Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет

им. И.И. Ползунова»

Факультет Информационных технологий

Кафедра Прикладная математика

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Я.Б.Еремин

(подпись преподавателя) (инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Отчет

по лабораторной работе №2

Багрепорты и юнит-тесты

по дисциплине «**Верификация и тестирование программного обеспечения**»

ЛР 09.03.04.00.10 ОТ

(обозначение документа)

Студент группы ПИ-82 A.C.Реутов

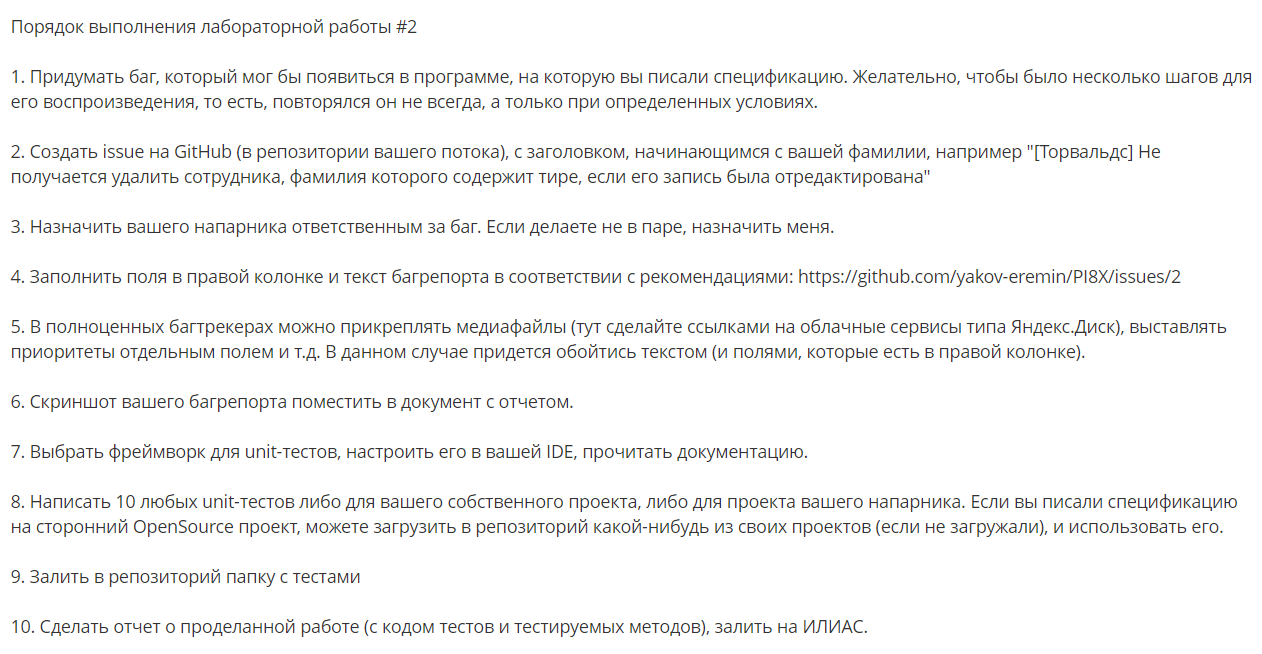
(инициалы, фамилия)

