ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

(СИБГУТИ)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

“Технологии разработки программного обеспечения”

на тему

ТЕЛЕФОННЫЙ СПРАВОЧНИК

Выполнил студент	Бородин Юрий Дмитриевич	
	Ф.И.О	
Группы	ЗП-91	
Работу принял		Пудов Сергей Григорьевич
	подпись	
Оценка		

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

1. ПОСТАНОВКА ЗАДАЧИ 4

2. ЭТАПЫ РАБОТЫ НАД ПРОЕКТОМ 5

ЗАКЛЮЧЕНИЕ 14

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 15

ВВЕДЕНИЕ

Цель работы: освоение курса по предмету «Технологии разработки программного обеспечения». Получение опыта использования системы контроля версий GIT, системы сборки приложений make, применения юнит-тестирования и использование системы Travis CI для автоматизации тестирования изменений.

Задачи:

- Разработка программы согласно ТЗ
- Ведение репозитория программы на Github
- Создание тестов для проверки работы программы
- Подключение средств для непрерывной интеграции

ПОСТАНОВКА ЗАДАЧИ

В рамках изучения дисциплины «Технологии разработки программного обеспечения» необходимо написать программу «Телефонный справочник». Данная программа должна удовлетворять следующим условиям:

Функциональность:

- Входные данные для работы программы считываются из файла
- Одна единица данных содержит в себе поля «Имя», «Город», «Номер телефона»
- Файл с данными форматируется заранее заданным образом
- В программе есть возможность дополнить данные вводом с клавиатуры
- Программа выполняет поиск по полю «Номер телефона». Поиск возможен по точному номеру телефона или по заданному диапазону. Поиск выводит на экран все совпадения, либо сообщение об отсутствии совпадений
- В программе реализована функциональность меню
- Программа работает с помощью командной строки (CLI)

В процессе выполнения на всех этапах разработки необходимо проводить автоматизированное тестирование кода на предмет наличия ошибок компиляции, а также юнит-тестирование различных участков кода на предмет корректного выполнения функций.

На всех этапах разработки необходимо использовать систему контроля версий Git, с подключением удаленного репозитория на сервисе Github. Данная система позволяет следить за изменениями кода, а также предоставляет удобный базовый функционал для командной разработки.

Для автоматизированного тестирования кода необходимо использовать систему Travis CI. Данная система позволяет проводить автоматизированное тестирование новых версий на предмет наличия ошибок сборки и тестов.

ЭТАПЫ РАБОТЫ НАД ПРОЕКТОМ

Для начала работы над проектом необходимо создать локальный репозиторий Git. После создания репозитория в корне необходимо создать файл .gitignore, для игнорирования файлов системой Git. В своем проекте я поместил туда строку вида *.exe для игнорирования исполняемых файлов.

Также нужно создать удаленный репозиторий на сайте Github. Для этого воспользуемся веб-интерфейсом сайта. После создания репозитория можно сразу же отправить туда наш локальный проект.

git add . – эта команда добавит в Git все файлы проекта, за исключением .gitignore

git commit –m “initial commit” – данная команда зафиксирует изменения в проекте

git add remote origin https://github.com/dev0nizer/coursework_trpo.git - эта команда подключит к локальному репозиторию удаленный

git push origin master – наконец, эта команда отправит локальные изменения на удаленный репозиторий в ветку master

В процессе разработки мной были использованы различные инструменты системы git, такие как создание и слияние веток, работа с удаленным репозиторием, откат изменений в ветке.

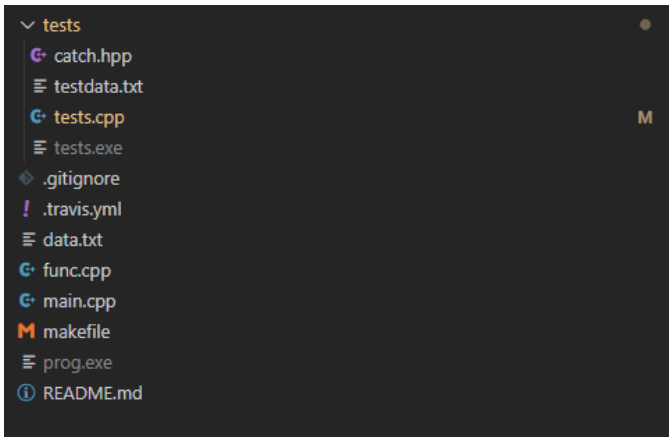
Для тестирования участков кода мной были написаны юнит-тесты для функций в моей программе. Файл с тестами был выделен в отдельную директорию, в которой также содержится тестовая база данных. Для тестирования был использован фреймворк «catch2».

