**Министр науки и высшего образования Российской̆ Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

Лабораторная работа №

*Rest – запросы на сайт ЦБ РФ*

**Выполнила студентка группы № M3119**

Ким Александр Николаевич

**Подпись:**

Изображение выглядит как зарисовка, линия

Автоматически созданное описание

**Проверил:**

Повышев Владислав Вячеславович

Санкт-Петербург

2024Текст задания

Написать программу, которая обеспечит следующий функции:

получение курса валюты к российскому рублю на выбранную дату

получение курса валюты к российскому рублю на выбранный период

сохранение полученной информации в файл

Программа выполняется из командной строки с параметрами. Параметры разделены пробелами. Параметр состоит из ключа и значения. Формат параметра: --<key>=<value>

Параметры и список возможных значение:

**Vname** – название валюты, значение - буквенный код валюты, например `. Полный список кодов: https://www.iban.ru/currency-codes. Обязательный паромер.

**Vnom** – наминал валюты, значение – целое положительное число, обычно 1, может быть 10, 100 и т.д. Не обязательный параметр, по умолчанию номинал считается равным номиналу определяемому ЦБ РФ.

**Date** – дата, на которую получают курс, значение – дата, формат: dddd.mmmm.yyyy. Не обязательный параметр, по умолчанию текущая дата.

**DateDate** – период на который получают курс. Значение – дата начала и дата окончания периода включительно, разделенная символом -. Формат: dddd.mmmm.yyyyyyyy-dddd.mmmm.yyyy. Не обязательный параметр. Нет значения по умолчанию. Не совместим с параметром **DateDate**.

**FileName** – имя, или полный путь, файла куда будет записан результат запроса. Значение – имя или путь в формате используемой операционной системы. Не обязательный параметр. Значения по умолчанию нет. При отсутствии параметры выводить информацию на экран.

Вывод информации в формате: <код валюты> <номинал> <дата> <курс>

Если выводится информация на период, то каждая дата выводится отдельной строкой.

Курс валют определяет ЦБ РФ.

Пример командной строки:

**Lab9.exe Vname=USD Vnom=10 Date=01.01.2024**

Получение курса 10 долларов США на 1 января 2024 года

Вывод:

**USD 10 01.01.2024 896 883**

Important!

Автотестов на GH для этой работы не предусмотрено. Работоспособность демонстрируется на защите.

Форматирование кода должно быть согласно правилам **clang-format**.

Необходимо задокументировать своё решение в виде доки по своей программе и набору классов/функций.

Решение с комментариями

1. Собрал библиотеку curl с помощью fetch content для CMake в Clion:

cmake\_minimum\_required(VERSION 3.27)  
project(OPLAB9  
 VERSION 1.0  
 DESCRIPTION "C++ Year 2024 labwork9"  
 LANGUAGES CXX  
)  
  
set(CMAKE\_CXX\_STANDARD 23)  
  
if (WIN32) # Install dlls in the same directory as the executable on Windows  
 set(CMAKE\_LIBRARY\_OUTPUT\_DIRECTORY ${CMAKE\_BINARY\_DIR})  
 set(CMAKE\_RUNTIME\_OUTPUT\_DIRECTORY ${CMAKE\_BINARY\_DIR}/bin)  
endif ()  
  
include(FetchContent)  
FetchContent\_Declare(  
 curl  
 GIT\_REPOSITORY https://github.com/curl/curl.git  
 GIT\_TAG curl-8\_7\_1  
)  
FetchContent\_MakeAvailable(curl)  
  
add\_executable(OPLAB9 main.cpp request\_class.h parser\_class.h currency\_class.h curl\_request.h)  
target\_link\_libraries(OPLAB9 PUBLIC libcurl)

1. Скачал библиотеку (не нужно устанавливать, все *include* файлы уже были в папке) pugixml, для работы с xml-деревом при получение ответа сайта на запрос