Преподаватель старший преподаватель Я.Б.Еремин

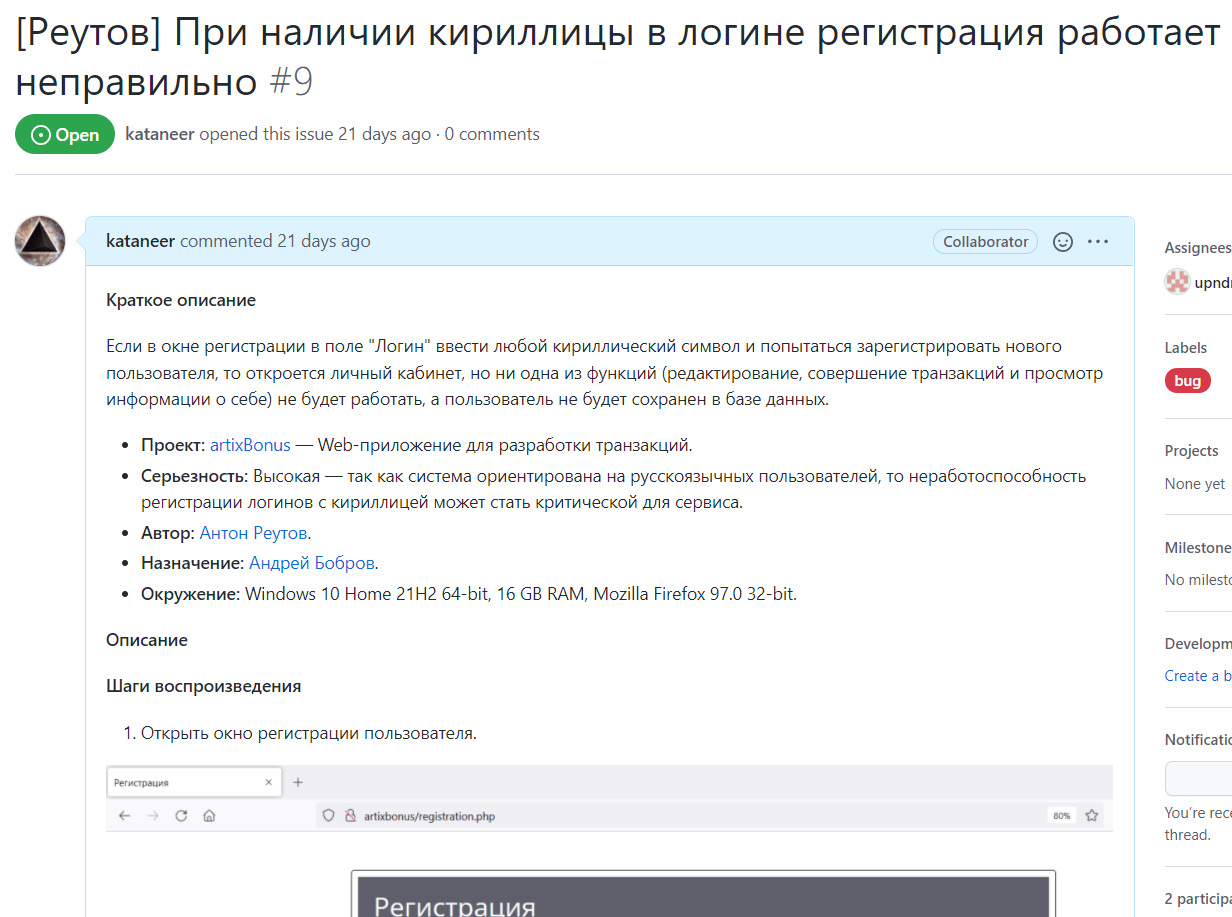
(должность, ученое звание) (инициалы, фамилия)

