UNIVERSITA' DEGLI STUDI DI PADOVA

PROGETTO DI PROGRAMMAZIONE AD OGGETTI 2016/2017

VIRUS_LAB

ALBERTO ROSSETTI
matricola:1050225

Sistema e Compilazione

Questo programma è stato creato utilizzando le librerie QT5.5.1 e compilato con gcc 5.4.0 attraverso l'IDE QTCreator 3.5.1 (OpenSource) sia come ambiente di sviluppo che come designer della GUI, su un ambiente Linux (distribuzione Ubuntu 16.04 LTS), per il progetto di programmazione ad oggetti.

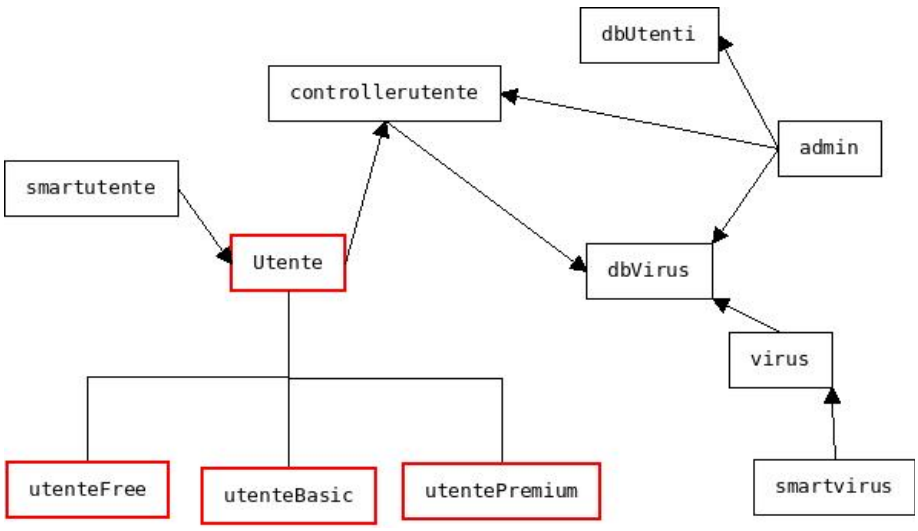
Per la compilazione nei computer del laboratorio è stato creato un file.pro specifico per qmake diverso da quello ottenibile tramite l'invocazione di qmake -project, che permette la generazione automatica tramite qmake del make file.
Quindi basta eseguire all'interno della cartella:
-qmake
-make

Il progetto compila ed esegue in locale e la compilazione è stata testata anche nei computer del laboratorio informatico del plesso Paolotti tramite tunnel ssh.
Il progetto contiene anche due file .txt dove sono contenuti dei dati prova, che riempiono i database e non devono essere modificati manualmente.

Abstract

Il progetto si basa sullo sviluppo di una applicazione che simula un *database di virus informatici* e include tutte le loro caratteristiche e attributi.
Lo scopo è quello di offrire ad un utente, registrato tramite un amministratore, un'ampia banca dati contenente i virus, dalla quale può accedere, tramite una ricerca, a dei risultati che gli verranno esposti a seconda del proprio account.
Esistono *tre tipi di account, Free,Basic e Premium. Un* account *Free* ha accesso a poche informazioni, in particolare riesce solo ad accertarsi dell'esistenza del virus ricercato, un account *Basic* riesce a vedere anche di che tipo di Virus si tratta e, infine, un utente *Premium* ha accesso a tutte le caratteristiche del virus, compresa una descrizione.
L'applicazione è gestita interamente da un amministratore che può inserire,modificare e rimuovere sia virus che utenti.

Gerarchia Di Classi



Come si può vedere dall'immagine, la gerarchia è stata creata a partire dalla classe base astratta *utente* che contiene un campo dati *username* , un campo dati *riferimenti* (per la gestione della memoria tramite gli smartpointer) e *un puntatore* protected alla classe controllerutente (classe intermedia tra il dbVirus e la classe utente), creata per separare il più possibile utente dal database dove sono presenti metodi riguardati la parte amministrativa.
La classe astratta pura utente definisce la funzione che filtra le ricerche in base ai privilegi dell'utente e che verrà poi implementata da classi concrete sue figlie. Le classi *utenteFree, utenteBasic ed utentePremium* ereditano dalla base astratta utente e definiscono quindi il metodo per filtrare la ricerca.
La gerarchia rispetta pienamente uno dei requisiti fondamentali di estensibilità ed evolvibilità del codice tramite polimorfismo. La gerarchia può essere poi ampliata con altri tipi di account e altre implementazioni di ricerche.

