****

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота № 7**

з дисципліни “ Основи програмування ”

тема: “TCP клієнт та публічні Web API”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Виконав  студент I курсу  групи КП-61  Cвинарчук Максим Владиславович  (*прізвище, ім’я, по батькові*)  варіант № 23 | |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 2017 р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) | |
| Штрафні бали:   |  |  | | --- | --- | | **Термін здачі** | **Оформлення звіту** | |  |  | | Нараховані бали:   |  |  |  | | --- | --- | --- | | **Корект. програм (2 бала)** | **Відп. на теор. питання (1 бал)** | **Відп. на прогр. питання (2 бала)** | |  |  |  |   Київ 2017 | | | Сумарний бал:   |  | | --- | |  | |

**Мета роботи**

Навчитися користуватись публічними API (Application Programming Interface) web-сайтів у мережі Інтернет за допомогою TCP клієнта та парсерів форматів даних.

**Постановка завдання**

**Завдання №1. Створення TCP-клієнта**

Реалізувати модуль, що використовує TCP клієнт для підключення до [тестового web-сервера](https://api.github.com/).

Для виконання HTTP(S) GET запиту на отримання даних із web-сервера через TCP потрібно:

1. Виконати підключення TCP клієнта до сервера із хост-іменем api.github.com (потрібно програмно отримати IP-адресу сервера). Порт обирається відповідно до протоколу доступу (80 для HTTP або 443 для HTTPS).
2. Після успішного з'єднання сформувати строку HTTP GET запиту виду: GET %s HTTP/1.0\r\nHost: %s\r\n\r\n, де замість першого параметру форматування задати URL-шлях, що розташований за хост-іменем (наприклад, /), а замість другого параметру - хост-ім'я сайту (наприклад, api.github.com).
3. Відправити сформовану строку на сервер і отримувати відповідь доки розмір отриманих у відповідь байт не буде 0. Всі частини відповіді об'єднати у одну строку. Видалити зі строки всі символи, що розміщені перед двома end-of-line підряд (end-of-line: \n (Unix) або \r\n (Windows) . Результат звірити з JSON строкою, що відображається при переході на сайт у браузері.

**Завдання №2. Отримання та парсинг даних з API**

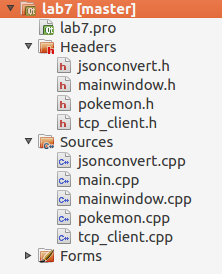
Дано публічний [API Pokemon](http://pokeapi.co/) за URI http://pokeapi.co/. Виконуючи HTTP(S) GET запити по різних шляхах даного сайту можна отримувати у відповідь дані у строці формату JSON. Використовуючи класи платформи Qt для роботи із JSON розпарсити отриману строку у JSON-дерево. Створити клас Pokemon із декількома характеристиками покемона, дані для яких можна отримати із API. На основі відповідей від API заповнити список різних покемонів.

Зверніть увагу на те, що кількість запитів до API протягом дня може бути обмеженою і після N-нного запиту в день дані стануть недоступні для вас аж до наступного дня.

**Завдання №3. Відображення даних у віконних програмах**

Створити віконну програму Qt, що дозволяє переглядати список отриманих поштових кодів із їх характеристиками (у формах) та фільтрувати відображення цього списку по певному критерію (наприклад, всі коди України). Користувач може обирати значення критерію.