Для удобной сборки используется утилита make. В корневой директории проекта был создан makefile, в котором предусмотрена сборка проекта. Утилита make при запуске автоматически использует makefile из данной директории.

Для интеграции с системой Travis CI в корне проекта создан файл .travis.yml. Данный файл используется системой Travis CI для автоматического тестирования приложения.

В процессе разработки были создан файл main.cpp с телом программы, файл func.cpp, содержащий в себе функции для работы программы и директория ./tests. В данной директории содержится файл tests.cpp с тестами функций из func.cpp и testdata.txt с тестовыми входными данными.

Окончательная структура проекта после завершения работы:



Инструменты Travis CI доступны через сайт. После подключения репозитория Travis будет использовать файл .travis.yml для проверки проекта.

После подключения Travis CI к описанию репозитория можно добавить динамическую иконку прохождения тестов. Код для вставки можно получить на сайте Travis CI и вставить в файл README.md. После commit в описании появится иконка с текущим статусом прохождения тестов.

Исходный код makefile:

```
all: hello

hello: main.o test.o

main.o:
    g++ -std=c++11 main.cpp -o prog

test.o:
    g++ -std=c++11 ./tests/tests.cpp -o ./tests/tests

test:
    ./tests/tests

run:
    ./prog
```

.travis.yml

```
language: cpp
compiler: g++
script:
  - make
  - make test
```

.gitignore

```
**/.vscode
*.exe
```

Func.cpp

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

class Citizen
{
    std::string name;
    std::string city;
    int phonenumber;
    Citizen* next;
    public:
        void Add(std::string add_name, std::string add_city, int add_phonenumber)
        {
            name = add_name;
            city = add_city;
            phonenumber = add_phonenumber;
        }

        void AddFromKeyboard()
        {
            std::cout << "Enter name" << std::endl;
            std::cin >> name;
            std::cout << "Enter city" << std::endl;
            std::cin >> city;
            std::cout << "Enter phone number" << std::endl;
            std::cin >> phonenumber;
        }

        int Getnumber()
        {
            return phonenumber;
        }

        void PutMember()
        {
            std::cout << name << " " << city << " " << phonenumber << std::endl;
        }
        std::string GetMember()
        {
            std::string temp;
            temp.append(name);
            temp.append("|");
            temp.append(city);
            temp.append("|");
            temp.append(std::to_string(phonenumber));
            return temp;
        }

};
```

```
std::vector <Citizen> LoadFromFile(std::string filepath)
{
    std::vector <Citizen> temppeople;
    std::ifstream file(filepath);
    if (file.is_open())
    {
        while(!file.eof()) // push from file to vector
        {
            std::string templine;
            std::string tempname;
            std::string tempcity;
            int tempphonenumber;
            std::getline(file, templine);
            int pos = templine.find("|");
            tempname = templine.substr(0, pos);
            templine.erase(0, pos + 1);
            pos = templine.find("|");
            tempcity = templine.substr(0, pos);
            templine.erase (0, pos +1);
            tempphonenumber = std::stoi(templine);
            Citizen temp;
            temp.Add(tempname, tempcity, tempphonenumber);
            temppeople.push_back(temp);
        }
        std::cout << "Successfully loaded from file!" << std::endl;
        return temppeople;
    }
    else
    {
        std::cout << "Error! Cannot open file";
        return temppeople;
    }
}

bool findbyexactphone(std::vector<Citizen> inputvector, int inputnumber)
{
    bool found = false;
    int i=0;
    for (i=0; i<inputvector.size(); i++)
    {
        if (inputvector[i].Getnumber() == inputnumber)
        {
            found = true;
            inputvector[i].PutMember();
        }
    }
    if (!found)
    {
        std::cout << "Member not found" << std::endl;
        return false;
    }
    else
    {
        return true;
    }

}
```

```cpp
bool findbyrange(std::vector<Citizen> inputvector, int inputrangebeg, int inputrangeend)
{
    bool found = false;
    for (int i=0; i<inputvector.size(); i++)
    {
        if(inputvector[i].Getnumber() > inputrangebeg && inputvector[i].Getnumber() < inputrangeend)
        {
            inputvector[i].PutMember();
            found = true;
        }

