Федеральное агентство связи
Сибирский Государственный Университет
Телекоммуникаций и Информатики
(СибГУТИ)

Пояснительная записка

к курсовоиу проекту по дисциплине “Технологии разработки программного обеспечения”

Выполнил: Хорнский А.Д.

Группа: ЗП-91

Номер студенческого билета:

73190184

Адрес электронной почты:

alexhornsky@yandex.ru

Проверил: _____

Новосибирск, 2020 г.

Содержание

ВВЕДЕНИЕ

Цель работы: освоение курса по предмету «Технологии разработки программного обеспечения». Получение опыта использования системы контроля версий GIT, системы сборки приложений make, применения юниттестирования 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» я взял программу «Калькулятор». Данная программа должна удовлетворять следующим условиям:

Функциональность:

- В программе реализовано меню.
- Запрашивает указать какое действие желает выполнить пользователь.
- Считывает вводимое пользователем значение, первое, затем второе.
- В выбранную пользователем желаемую функцию, передает считанные значения, как фактические аргуметы.
- Результат выодится на экран.

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
- src/*.exe
- test/*.exe

для игнорирования исполняемых файлов.

Также нужно создать удаленный репозиторий на сайте Github. Для этого воспользуемся GitBash. 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 “commit text” – данная команда подпишет и зафиксирует изменения в проекте.

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

git push origin master – 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.

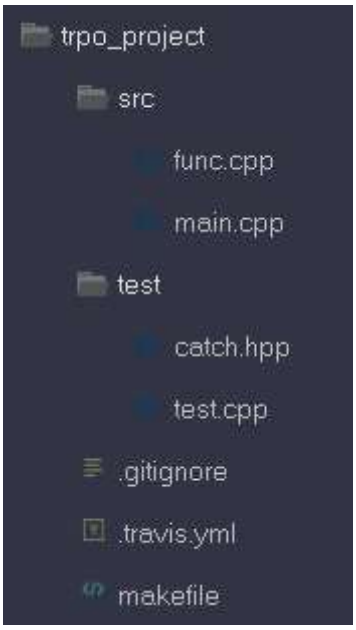
Для тестирования участков кода мной были написаны юнит-тесты для функций в моей программе. Файл с тестами был выделен в отдельную директорию, в которой также содержится тестовая база данных. Для тестирования был использован фреймворк «catch2».

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 созданы: директория ./src содержащая файлы, main.cpp с телом программы и файл func.cpp, содержащий в себе функции для работы программы, текже директория ./test, в данной директории содержится файл test.cpp с тестами функций из func.cpp с тестовыми входными данными.

Окончательная структура проекта после завершения работы:



Инструменты Travis CI доступны через сайт. После синхронизации репозитория с GitHub и подключения, 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 в формате Markdown. После commit в описании появится иконка с текущим статусом прохождения тестов.

makefile:

```
all: hello

hello: main.o test.o

main.o:
    g++ -std=c++11 src/main.cpp -o src/main -lncurses

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

test:
    ./test/test

run:
    .src/main

.PHONY: clean

clean:
    rm -rf src/*.o test/*.o

```

.travis.yml

```
language: cpp

compiler : g++

script:
  - make
  - make test

```

.gitignore

```
*.exe
src/*.exe
test/*.exe
```

./src/func.cpp

```cpp
#include <iostream>
#include <stdlib.h>
#include <math.h>

using namespace std;

float Addition(float x, float y) {return x + y;}

float Subtraction(float x, float y)  {return x - y;}

float Multiplication(float x, float y)  {return x * y;}

float Division(float x, float y)
{
    if (y == 0)
    {
        cout << "test_Division : isn't possible by zero" << endl;
        return 0;
    }
    else
        return x / y;
}

unsigned int Factorial(unsigned int b)
{
    return ((0 < b) && (b < 13)) ? Factorial(b - 1) * b : 1;
}

unsigned int body_of_factorial()
{
    unsigned int b;
    cout << "Enter unsigned int number : ";
    cin >> b;

    if ((b >= 13) || (b < 0))
    {
        cout << "Number isn't unsigned or Your factorial will be greater than 4.294.967.295. Try using a lower number." << endl;
        return 0;
    }

    cout << "Your Factorial of " << b << " is ";
    return Factorial(b);
}

float Exponentiation(float x, float y)
{
    return pow(x,y);
}
```

