- Operatore di uguagliata e disuguaglianza.
- Metodo toString() che ritorna una std::string che rappresenta uno specifico punto.

Inoltre sono disponibili i seguenti metodi esterni alla classe:

- double distanza(p1, p2) const; che ritorna la distanza tra il punto p1 e il punto p2.
- double determinante(p1,p2) const; che calcola il determinante dei punti p1 e p2.

Vertice: classe concreta derivata da Punto che rappresenta un vertice di una figura.
La classe vertice e' rappresentata da un solo campo dati privato:

- Un informazione di tipo double, che rappresenta l'informazione numerica associata a quel vertice.

La classe Vertice ridefinisce i seguenti metodi:

- Metodo di uguaglianza e disuguaglianza, e metodo toString().

Inoltre e' disponibile il seguente metodo esterno alla classe:

- double AngoloVertici(v1,v2,v3) const; che calcola l'angolo del vertice v2, compreso tra v1 e v3.

Figura: classe base polimorfa astratta che rappresenta una qualsiasi figura geometrica.
Una figura, come per esempio un cerchio, un poligono convesso o non convesso, e' rappresentata solo dal campo dati protetto:

- Nome, di tipo QString

La classe, che definisce il distruttore virtuale = default, e' astratta in quanto prevede i seguenti metodi virtuali puri:

- metodo che calcola il perimetro: virtual double getPerimetro() const =0;
- metodo che calcola l'area: virtual double getArea() const =0;
- bool operator<(const Figura&) const =0; e operator>
- virtual std::string toString() const =0;
- virtual void print(std::ostream&) const = 0;

PoligonoConvesso: classe base polimorfa astratta, derivata da Figura, che rappresenta un generico poligono convesso, su cui poter effettuare le operazioni; funge da interfaccia comune per le classi derivate da essa.
Ogni poligono convesso e' rappresentato da:

- un array di Vertici: std::vector<Vertice>, che contiene i vertici di un poligono (campo dati protetto)

PoligonoConvesso e' astratta perché, oltre a non implementare i metodi astratti di figura, di cui e' sottoclasse, contiene i seguenti metodi virtuali puri:

- PoligonoConvesso* clone() metodo polimorfo di clonazione standard; e' l'unico metodo polimorfo presente nel progetto.

- double getSommaAngolo() che ritorna la somma degli angoli interni di un poligono
- unsigned int getNumLati() che ritorna il numero di lati di un poligono

PoligonoConvesso contiene molti metodi pubblici, tra cui:

- double getLato(double i, double p) const; che ritorna la lunghezza del lato tra il vertice con info=i e il vertice successivo; se i campi info sono tutti uguali o si vuole un vertice in una posizione specifica, si può passare la posizione desiderata come secondo parametro.
- double getAngolo(double, double)const; come il metodo precedente, ma ritorna l'angolo del vertice con info=i.
- void modificaInfo(unsigned int p, double i); che modifica il campo info del vertice in posizione p con il nuovo info i; se la posizione p non e' valida (per esempio p<0 o p maggiore del numero di vertici), allora il metodo solleva un'eccezione di tipo std::invalid_argument.
- PoligonoConvesso* inserisciVertice(QString s, Vertice v) const; che inserisce il vertice v nel poligono di invocazione creando un nuovo poligono convesso di nome s e lo ritorna; se il nuovo poligono non e' convesso, viene sollevata un'eccezione di tipo std::invalid_argument. Per come e' estesa ora la gerarchia si può inserire solo in un triangolo, per formare uno dei quadrilateri.
- PoligonoConvesso* eliminaVertice(QString s, unsigned int p) const; che elimina dal poligono di invocazione (almeno un quadrilatero) il vertice in posizione s, creando un nuovo poligono convesso di nome s e lo ritorna; viene sollevata un eccezione std::invalid_argument solo nel caso in cui la posizione fornita non e' valida (p<0 o maggiore del numero di vertici). Infatti, eliminando un vertice da un poligono convesso, il poligono risultante sarà sempre ancora convesso.
- static PoligonoConvesso* costruisci(QString s, vector<Vertice>); metodo statico privato (perche' e' gia' stato controllato che i vertici formano un poligono convesso) che ritorna il poligono di nome s rappresentato dal vector di vertici.
- Int WindingNumberInclusion(Vertice v) const; metodo che determina l'inclusione di un punto, il vertice v, nel poligono di invocazione: il punto si trova fuori dall'area del poligono solo se il winding number =0, altrimenti e' all'interno.

