

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 №** 8

з дисципліни “ Основи програмування ”

Тема “**HTTP сервер** ”

|  |  |  |
| --- | --- | --- |
| Виконав  студент I курсу  групи КП-62  Лук’янець Михайло Олександрович  (*прізвище, ім’я, по батькові*)  варіант №15 |  | 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)** | |  |  |  | | Сумарний бал:   |  | | --- | |  | |

Київ 2017

**Мета роботи**

Навчитися створювати простий web-сервер із взаємодією через протокол HTTP. Навчитися оформляти віддалений Git репозиторій із кодом, перевіряти збірки проекту за допомогою безперервної інтеграції та створювати документацію коду проекту.

**Постановка завдання**

#### **Завдання №1. Створення web-сервера**

Створити TCP-сервер, що слухає запити на підключення TCP клієнтів і взаємодіє з ними за допомогою протоколу HTTP. Сервер повинен обробляти лише HTTP GET запити і повертати коректно-сформовані HTTP-відповіді. Інтерфейс сервера дозволяє звертатися до нього по HTTP-шляхах:

* / - відправити у відповідь JSON-об'єкт із інформацією про сервер:
* {  
   "title": "%SERVER\_NAME%",   
   "developer": "%YOUR\_NAME\_SURNAME%",   
   "time": "%SERVER\_TIME%"  
  }
* /favorites - JSON-список із об'єктів, що позначають список ваших улюблених хімічних речовин. Кожен такий об'єкт має мати декілька полів, серед яких обов'язково має бути числове поле id із унікальним для кожного об'єкта значенням.
* /favorites?{key}={value} - JSON-підсписок тих об'єктів зі списку /favorites, у яких поле {key} має значення {value}. Ключ {key}, по якому відбувається фільтрація списку, обрати довільно.
* /favorites/{id} - JSON-об'єкт зі списку /favorites за ідентифікатором з числовим значенням {id}. Якщо об'єкта із таким значенням id не було знайдено, повертати у відповідь HTTP статус 404 Not Found.
* /file - зчитати вміст файлу data.txt, який попередньо розмістити у директорії data/ проекту. Відправити клієнту JSON-об'єкт, що містить у собі інформацію про назву, розмір (у байтах) і текстовий вміст файлу.
* /file/data - у відповідь клієнту відправити JSON-список всіх слів у файлі (файл data.txt).

#### **Завдання №2. Створення документації проекту за допомогою Doxygen**

##### **Коментування коду у форматі Doxygen**

Виконати коментування інтерфейсів модулів у форматі Doxygen ([приклад](https://github.com/PublicHadyniak/libprogbase/blob/master/include/progbase/list.h)).

Для цього кожен заголовочний файл повинен починатися на коментар виду:

*/\*\* @file  
 @brief Some text about this header  
\*/*

Перед кожною визначеною структурою даних має бути коментар:

*/\*\*  
 @struct List  
 @brief defines a generic List data type  
\*/*

Перед кожною функцією інтерфейсу також розмістити коментар із інформацією про призначення функції, її параметри та результат, що повертається:

*/\*\*  
 @brief remove item from list by value  
 @param ref - reference to memory from where to read compared value  
 @returns true if success removal  
\*/*  
**bool** **List\_remove**(List \* self, **void** \* ref);

##### **Генерація HTML-документації утилітою Doxygen**

Встановити Doxygen:

sudo apt-get install doxygen graphvis

Перейти у директорію проекта і згенерувати файл із конфігурацією (Doxyfile):

doxygen -g

Задати у Doxyfile такі значення для вказаних налаштувань:

PROJECT\_NAME = "Project Name Here"  
EXTRACT\_ALL = NO  
FILE\_PATTERNS = \*.c \*.cpp \*.h  
RECURSIVE = YES

Згенерувати документацію командою:

doxygen Doxyfile

Doxygen створить директорію docs/ із піддиректоріями html/ і latex/, які міститимуть документацію у різних форматах (не забудьте додати docs/ у .gitignore).

Для перегляду згенерованої документації перейдіть у директорію html/ і відкрийте файл index.html у веб-браузері.

#### **Завдання №3. Безперервна інтеграція (Continuous Integration)**

##### **CMake**

Налаштувати збірку проекта із сервером за допомогою утиліти cmake та файлу CMakeLists.txt ([приклад](https://github.com/PublicHadyniak/c-project-template/blob/master/CMakeLists.txt)).