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

1. Написал парсер для аргументов int argc, char \*argv[]:
2. class ParseArguments {  
    private:  
    std::string v\_name;  
    std::string v\_nominal;  
    const char \*date\_format = "%d.%m.%Y";  
    static std::string GetCurrentDate(const char \*format) {  
    time\_t now = time(nullptr);  
    tm \*tmPtr = localtime(&now);  
    char arr[80];  
    strftime(arr, sizeof(arr), format, tmPtr);  
    std::string return\_str = std::string(arr);  
    return return\_str;  
    }  
    std::string date;  
    std::string date\_date;  
    std::string date\_date\_start;  
    std::string date\_date\_finish;  
    std::string filename;  
     
    bool have\_filename = false;  
    bool have\_date = false;  
    bool have\_v\_nom = false;  
    bool have\_v\_name = false;  
    bool have\_date\_date = false;  
    std::vector<std::string> dates{};  
     
    void FormatDate() {  
    std::replace(date.begin(), date.end(), '.', '/');  
    std::replace(date\_date.begin(), date\_date.end(), '.', '/');  
    std::replace(date\_date\_start.begin(), date\_date\_start.end(), '.', '/');  
    std::replace(date\_date\_finish.begin(), date\_date\_finish.end(), '.', '/');  
    }  
     
    static std::vector<std::string> SplitDate(const std::string &s, char delim) {  
    std::vector<std::string> elems;  
    std::stringstream ss(s);  
    std::string item;  
    while (std::getline(ss, item, delim)) {  
    elems.push\_back(item);  
    }  
    return elems;  
    }  
     
    public:  
    ParseArguments() = default;  
    [[nodiscard]] std::string get\_v\_name() const {  
    return this->v\_name;  
    }  
    [[nodiscard]] std::string get\_v\_nominal() const {  
    return this->v\_nominal;  
    }  
    [[nodiscard]] std::string get\_date() const {  
    return this->date;  
    }  
    [[nodiscard]] std::string get\_date\_date\_start() const {  
    return this->date\_date\_start;  
    }  
    [[nodiscard]] std::string get\_date\_date\_finish() const {  
    return this->date\_date\_finish;  
    }  
    [[nodiscard]] bool date\_date\_exists() const {  
    return this->have\_date\_date;  
    }  
    [[nodiscard]] std::vector<std::string> get\_dates() const {  
    return this->dates;  
    }  
    [[nodiscard]] size\_t get\_dates\_size() const {  
    return this->dates.size();  
    }  
    [[nodiscard]] std::string get\_filename() const {  
    return this->filename;  
    }  
    [[nodiscard]] bool filename\_exists() const {  
    return this->have\_filename;  
    }  
     
    void Parse(int argc, char \*argv[]) {  
    std::string argument;  
    for (int i = 1; i < argc; ++i) {  
    argument = argv[i];  
    if (argument.starts\_with("--Vname=")) {  
    have\_v\_name = true;  
    for (int j = 8; j < argument.size(); ++j) {  
    v\_name += argument[j];  
    }  
    } else if (argument.starts\_with("--Vnom=")) {  
    have\_v\_nom = true;  
    for (int j = 7; j < argument.size(); ++j) {  
    v\_nominal += argument[j];  
    }  
    } else if (argument.starts\_with("--Date=")) {  
    have\_date = true;  
    for (int j = 7; j < argument.size(); ++j) {  
    date += argument[j];  
    }  
    } else if (argument.starts\_with("--DateDate=")) {  
    have\_date\_date = true;  
    for (int j = 11; j < argument.size(); ++j) {  
    date\_date += argument[j];  
    }  
    } else if (argument.starts\_with("--Filename=")) {  
    have\_filename = true;  
    for (int j = 11; j < argument.size(); ++j) {  
    filename += argument[j];  
    }  
    }  
    }  
    if (!have\_date && !have\_date\_date) {  
    date = GetCurrentDate(date\_format);  
    }  
    if (!have\_date && have\_date\_date) {  
    SeparateDateDate('-');  
    }  
    FormatDate();  
    if (!have\_v\_nom || !have\_v\_name) {  
    std::cerr << "Parsing failed. No required arguments.";  
    exit(EXIT\_FAILURE);  
    }  
    }
3. Сделал функции класса для распознавания дат через разделители и калькуляции дат между двумя какими либо датами:
4. void SeparateDateDate(const char delim) {  
    std::vector<std::string> out;  
    size\_t start;  
    size\_t end = 0;  
    while ((start = date\_date.find\_first\_not\_of(delim, end)) != std::string::npos) {  
    end = date\_date.find(delim, start);  
    out.push\_back(date\_date.substr(start, end - start));  
    }  
    date\_date\_start = out.front();  
    date\_date\_finish = out.back();  
    }  
     