**Структура програми**

****

**Тексти коду програм**

|  |
| --- |
| main.cpp |
| **#include** "mainwindow.h"  **#include** <QApplication>  **int** **main**(**int** argc, **char** **\***argv[])  {  QApplication a(argc, argv);  MainWindow w;  w.show();    **return** a.exec();  } |
| postcodeinfo.h |
| **#ifndef POSTCODEINFO\_H**  #define POSTCODEINFO\_H  #include <QString>  #include <QMetaType>  class PostCodeInfo  {  QString postcode;  QString country;  double longitude;  double latitude;  QString europeanElectoralRegion;  QString adminDistrict;  public:  PostCodeInfo();  ~PostCodeInfo();  void loadFromJson(QString string);  void print();  void setPostCode(QString postcode);  void setCountry(QString country);  void setLongitude(double longitude);  void setLatitude(double latitude);  void setEuropeanElectoralRegion(QString europeanElectoralRegion);  void setAdminDistrict(QString adminDistrict);  QString getPostCode(void);  QString getCountry(void);  double getLongitude(void);  double getLatitude(void);  QString getEuropeanElectoralRegion(void);  QString getAdminDistrict(void);  };  Q\_DECLARE\_METATYPE(PostCodeInfo)  #endif // POSTCODEINFO\_H |
| postcodeinfo.cpp |
| #include "postcodeinfo.h"  #include <iostream>  #include <QJsonDocument>  #include <QJsonObject>  #include <QJsonArray>  #include <QString>  void PostCodeInfo::setPostCode(QString postcode)  {  this->postcode = postcode;  }  void PostCodeInfo::setCountry(QString country)  {  this->country = country;  }  void PostCodeInfo::setLongitude(double longitude)  {  this->longitude = longitude;  }  void PostCodeInfo::setLatitude(double latitude)  {  this->latitude = latitude;  }  void PostCodeInfo::setEuropeanElectoralRegion(QString europeanElectoralRegion)  {  this->europeanElectoralRegion = europeanElectoralRegion;  }  void PostCodeInfo::setAdminDistrict(QString adminDistrict)  {  this->adminDistrict = adminDistrict;  }  QString PostCodeInfo::getPostCode()  {  return this->postcode;  }  QString PostCodeInfo::getCountry()  {  return this->country;  }  double PostCodeInfo::getLongitude()  {  return this->longitude;  }  double PostCodeInfo::getLatitude()  {  return this->latitude;  }  QString PostCodeInfo::getEuropeanElectoralRegion()  {  return this->europeanElectoralRegion;  }  QString PostCodeInfo::getAdminDistrict()  {  return this->adminDistrict;  }  PostCodeInfo::PostCodeInfo()  {  this->postcode = "";  this->country = "";  this->longitude = 0;  this ->latitude = 0;  this->europeanElectoralRegion = "";  this->adminDistrict = "";  }  PostCodeInfo::~PostCodeInfo() { }  void PostCodeInfo::loadFromJson(QString string)  {  QJsonDocument jdoc = QJsonDocument::fromJson(string.toUtf8());  if (!jdoc.isObject()) {  std::cerr << "Invalid JSON doc" << std::endl;  return;  }  QJsonObject jobj = jdoc.object();  QJsonObject result = jobj["result"].toObject();  this->postcode = result["postcode"].toString();  this->country = result["country"].toString();  this->longitude = result["longitude"].toDouble();  this->latitude = result["latitude"].toDouble();  this->europeanElectoralRegion = result["european\_electoral\_region"].toString();  this->adminDistrict = result["admin\_district"].toString();  }  void PostCodeInfo::print()  {  std::cout << "Postcode: " << postcode.toStdString() << "\nCountry: " << country.toStdString() << "\nLongitude: "  << longitude << "\nLatitude: " << latitude << "\nEuropean electoral region: "  << europeanElectoralRegion.toStdString() << "\nAdmin district: " << adminDistrict.toStdString() << "\n" << std::endl;  } |
| mainwindow.h |
| #ifndef MAINWINDOW\_H  #define MAINWINDOW\_H  #include <QMainWindow>  //#include <vector>  #include <postcodeinfo.h>  namespace Ui {  class MainWindow;  }  class MainWindow : public QMainWindow  {  Q\_OBJECT  public:  explicit MainWindow(QWidget \*parent = 0);  ~MainWindow();  void loadPostCodesToVector(int count);  void loadAllPostCodesToListWidget(void);  void loadPostCodeToListWidget(PostCodeInfo \* postCode);  void loadComboBox(void);  private slots:  void closeEvent(QCloseEvent \* event);  void on\_pushButton\_clicked();  void on\_listWidget\_itemSelectionChanged();  void on\_comboBox\_currentIndexChanged(const QString &arg1);  void showTime();  private:  Ui::MainWindow \*ui;  std::vector<PostCodeInfo \*> postCodes;  std::vector<QString> europRegions;  };  #endif // MAINWINDOW\_H |