##### **GitHub репозиторій**

Зареєструватись на сайті [Github](https://github.com/) та створити віддалений публічний репозиторій, у який надіслати файли проекту. Додатково створити у корені проекту файл README.md, у якому у форматі [Markdown](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) описати варіант завдання лабораторної роботи та короткий опис модулів проекту.

##### **Додаткові скрипти для збірки проекта**

Створити у проекті директорію scripts/ і додати у неї файл install-libprogbase.sh:

**#!/bin/sh**  
wget -O libprogbase.zip https://github.com/PublicHadyniak/libprogbase/archive/master.zip  
unzip libprogbase.zip -d libprogbase  
cd libprogbase/libprogbase-master ; sudo make install  
cd ../../ ; rm libprogbase/ libprogbase.zip -rf

##### **Travis Continuous Integration**

Додати у кореневу директорію проекта файл .travis.yml із вмістом:

sudo: required  
language: c  
os:  
 - linux  
compiler:   
 - gcc  
  
before\_install:  
 - sudo apt-get install libssl-dev  
 - sh scripts/install-libprogbase.sh  
  
script:   
 - cmake .   
 - make .

Зареєструватись на сайті [Travis-CI](https://travis-ci.org/) та синхронізувати його із профілем Github. Налаштувати у Travis-CI збірку репозиторія зі сервером. Після кожного push комітів у віддалений репозиторій GitHub, Travis-CI збиратиме проект і оновлюватиме статус його збірки (passing | failing). Додати у файл README.md репозиторія мітку із зображенням статусу збірки та посиланнями на історію збірок :

[![Travis-CI][travis-badge]][travis-builds]  
  
[travis-badge]: https://travis-ci.org/PublicHadyniak/libprogbase.svg?branch=master  
[travis-builds]: https://travis-ci.org/PublicHadyniak/libprogbase/builds

де замінити PublicHadyniak на ваш нікнейм на GitHub, а libprogbase на назву репозиторія із проектом сервера.

**Тексти коду програм**

|  |
| --- |
| main.cpp |
| #include <iostream>  #include <time.h>  #include <progbase-cpp/net.h>  #include <sstream>  #include <vector>  #include <iterator>  #include "include/processor.h"  **using namespace** std;  **using namespace** progbase::net;  **int** main(**int** argc, **char** \* argv[]) {  vector<Chemical> chemicals;  chemicals.push\_back(Chemical("Oxygen", 16, "Oxydyzer"));  chemicals.push\_back(Chemical("Lysergsäurediethylamid", 323.4, "Psycho"));  chemicals.push\_back(Chemical("DihidrohenaMonooksid", 18, "Solvent"));  chemicals.push\_back(Chemical("tetrahydrocannabinol", 314, "Psycho"));  chemicals.push\_back(Chemical("methylcarbinol", 46, "Depressant"));  **const int** serverPort = 8006;  TcpListener listener;  NetMessage message(10000);  **try** {  listener.bind(IpAddress("127.0.0.1", serverPort));  listener.start();  **while** (**true**) {  cout << ">> Waiting for clients at " << serverPort << "..." << endl;  TcpClient \* client = listener.accept();  client->receive(message);  cout << ">> Received: " << endl << message.dataAsString() << endl;  std::string data = message.dataAsString();  **auto** tokens = getTokens(data);  **int** status = analyzeTokens(tokens);  string backMessage = makeResponse(tokens, status, chemicals);  cout << "RESOPNSE \n\n" << backMessage;  message.setDataString(backMessage);  client->send(message);  cout << ">> Response sent." << endl;  **delete** client;  }  } **catch**(NetException **const** & exc) {  cerr << exc.what() << endl;  }  } |

