Università degli Studi di Padova ~ Dipartimento di Matematica

Progetto di Programmazione ad Oggetti

A.A. 2015/2016

Daniel De Gaspari, matricola n° 1078058

Nome progetto: Qultura

Indice:

1) Scopo del progetto:

Il progetto Qultura nasce dall'idea di realizzare un'applicazione in C++/Qt per la gestione minimale dei dati riguardanti le vendite di un'editoria/negozio che tratta libri e riviste, mediante utilizzo di interfaccia utente grafica.

L'applicazione consente di:

- creare nuovi fogli di lavoro, i quali conterranno le varie informazioni relative ai prodotti venduti;
- modificare fogli di lavoro esistenti, ovvero consente di inserire nuovi dati, e di modificare/eliminare i dati presenti in memoria;
- archiviare e recuperare fogli di lavoro già esistenti;
- visualizzare grafici relativi alle vendite dei prodotti, raggruppati per categoria di prodotto; sono presenti due differenti tipologie di grafico : istogramma e grafico a linee.

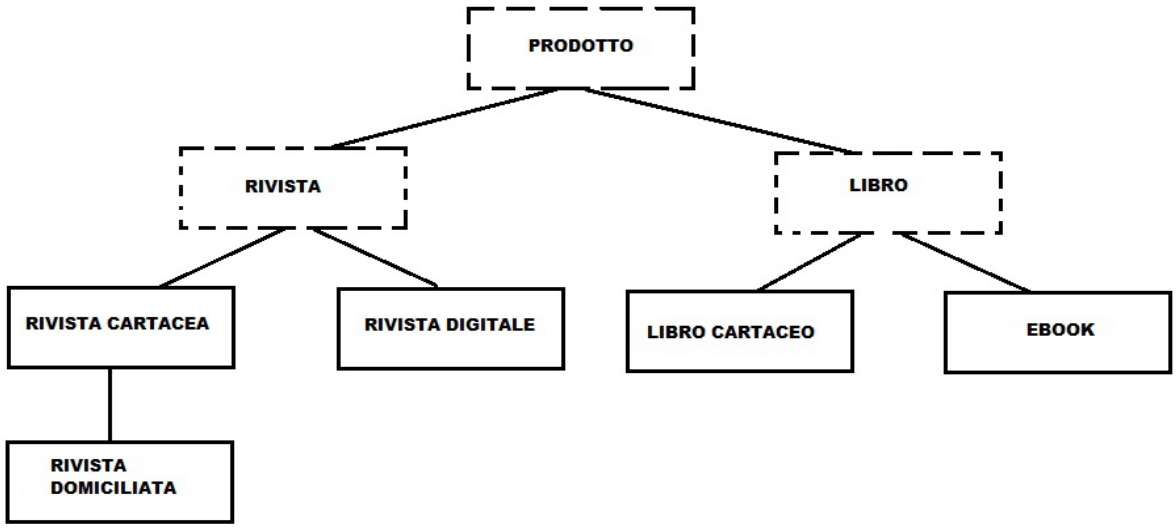
2) Ambiente di sviluppo:

- Sistema operativo: Ubuntu 15.10 32-bit
- Compilatore: GCC 5.2.1
- Versione libreria Qt : 5.3

- Sistema operativo: Windows 10 Pro 32-bit
- Compilatore: MinGW 4.8.2 (32-bit)
- Versione libreria Qt: 5.3

3) Descrizione della gerarchia:

Di seguito, verranno elencate le classi presenti nel progetto, e le relative caratteristiche di ognuna di esse.



- **Prodotto**: E' la classe base polimorfa astratta, che funge da interfaccia comune per le classi derivate da essa. Gli oggetti “Prodotto” rappresentano un generico prodotto vendibile nel negozio. Ogni prodotto, è caratterizzato da :
  - Titolo
  - Editore
  - Data di pubblicazione

  La classe è astratta, in quanto prevede i seguenti metodi virtuali puri:
  - Un metodo di “clonazione”: Prodotto* clone()
  - Un metodo bool isEqual(const Prodotto&)
  - Un metodo double prezzo();

- **Rivista**: E' una classe astratta , derivata da Prodotto, i cui oggetti rappresentano una generica rivista vendibile nel negozio. Ogni Rivista, è caratterizzata da :
  - Categoria : es: film, informatica, sportiva, ecc