./src/main.cpp

```cpp
#include <ncurses.h> //<ncurses.h> -lncurses
#include <iostream>
#include <stdlib.h>
#include <math.h>

#include "func.cpp"

using namespace std;

float x, y;
unsigned int b;
char c;

float Addition();
float Subtraction();
float Multiplication();
float Division();
float Exponentiation();
unsigned int Factorial();
unsigned int body_of_Factorial();

int main()
{
    while (1)
    {
        system("clear"); //system("clear");

        puts("1. + Addition");
        puts("2. - Subtraction");
        puts("3. * Multiplication");
        puts("4. / Division");
        puts("5. ! Factorial");
        puts("6. ^ Exponentiation");
        puts("0. EXIT");

        c = getch();

        system("clear"); //system("clear");

        if ((c == '1') || (c == '2') || (c == '3') || (c == '4') || (c == '6'))
        {
            cout << "Enter first number : ";
            cin >> x;
            cout << "Enter second number : ";
            cin >> y;
        }

        switch (c)
        {
            case '1': cout << Addition(x, y) << endl; getch(); break;
            case '2': cout << Subtraction(x, y) << endl; getch(); break;
            case '3': cout << Multiplication(x, y) << endl; getch(); break;
            case '4': cout << Division(x, y) << endl; getch(); break;
            case '5': cout << body_of_Factorial() << endl; getch(); break;
            case '6': cout << Exponentiation(x, y) << endl; getch(); break;
            case '0': return 0;
            default : puts("WRONG MODE"); getch();
        }
    }
}

```

./test/test.cpp

```cpp
#include <iostream>
#include <stdlib.h>
#include "../src/func.cpp"

#define CATCH_CONFIG_MAIN
#include "catch.hpp"

TEST_CASE("test_Addition")
{
    REQUIRE(Addition(0,0) == 0);
    REQUIRE(Addition(0,3) == 3);
    REQUIRE(Addition(-10,0) == -10);
    REQUIRE(Addition(9,0) == 9);
    REQUIRE(Addition(-3,-3) == -6);
    cout << "test_Addition : correct\n" << endl;
}

TEST_CASE("test_Subtraction")
{
    REQUIRE(Subtraction(3,3) == 0);
    REQUIRE(Subtraction(0,3) == -3);
    REQUIRE(Subtraction(-10,0) == -10);
    REQUIRE(Subtraction(9,0) == 9);
    REQUIRE(Subtraction(0,9) == -9);
    REQUIRE(Subtraction(-3,-3) == 0);
    cout << "test_Subtraction : correct\n" << endl;
}

TEST_CASE("test_Multiplication")
{
    REQUIRE(Multiplication(3,3) == 9);
    REQUIRE(Multiplication(0,3) == 0);
    REQUIRE(Multiplication(22,22) == 484);
    REQUIRE(Multiplication(9,0) == 0);
    REQUIRE(Multiplication(-3,-3) == 9);
    REQUIRE(Multiplication(-7,3) == -21);
    cout << "test_Multiplication : correct\n" << endl;
}

TEST_CASE("test_Division")
{
    REQUIRE(Division(3,3) == 1);
    REQUIRE(Division(0,3) == 0);
    REQUIRE(Division(22,22) == 1);
    REQUIRE(Division(-8,4) == -2);
    REQUIRE(Division(-8,0) == 0); // + test error
    cout << "test_Division : correct\n" << endl;
}

TEST_CASE("test_Factorial")
{
    REQUIRE(Factorial(55) == 1); //0
    cout << "test_Factorial : Number isn't unsigned or Your factorial will be greater than 4.294.967.295. Try using a lower number." << endl;
    REQUIRE(Factorial(-5) == 1); //0
    cout << "test_Factorial : Number isn't unsigned or Your factorial will be greater than 4.294.967.295. Try using a lower number." << endl;
    REQUIRE(Factorial(0) == 1);
    REQUIRE(Factorial(12) == 479001600);
    REQUIRE(Factorial(3) == 6);
    REQUIRE(Factorial(1) == 1);
    cout << "test_Factorial : correct\n" << endl;
}

TEST_CASE("test_Exponentiation")
{
    REQUIRE(Exponentiation(0,5) == 0);
    REQUIRE(Exponentiation(3,3) == 27);
    REQUIRE(Exponentiation(9,3) == 729);
    REQUIRE(Exponentiation(1,0) == 1);
    REQUIRE(Exponentiation(10,0) == 1);
    cout << "test_Exponentiation : correct" << endl;
}
```

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана программа «Калькулятор». В процессе выполнения работы были применены различные средства и методы разработки, такие как git, Travis CI, make, юниттестирование. 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/hornsky/mainTRPO