Software Assignment

Guang Heng Xian (21153324)

Jamie Lee (18029770)

Jamie Corstorphine (21155713)

**Program Design:**

The program was designed using the C# language, running a console application. The user is prompted to input at least four currencies out of the allowed currencies, and the program will attempt to find an arbitrage opportunity and print it out to the user. Next, the program prompts the user to input a pair of currencies (source and terminal), and the program will attempt to output the best conversion rate path from the source currency to the terminal currency.

The program reads a key from the **key.txt** file to call the API. If the program begins crashing after trying to retrieve the exchange rates, you may create a new email address using the 10 minute mail (<https://10minutemail.com/>) and register a new account at the API service (<https://www.exchangerate-api.com/>) to generate a new key. Once you have generated a new key, you may paste it into the **key.txt** file, and the program will continue working.

The entry point of the program is the **Main()** function in the **Program.cs** file.

**How to run the program:**

You may run the program directly by running the .exe file. If your computer does not allow you to run the .exe program, you will need to install Visual Studio (<https://visualstudio.microsoft.com/vs/>) and .NET6.0 (<https://dotnet.microsoft.com/en-us/download/dotnet/6.0>), then open the .sln file in Visual Studio. The program will be runnable afterwards.

**Findings**

Here is an example of the input and output of the program:

A screen shot of a computer

Description automatically generated

In this first example, we want to draw conclusions between the currencies AUD, NZD, USD, CNY, HKD, and EUR. The program has found an arbitrage opportunity from AUD->CNY->HKD->USD->AUD. The program then displays the best conversion rates between any two currencies, and upon entering CNY and EUR, the program has detected that there is a negative cycle along the path from CNY->EUR, so the program is unable to detect the exact path. However, the program will still output the exact best change rate from CNY->EUR, which precisely is 0.8641960203285104.

A screen shot of a computer

Description automatically generated

In this second example, the currencies we wish to draw conclusions on is CNY, CAD, CHF, KRW, SEK, INR, and BRL. In this case, there is an arbitrage opportunity found using the cycle CNY->SEK->INR->BRL->CNY. When we ask the program for the best conversion rate from CAD->CHF, the program has found the path and the path is not included inside a negative cycle, so the program is able to output the path. In this case, the path of the best conversion rate from CAD->CHF is CAD->CNY->CHF.

Please note that the API updates every hour on the free tier. The results produced in the future may not be identical to the results produced at the time which this report was written.

**Problems**

The first problem is associated with program memory and data type memory limits. The program was written with C#, where the most precise floating-point numeric type is the *decimal* type, having a size of 16 bytes. There are many operations in the C# **System.Math** namespace which support decimal calculations, but some operations only support up to the *double* type (8 bytes). Therefore precision may be lost amongst some calculations, however the general result will not be different by too much.

Another problem arises with currency exchange rates. In some currencies, one unit may be equal to tens or hundreds of thousands of units in another currency. Attempting to input the entire currency list at once may result in the program being unable to calculate a correct result and returning infinity.

The third bug is a display bug, but not a calculation error, happens when a large number is trying to be printed in the console. Due to the hardcoding of string padding widths, if a large number is attempted to be output, the entire row will have its formatting offset by a few characters, resulting in an uglier table.

Fourth, the program uses a constant value to represent infinity. If a calculation by coincidence equals to the infinity constant value, then the program will be unable to return the correct result. Although there does exist a **Double.PositiveInfinity** constant value in the C# language, but there is not a similar constant for decimals, so using a double would still result in loss of precision.

Finally, if a value has exceeded the maximum value of the decimal type, the calculation will result in an overflow, also deeming the calculation inaccurate.

All problems stated above only have a small chance of occurring. If the input currencies do not contain an extremely large exchange rate, then a formatted and correct output will be displayed.