  La classe è astratta, in quanto prevede i seguenti metodi virtuali puri, ereditati da Prodotto:
  - Un metodo di “clonazione”: Rivista* clone()
  - Un metodo bool isEqual(const Prodotto&)
  - Un metodo double prezzo();

- **Rivista Cartacea** : E' una classe concreta , derivata da Rivista, i cui oggetti rappresentano una rivista cartacea acquistabile direttamente nel negozio. Ogni rivista cartacea, è caratterizzata da :
  - Numero pagine
  - Costo per pagina, ovvero il costo richiesto per la stampa di una singola pagina

  La classe implementa i metodi virtuali puri di Rivista :
  - per ogni puntatore p a Rivista Cartacea, p->clone() ritorna un puntatore ad un oggetto Rivista Cartacea che è una copia di *p;
  - per ogni puntatore p a Rivista Cartacea, p->isEqual(const Prodotto& q) ritorna true solamente se i campi dati di *p sono tutti contenuti nel prodotto q, ed hanno gli stessi valori per i campi dati. [Questo metodo, in questa e nelle altre classi concrete appartenenti alla gerarchia, non controlla quindi se i prodotti sono esattamente uguali; quest'operazione e' lasciata all' operator== ridefinito; per maggiori informazioni su quanto detto, vedere pagina 6, primo punto.]
  - per ogni puntatore p a Rivista Cartacea, p->prezzo() ritorna il prezzo di vendita di *p , calcolato come segue: Costo per pagina * Numero pagine , con arrotondamento alla seconda cifra decimale;

- **Rivista Domiciliata**: E' una classe concreta , derivata da Rivista Cartacea, i cui oggetti rappresentano una rivista cartacea domiciliata , ovvero una rivista cartacea che arriva direttamente a casa, comportando così un aumento del prezzo a causa delle spese di spedizione. Ogni rivista domiciliata, è caratterizzata da :
  - Maggiorazione, ovvero il costo aggiuntivo richiesto per la spedizione. E' stato scelto di non dichiarare questo campo dati come statico, in considerazione del fatto che la spesa per l'invio di un prodotto varia a seconda della distanza.

  La classe implementa i metodi virtuali puri di Rivista :
  - per ogni puntatore p a Rivista Domiciliata, p->clone() ritorna un puntatore ad un oggetto Rivista Domiciliata che è una copia di *p;
  - per ogni puntatore p a Rivista Domiciliata, p->isEqual(const Prodotto& q) ritorna true solamente se i campi dati di *p sono tutti contenuti nel prodotto q, ed hanno gli stessi valori per i campi dati;
  - per ogni puntatore p a Rivista Domiciliata, p->prezzo() ritorna il prezzo di vendita di *p , calcolato come segue: prezzo della rivista cartacea (= Costo per pagina * Numero pagine) + maggiorazione , con arrotondamento alla seconda cifra decimale;

- **Rivista Digitale**: E' una classe concreta , derivata da Rivista, i cui oggetti rappresentano una rivista digitale , ovvero una rivista in formato digitale. Ogni rivista digitale, è caratterizzata da :
  - Costo per KB , ovvero il costo per ogni singolo KB di dati inviato
  - Dimensione in KB della rivista digitale

  La classe implementa i metodi virtuali puri di Rivista :
  - per ogni puntatore p a Rivista Digitale, p->clone() ritorna un puntatore ad un oggetto Rivista Digitale che è una copia di *p;
  - per ogni puntatore p a Rivista Digitale, p->isEqual(const Prodotto& q) ritorna true solamente se i campi dati di *p sono tutti contenuti nel prodotto q, ed hanno gli stessi valori per i campi dati;
  - per ogni puntatore p a Rivista Digitale, p->prezzo() ritorna il prezzo di vendita di *p , calcolato come segue: Costo per KB * Dimensione , con arrotondamento alla seconda cifra decimale;

- **Libro**: E' una classe astratta , derivata da Prodotto, i cui oggetti rappresentano un generico libro vendibile nel negozio. Ogni Libro, è caratterizzato da :
  - Autore
  - ISBN , ovvero una stringa che rappresenta univocamente una specifica edizione di un libro
  - Edizione
  - Genere
  - Numero pagine