|  |
| --- |
| processor.cpp |
| //  // Created by tgifr on 01.05.17.  //  #include "../include/processor.h"  #include <time.h>  #include <jansson.h>  #include <cstring>  **const** string filepath = "data/text.txt";  vector<std::string> getTokens(std::string str){  istringstream iss(str);  vector<std::string> tokens{istream\_iterator<std::string>{iss},  istream\_iterator<std::string>{}};  **return** tokens;  }  **int** analyzeTokens(vector<std::string> tokens){  **int** size = tokens.size();  **if**(size > 1){  std::string first = tokens.at(0);  std::string second = tokens.at(1);  **if**(first.compare("GET")) **return** *UNKNOWN*;  **if**(!second.compare("/file") ) **return** *FILE\_INF*;  **if**(!second.compare("/") ) **return** *ROOT*;  **if**(!second.compare("/file/data")) **return** *FILE\_DATA*;  **if**(!second.compare("/favorites")) **return** *FAVORITES*;  std::size\_t found = second.find("/favorites?");  **if** (found!=std::string::npos) **return** *FAVORITES\_BY\_VALUE*;  found = second.find("/favorites/");  **if** (found!=std::string::npos) **return** *FAVORITES\_BY\_ID*;  }  **return** *UNKNOWN*;  }  string makeResponse(vector<std::string> tokens, **int** analyzingStatus, vector<Chemical> chemicals) {  **switch**(analyzingStatus){  **case** *UNKNOWN*:  **return** make404Response();  **case** *ROOT*:  **return** makeRootResponse();  **case** *FAVORITES*:  **return** makeFavoritesResponse(chemicals);  **case** *FAVORITES\_BY\_ID*:  **return** makeFavoritesResponseById(chemicals, tokens);  **case** *FAVORITES\_BY\_VALUE*:  **return** makeFavoritesResponseByValue(chemicals, tokens);  **case** *FILE\_DATA*:  **return** makeResponseFileData();  **case** *FILE\_INF*:  **return** makeResponseFileInf();  }  }  string makeResponseFileInf(){  string fileDataResponse = "HTTP/1.1 200 OK\n";  fileDataResponse += "Connection: Closed\n"  "Content-Type: text/html; charset=iso-8859-1\r\n\r\n";  string data = readFileContents(filepath.c\_str());  json\_t \* json = json\_object();  json\_object\_set\_new(json, "name", json\_string( "text.txt"));  json\_object\_set\_new(json, "size", json\_integer(getFileSize(filepath.c\_str())));  json\_object\_set\_new(json, "content", json\_string(data.c\_str()));  **char** \* jsonString = json\_dumps(json, JSON\_INDENT(2));  fileDataResponse += jsonString;  **return** fileDataResponse;  }  string makeResponseFileData(){  string fileDataResponse = "HTTP/1.1 200 OK\n";  fileDataResponse += "Connection: Closed\n"  "Content-Type: text/html; charset=iso-8859-1\r\n\r\n";  string data = readFileContents(filepath.c\_str());  **auto** tokens = getTokens(data);  json\_t \* json = json\_object();  json\_t \* arr = json\_array();  **for** (**auto** str : tokens) // access by reference to avoid copying  {  json\_t \* wordJ = json\_object();  json\_object\_set\_new(wordJ, "word", json\_string(str.c\_str()));  json\_array\_append(arr, wordJ);  }  json\_object\_set\_new(json, "words", arr);  **char** \* jsonString = json\_dumps(json, JSON\_INDENT(2));  fileDataResponse += jsonString;  **return** fileDataResponse;  }  **char** \* readFileContents(**const char** \* fileName) {  FILE \* textFile = fopen(fileName, "r");  **if** (textFile == NULL) {  **return** NULL;  }  fseek(textFile, 0, SEEK\_END);  **long** size = ftell(textFile);  fseek(textFile, 0, SEEK\_SET);  **char** \* fcontent = (**char** \* ) malloc(size);  fread(fcontent, 1, size, textFile);  fclose(textFile);  **return** fcontent;  }  **int** getFileSize(**const char** \* fileName) {  FILE \* textFile = fopen(fileName, "r");  **if** (textFile == NULL) {  **return** 0;  }  fseek(textFile, 0, SEEK\_END);  **return** ftell(textFile);  }  string makeFavoritesResponseByValue(vector<Chemical> chemicals, vector<std::string> tokens){  string favoritesResponse = "HTTP/1.1 200 OK\n"  "Connection: Closed\r\n\r\n";  **auto** str = tokens.at(1);  size\_t last\_index = str.find("?");  string result = str.substr(last\_index + 1);  cout << "SUBSTR\n\n\n"<< result.substr(0,10);  **if**(result.substr(0,9).compare("influence"))  **return** make404Response();  last\_index = str.find("=");  result = str.substr(last\_index + 1);  json\_t \* json = json\_object();  json\_t \* arr = json\_array();  **for** (**int** i = 0; i < chemicals.size(); i++) {  Chemical chem = chemicals.at(i);  **if**(chem.influence.compare(result)) **continue**;  json\_t \* chemJ = json\_object();  json\_object\_set\_new(chemJ, "name", json\_string(chem.name.c\_str()));  json\_object\_set\_new(chemJ, "weight", json\_real(chem.weight));  json\_object\_set\_new(chemJ, "influence", json\_string(chem.influence.c\_str()));  json\_array\_append(arr, chemJ);  }  json\_object\_set\_new(json, "chemicals", arr);  **char** \* jsonString = json\_dumps(json, JSON\_INDENT(2));  favoritesResponse += jsonString;  **return** favoritesResponse;  }  string makeFavoritesResponseById(vector<Chemical> chemicals, vector<std::string> tokens){  string favoritesResponse = "HTTP/1.1 200 OK\n"  "Connection: Closed\r\n\r\n";  **auto** str = tokens.at(1);  size\_t last\_index = str.find\_last\_not\_of("0123456789");  string result = str.substr(last\_index + 1);  **int** index = stoi(result);  **if**(index < 0 || index >= chemicals.size()){  **return** make404Response();  }  json\_t \* json = json\_object();  json\_t \* arr = json\_array();  Chemical chem = chemicals.at(index);  json\_t \* chemJ = json\_object();  json\_object\_set\_new(chemJ, "name", json\_string(chem.name.c\_str()));  json\_object\_set\_new(chemJ, "weight", json\_real(chem.weight));  json\_object\_set\_new(chemJ, "influence", json\_string(chem.influence.c\_str()));  json\_array\_append(arr, chemJ);  json\_object\_set\_new(json, "chemicals", arr);  **char** \* jsonString = json\_dumps(json, JSON\_INDENT(2));  favoritesResponse += jsonString;  **return** favoritesResponse;  }  string makeFavoritesResponse(vector<Chemical> chemicals){  string favoritesResponse = "HTTP/1.1 200 OK\n"  "Connection: Closed\r\n\r\n";  json\_t \* json = json\_object();  json\_t \* arr = json\_array();  **for** (**int** i = 0; i < chemicals.size(); i++) {  Chemical chem = chemicals.at(i);  json\_t \* chemJ = json\_object();  json\_object\_set\_new(chemJ, "name", json\_string(chem.name.c\_str()));  json\_object\_set\_new(chemJ, "weight", json\_real(chem.weight));  json\_object\_set\_new(chemJ, "influence", json\_string(chem.influence.c\_str()));  json\_array\_append(arr, chemJ);  }  json\_object\_set\_new(json, "chemicals", arr);  **char** \* jsonString = json\_dumps(json, JSON\_INDENT(2));  favoritesResponse += jsonString;  **return** favoritesResponse;  }  string makeRootResponse(){  string rootResponse = "HTTP/1.1 200 OK\n";  time\_t rawtime;  **struct** tm \* timeinfo;  time ( &rawtime );  timeinfo = localtime ( &rawtime );  rootResponse += "Connection: Closed\n"  "Content-Type: text/html; charset=iso-8859-1\r\n\r\n";  json\_t \* json = json\_object();  json\_object\_set\_new(json, "title", json\_string( "RuntimeJunkie 3"));  json\_object\_set\_new(json, "developer", json\_string("TGIfr"));  json\_object\_set\_new(json, "time", json\_string(asctime(timeinfo)));  **char** \* jsonString = json\_dumps(json, JSON\_INDENT(2));  rootResponse += jsonString;  **return** rootResponse;  }  string make404Response(){  string response404 = "HTTP/1.1 404 Not Found\n";  // "Date: ";  // time\_t rawtime;  // struct tm \* timeinfo;  // time ( &rawtime );  // timeinfo = localtime ( &rawtime );  // response404 += asctime(timeinfo);  // response404 += "Server: Apache/2.2.14 (Win32)\n"  // "Content-Length: 230\n"  // "Connection: Closed\n"  // "Content-Type: text/html; charset=iso-8859-1\r\n\r\n"  // "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 2.0//EN\">\n"  // "<html>\n"  // "<head>\n"  // " <title>404 Not Found</title>\n"  // "</head>\n"  // "<body>\n"  // " <h1>Not Found</h1>\n"  // " <p>The requested URL /t.html was not found on this server.</p>\n"  // "</body>\n"  // "</html>";  **return** response404;  } |