Descrizione del codice polimorfo

Le classi concrete all'interno della gerarchia chiamano una funzione ricerca all'interno della classe controllerutente che, a sua volta, chiama una funzione di ricerca all'interno del dbvirus, che ritorna una lista di risultati alla classe chiamante, la quale si preoccuperà solamente di ritornare il risultato all'overriding della funzione utentesearch.
Le classi figlie *utenteFree, utenteBasic ed utentePremium* ereditano dalla base astratta utente e definiscono quindi il metodo per filtrare la ricerca, che utilizza una classe annidata ad utente *SearchFunctor*. Questa, una volta ricevuta la ricerca generale effettuata nel db, restituirà i risultati a seconda del tipo di account che ha richiesto la ricerca.
Nella sezione utente della GUI, l'utente si preoccuperà solamente di inserire il proprio username: dopo aver effettuato un controllo all'interno del contenitore del dbutenti, per capire di che tipo di utente si tratta, partirà la chiamata polimorfa alla funzione utenteSearch che inizierà la ricerca dei virus.
Per riassumere, una chiamata della funzione utenteSearch, da parte di un puntatore polimorfo utente, eseguirà l'overriding della funzione appartenente alla classe figlia, che corrisponde al tipo dinamico del puntatore polimorfo. La funzione riceverà la ricerca basata sul nome del virus e la filtrerà tramite un funtore.

Principali scelte progettuali

-Puntatori

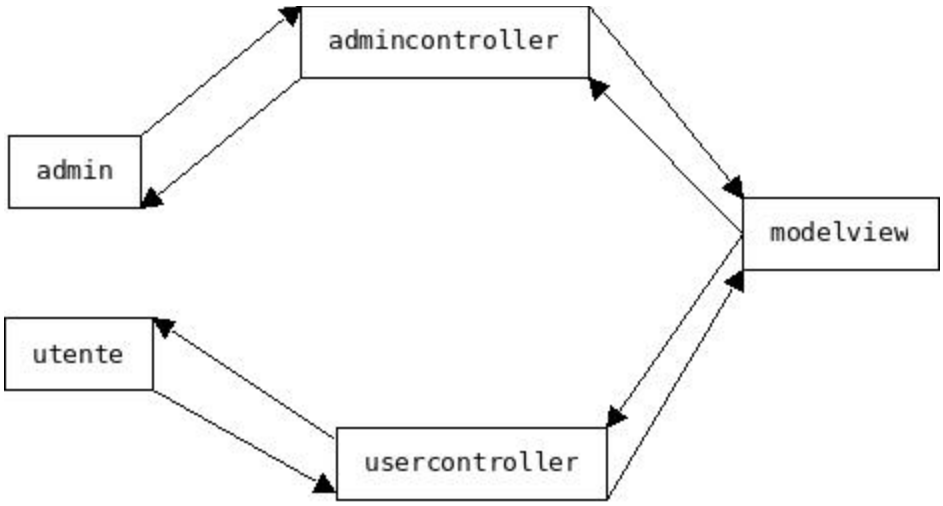
Ho deciso di implementare degli smartpointer attraverso due classi, smartvirus e smartutente, che hanno come campo dati un puntatore al corrispondente oggetto puntato. Inoltre ho inserito, come campo dati della classe utente e virus, un contatore "riferimenti" che permette l'autodistruzione dell'oggetto quando non è più puntato da nulla: questo evita garbage nello heap.
La gestione smart del campo dati riferimenti è assegnata alla ridefinizione degli operatori delle classi smartutente/virus.