  La classe è astratta, in quanto prevede i seguenti metodi virtuali puri, ereditati da Prodotto:
  - Un metodo di “clonazione”: Libro* clone()
  - Un metodo bool isEqual(const Prodotto&)
  - Un metodo double prezzo();

- **Libro Cartaceo**: E' una classe concreta , derivata da Libro, i cui oggetti rappresentano un libro cartaceo, ovvero un libro acquistabile direttamente nel negozio. Ogni libro cartaceo, è caratterizzato da :
  - Costo per pagina, ovvero il costo richiesto per la stampa di una singola pagina

  La classe implementa i metodi virtuali puri di Libro :
  - per ogni puntatore p a Libro Cartaceo, p->clone() ritorna un puntatore ad un oggetto Libro cartaceo che è una copia di *p;
  - per ogni puntatore p a Libro Cartaceo, p->isEqual(const Prodotto& q) ritorna true solamente se i campi dati di *p sono tutti contenuti nel prodotto q, ed hanno gli stessi valori per i campi dati;
  - per ogni puntatore p a Libro Cartaceo, p->prezzo() ritorna il prezzo di vendita di *p , calcolato come segue: Numero Pagine * Costo per pagina, con arrotondamento alla seconda cifra decimale;

- **Ebook**: E' una classe concreta , derivata da Libro, i cui oggetti rappresentano un Ebook , ovvero un libro in formato digitale. Ogni ebook, è caratterizzata da :
  - Costo per KB , ovvero il costo per ogni singolo KB di dati inviato
  - Dimensione in KB dell' ebook

  La classe implementa i metodi virtuali puri di Libro :
  - per ogni puntatore p a Ebook, p->clone() ritorna un puntatore ad un oggetto Ebook che è una copia di *p;
  - per ogni puntatore p a Ebook , p->isEqual(const Prodotto& q) ritorna true solamente se i campi dati di *p sono tutti contenuti nel prodotto q, ed hanno gli stessi valori per i campi dati.;
  - per ogni puntatore p a Ebook, p->prezzo() ritorna il prezzo di vendita di *p , calcolato come segue: Costo per KB * Dimensione , con arrotondamento alla seconda cifra decimale;

E' inoltre presente un'ulteriore gerarchia, relativa ai grafici:

- **Grafico**: E' la classe base polimorfa astratta, che funge da interfaccia comune per le classi derivate da essa. Gli oggetti “grafico” rappresento un generico grafico. Ogni grafico e' caratterizzato da :
  - Origine, ovvero un riferimento costante ad un QpointF, che indica le coordinate relative all'origine, necessario per posizionare correttamente il grafico tracciato relativamente agli assi.
  - DG (= DatiGrafico), un puntatore costante ad un oggetto ti tipo DatiGrafico, classe creata appositamente per gestire separatamente i dati del grafico.
  - Punti, è un Qvector costante, contenente puntatori costanti a QpointF : viene inizializzato grazie ad un metodo presente nella classe DatiGrafico, che come detto prima gestisce i dati del grafico, il quale restituisce un Qvector<const QpointF*> a partire dai dati del grafico. Punti rappresenta dunque i punti che verranno disegnati, relativi al grafico.

Nota: è stato scelto di inserire i campi dati di grafico nella parte Protected della classe, in quanto utili solamente alle classi (concrete) derivate da grafico, le quali acquisiranno dunque i dati e tracceranno i vari punti. L'alternativa sarebbe stata quella di esporre i campi dati alla parte pubblica, o inserendoli direttamente nella parte pubblica della classe, oppure esponendo i vari campi dati , che sarebbero stati quindi privati, mediante dei metodi get per tutti i campi dati. E' stato scelto di inserirli nella parte protetta in quanto i campi dati hanno senso solamente all'interno della classe, e sono pertanto inaccessibili alle classi esterne a Grafico.

La classe è astratta, in quanto prevede il seguente metodo virtuale puro:
- Un metodo void disegna(QPainter& painter)

- **GLinea**: E' una classe concreta , derivata da Grafico, i cui oggetti rappresentano dei grafici linea. La classe, si occupa solamente dell' implementazione del metodo virtuale puro di Grafico:
  - per ogni puntatore p a Grafico Linea, p->disegna(painter) si preoccupa di tracciare il grafico linea, con i relativi valori di riferimento per l'asse delle ascisse e delle ordinate, servendosi dei dati di “Punti”.