| mainwindow.cpp |
| **#include** "mainwindow.h"  **#include** <QDebug>  **#include** <QCloseEvent>  **#include** <QApplication>  **#include** <QListWidget>  **#include** <dialog.h>  **#include** "ui\_mainwindow.h"    MainWindow**::**MainWindow(QWidget **\***parent) **:**  QMainWindow(parent),  ui(**new** Ui**::**MainWindow)  {  ui**->**setupUi(**this**);  languages.push\_back(**new** ProgLang("C", 1972, 1.3, "gcc", 0.33));  languages.push\_back(**new** ProgLang("C++", 1983, 6.4, "g++", 0.61));  languages.push\_back(**new** ProgLang("C#", 2001, 15.4, ".NET\_Framework", 0.80));  languages.push\_back(**new** ProgLang("Java", 1995, 26.3, "JVM", 0.78));  languages.push\_back(**new** ProgLang("JavaScript", 1995, 13.8, "SpiderMonkey", 0.71));  ui**->**listWidget**->**setIconSize(QSize(32, 32));  foreach (ProgLang **\*** cur, languages) {  QVariant qVariant;  qVariant.setValue(**\***cur);  QListWidgetItem **\***qLanguagesListItem **=** **new** QListWidgetItem();  qLanguagesListItem**->**setText(cur**->**getFullname());  qLanguagesListItem**->**setIcon(QIcon("/home/valzavator/progbase2/labs/lab6/images/" **+** cur**->**getFullname() **+** ".jpeg"));  qLanguagesListItem**->**setData(Qt**::**UserRole, qVariant);  ui**->**listWidget**->**addItem(qLanguagesListItem);  }    }    MainWindow**::~**MainWindow()  {  **delete** ui;  }    **void** MainWindow**::**closeEvent(QCloseEvent **\***event)  {  // use this to cleanup allocated eventmemory    languages.erase(languages.begin(), languages.end());  qDebug() **<<** "Window closed!";  event**->**accept();  }    **void** MainWindow**::**on\_listWidget\_itemSelectionChanged()  {  qDebug() **<<** "changed!";  **auto** items **=** ui**->**listWidget**->**selectedItems();  **auto** hasSelected **=** items.size() **>** 0;  **if** (hasSelected) {  ui**->**pushButton\_3**->**setEnabled(true);  ProgLang lang **=** items[0]**->**data(Qt**::**UserRole).value**<**ProgLang**>**();  ui**->**label\_2**->**setText(lang.getFullname());  ui**->**label\_3**->**setText(QString**::**number(lang.getDateOfOccurrence()));  ui**->**label\_4**->**setText(QString**::**number(lang.getMarketShare(), 'f', 2));  ui**->**label\_5**->**setText(lang.getData().getMajorImpl());  ui**->**label\_6**->**setText(QString**::**number(lang.getData().getSatisfactionIndex(), 'f', 2));  } **else** {  ui**->**pushButton\_3**->**setEnabled(false);  ui**->**label\_2**->**setText("");  ui**->**label\_3**->**setText("");;  ui**->**label\_4**->**setText("");;  ui**->**label\_5**->**setText("");;  ui**->**label\_6**->**setText("");;  }  }    **void** MainWindow**::**on\_pushButton\_3\_clicked()  {  QList**<**QListWidgetItem**\*>** items **=** ui**->**listWidget**->**selectedItems();    foreach(QListWidgetItem **\*** item, items)  {  **int** toDelete **=** ui**->**listWidget**->**row(item);  **delete** ui**->**listWidget**->**takeItem(ui**->**listWidget**->**row(item));  languages.erase(languages.begin() **+** toDelete);  }    }    **void** MainWindow**::**on\_pushButton\_4\_clicked()  {  ui**->**listWidget\_2**->**clear();  **int** year **=** ui**->**spinBox**->**value();  **int** count **=** languages.size();  **for** (**int** i **=** 0; i **<** count; i**++**) {  **if** (languages[i]**->**getDateOfOccurrence() **<** year) {  QVariant qVariant;  qVariant.setValue(**\***languages[i]);  QListWidgetItem **\***qLanguagesListItem **=** **new** QListWidgetItem();  qLanguagesListItem**->**setText(languages[i]**->**getFullname());  ui**->**listWidget\_2**->**setIconSize(QSize(32, 32));  qLanguagesListItem**->**setText(languages[i]**->**getFullname());  