-Contenitori

Per memorizzare gli utenti e i virus, ho deciso di dividere il database in due classi (dbVirus e dbUtenti) contenenti due liste della libreria std, riempite da oggetti smartutente e smartvirus.
La separazione è stata adottata per dividere al meglio la parte amministrativa da quella user.
La scelta dei contenitori std::list è dovuta al fatto che, per rimozioni, le liste sono molto più efficienti dei vector in quanto evitano il compattamento del contenitore dopo una remove.

-Modello MVC

Per implementare l'interfaccia utente, ho cercato di utilizzare il design pattern MVC provando ad ottenere una massima separazione tra GUI e parte logica.
Il pannello utente e i due pannelli admin(uno per la gestione virus e uno per quella degli utenti) hanno entrambi tre classi: una classe View, dove definisco e disegno tutti gli oggetti visibili a video, una classe Model, dove definisco i metodi per interagire direttamente con la parte logica del programma, e una classe Control, che provvede a catturare gli eventi generati dall'utente attraverso la GUI e a interagire con la classe Model per produrre risultati. La classe Control ha il compito di convertire i tipi QT in tipi propri del c++.

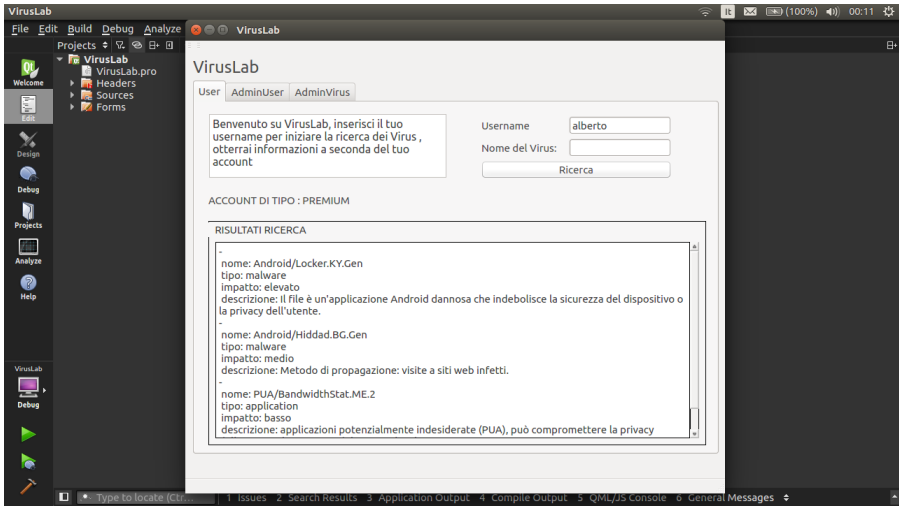


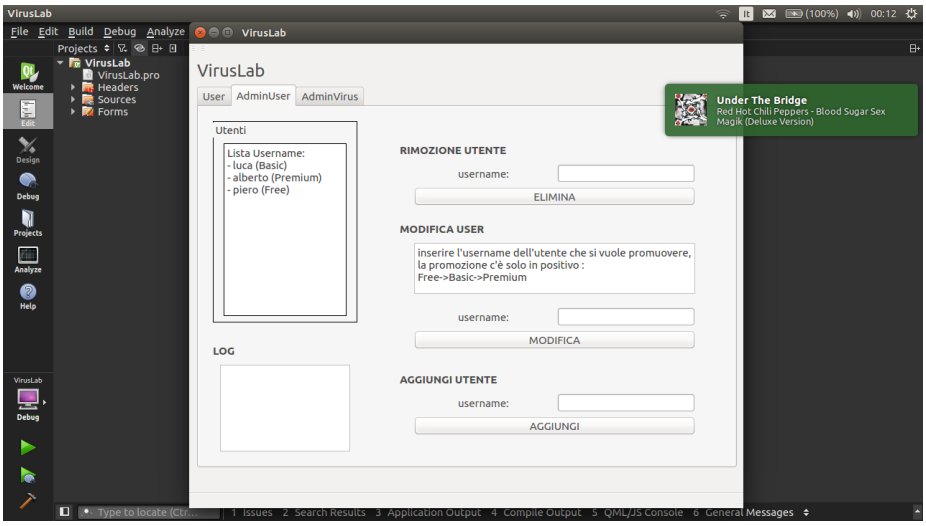
-Salvataggio File

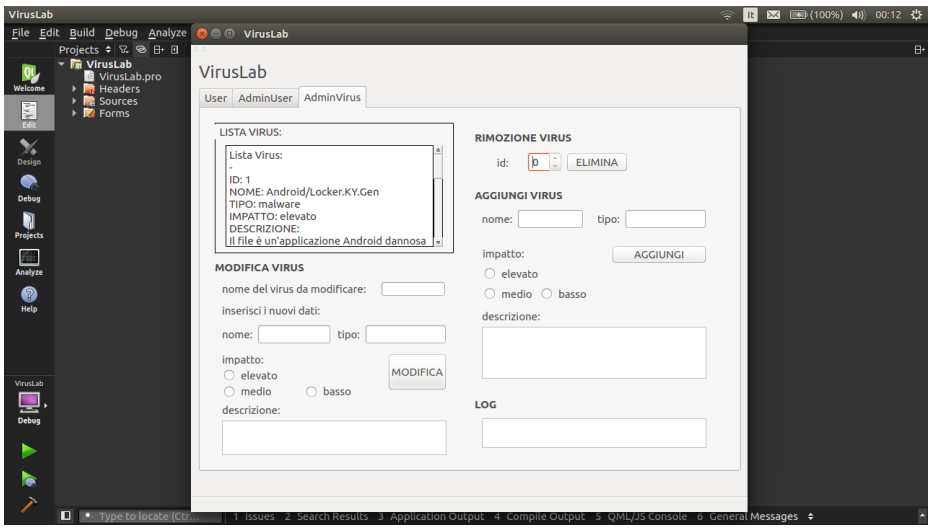
Per salvare gli oggetti presenti all'interno dei contenitori std::list e consentire lettura da file a memoria e scrittura da memoria su file dei contenitori, ho scelto di usare dei semplici .txt e creare dei metodi all'interno delle due classi db di caricamento e salvataggio nei file.
Load e save leggono e scrivono i file dbUtenti.txt e dbVirus.txt, separando i campi con delle relative sentinelle.
dbVirus ha come sentinella finale un "999" che deve sempre esserci anche a file vuoto.