- **GIstogramma**: E' una classe concreta , derivata da Grafico, i cui oggetti rappresentano degli istogrammi. La classe, si occupa solamente dell' implementazione del metodo virtuale puro di Grafico:
  - per ogni puntatore p a Grafico Linea, p->disegna(painter) si preoccupa di tracciare l'istogramma, con i relativi valori di riferimento per l'asse delle ascisse e delle ordinate, servendosi dei dati di “Punti”.

4) Descrizione dell'uso di codice polimorfo

Gerarchia Prodotto:

- Nella classe “Ordine”, e' presente l'utilizzo del metodo virtuale clone() nel metodo “allunga” e nel metodo statico “copia” : in entrambi i casi, grazie al polimorfismo, e' possibile effettuare la copia profonda di un prodotto, ricorrendo al late binding. Verrà quindi invocato il metodo clone() sulla base del TD di Prodotto*.
- Nella classe “Dati Grafico”, si ricorre all'utilizzo del metodo virtuale prezzo() nell'inizializzazione di “value”, un Qvector di double che serve a contenere la somma dei prezzi dei singoli prodotti, raggruppati per categoria di appartenenza.
  [Nota: Nell'inizializzazione di “value”, è stato comunque necessario ricorrere all'utilizzo di RTTI, e in particolare all'operatore typeid, per determinare la categoria di appartenenza del prodotto, in modo tale da rispettare la corretta rappresentazione dei dati nel Qvector, ovvero per il raggruppamento per categoria.]
- Nella classe “Ordine”, viene utilizzato virtual bool operator!= tra prodotti. Questo serve per determinare se due oggetti di tipo Ordine sono uguali. Questa funzionalità viene utilizzata nella mainwindow, nel metodo “checkModifiche()”, per controllare se il file aperto ha subito modifiche o meno. Grazie alla disuguaglianza tra prodotti, e' possibile stabilire se due prodotti sono uguali o meno, ricorrendo al late binding.

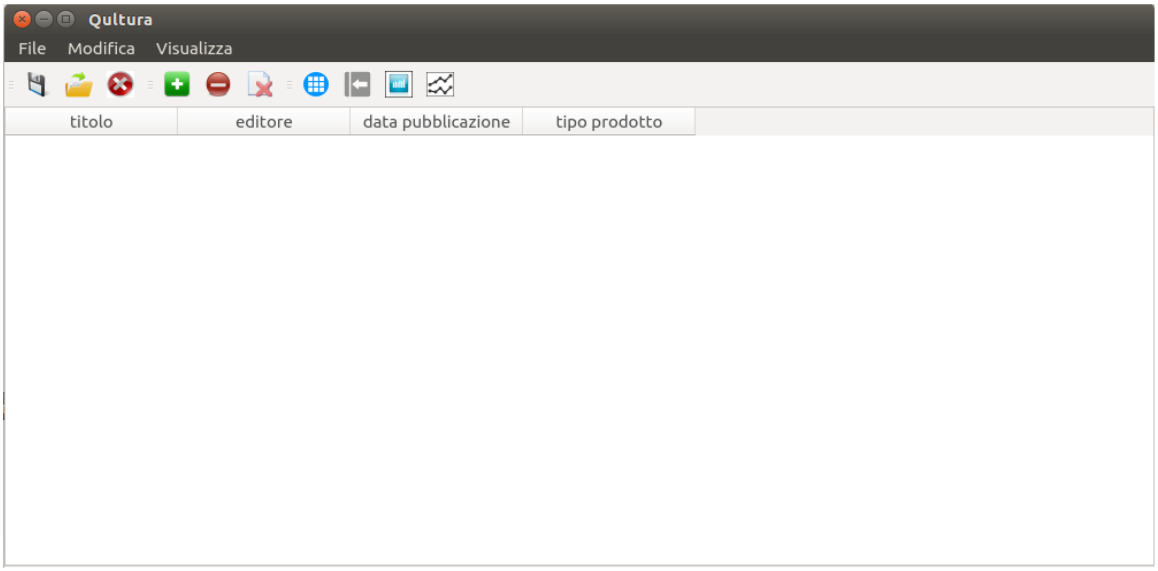
- Nelle classi concrete, derivate da Prodotto, viene utilizzato il metodo virtuale "isEqual(const Prodotto& q)" nelle definizioni degli operator== delle rispettive classi. Tale metodo, viene utilizzato per stabilire se i campi dati dell'oggetto di invocazione sono tutti contenuti nel prodotto q, ed hanno gli stessi valori per i campi dati.
  Ciò non è sufficiente però per dire che due prodotti sono esattamente uguali, dal momento che il prodotto q potrebbe avere ulteriori campi dati, oltre a quelli presenti nell'oggetto di invocazione. Il caso in questione potrebbe accadere ad esempio tra due Prodotti, p e q, appartenenti rispettivamente a Rivista Cartacea e Rivista Domiciliata.
  La definizione di operator== tiene conto di questo caso limite, controllando per prima cosa che i due oggetti siano esattamente dello stesso tipo, e successivamente, grazie a "isEqual", controlla se hanno anche gli stesso valori per i campi dati.
- Infine, ma non meno importante, e' presente un distruttore virtuale. Grazie a ciò, alla distruzione di un prodotto, viene invocato il distruttore standard della classe concreta del prodotto specifico, senza così lasciare garbage nella memoria.

