CMPE 230

System Programming

Project 3

-

Author 1: Altay Acar
Author 1's ID: 2018400084
Author 2: Engin Oğuzhan Şenol
Author 2's ID: 2020400324

-

Introduction

In this project, we are asked to design a QT program (in C++ programming language) which gives a table as the output. The program takes the data from a txt file containing the names or symbols of cryptocurrencies as input and places them in the table by taking the euro, dollar and British Pound values of each. The path to the input file should be obtained from an environment variable named MYCRYPTOCONVERT.

For this project, we created four different files which are mycryptoconvert.pro, accessor.h, accessor.p, and mainwindow.cpp. Also, we need another file as an input file, in which we take the name or symbols of the cryptocurrencies, to display them in the Currency Conversion Rate table.

mycryptoconvert.pro

This is the project file of our program. It contains all the information required by qmake to build our application. The resources that we have used while implementing our program are specified using a series of declarations in the file. Headers, source files, and distant files (input txt file) are specified in their respective declarations.

accessor.h

First, we included all the classes that we used in our Project to our header file. Our Accessor object inherits the QTableWidget class, as it functions like a unique QTableWidget with

additional features for our project. While QTableWidget allows us to create the table structure and perform operations on it, QNetworkAccessManager provides us the functionality to send requests to the network and get replies accordingly. We used these classes to create the table structure, get data from the given URL and fill the created table from the json data that we received.

Then, we declared our Accessor class, which inherits QTableWidget. We publicly created 3 different QMap objects in order to store key value pairs we needed to access during the operation to fill the table. Those key-value pairs are symbols and ids of the cryptocurrencies in `symbol_dict`, names and ids of cryptocurrencies in `name_dict`, and ids and names in `id_dict`. Also, we defined an integer named `rowsize` to store the size of the rows for sizing operation of the table display. Then, we created two functions under slots: `replyFinished`, which fills the Accessor object's table structure's appropriate fields from the data received after the reply from the given URL, `dictAcquired` helps us to get data of all the coins' names, ids, and symbols to build QMap structures that stores them as key-value pairs. After that it sends the request to get conversion rates to fill the table. Last, we defined `manager` and `archiver` QNetworkAccessManager object pointers privately. These are essential for us to access the data from the given URLs.

accessor.cpp

accessor.cpp is the source file for our Accessor class. We created a global variable QVector named `currencies` to store the name of the currencies that we take from the input file. `Accessor` is the most important class of our project. It inherits the QTableWidget for creating the table display and doing the operations regarding. In this part, we use two QNetworkAccessManager objects to receive data from two distinct URLs and make some appropriate operations accordingly. Our table display's number of rows is determined by the number of coins given in the input file, which corresponds to the size of the `currencies` QVector. Because our table display only shows the conversion rates of three currencies, named USD, EUR, GBP, the number of columns is four (plus the name column).

`replyFinished`: We started this part with creating a QJsonObject from the reply we received from the given URL for key-value names. Then, we implemented two nested for loops, which fills the table with the conversion rates of the given cryptocurrencies to USD, EUR and GBP in appropriate fields. We received the data about conversion rates from "coingecko.com".

`dictAcquired`: First, we started this part with creating a list which can store all the coins' data as a map for each coin. Then, we created a for statement that fills the information about coins as key values to appropriate maps we created earlier. After that, we implemented another for statement which includes two if statements. These statements stand for changing the name and symbols of the cryptocurrencies that are given in the input file to their ids. We use these ids to use in the URL. Because the URL we get reply about the conversion rates only accepts ids of the coins. So, we built the appropriate URL for the request according to the coins' ids given in the input file, which are stored in QVector

`currencies`. But, we also use the same data structure `currencies` to write names of the coins in the first column of our table display. So, we created a for loop which replaces the ids of the coins with their respective names.

mainwindow.cpp

This file is the main display of our program. We created our unique Accessor object `myAcc` for table display and network request operations. Our QApplication `app` runs the whole application operation and the central widget that contains `myAcc` is placed in the QVBoxLayout `vbox`. `myAcc` is the sole element of the display feature. So, our application window only shows the table we built in the previous procedures when `centralwidget->show()` function is called.

Problems We Encountered While Implementing

In this project, we faced several issues that blocked us from implementing our program further and we needed to overcome them to continue. Some of them are listed below:

Information about every coin: We created three different QMap objects for the information about coins and expected the reply we received from the URL as json objects. However the network reply was not a json object, instead it was structured as a list. So accessing the data from an URL, which is not a JsonDocument nor a JsonObject and it pushed us to parse the reply as a list structure. In order to overcome the issue, we casted the reply to a QVariant object, which is very flexible and can be further casted to QList object. Thanks to this, we could get the information and parse them as given order from the URL.

Accessing data outside of the scope of reply functions: After we receive a network reply, our program makes some adjustments and performs some operations accordingly. Those adjustments and operations change the data structures we use in our program. For example after receiving the reply from `archiver`, which we use to fill the QMap objects about coin's names, ids, and symbols, this change in these QMap objects didn't reach outside of the scope of `dictAcquired` function. So after data is received and the reply function is called, we couldn't see those changes when we tried to continue from the main field of Accessor. We tried to overcome this issue with declaring those objects as pointers. But, this time when we try to reach those objects' data the application crashed. So, we changed our algorithm in such a way that the network request `replyFinished`, which receives the coin's conversion rates and fills up the table accordingly is called inside the `dictAcquired` function. Hence, our program functioned errorless.

Parts of the Code That Can Be Improved

We know this project is a kind of a design project that expects us to fill the blanks about our implementation. So, if we had more time, we could design our table display more advanced by adding some extra features. For example, we can display the cryptocurrencies with their logo besides their names and we could also implement a code which can reload the table within a certain time to keep the conversion rates up to date after running the program once.