dbUtenti ha come sentinella finale un "XX" che deve sempre esserci anche a file vuoto.

-GUI

Realizzata totalmente con il designer di QtCreator.
L'interfaccia utente consiste in una finestra principale nella quale è possibile, attraverso un menù a tendina contenente tre schede, interagire come amministratore (per modificare, aggiungere ed eliminare virus o utenti) oppure accedere all'area user che consente solo ricerche di virus dopo aver inserito l'username.
La ricerca ci darà informazioni sui virus a seconda del tipo di account appartenente all'username.
Non inserendo niente nella casella di ricerca, saranno stampati tutti i virus.







Come si può notare dalle immagini, nelle due sezioni di amministrazione sono presenti due box identificate come “LOG”. Queste box stamperanno dei messaggi di buona riuscita o meno dopo aver effettuato un’azione di modifica,aggiunta o eliminazione di virus o utenti.

*O*re complessive di lavoro:

Il progetto, compreso di progettazione modello logico e GUI, codifica modello e GUI, debuggin e test, ha richiesto circa 70/80 ore di lavoro, probabilmente perché ho iniziato subito dopo la consegna del 12/12/2016, quando ancora non avevo compreso benissimo molti concetti del corso.
La progettazione della parte logica è stata utile per applicare e capire i concetti spiegati a lezione, quindi alcune ore del progetto sono state usate anche come esercizio e studio.
Per l’imminente consegna, non mi sono soffermato troppo sulla GUI che, per essere implementata nel dettaglio, mi avrebbe richiesto più tempo di quello rimasto a disposizione.

Esempi di username o virus da ricercare:

Utenti:

username: piero (Free)
username: luca (Basic)
username: alberto (Premium)

Virus:

-Android/Locker.KY.Gen
-Android/Hiddad.BG.Gen
-PUA/BandwidthStat.ME.2

Test laboratorio