* *Fonte:* Winding number http://www.geomalgorithms.com/a03-_inclusion.html

Triangolo: classe concreta derivata da PoligonoConvesso che rappresenta un generico triangolo. La classe e' marcata final perché nella seguente gerarchia non può essere definita una classe derivata da Triangolo, che rappresenta già ogni tipo di triangolo a due dimensioni.

Ogni triangolo contiene solo un campo dati statico privato:

- static const unsigned int numLati che indica il numero di lati di ogni triangolo.

Triangolo e' concreta perché fa l'overriding dei seguenti metodi virtuali puri:

- Triangolo* clone() const; che e' un override del metodo clone di PoligonoConvesso: per ogni puntatore p a Triangolo, p→clone() ritorna un puntatore ad un oggetto PoligonoConvesso che e' una copia di p.
- getPerimetro() e getAree() che calcolano rispettivamente il perimetro e l'area di un triangolo generico.
- GetSommaAngoli() e getNumLati() che ritornano la somma degli angoli e il numero di lati di ogni triangolo.

- Operator < che ritorna true solo se l'area del triangolo di invocazione e' maggiore dell'area della figura passata come parametro. Viceversa per >.
- toString(); che ritorna un Triangolo rappresentato come una stringa.

Quadrilatero: classe concreta derivata da PoligonoConvesso che rappresenta un generico quadrilatero.
Quadrilatero contiene un unico campo dati protetto:
- numLati, di tipo static const unsigned int, che rappresenta il numero di lati.

Fa l'overriding di tutti i metodi virtuali di PoligonoConvesso:
- metodo clone() di clonazione standard.
- getPerimetro, getArea, getSommaAngoli e getNumLati.
- operator< e > e toString.

Inoltre fornisce il seguente metodo per calcolare le diagonali di qualsiasi quadrilatero:
- double getDiagonale(unsigned int p) const; che ritorna la lunghezza della diagonale tra il vertice in posizione p e il vertice in posizione p+2.

Rettangolo: classe concreta derivata da Quadrilatero che rappresenta un generico rettangolo; Rettangolo non contiene campi dati.
Fa l'overriding dei seguenti metodi ereditati:
- metodo di clonazione standard clone()
- getPerimetro() e getArea()

Mentre invece ridefinisce (overloading) i seguenti metodi:
- getAngolo(); che non contiene più il parametro posizione, perché ovviamente tutti gli angoli del rettangolo sono uguali. Lo stesso vale per la sottoclasse Quadrato.
- GetDiagonale(); come per il precedente, perché le due diagonali del rettangolo sono uguali.

Quadrato: classe concreta derivata da Rettangolo che rappresenta un generico quadrato; come un Rettangolo, Quadrato non contiene campi dati.
Anche questa classe e' marcata final perche', anche nel caso in cui venisse estesa la gerarchia dei quadrilateri, aggiungendo per esempio il Rombo o il Parallelogramma, tutte queste classi sarebbero superclassi di Quadrato.

Quadrato fa l'overriding dei seguenti metodi ereditati:
- metodo di clonazione standard clone()
- getPerimetro() e getArea()

Mentre ridefinisce i seguenti metodi:
- double getLato() const; che ritorna la lunghezza del lato del quadrato; senza parametro perché i lati sono tutti uguali.
- double getApotema() const; che ritorna la lunghezza del raggio della circonferenza inscritta all'interno del quadrato.

3) Manuale utente della GUI

L'interfaccia della calcolatrice e' molto intuitiva, ma alcune operazioni devono essere descritte con chiarezza:

- Innanzitutto, occorre inserire almeno una nuova figura, altrimenti tutte le operazioni sono disabilitate.