qLanguagesListItem**->**setIcon(QIcon("/home/valzavator/progbase2/labs/lab6/images/" **+** languages[i]**->**getFullname() **+** ".jpeg"));  qLanguagesListItem**->**setData(Qt**::**UserRole, qVariant);  ui**->**listWidget\_2**->**addItem(qLanguagesListItem);  }  }  }    **void** MainWindow**::**on\_pushButton\_2\_clicked()  {  Dialog **\*** dialog **=** **new** Dialog();  dialog**->**setWindowTitle("Add new language");  **if** (dialog**->**exec()) {  qDebug() **<<** "Accepted!";  ProgLang **\*** newLang **=** dialog**->**getData();  languages.push\_back(newLang);  QVariant qVariant;  qVariant.setValue(**\***newLang);  QListWidgetItem **\***qLanguagesListItem **=** **new** QListWidgetItem();  ui**->**listWidget**->**setIconSize(QSize(32, 32));  qLanguagesListItem**->**setText(newLang**->**getFullname());  qLanguagesListItem**->**setIcon(QIcon("/home/valzavator/progbase2/labs/lab6/images/" **+** newLang**->**getFullname() **+** ".jpeg"));  qLanguagesListItem**->**setData(Qt**::**UserRole, qVariant);  ui**->**listWidget**->**addItem(qLanguagesListItem);  } **else** {  qDebug() **<<** "Rejected!";  }  }#include "mainwindow.h"  #include "ui\_mainwindow.h"  #include <QDebug>  #include <QCloseEvent>  #include <QString>  #include <iostream>  #include <QTimer>  #include <QTime>  #include <https\_client.h>  #include <postcodeinfo.h>  using namespace std;  MainWindow::MainWindow(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  QTimer \*timer = new QTimer(this);  connect(timer, SIGNAL(timeout()), this, SLOT(showTime()));  timer->start(500);  ui->setupUi(this);  setWindowTitle("Lab 7");  ui->progressBar->setValue(0);  europRegions.push\_back("All region");  }  void MainWindow::showTime()  {  QTime time = QTime::currentTime();  QString text = time.toString("hh:mm");  if ((time.second() % 2) == 0)  text[2] = ' ';  ui->lcdNumber->display(text);  }  MainWindow::~MainWindow()  {  delete ui;  }  void MainWindow::closeEvent(QCloseEvent \*event)  {  // use this to cleanup allocated eventmemory  foreach (PostCodeInfo \* cur, postCodes) {  delete cur;  }  qDebug() << "Window closed!";  event->accept();  }  void MainWindow::on\_pushButton\_clicked()  {  int count = ui->spinBox->value();  loadPostCodesToVector(count);  loadAllPostCodesToListWidget();  }  void MainWindow::on\_listWidget\_itemSelectionChanged()  {  qDebug() << ">>Selection changed<<";  auto items = ui->listWidget->selectedItems();  auto hasSelected = items.size() > 0;  if (hasSelected) {  ui->postCodeLabel->setEnabled(true);  ui->countryLabel->setEnabled(true);  ui->latitudeLabel->setEnabled(true);  ui->longitudeLabel->setEnabled(true);  ui->europeanRegionLabel->setEnabled(true);  ui->adminDistrictLabel->setEnabled(true);  PostCodeInfo postCode = items[0]->data(Qt::UserRole).value<PostCodeInfo>();  ui->label->setText(postCode.getPostCode());  ui->label\_2->setText(postCode.getCountry());  ui->label\_3->setText(QString::number(postCode.getLongitude(), 'f', 4));  ui->label\_4->setText(QString::number(postCode.getLatitude(), 'f', 4));  ui->label\_5->setText(postCode.getEuropeanElectoralRegion());  ui->label\_6->setText(postCode.getAdminDistrict());  } else {  ui->postCodeLabel->setEnabled(false);  ui->countryLabel->setEnabled(false);  ui->latitudeLabel->setEnabled(false);  ui->longitudeLabel->setEnabled(false);  ui->europeanRegionLabel->setEnabled(false);  ui->adminDistrictLabel->setEnabled(false);  ui->label->setText("");  ui->label\_2->setText("");  ui->label\_3->setText("");;  ui->label\_4->setText("");;  ui->label\_5->setText("");;  ui->label\_6->setText("");;  }  }  void MainWindow::loadPostCodesToVector(int count)  {  ui->progressBar->setValue(0);  double status = 0;  ui->label\_7->setText("Loading...");  for (int i = 0; i < count; i++) {  PostCodeInfo \* code = new PostCodeInfo();  QString buffer = QString::fromStdString(httpsClient());  code->loadFromJson(buffer);  if (code->getEuropeanElectoralRegion().size() == 0) {  qDebug() << ">>Incorrect data<<";  continue;  }  postCodes.push\_back(code);  