# Elaborato Fondamenti di Informatica - Consumi Idrici

## Analisi dei requisiti

Lo scopo del programma è quello di analizzare i consumi idrici di un gruppo di utenti relativi esclusivamente all'anno 2015 in maniera interattiva. I dati sono forniti in un file formato .csv strutturato in questo modo: ogni riga del file corrisponde ad una lettura di un contatore, ciascuna riga riporta in ordine data, valore e codice utente della lettura separati da una virgola.

* La data è in formato "yyyy-MM-dd hh:mm:ss"
* Il valore della lettura è un numero decimale progressivo che indica quanti metri cubi sono stati consumati da quell'utente a quella data
* Il codice utente è un identificativo alfanumerico univoco che si riferisce ad una utenza domestica.

Non è garantito alcun ordine nell'insieme dei dati e non è garantita nessuna continuità cronologica: ogni utente può avere un numero variabile di letture e tra lettura successive può essere passata una quantità di tempo variabile.

Il programma deve fornire tre funzionalità principali: visualizzazione dei consumi, interrogazione dei dati ed analisi dei consumi.

La visualizzazione dei consumi permette, inserito un codice cliente, di visualizzare il consumo totale dello stesso, ossia il valore dell'ultima lettura in ordine cronologico. Inoltre permette di visualizzare un istogramma dei consumi. Tale grafico ha quattro modalità di visualizzazione:

* Annuale: il grafico mostra 12 barre dei consumi di ciascun mese dell'anno 2015
* Mensile per giorni: il grafico mostra da 28 a 31 barre relative ai consumi di ciascun giorno di un mese scelto dall'utente
* Mensile per settimane: il grafico mostra da 4 a 6 barre relative ai consumi delle settimane di un mese scelto dall'utente. Le settimane si considerano da lunedì a domenica e possono ricadere parzialmente all'interno del mese stesso
* Giornaliero: il grafico mostra i 24 consumi orari di una data scelta dall'utente.

Inoltre il grafico deve indicare il valore minimo, medio e massimo.

L'interrogazione dei dati è una funzionalità del programma che deve richiedere all'utente un codice identificativo, una data di inizio ed una di fine periodo. In base all'input deve mostrare il consumo dell'utente cercato all'interno del periodo ed i valori medi di consumo orario, giornaliero, settimanale e mensile. Devono essere indicati solo i dati ritenuti validi in base all'estensione del periodo.

L'analisi dei consumi, al contrario delle altre sezioni, riguarda l'intero insieme di utenze e non una singola e si suddivide in due parti. La prima deve mostrare un elenco di utenti che si ritenga possano avere perdite, ossia utenti con consumi superiori ad una soglia in orario notturni (compresi tra le 00:00 e le 05:00). Per ogni utente devono essere indicati i periodi in cui tali consumi sospetti si sono verificati. La seconda parte è quella che ricerca le utenze devianti, ossia un elenco di tutte le utenze con consumi medi superiori di più del doppio rispetto alla media di tutte le utenze.

## Progettazione

...

### Classi e strutture dati

Il programma è realizzato secondo il paradigma della programmazione a oggetti. Le classi realizzate sono:

* MainWindow: classe standard per la gestione dell'interfaccia grafica. Essendo il programma formato da una sola finestra è l'unica classe di questo tipo. Contiene tutte le interazioni con l'utente gestite in base agli eventi dell'interfaccia e richiama gli altri oggetti per la manipolazione e la lettura dei dati. Il suo metodo show() viene invocato dal main.
* InputFile: oggetto che rappresenta il file contenente i dati. Il suo metodo principale read() consente di estrapolare dal file i dati in una struttura ideale per essere analizzati facilmente.
* Consumption: rappresenta una lettura di un contatore indipendentemente dall'utente cui viene associata. La parte privata contiene quindi solo data e valore della lettura. Sono implementati gli operatori di (dis)uguaglianza e ordinamento. Garantisce la correttezza dei dati.
* ConsumptionSet: basato sulla std::set, rappresenta una collezione di consumption ordinate. Gestisce l'inserimento delle consumption nella collezione, garantendo la coerenza tra tutti i dati. Permette inoltre di calcolare i dati necessari alla realizzazione del programma.
* Plot: rappresenta il grafico della parte di visualizzazione. Permette di disegnare e cancellare l'istogramma e di ottenere i valori minimo, medio e massimo.
* AvgTableModel e LeaksTableModel: derivano dalla classe QAbstractModel e sono i modelli delle due tabelle che visualizzano gli utenti con perdite e devianti.

