Real-time Programming

Assignment 1

Currency app

I M Bouaraba

```cpp
#include "windows.h"
#include "iostream"
#include <string>
#include "tchar.h"
#include "wininet.h"
#include "stdlib.h"
#include <vector>
#include <fstream>
#include <list>
#include <iterator>
#include <algorithm>
#include <sstream>
//#include <stdafx.h>
#pragma comment (lib,"wininet.lib")
using namespace std;
```

As you can see most of the library's used are quite standard, had quite a lot of trouble with wininet.h and getting it to connect but that turned out to be a bad URL in the end.

```cpp
::vector<std::string> splitString(const std::string &s, char delim) {
    std::stringstream ss(s);
    std::string item;
    std::vector<std::string> elems;
    while (std::getline(ss, item, delim)) {
        elems.push_back(item);
    }
    return elems;
}
```

This function takes data recovered from the API and breaks it down into a vector string for better storage and latter to be turned in Doubles and strings.

```cpp
int dataRead(std::vector<double> & rates)
{
    string line;

    ifstream inFile("Data.txt");
    if (!inFile)
    {
        cerr << "File Data.txt not found." << endl;
        return -1;
    }

    std::vector<string> line2split;
    double rate;

    while (getline(inFile, line))
    {
        if (line.empty()) continue;

        line2split = splitString(line, ',');

        rate = std::stod(line2split.at(1));

        rates.push_back(rate);
    }

    inFile.close();
    return 0;
}
```

This function reads from the the data file from the yahoo API and extracts the rate from each line after turning the rate into a double for easy processing. Using the handy comma's in the API feed and a separator. In the final part it pushes the values down the list and writes them to file.

```
int GBPtoUSD(int sampleSize, int interval /*, const std::string& currency = "GBPUSD"*/)
{

    HINTERNET connect = InternetOpen(_T("MyBrowser"),
    INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
    if (!connect)
    {
        cout << "Connection with server failed";
        system("PAUSE");
        return -1;
    }
    ofstream output_file;
    output_file.open("Data.txt");
    char DataReceived1[4096] = { '\0' };
    DWORD NumberOfBytesRead = 0;

    while ( sampleSize >= 0) {

            HINTERNET OpenAddress1 = InternetOpenUrl(connect, _T("http://finance.yahoo.com/d/quotes.csv?e=.csv&f=sl1d1t1&s=GBPUSD=X")
            _T("&format=json&env=store://datatables.org/alltableswithkeys&callback="), NULL, 0,
            INTERNET_FLAG_PRAGMA_NOCACHE | INTERNET_FLAG_KEEP_CONNECTION | INTERNET_FLAG_SECURE, 0);

        if (!OpenAddress1)
        {
            DWORD ErrorNum = GetLastError();
            cout << "failed to connect to API Url \nError No: " << ErrorNum;
            InternetCloseHandle(connect);
            //system("PAUSE");
            return -1;
        }
        InternetReadFile(OpenAddress1, DataReceived1, 4096, &NumberOfBytesRead) && NumberOfBytesRead;
        {
            cout << ".";
            output_file << DataReceived1;
            sampleSize = sampleSize - 1;
        }
        InternetCloseHandle(OpenAddress1);
        Sleep(interval);
    }
    InternetCloseHandle(connect);
    output_file.close();
    return 0;
}
```

This function establishes a connection with the API (also includes fail safes if connection is not made), it then opens a file, if it's not present it makes it. It then stores the whole data string as a char taking advantage of the while loop to propagate the list of data. Then makes sure to close any connection left open after.

```cpp
void processData(const std::vector<double> & rates)
{
    // do the processing
    double totalRate = rates.front();
    for (unsigned int i = 1; i < rates.size(); ++i)
    {
        totalRate += rates.at(i);
    }
    cout << "sum of rates is " << totalRate << " average " << totalRate / rates.size() << "\n";
}
```

This code works out a possible future value by calculating and average of data file collected by the user, it quickly

```cpp
int main()
{
    bool menuon;
    char mainmenu;
    int sampleSize = 10; /// total sample count
    int interval = 1000; /// sample taken every x milliseconds

    std::vector<double> rates;

    cout << "Welcome to Ibby's money rate guess app" << endl;

    do {
        cout << "\n[1] Define sample size" << endl;
        cout << "[2] Define sample interval (milliseconds)" << endl;
        cout << "[3] Record GBP-USD data" << endl;
        cout << "[4] predict rate based on cumalitive average" << endl;
        cout << "[5] current rate for GBP-USD" << endl;
        cin >> mainmenu;
        switch (mainmenu)
        {
        case '1':
            cout << "please input how many samples you would like to take.."<< endl;
            cin >> sampleSize;
            break;
        case '2':
            cout << "please define how often you would like to sample data" << endl;
            cin >> interval;
            break;
        case '3':
            //int result =
            cout << "Starting data collection...." << endl;
            GBPtoUSD(sampleSize, interval);

            break;
        case '4':
            dataRead(rates);
            processData(rates);
            break;
        case '5':
            break;
        }
    } while (menuon = true);
    return 0;
}
```

shuffles through the list using a for loop.

The main function acts as a main menu for the user to choose what they want to do and also preconfigure the collection process of data collection. This is done via option 1 and 2. sample size defines how many samples the app will take in total regardless of amount of time taken between each sample, that's where sample interval comes in. it is recommended for the best prediction to set the sample size in the range of 250-500 with an interval of 2000-5000 millisecond interval.

Example Raw data.

"GBPUSD=X",1.4219,"3/9/2016","7:14pm"
"GBPUSD=X",1.4218,"3/9/2016","7:14pm"
"GBPUSD=X",1.4219,"3/9/2016","7:14pm"
"GBPUSD=X",1.4219,"3/9/2016","7:14pm"
"GBPUSD=X",1.4218,"3/9/2016","7:14pm"
"GBPUSD=X",1.4218,"3/9/2016","7:14pm"
"GBPUSD=X",1.4219,"3/9/2016","7:14pm"
"GBPUSD=X",1.4218,"3/9/2016","7:14pm"
"GBPUSD=X",1.4218,"3/9/2016","7:14pm"
"GBPUSD=X",1.4220,"3/9/2016","7:14pm"
"GBPUSD=X",1.4219,"3/9/2016","7:14pm"
"GBPUSD=X",1.4219,"3/9/2016","7:14pm"
"GBPUSD=X",1.4219,"3/9/2016","7:14pm"

Conclusion

Keeping in mind this is one of the first c++ programmes I've written and prior to starting this module had never written a thing in c++ apart from a dice roller, I feel I have learned a lot about debugging code that doesn't always have a directly apparent issue, also learned that there is a whole lot of decent library that have so many uses, looking forward to exploring more of c++ and the many perils it has to offer.

In retrospect I would have liked to diversify the app further to provide more insightful predictions and features. But also knowledge is key and feel that as coming from a ART and design background the experience of this module so far has enriched my pallet for programming.