Gerarchia Grafico:

- Come per la gerarchia precedente, anche in questa e' presente un distruttore virtuale, in modo tale da ripulire per bene la memoria senza lasciare "spazzatura" alla distruzione di un Grafico.
- Nella classe "FormGrafico", e in particolare nella ridefinizione del metodo "paintEvent" , l'oggetto di invocazione "graph", che avrà TS = Grafico, e TD = il tipo relativo al grafico che si intende rappresentare, invocherà il metodo "disegna(painter)"; così facendo, verrà disegnato il grafico della tipologia corretta, sulla base del TD di graph.

5) Manuale utente della GUI:

La GUI dell'applicativo è stato pensato per essere il più semplice e user-friendly possibile:
Ecco come si presenta all'avvio:



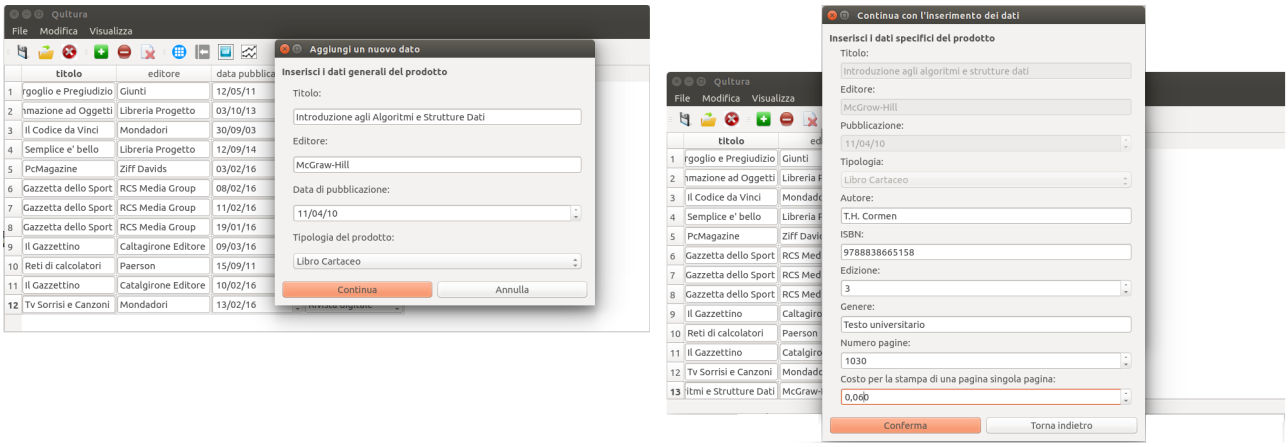
Dal menù, sono possibili le seguenti operazioni:
File:

- Apri: Per aprire un foglio di lavoro esistente; (se è già presente un file aperto, prima che questo venga chiuso, viene controllato se il file aperto ha subito modifiche, e in caso positivo, viene data l'opportunità di salvare le modifiche, prima di perdere tutte le modifiche apportate).
- Salva: per archiviare un foglio di lavoro aperto; se il file aperto esiste è già presente in memoria, la versione vecchia viene sovrascritta. Se invece il file non esiste, ne viene richiesta la creazione.
- Salva con nome: serve a creare un file nuovo, o salvare il file aperto con un nuovo nome.
- Chiudi file corrente: chiude il file aperto, se presente. (viene controllato se il file aperto ha subito modifiche, e in caso positivo, viene data l'opportunità di salvare le modifiche, prima di perdere tutte le modifiche apportate).
- Esci: Chiude l'applicazione. (viene controllato se il file aperto ha subito modifiche, e in caso positivo, viene data l'opportunità di salvare le modifiche, prima di perdere tutte le modifiche apportate).