La struttura dati più importante del programma è quella che contiene tutti i dati ricavati dal file. È una std::map<QString, ConsumptionSet> che associa ad ogni codice utente (QString) l'elenco di tutte le sue letture (ConsumptionSet). Questa soluzione semplifica la rappresentazione dei dati perché elimina ogni informazione non necessaria (il codice utente ripetuto ad ogni lettura) e li organizza in maniera strutturata e ne velocizza l'accesso perché le chiavi (codici utente) sono ordinati alfabeticamente e gli oggetti a cui si riferiscono (ConsumptionSet) contengono al loro interno le letture ordinate cronologicamente. La ConsumptionSet sfrutta tutti i vantaggi della std::set, in particolare l'ordinamento, estendendo le sue funzioni con quelle di calcolo dei consumi.

### Algoritmi

***InputFile::read() : legge il file e produce una std::map contenente i dati***

Legge il file csv una riga alla volta, da cui ricava i tre parametri data, valore e codice utente. Utilizza i primi due per costruire un oggetto Consumption ed aggiunge quest'ultimo alla ConsumptionSet indicizzata nella mappa dalla chiave uguale al codice utente. Robusto ad errori causati da dati errati, può ignorare righe invalide chiedendo conferma all'utente.

***ConsumptionSet::getLast() : restituisce il consumo totale dell'utente e la data di ultimo aggiornamento***

Sfruttando l'ordinamento dei dati già descritto, il consumo totale e la sua data di lettura sono semplicemente l'ultima Consumption nella collezione di dati

***ConsumptionSet::getConsAtDate(date) : restituisce il consumo in un momento preciso***

I dati non garantiscono una continuità ed indicano solo il consumo totale solo in alcuni momenti. Nei periodi compresi tra due lettura l'andamento del consumo è ignoto e viene quindi considerato lineare. Perciò, il consumo in un determinato momento è dato da una proporzione. Si cercano l'ultima registrazione precedente alla data richiesta e la prima successiva e la formula per calcolarlo è:

consumo a una data = (differenza di tempo tra le registrazioni più vicine) \* (differenza di consumi tra le registrazioni più vicine) / (differenza di tempo tra la data e la registrazione precedente più vicina) + (consumo alla registrazione precedente più vicina)

Nel caso in cui non sia presente una registrazione successiva, significa che il consumo è rimasto invariato dall'ultima registrazione. Il consumo alla data richiesta è quindi lo stesso di quello dell'ultima lettura. Nel caso in cui manchi una lettura precedente vale il discorso inverso.

***ConsumptionSet::getPeriodConsumption( firstDate, lastDate) : restituisce il consumo totale in un periodo***

Semplicemente la differenza tra il consumo totale alla seconda data e il consumo totale alla prima data. Grazie ai controlli sull'inserimento dei dati non può risultare negativo.

***ConsumptionSet::getHistogramData( begin, end, step) : restituisce i valori delle barre del grafico***

Richiede un periodo di tempo ed uno step tra un intervallo e l'altro.

Per ogni intervallo calcola la differenza di consumi tra il suo inizio e la sua fine. Ad ogni iterazione il consumo alla fine di un intervallo diventa quello all'inizio dell'intervallo successivo. Ripete finché la data di inizio incrementata ogni volta non raggiunge la data di fine.

L'insieme dei consumi dei vari intervalli sono i valori delle barre del grafico.

***ConsumptionSet::getNightLeaks( threshold) : restituisce l'insieme di consumi notturni superiori alla soglia***

Per ogni giorno del 2015 calcola il consumo nel periodo tra le 00:00 e le 05:00. Se tale consumo è superiore alla threshold lo aggiunge (insieme alla data di riferimento) alla lista delle possibili perdite notturne.

## Note di realizzazione: aspetti particolari della codifica

***Conversione in CET (Central European Time)***

Il tipo scelto per rappresentare le date e ore è il QDateTime. Durante le lettura del file, però, causa problemi in quanto esistono letture dei contatori nelle notti dei giorni in cui avviene il cambio ora legale-solare e ciò risulta in oggetti QDateTime invalidi. Per risolvere questo problema è necessario convertire il tutto in UTC tramite il metodo apposito e poi tornare in CET tenendo però conto dell'ora legale. Alle date UTC vengono quindi aggiunte due ore nel caso di date estive (comprese tra l'ultima domenica di marzo e l'ultima di ottobre) o un'ora sola nel caso di date invernali.