    void DatesBetween(const std::string &startDate, const std::string &endDate) {  
    std::vector<std::string> startParts = SplitDate(startDate, '/');  
    std::vector<std::string> endParts = SplitDate(endDate, '/');  
    int startDay = std::stoi(startParts[0]);  
    int startMonth = std::stoi(startParts[1]);  
    int startYear = std::stoi(startParts[2]);  
    int endDay = std::stoi(endParts[0]);  
    int endMonth = std::stoi(endParts[1]);  
    int endYear = std::stoi(endParts[2]);  
    for (int year = startYear; year <= endYear; ++year) {  
    int monthStart = (year == startYear) ? startMonth : 1;  
    int monthEnd = (year == endYear) ? endMonth : 12;  
    for (int month = monthStart; month <= monthEnd; ++month) {  
    int dayStart = (year == startYear && month == startMonth) ? startDay : 1;  
    int dayEnd = (year == endYear && month == endMonth) ? endDay : 31;  
    for (int day = dayStart; day <= dayEnd; ++day) {  
    std::string single\_date = (day < 10 ? "0" : "") + std::to\_string(day) + "/" +  
    (month < 10 ? "0" : "") + std::to\_string(month) + "/" +  
    std::to\_string(year);  
    dates.push\_back(single\_date);  
    }  
    }  
    }  
    }  
   };
5. Написал базовые функции для использования ссылок, и создания заппроса, а также для получения данных из сайта
6. class Request {  
    private:  
    std::string url;  
    std::string current\_date;  
    const std::string url\_template = "www.cbr.ru/scripts/XML\_daily.asp?date\_req=";  
    void MakeURL(const ParseArguments &Parser) {  
    url = url\_template + Parser.get\_date();  
    }  
    public:  
    explicit Request(const ParseArguments &Parser) {  
    current\_date = Parser.get\_date();  
    MakeURL(Parser);  
    }  
    void MakeURLByDate(const std::string &date) {  
    current\_date = date;  
    url = url\_template + date;  
    }  
    [[nodiscard]] std::string get\_url() const {  
    return this->url;  
    }  
    [[nodiscard]] std::string get\_curr\_date() const {  
    return this->current\_date;  
    }  
   };  
     
   static size\_t CollectData(char \*p, size\_t s, size\_t n, std::string \*d) {  
    d->append(p, s \* n);  
    return s \* n;  
   }
7. Реализовал функции для парсинга XML-дерева и отображение итогового запроса в консоль:
8. #pragma once  
     
   namespace fs = std::filesystem;  
     
   struct XmlCurrency {  
    private:  
    std::string num\_code;  
    std::string char\_code;  
    int nominal{};  
    std::string name;  
    long double value{};  
    std::vector<XmlCurrency> parsed\_data;  
    public:  
    void ParseXml(const std::string &xml\_data) {  
    pugi::xml\_document doc;  
    if (!doc.load\_string(xml\_data.c\_str())) {  
    std::cerr << "Failed to load XML data.";  
    return;  
    }  
    pugi::xml\_node val\_curs = doc.child("ValCurs");  
    for (pugi::xml\_node v\_currency = val\_curs.child("Valute"); v\_currency;  
    v\_currency = v\_currency.next\_sibling("Valute")) {  
    XmlCurrency curr;  
    curr.num\_code = v\_currency.child\_value("NumCode");  
    curr.char\_code = v\_currency.child\_value("CharCode");  
    curr.nominal = std::stoi(v\_currency.child\_value("Nominal"));  
    curr.name = v\_currency.child\_value("Name");  
    std::string str\_value = v\_currency.child\_value("Value");  
    std::replace(str\_value.begin(), str\_value.end(), ',', '.');  
    curr.value = std::stold(str\_value);  
    parsed\_data.push\_back(curr);  
    }  
    }  
     
    static void ColorDraw(const std::string &message) {  
    if (message.empty()) {  
    std::cerr << "Non-existing request.";  
    return;  
    }  
    system("Color 02");  
    std::vector<std::string> words;  
    std::istringstream iss(message);  
    std::string word;  
    while (iss >> word) {  
    words.push\_back(word);  
    }  
    for (const auto &kWord : words) {  
    std::cout << kWord << " ";  
    }  
    std::cout << "\n";  
    }  
     