status += 100.0 / count;  ui->progressBar->setValue(status);  }  ui->progressBar->setValue(100);  ui->label\_7->setText("Loading complete");  }  void MainWindow::loadAllPostCodesToListWidget(void)  {  ui->listWidget->clear();  foreach (PostCodeInfo \* cur, postCodes) {  QVariant qVariant;  qVariant.setValue(\*cur);  QListWidgetItem \*qPostCodeListItem = new QListWidgetItem();  qPostCodeListItem->setText(cur->getPostCode());  qPostCodeListItem->setData(Qt::UserRole, qVariant);  ui->listWidget->addItem(qPostCodeListItem);  }  loadComboBox();  }  void MainWindow::loadPostCodeToListWidget(PostCodeInfo \* postCode)  {  QVariant qVariant;  qVariant.setValue(\*postCode);  QListWidgetItem \*qPostCodeListItem = new QListWidgetItem();  qPostCodeListItem->setText(postCode->getPostCode());  qPostCodeListItem->setData(Qt::UserRole, qVariant);  ui->listWidget->addItem(qPostCodeListItem);  }  void MainWindow::loadComboBox()  {  bool cont = false;  QString value = ui->comboBox->currentText();  ui->comboBox->clear();  foreach (PostCodeInfo \* cur, postCodes) {  cont = false;  foreach (QString str, europRegions) {  if (str == cur->getEuropeanElectoralRegion()) cont = true;  }  if (cont) continue;  else europRegions.push\_back(cur->getEuropeanElectoralRegion());  }  foreach (QString str, europRegions) {  ui->comboBox->addItem(str);  }  ui->comboBox->setCurrentText(value);  }  void MainWindow::on\_comboBox\_currentIndexChanged(const QString &arg1)  {  ui->listWidget->clear();  for(PostCodeInfo \* code: postCodes){  if(code->getEuropeanElectoralRegion() == arg1 || arg1 == "All region") {  loadPostCodeToListWidget(code);  }  }} |
| https\_client.h |
| #ifndef HTTPS\_CLIENT\_H  #define HTTPS\_CLIENT\_H  #include <iostream>  std::string httpsClient(void);  std::string URI\_getProtocol(std::string uri);  std::string URI\_getHost(std::string uri);  std::string URI\_getURL(std::string uri);  #endif // HTTPS\_CLIENT\_H |
| https\_client.cpp |
| #include <iostream>  #include <progbase-cpp/net.h>  #include <QDebug>  #include <https\_client.h>  using namespace std;  using namespace progbase::net;  string httpsClient(void) {  string buffer;  TcpClient client;  NetMessage message(100);  string uri = "http://api.postcodes.io/random/postcodes";  try {  if(URI\_getProtocol(uri) == "http") {  client.connect(IpAddress(Ip::resolveHostname(URI\_getHost(uri)), 80));  }  else if (URI\_getProtocol(uri) == "https"){  auto ipAddress = IpAddress(Ip::resolveHostname(URI\_getHost(uri)), 443);  Ssl ssl;  client.connect(ipAddress, ssl);  } else throw "Incorrect URI";  message.setDataString(  "GET " + URI\_getURL(uri) + " HTTP/1.0\r\n"  "User-Agent: Valzavator\r\n"  "Host: " + URI\_getHost(uri) + "\r\n\r\n");  client.send(message);  do {  client.receive(message);  string str = message.dataAsString();  buffer += str;  } while (!message.isEmpty());  size\_t found = buffer.find("\r\n\r\n");  buffer.erase(buffer.begin(), buffer.begin() + found + 4);  } catch(NetException const & exc) {  cerr << exc.what() << endl;  } catch(const char \* er) {  cerr << er << endl;  }  if (buffer.size() != 0) qDebug() << ">>Request get<<";  return buffer;  }  string URI\_getProtocol(string uri) {  if (uri.find("https://") != string::npos) return "https";  else if (uri.find("http://") != string::npos) return "http";  else return "Incorrect URI";  }  std::string URI\_getHost(string uri) {  string host;  int indexAf;  int indexBf = uri.find("https://");  if (indexBf == (int)string::npos) {  indexBf = uri.find("http://");  if (indexBf == (int)string::npos) return "Incorrect URI";  else indexBf += 7;  } else indexBf += 8;  indexAf = uri.find("/", indexBf);  if (indexAf != (int)string::npos) host = uri.substr(indexBf, indexAf - indexBf);  else host = uri.substr(indexBf, uri.size() - indexBf);  return host;  }  std::string URI\_getURL(string uri) {  string host = URI\_getHost(uri);  int index = uri.find(host) + host.size();  string URL = uri.substr(index, uri.size() - index);  return URL;  } |