|  |
| --- |
| processor.h |
| //  // Created by tgifr on 01.05.17.  //  #ifndef HTTPSERVERLAB\_PROCESSOR\_H  #define HTTPSERVERLAB\_PROCESSOR\_H  #include <iostream>  #include <sstream>  #include <vector>  #include <iterator>  #include "Chemical.h"  **using namespace** std;  /\*\*  \* @brief enum of responses from analyzeTokens  \*/  **enum** {  *OK*,  *UNKNOWN*,  *WRONG\_INDEX*,  *ROOT*,  *FAVORITES*,  *FILE\_INF*,  *FILE\_DATA*,  *FAVORITES\_BY\_ID*,  *FAVORITES\_BY\_VALUE*  };  /\*\*  \*  \* @param str - string to divide  \* @return vector of separated words  \*/  vector<std::string> getTokens(std::string str);  /\*\*  \*  \* @param tokens - vector of separated words  \* @return int representing command  \*/  **int** analyzeTokens(vector<std::string> tokens);  /\*\*  \*  \* @param chemicals vector of chemicals to be turned into response  \* @return response on /favorites  \*/  string makeFavoritesResponse(vector<Chemical> chemicals);  /\*\*  \* @brief decides what response to do  \* @param tokens - words of http-get  \* @param analyzingStatus - value returned by analyzeTokens  \* @param chemicals - vector of chemicals  \* @return final response  \*/  string makeResponse(vector<std::string> tokens, **int** analyzingStatus, vector<Chemical> chemicals);  /\*\*  \*  \* @param chemicals vector of chemicals to be turned into response  \* @return response on /favorites/{id}  \*/  string makeFavoritesResponseById(vector<Chemical> chemicals, vector<std::string> tokens);  /\*\*  \*  \* @param chemicals vector of chemicals to be turned into response  \* @return response on /favorites?influence={value}  \*/  string makeFavoritesResponseByValue(vector<Chemical> chemicals, vector<std::string> tokens);  /\*\*  \*  \* @param fileName  \* @return all data from file in str  \*/  **char** \* readFileContents(**const char** \* fileName);  /\*\*  \*  \* @param fileName  \* @return size of file  \*/  **int** getFileSize(**const char** \* fileName);  /\*\*  \*  \* @return response to /file  \*/  string makeResponseFileInf();  /\*\*  \*  \* @return response to /file/data  \*/  string makeResponseFileData();  /\*\*  \*  \* @return 404 statused empty response  \*/  string make404Response();  /\*\*  \*  \* @return response to /  \*/  string makeRootResponse();  #endif //HTTPSERVERLAB\_PROCESSOR\_H |

