Cognome e Nome: Porto Francesco
Matricola: 816042
Mail: f.porto2@campus.unimib.it

Relazione Progetto Programmazione e Amministrazione di Sistema

Parte C++

Il progetto prevede la creazione di una classe templata che implementa un Set, ovvero un insieme in senso matematico. Oltre al file **set.h** (contenente la classe templata), **setexception.h**, **setexception.cpp** (contenenti il tipo custom di eccezioni set_exception) e **main.cpp** (contenente i vari test), troverà anche i file voce.h e voce.cpp (utilizzati per testare la classe templata con tipi non standard – da ignorare ai fini della correzione).

- La classe set è templata sul tipo **T** in quanto deve avere la possibilità di contenere un qualsiasi tipo di dati, e sul funtore **E** che mi permette di definire la nozione di uguaglianza tra due generici elementi: la classe T potrebbe infatti non avere un operator ==.
- Il set è stato implementato usando una **lista concatenata doppia**: ho fatto questa scelta in quanto mi sembrava ragionevole avere la possibilità di muoversi in entrambe le direzioni in un certo set, mentre una lista semplice avrebbe sarebbe stata troppo limitante. Erano possibili altre implementazioni (es: alberi …) ma sarebbero state inutilmente complicate.
- Oltre ai metodi richiesti (e quelli fondamentali) è stato inserito un metodo ausiliario **contains**, che dato un certo elemento ritorna true se l'elemento è compreso nel set, false altrimenti. Lo utilizzo in molti metodi, e a mio parere migliora la leggibilità del codice.
- E' stato implementato un const_iterator di tipo **random-access**. Ho fatto questa scelta in quanto veniva esplicitamente richiesto (nella classe) un operator[]: questo mi ha fatto capire che gli elementi all'interno di un set non vengono necessariamente acceduti in maniera sequenziale. Gli iteratori forward e bidirectional sembravano troppo limitanti in questo senso. E' vero che si va a creare una certa ridondanza tra l'operator[] della classe e quello dell'iteratore, averli separati può tornare comunque utile (es: passo ad una funzione l'iteratore e non necessariamente l'oggetto …)
    - Un operatore a cui ho fatto particolare attenzione è l'operator-. Esso risulta fondamentale per avere la possibilità di fare confronti tra iteratori.
- La funzione **filter_out** è templata su T,E (per il set) e su **P** (il predicato). Il set s viene passato per reference per velocizzare le performance. Il **ritorno** avviene **per copia** in quanto result viene creato all'interno della funzione e deallocato all'uscita dello scope.
- L' **operator+** è templato su T,E (per il set). Valgono le stesse considerazioni su tipo di ritorno e parametri della funzione filter_out.

Parte Qt

Il progetto prevede la creazione di un'interfaccia grafica per la visualizzazione, sotto forma tabellare e di grafico, di dati relativi ad alcune regioni italiane.

- Viene utilizzata una struttura dati
  QMap<QString, QMap<QString,QMap<QString,int>>> values.
  Anche se apparentemente complicata, semplifica di molto la memorizzazione e l'accesso ai dati: basta fare **values[regione][eta][M/F].** (Python potrebbe avermi ispirato in questa scelta!)
    - Viene preferita QMap a QHash in quanto QMap ordina automaticamente i dati in ordine lessicografico.
- All'apertura del programma viene caricato il file in posizione *./dataset/data.txt*.
    - Se il file non viene trovato, viene aperto un **QFileDialog** per permettere all'utente di scegliere il file da caricare
- Nel widget **QTableWidget** viene impedita la modifica dei dati da parte dell'utente: sarebbe poco sensato in questo caso, inoltre renderebbe i dati non più coerenti con i grafici.
- La parte di inserimento dati nella tabella e nel grafico utilizza due "casi particolari": "100+" e "5-9". Entrambi davano problemi con l'ordinamento lessicografico ("100+" veniva messo tra "10" e "20", "5-9" tra "50" e "60").
    - I controlli relativi a questi due casi sono all'interno di due if in quanto volevo evitare problemi nel caso in cui le due categorie non fossero presenti nel dataset di partenza. Ad ogni modo per questo dataset specifico avrei potuto evitare gli if.
- Per la visualizzazione dei grafici sono stati utilizzati dei widget **QGraphicsView**.
    - Essendo i grafici molto grandi ho dovuto allargare molto questi widget per evitare problemi di visualizzazione.
    - Il grafico riporta le etichette del tipo: range-percentuale. Avrei preferito visualizzare nella legenda i range e nel grafico le percentuali (come indicato nel mockup), ma non è possibile dividere le due (le etichette del grafico sono le stesse della legenda).