**Приклад результатів**

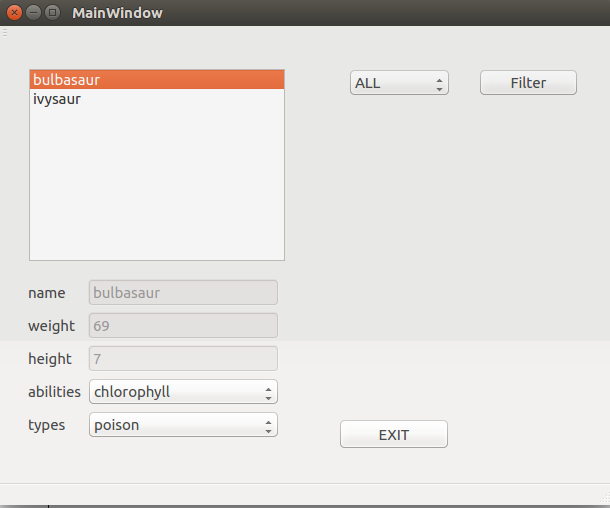
**Завдання 1**

Був реалізований модуль, що використовує TCP клієнт для підключення до [тестового web-сервера](https://api.github.com/). TCP клієнт реалізований таким чином, щоб користувачу достатньо ввести URI серверу, а програма самостійно розпарсить його і сформує запит. TCP клієнт здатний взаємодіяти з web-серверами за допомогою протоколів HTTP та HTTPS.

**Завдання 2**

Сервер виконує N запитів(де N задається користувачем) та парсить отримані поштові коди із JSON формату й зберігає дані в клас PostCodeInfo.

**Завдання 3** Була створенна вікнна програма Qt, що дозволяє переглядати список отриманих покемонів із їх характеристиками (у формах) та фільтрувати відображення цього списку по типах. Користувач може обирати значення критерію та в будь-який момент часу запросити від web-сервера ще покемонів.



**Висновок**

Виконавши дану лабораторну роботу я навчитися користуватись публічними [API (Application Programming Interface)](https://en.wikipedia.org/wiki/Application_programming_interface) web-сайтів у мережі Інтернет за допомогою TCP клієнта та парсерів форматів даних.

Я навчився створювати віконні програми в ОС Linux за допомогою фреймворку Qt та редактора Qt Creator.

Були отримані навички роботи з QDialog, QListWidget, Label, Button, та іншими елементами віконних програм, їх програмування та взаємодія між собою.