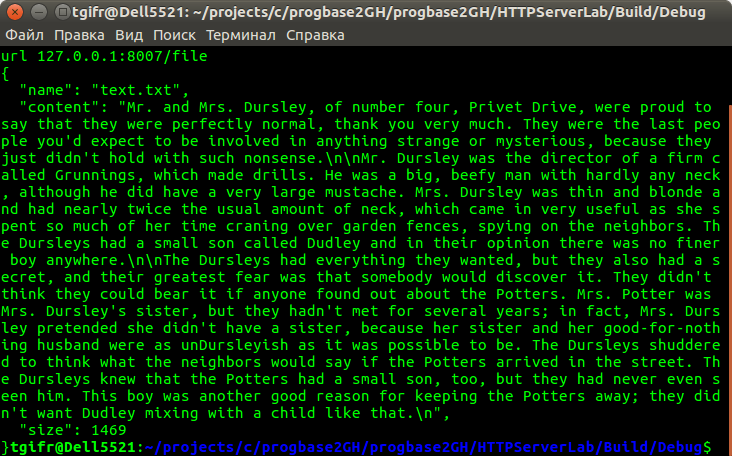
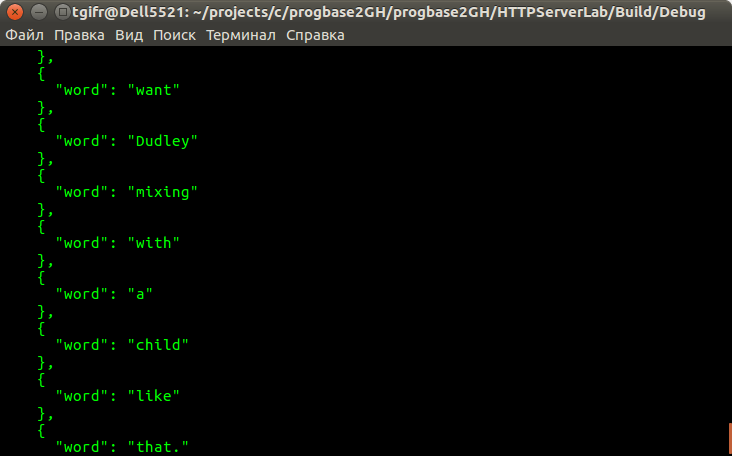
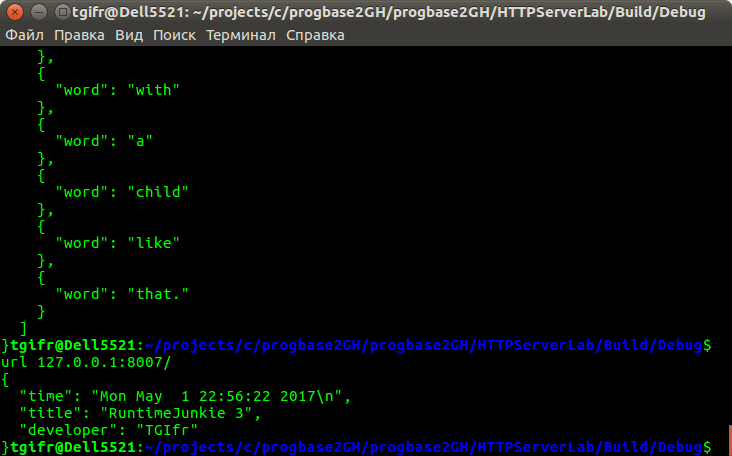
|  |
| --- |
| chemical.cpp |
| //  // Created by tgifr on 01.05.17.  //  #include "../include/Chemical.h"  Chemical::Chemical(string name , **double** weight, string influence){  **this**->influence = influence;  **this**->name = name;  **this**->weight = weight;  } |