    void Display(ParseArguments &Parser, Request &Req) {  
    std::string result\_str;  
    std::string v\_name = Parser.get\_v\_name();  
    std::string v\_nominal = Parser.get\_v\_nominal();  
    for (auto &c : parsed\_data) {  
    if (c.char\_code == v\_name) {  
    result\_str += c.char\_code + " ";  
    long double value\_res = (stold(v\_nominal) / c.nominal) \* c.value;  
    std::stringstream ss\_value\_res;  
    ss\_value\_res << std::fixed << std::setprecision(std::numeric\_limits<long double>::digits10 + 1) << value\_res;  
    std::string str\_value = ss\_value\_res.str();  
    result\_str += v\_nominal + " " + Req.get\_curr\_date() + " " + ss\_value\_res.str();  
    // std::replace(result\_str.begin(), result\_str.end(), '/', '.');  
    break;  
    }  
    }  
    if (!Parser.filename\_exists()) {  
    XmlCurrency::ColorDraw(result\_str);  
    } else {  
    fs::path project\_dir = "D:/ITMO/2\_SEM/OP/OPLAB9";  
    fs::path file\_path = project\_dir / Parser.get\_filename();  
    if (!fs::exists(project\_dir)) {  
    std::cerr << "No existing directory for the file.";  
    return;  
    }  
    std::ofstream outFile(file\_path, std::ios::app);  
    if (!outFile.is\_open()) {  
    std::cerr << "Can not open the file.";  
    return;  
    }  
    outFile << result\_str << "\n";  
    outFile.close();  
    }  
    }  
   };
9. Реализовал подачу запросов на сайт через установленный модуль curl и прикрутил к его классу реализованные выше методы
10. #pragma once  
    #include "parser\_class.h"  
    #include "request\_class.h"  
    #include "currency\_class.h"  
    #include "curl/curl.h"  
      
    void GetCurlRequest(ParseArguments& Parser, Request& Req) {  
     CURL \*curl;  
     CURLcode c;  
     curl\_global\_init(CURL\_GLOBAL\_ALL);  
     curl = curl\_easy\_init();  
     if (curl) {  
     if (!Parser.date\_date\_exists()) {  
     std::string xml\_s\_request;  
     curl\_easy\_setopt(cur*l, CURLOPT\_*URL, Req.get\_url().data());  
     curl\_easy\_setopt(cur*l, CURLOPT\_WRITEFUNCT*ION, CollectData);  
     curl\_easy\_setopt(cur*l, CURLOPT\_WRITED*ATA, &xml\_s\_request);  
     c = curl\_easy\_perform(curl);  
     if (c *!= CURLE*\_OK) {  
     std::cerr << "Curl parsing failed.";  
     return;  
     } else {  
     XmlCurrency XmlCrc;  
     XmlCrc.ParseXml(xml\_s\_request);  
     XmlCrc.Display(Parser, Req);  
     }  
     } else {  
     Parser.DatesBetween(Parser.get\_date\_date\_start(), Parser.get\_date\_date\_finish());  
     for (int i = 0; i < Parser.get\_dates\_size(); ++i) {  
     std::string xml\_s\_request;  
     Req.MakeURLByDate(Parser.get\_dates()[i]);  
     curl\_easy\_setopt(curl, CURLOPT\_URL, Req.get\_url().data());  
     curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, CollectData);  
     curl\_easy\_setopt(cu*rl, CURLOPT*\_WRITEDATA, &xml\_s\_request);  
     c = curl\_easy\_perfo*rm(curl);  
     if* (c != CURLE\_OK) *{  
     std::*cerr << "Curl parsing failed.";  
     ret*urn;* } else {  
     XmlCurrency XmlCrc;  
     XmlCrc.ParseXml(xml\_s\_request);  
     XmlCrc.Display(Parser, Req);  
     }  
     }  
     }  
     } else {  
     curl\_easy\_cleanup(curl);  
     std::cerr << "No parsing was enabled.";  
     return;  
     }  
    }
11. Разнес классы по .h файлам и сделал для некоторых #pragma once, чтобы избежать проблем на этапе линковки.
12. Собрал