***ConsumptionSet::insert ( cons )***

Una ConsumptionSet è sostanzialmente una std::set<Consumption> e pertanto una collezione ordinata di Consumption. L'ordinamento tra Consumption è definito tramite l'operatore < che stabilisce che una Consumption è minore di un'altra se lo è la sua data. In caso di date uguali l'ordinamento viene stabilito dal valore. Ciò può sfociare nel caso in cui un oggetto Consumption ha data inferiore ad un'altra, ma consumo maggiore. Perciò è necessario escludere questi casi quando viene aggiunta una nuova Consumption ad una ConsumptionSet. Questi controlli sono implementati nella ConsumptionSet::insert() e, nel caso in cui non vengano superati, la nuova Consumption non viene aggiunta. In caso di dati incoerenti vengono quindi scelti quindi inseriti per primi. Nel file consumption\_all.csv questo avviene più volte anche in utenti diversi.

Qualche esempio in cui una lettura ha un consumo inferiore rispetto alla lettura precedente:

"2015-05-28 20:00:00",45.099,16

"2015-05-28 21:00:00",44.739,16

"2015-05-28 22:00:00",45.099,16

"2015-06-10 20:00:00",54.893,16

"2015-06-10 21:00:00",54.535,16

"2015-06-10 22:00:00",54.893,16

"2015-06-18 20:00:00",57.356,16

"2015-06-18 21:00:00",56.988,16

"2015-06-18 22:00:00",57.356,16

"2015-06-25 20:00:00",62.356,16

"2015-06-25 21:00:00",61.995,16

"2015-06-25 22:00:00",62.356,16

"2015-06-30 20:00:00",66.695,16

"2015-06-30 21:00:00",66.341,16

"2015-06-30 22:00:00",66.695,16

"2015-02-04 08:25:00",23.445,112

"2015-02-04 08:28:00",9.726,112

"2015-04-09 09:00:00",20.331,206

"2015-04-09 10:00:00",20.330,206

***Visualizzazione grafico per settimane***

La funzione ConsumptionSet::getHistogramData() funziona solo se il periodo considerato è formato da un numero intero di intervalli (cioè all'inizio del periodo inizia anche il primo intervallo e alla fine del periodo finisce l'intervallo). Questo non crea problemi nella visualizzazione annuale (12 mesi, il primo giorno dell'anno è il primo di Gennaio e l'ultimo è l'ultimo è l'ultimo di Dicembre), in quella mensile per giorni (28-31 giorni) e in quella giornaliera (24 ore). Nella visualizzazione settimanel, però, può capitare che la prima e/o l'ultima settimana siano comprese solo parzialmente nel mese scelto. Per questo motivo quando viene scelta la visualizzazione settimanale si tratta in realtà di una visualizzazione del mese a giorni, i cui valori vengono però accorpati da lunedì (o dal primo giorno del mese per la prima settimana) a domenica (o all'ultimo giorno del mese per l'ultima). L'algoritmo che fa ciò è molto semplice: una volta ottenuti i dati della visualizzazione del mese per giorni, li scorre tutti e li somma fino a quando non raggiunge una domenica o la fine del mese; a quel punto memorizza la somma in un altro vettore, la azzera e ricomincia dal lunedì.

***Utenze devianti - consumo medio***

Nella ricerca delle utenze devianti si chiede di riportare le utenze con valor medio del consumo mensile, settimanale o giornaliero superiore di più del doppio rispetto alla media dell'intero insieme di utenze. Dato che i valori medi mensile, settimanale e giornaliero sono calcolati dividendo lo stesso consumo totale per il numero di mesi, settimane e giorni del periodo, questi sono multipli tra loro. Perciò non ha senso confrontare consumo medio giornaliero, settimanale e mensile di un utente con la media giornaliera, settimanale e mensile dell'intero insieme di utenze, ma è sufficiente confrontarne uno solo di questi, o addirittura confrontare i due consumi totali. Una volta trovate le utenze devianti si è comunque scelto di mostrare nella relativa tabella i valori medi giornalieri, settimanali e mensili.

***Utenze devianti - periodo***

Essendo una ricerca effettuata sui valori medi all'interno di un periodo, non ha senso parlare di "tutti i periodi in cui il consumo ha superato il doppio della media" perché l'unico periodo in cui ciò è avvenuto è quello in cui si è cercato.

***Rispetto paradigma***

Si è cercato di rispettare ogni regola di buona programmazione:

* Le classi sono strutturate in maniera corretta
* Sono state usate le liste di inizializzazione
* Gli errori non sono comunicati con valori di ritorno particolari ma con eccezioni, le quali vengono intercettate e gestite. La stessa cosa viene fatta nel costruttore della classe Consumption nel caso in cui i dati non siano validi.
* Avendo definito gli operatori == e < per la classe Consumption, sono stati definiti anche !=, <=, > e >=
* Si è evitato di causare memory leak. L'unica volta che viene allocata memoria dinamica viene fatto in un costruttore e nel corrispettivo distruttore la memoria viene deallocata
* È rispettata la const correctness

## Test

...