- Per **inserire una nuova figura** bisogna:
  - Inserire il nome: deve iniziare con una lettera, e può contenere numeri; lunghezza massima 14 caratteri.
  - Inserire il numero di lati: al momento possono essere o 3 o 4.
  - Inserire coordinate del primo vertice: per la coordinata x un numero tra 0 e 160 (compresi), di cui al massimo 6 cifre dopo la virgola. Analogamente per y, ma il range si abbassa a 0 e 105, sempre compresi.
  - Inserire l'informazione del primo vertice: numero positivo o negativo, di al massimo 7 cifre senza virgola, o 7 cifre intere + 3 cifre dopo la virgola.

  Una volta compilati tutti i campi correttamente, cliccando sulla freccia ('Avanti'), viene aggiungo il primo vertice, e *non e' più possibile modificare il nome e il numero di lati*. L'inserimento dei vertici successivi e' analogo, finché, una volta inseriti tutti, il bottone 'Avanti' viene disabilitato, e si può confermare l'inserimento.

Attenzione:

* Ogni figura deve avere un nome unico, diverso da tutte le altre.

* I vertici di ogni figura vanno inseriti in senso antiorario, altrimenti il controllo sulla convessita' fallisce. Esempio: i vertici (10,10) > (5,30) > (30,10) creano correttamente un triangolo; gli stessi vertici, in ordine inverso, falliscono.

- Con un **click** in una qualsiasi parte della calcolatrice, si deseleziona un eventuale selezionata figura, e si disattivano i bottoni non disponibili per le figure del piano corrente.
  Con un **doubleClick** nell'area di una figura, questa viene selezionata e stampata, ed e' possibile effettuare le operazioni disponibili su di essa.
- Per **inserire un vertice**, una volta selezionata una figura e cliccato su 'Inserisci Vertice', si apre una finestra identica a quella di 'Inserisci Figura': in questo caso pero' viene chiesto solo di inserire un nuovo nome per la nuova figura, e le coordinate e l'informazione del nuovo vertice.
  Una volta inserite le informazioni e cliccato su 'Avanti', si può confermare l'operazione.

Attenzione:

* Il nuovo vertice viene inserito tra l'ultimo e il primo vertice della figura corrente, quindi attenzione alla convessita'.

- Per **eliminare un vertice**, in modo simile all'inserimento, va inserito un nuovo nome per la nuova figura, e attraverso il pulsante 'Avanti' si puo' scorrere ordinatamente i vertici: una volta scelto quello da eliminare, basta premere 'Elimina', e verra' stampata la nuova figura risultante.
- Per **modificare l'informazione di un vertice** si apre il classico form, come per le altre operazioni: in questo caso, ovviamente, non e' possibile modificare nessun campo tranne il campo informazione di un vertice. E' possibile, come prima, scorrere ordinatamente i vertici della figura: una volta scelto e modificato l'info da cambiare, va premuto ancora 'Avanti' e poi 'Conferma'.
  Se si lascia il campo info vuoto, viene messo automaticamente a 0.
- La **Stampa** di un poligono non e' subito chiara a primo impatto, ma fornisce molte informazioni su di esso:

Nome: (x1,y1) ‘campo_info1’ [gradi_vertice1] > lunghezza_lato > (x2,y2) ‘info2’ ...

4) Ambiente di sviluppo

- OS: Ubuntu 17.04 – 64bit
- Compilatore GCC 6.3.0 (x86 64bit)
- Versione libreria Qt: 5.7.1 (qt5)

Il progetto ha richiesto un lavoro complessivo individuale di circa ~85 ore, di cui:

- 5 ore per la progettazione
- 35 ore per implementare il modello
- 35 ore per implementare la GUI
- 10 ore di testing

5) Materiale consegnato

La cartella consegnata contiene:

- Cartella contenente la versione C++ del progetto, con tutti i file .h, .cpp, il main, Calcolatrice.pro e un file icone.qrc per le icone.

* Attenzione: per generare il MakeFile va usato il file Calcolatrice.pro presente nella cartella.

- Cartella contenente i file .java della versione Java del progetto; tutti i file si trovano nel package di default, quindi non necessitano di istruzioni particolari di import per la compilazione.

Fine.