    }

    if(!found)
    {
        std::cout << "Empty result" << std::endl;
        return false;
    }
    else
    {
        return true;
    }

}
```

Main.cpp

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "func.cpp"

#define filepath "data.txt"

int main()
{
    std::vector <Citizen> people;
    bool exit = false;
    while (!exit)
    {
        std::cout << "Choose option:" << std::endl;
        std::cout << "1) Load from file(re-write current)" << std::endl;
        std::cout << "2) Add member to data" << std::endl;
        std::cout << "3) Find member by exact input" << std::endl;
        std::cout << "4) Find member by range" << std::endl;
        std::cout << "5) Print current dataset" << std::endl;
        std::cout << "6) Exit prog" << std::endl;
        int x;
        std::cin >> x;
        switch (x)
        {
        case 1:
            {
                people = LoadFromFile(filepath);
                break;
            }
        case 2:
        {
            Citizen temp;
            temp.AddFromKeyboard();
            people.push_back(temp);
            break;
        }
        case 3:
        {
            std::cout << "Enter number" << std::endl;
            int input;
            std::cin >> input;
            bool found = findbyexactphone(people, input);

            break;
        }


        case 4:
        {
            int phone_range_begin;
            int phone_range_end;
            std::cout << "Enter range begin" << std::endl;
            std::cin >> phone_range_begin;
            std::cout << "Enter range end" << std::endl;
            std::cin >> phone_range_end;
            bool found = findbyrange(people, phone_range_begin, phone_range_end);

            break;
        }
```

```
        case 5:
        {
            for (int i=0; i<people.size(); i++)
            {
                people[i].PutMember();
            }
            break;
        }
        case 6:
        {
            return 0;
        }

        default:
            {
                std::cout << "Invalid input" << std::endl;
                break;
            }
        }
    }

    return 0;
}
```

Пример входных данных (data.txt)

```
Putin|Moscow|3214
Trump|Washington|6587
Kim Jong Un|Pheniang|4552
Merkel|Berlin|1115
Johnson|London|9864
Macron|Paris|3762
Conte|Rome|4444
Duda|Warsaw|7777
Erdogan|Ankara|6666
Xi Jinping|Beijing|1111
```

./tests/tests.cpp

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "../func.cpp"

#define CATCH_CONFIG_MAIN
#include "catch.hpp"

#define testfilepath "./tests/testdata.txt"

TEST_CASE("testloadfromfile")
{
    std::vector <Citizen> testvector;
    testvector = LoadFromFile(testfilepath);
    std::ifstream file (testfilepath);
    std::string teststring;
    std::getline(file, teststring);
    REQUIRE(teststring == testvector[0].GetMember());
}

TEST_CASE("testfindbyexact")
{
    std::vector <Citizen> testvector;
    Citizen temp;
    temp.Add("Putin", "Moscow", 1111);
    testvector.push_back(temp);
    temp.Add("Trump", "Washington", 2222);
    testvector.push_back(temp);
    REQUIRE(findbyexactphone(testvector, 2222));
    REQUIRE(!findbyexactphone(testvector, 2000));
}

TEST_CASE("testfindbyrange")
{
    std::vector <Citizen> testvector;
    Citizen temp;
    temp.Add("Putin", "Moscow", 1111);
    testvector.push_back(temp);
    temp.Add("Trump", "Washington", 2222);
    testvector.push_back(temp);
    REQUIRE(findbyrange(testvector, 1000,1200));
    REQUIRE(!findbyrange(testvector, 1200,2000));
}
```

./tests/testdata.txt

```
Putin|Moscow|3214
Trump|Washington|6587
```

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана программа «Телефонный справочник». В процессе выполнения работы были применены различные средства и методы разработки, такие как git, Travis CI, make, юнит-тестирование. В конечном итоге на странице проекта на сайте github в ветку master была загружена рабочая версия программы, готовая к использованию. Данная версия программы удовлетворяет всем требованиям и прошла тестирование. Проект привязан к системе Travis CI для непрерывной интеграции. Соответствующий индикатор расположен в секции README.

https://github.com/dev0nizer/coursework_trpo

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

- https://git-scm.com/docs
- https://docs.travis-ci.com/
- https://www.gnu.org/software/make/manual/make.html
- https://stackoverflow.com/