|  |
| --- |
| chemical.h |
| //  // Created by tgifr on 01.05.17.  //  #ifndef HTTPSERVERLAB\_CHEMICAL\_H  #define HTTPSERVERLAB\_CHEMICAL\_H  #include <iostream>  #include <sstream>  #include <vector>  #include <iterator>  #include "Chemical.h"  **using namespace** std;  /\*\*  \* @class - describes a chemical  \*/  **class** Chemical{  **public**:  string name;  **double** weight;  string influence;  /\*\*  \*  \* @brief - simple constructor  \*/  Chemical(string , **double**, string);  };  #endif //HTTPSERVERLAB\_CHEMICAL\_H |

|  |
| --- |
| **install-libprogbace.sh** |
| **#!/bin/sh**  **wget -O libprogbase.zip https://github.com/PublicHadyniak/libprogbase/archive/master.zip**  **unzip libprogbase.zip -d libprogbase**  **cd libprogbase/libprogbase-master ; sudo make install**  **cd ../../ ; rm libprogbase/ libprogbase.zip -rf**  **wget -O libprogbase-cpp.zip https://github.com/PublicHadyniak/libprogbase-cpp/archive/master.zip**  **unzip libprogbase-cpp.zip -d libprogbase-cpp**  **cd libprogbase-cpp/libprogbase-master ; cmake src/..; sudo make install**  **cd ../../ ; rm libprogbase/ libprogbase.zip -rf** |

|  |
| --- |
| **.travis.yml** |
| **sudo: required**  **language: c++**  **os:**  **- linux**  **compiler:**  **- gcc**  **before\_install:**  **- sudo apt-get install libssl-dev**  **- sudo apt-get install libjansson-dev**  **- sh scripts/install-libprogbase.sh**  **script:**  **- cmake .**  **- make .** |

**Приклади результатів**



**Висновки**

Виконавши дану лабораторну роботу було створено HTTP сервер формлено віддалений Git репозиторій із кодом, перевірено збірки проекту за допомогою безперервної інтеграції та створено документацію коду проекту.