**BARBARA RATTO S4952202**

**1 Documentazione / Commenti**

**1.a Generare la documentazione usando Doxygen e verificare se la documentazione prodotta permette di orientarsi nella struttura del progetto software. Indicare i punti che potrebbero essere migliorati.**

***Class List*:**

**Classe polygon**

Sono presenti due documentazioni diverse per la classe **polygon**: *Polygon* e *polygon*.

* *Polygon*: manca un breve riassunto che descriva la classe, però è possibile visualizzare la documentazione senza problemi
* *polygon*: è presente il breve riassunto ma non è possibile vedere la documentazione

Quest’errore potrebbe essere dovuto al fatto che nel file polygon.cpp, alla riga 3 del codice, ci sia scritto:

@file rectangle.cpp

invece che

@file polygon.cpp

La descrizione della funzione GetArea è *GETFUNCTIONS*, sarebbe da migliorare, scrivendo *get the area of the object.*

**Classe rectangle**

In generale la documentazione è fatta in modo preciso e ordinato, è sempre presente una breve spiegazione del compito svolto da ogni funzione. Riporto però di seguito alcune imprecisioni.

Nella breve descrizione della classe c’è scritto ‘*the class rectangle is a poligonal figure with 4 sides and*…’; avrei scritto che la classe rectangle permette di lavorare con figure con 4 lati…

La breve spiegazione delle funzioni SetDim e GetDim è *SETFUNCIONS* e *GETFUNCIONS*, un po’ generica.

Avrei separato in modo più evidente le funzioni di tipo ‘virtual = 0’ dalle altre *access functions.*

Nell’ottica di una possibile aggiunta di altre funzioni virtual o virtual=0 penso sarebbe utile averle tutte insieme, per fare poi altri eventuali sottogruppi all’interno di questo blocco.

Per il resto la documentazione è stata fatta in modo adeguato.

**Classe rhombus**

Stesse considerazioni fatte per la classe rectangle.

**Classe isoTriangle**

Stesse considerazioni fatte per la classe rectangle.

***File List*:**

Sono presenti tutti i file, con la dovuta descrizione, eccetto *polygon.cpp* (legato al problema sopra riportato).

Anche la documentazione del *main* è chiara e sintetica.

**1.b Utilizzando la documentazione e i commenti inseriti nei file .ccp e .h verificare se il progetto software aderisce alle specifiche assegnate durante la prima settimana. Indicare le eventuali specifiche non rispettate.**

main

**Classe isoTriangle**

Le funzioni **Area** e **Perimeter**, virtualizzate a 0 in *polygon*, non svolgono propriamente il loro ruolo: calcolano il valore dell’area e del perimetro e lo restituiscono come dovuto, ma **non** **aggiornano** il valore delle variabili *area* e *perimeter*, ereditate dalla classe base polygon.

Inoltre non mi sembra che abbia molto senso mettere le funzioni **Area** e **Perimeter** di tipo virtual, perlomeno per le specifiche dell’esercitazione.

Sempre riguardo all’area e al perimetro, sono implementate anche le funzioni **GetArea** e **GetPerimeter**, **non necessarie** in quanto già implementate da polygon con la stessa funzionalità.

Infatti, analizzando il codice si può vedere che nelle funzioni GetArea e GetPerimeter implementate in polygon, il valore delle variabili area e perimeter viene correttamente aggiornato, ma pare che queste funzioni **non vengano mai chiamate**.

Sarebbe più corretto mettere di tipo ‘**const’** i parametri che prendono in ingresso le seguenti funzioni: la Init di copia, l’overload dell’operatore = e l’overload dell’operatore ==.

La funzione **Draw**, quando deve stampare i valori di area e perimetro, chiama le funzioni Area e Perimeter invece che prendere direttamente il valore dalle variabili ereditate da polygon (area e perimeter) che, come ho già scritto, non vengono mai aggiornate.

Per come è stata costruita la classe polygon, avrei fatto in modo che la funzione **Dump** della classe derivata chiamasse la funzione Dump della classe base, la quale stampa i valori di area e di perimetro. In questo modo si sfrutterebbe quello che la classe derivata ‘eredita’ da quella di base.