Modifica:

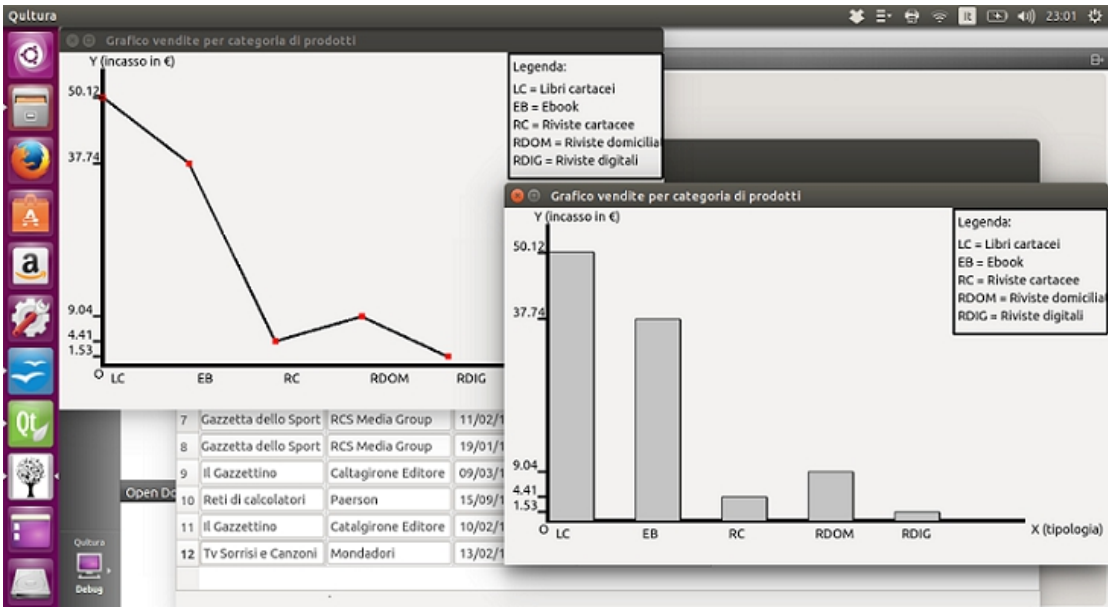
- Aggiungi elemento: Aggiunge un nuovo prodotto.
- Rimuovi elemento: Rimuove l'elemento relativo alla riga selezionata. Di default, viene altrimenti rimossa l'ultima riga.
- Rimuovi tutti gli elementi: Pulisce il foglio di lavoro, eliminando tutti i prodotti presenti.

Form per l'inserimento di un nuovo prodotto:



Visualizza:

- Mostra informazioni complete: Di default, vengono visualizzate solamente le informazioni principali dei prodotti, in comune a tutti, ovvero: Titolo, Editore, Data di Pubblicazione, Tipologia del prodotto. Questa opzione permette di visualizzare la tabella completa, con tutti i dati specifici dei singoli prodotti.
- Nascondi informazioni complete: Vengono visualizzate solamente le informazioni principali dei prodotti.
- Visualizza istogramma: Mostra il grafico della tipologia “istogramma”, contenente gli incassi pervenuti dalla vendita dei prodotti, suddivisi per categoria di prodotto (il grafico si adatta alle dimensioni della finestra).
- Visualizza grafico linea: Mostra il grafico della tipologia “grafico linea”, contenente gli incassi pervenuti dalla vendita dei prodotti, suddivisi per categoria di prodotto (il grafico si adatta alle dimensioni della finestra) .



6) Materiale consegnato:
   - File sorgente .h e .cpp necessari per la compilazione del progetto, oltre ad un file .pro
   - relazione.pdf
   - esempio.xml , contenente un elenco di dati di prova
   - immagini.qrc , necessario per la visualizzazione delle immagini/loghi presenti nell'applicativo, memorizzate all'interno della cartella “images”.

Fine.