Per il resto il progetto software aderisce alle specifiche assegnate.

**Classe rectangle e rhombus**

Si verificano gli stessi problemi della classe isoTriangle.

Infine vorrei fare un **commento generale** sul programma.

Nonostante sia funzionante (come ho scritto nei punti successivi), trovo il main un po’ troppo **annidato** e **poco modulare**: spesso le stesse righe di codice sono ripetute 2 o 3 volte, e per questo esso risulta anche molto lungo e **ripetitivo**.

Ritengo che creando delle funzioni che raccolgano gruppi di istruzioni ricorrenti, si alleggerirebbe molto il main, diminuendo anche il rischio di incorrere in bug (quando per esempio si deve correggere un’istruzione che è presente più volte nel codice, lo si può fare una volta per tutte nella funzione che la include, evitando di dimenticarsi di ‘correggerla da altre parti’).

**1.c Verificare che il file README introduca correttamente lo scopo del progetto software e che dia sufficienti informazioni per un corretto uso dell’interfaccia a riga di comando. Indicare eventuali mancanze e/o possibili migliorie.**

Il file README è stato implementato **correttamente**, forse potrebbe essere migliorato descrivendo più nei dettagli il menù di interfaccia del programma.

**2 Compilazione e prima sessione di test**

**2.a Verificare se è possibile compilare il progetto**

Il progetto compila correttamente.

**2.b Test dell’interfaccia a riga di comando: l’interfaccia funziona correttamente? L’interfaccia è di facile utilizzo? Quali prove sono state eseguite per fare il test?**

L’interfaccia funziona correttamente, è **chiara** e **completa**. Sono presenti tutte le opzioni richieste. Di seguito riporto i test effettuati e le eventuali **problematiche**.

**Test effettuati:**

1. Creazione di un **rettangolo** e un **rombo**, utilizzo dell’opzione **Draw** ed **Exit**.

Tutto funziona secondo specifiche.

1. Creazione di un **triangolo isoscele**.

Viene richiesto se creare l’oggetto per default (opzione 1) o per inizializzazione (opzione 2) e se viene inserito dall’utente un carattere qualsiasi diverso da 1 o 2, l’oggetto viene correttamente **NON** creato, ma **non viene visualizzato alcun messaggio di errore**: si riparte dal menu iniziale (vale la stessa cosa per la costruzione di rettangolo e rombo).

1. Creazione di un rettangolo/rombo/triangolo di tipo ‘**Normal’** (cioè con l’inserimento del valore dei lati da parte dell’utente).

Una volta scelta questa opzione, se al posto di due valori numerici per i lati vengono inseriti due altri **caratteri qualsiasi**, **l’oggetto** in questione **viene costruito** per default, **senza** la visualizzazione di alcun **messaggio d’errore**.

1. All’apertura del programma, **inserendo un carattere** che non sia un numero, il programma stampa il menu di continuo interrompendo la possibilità di interagire con esso e quindi di utilizzarlo per il suo scopo. Sarebbe necessario eseguire un **controllo** sul **parametro inserito**.
2. Test per verificare l’overflow.

Provando a cambiare il valore di *SIZE* (numero massimo di oggetti nella lista) e mettendolo a 3, il numero di creazioni ancora disponibili viene **correttamente aggiornato**, però se l’utente, una volta che si trova di fronte al menù iniziale, inserisce nuovamente i numeri 1,2 o 3, gli viene **erroneamente** chiesto il tipo di costruzione da effettuare, e appare la scritta che riporta che la **costruzione per default** è andata a buon fine (anche se in realtà fortunatamente l’oggetto non è stato costruito, altrimenti sarebbe stata generata un’eccezione). Poi, come specificato nel loro codice, il programma termina correttamente. Bisognerebbe però far effettuare l’operazione di chiusura, o comunque stampare un messaggio di errore, **prima di dare la possibilità all’utente** di scegliere un altro oggetto da creare e di assegnare i valori dei lati.

**3 Seconda sessione di test**

**3.a Definire una procedura di test che permetta di identificare possibili bug nel codice prodotto per le singole classi e i singoli metodi di ogni classe**

(devo scrivere io il main)

**3.b Riportare i risultati del test**