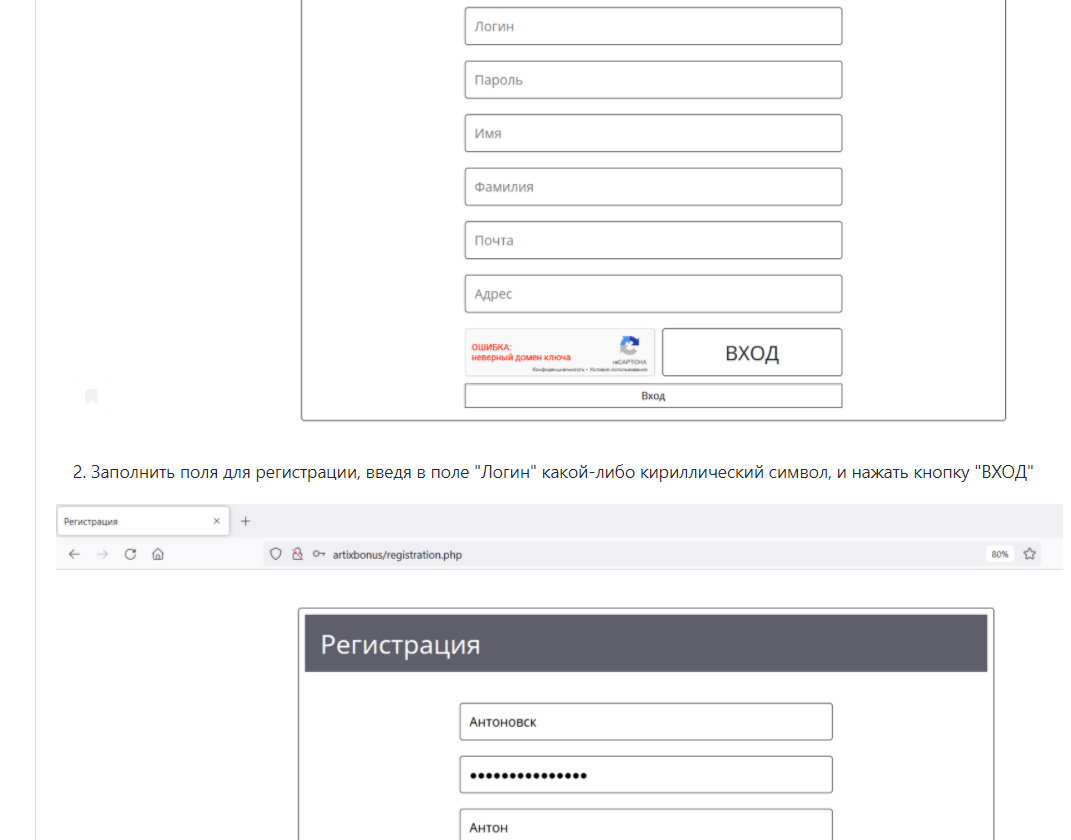
Барнаул 2022

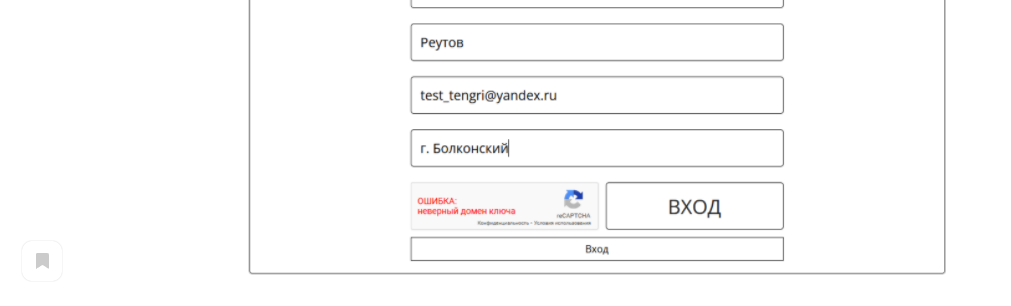
Задание:

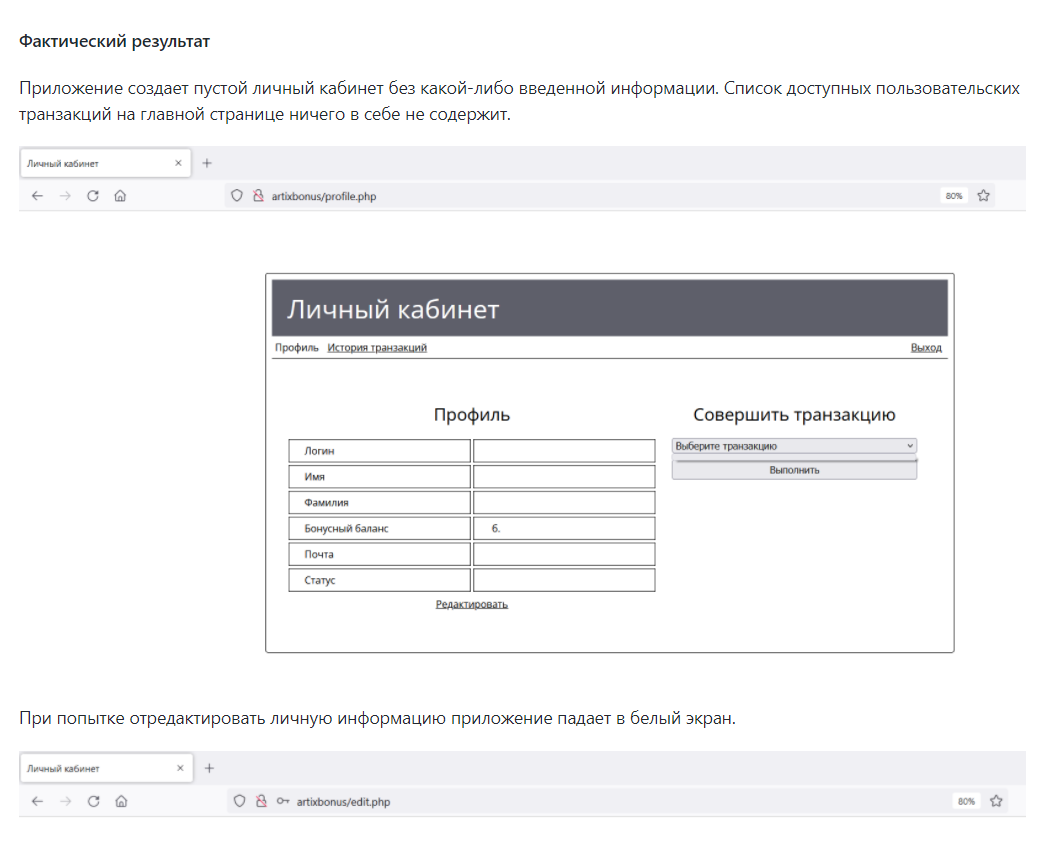


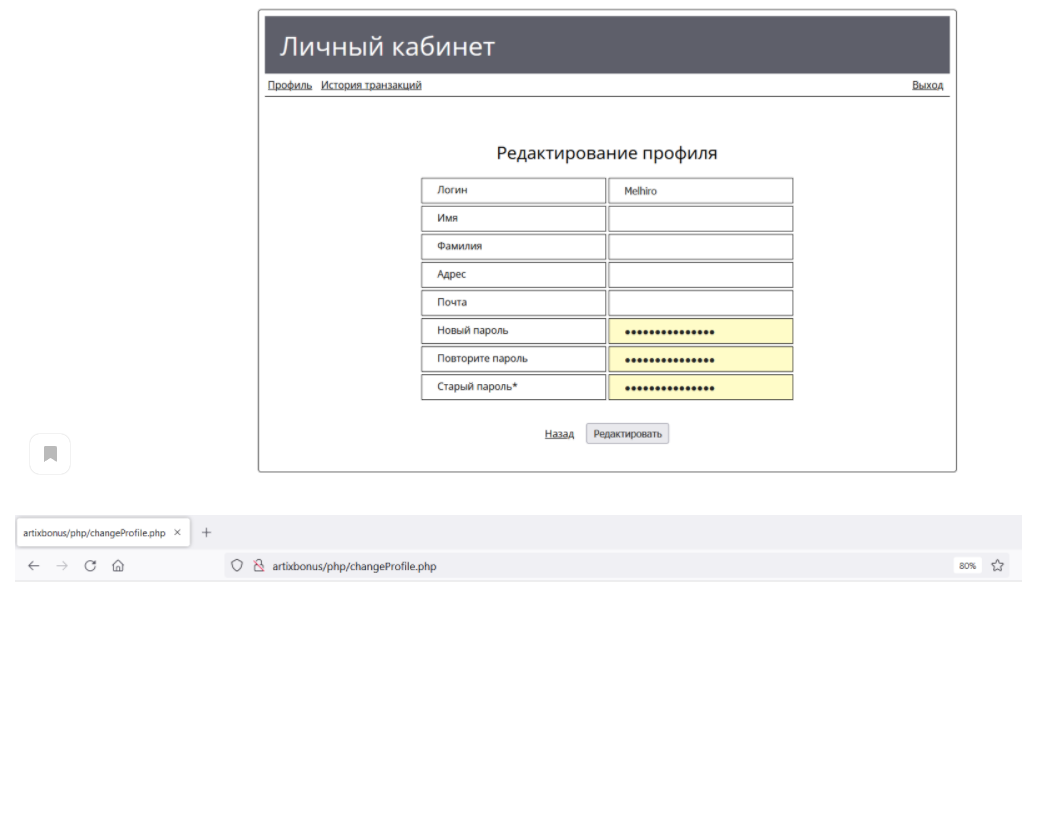
Багрепорт:

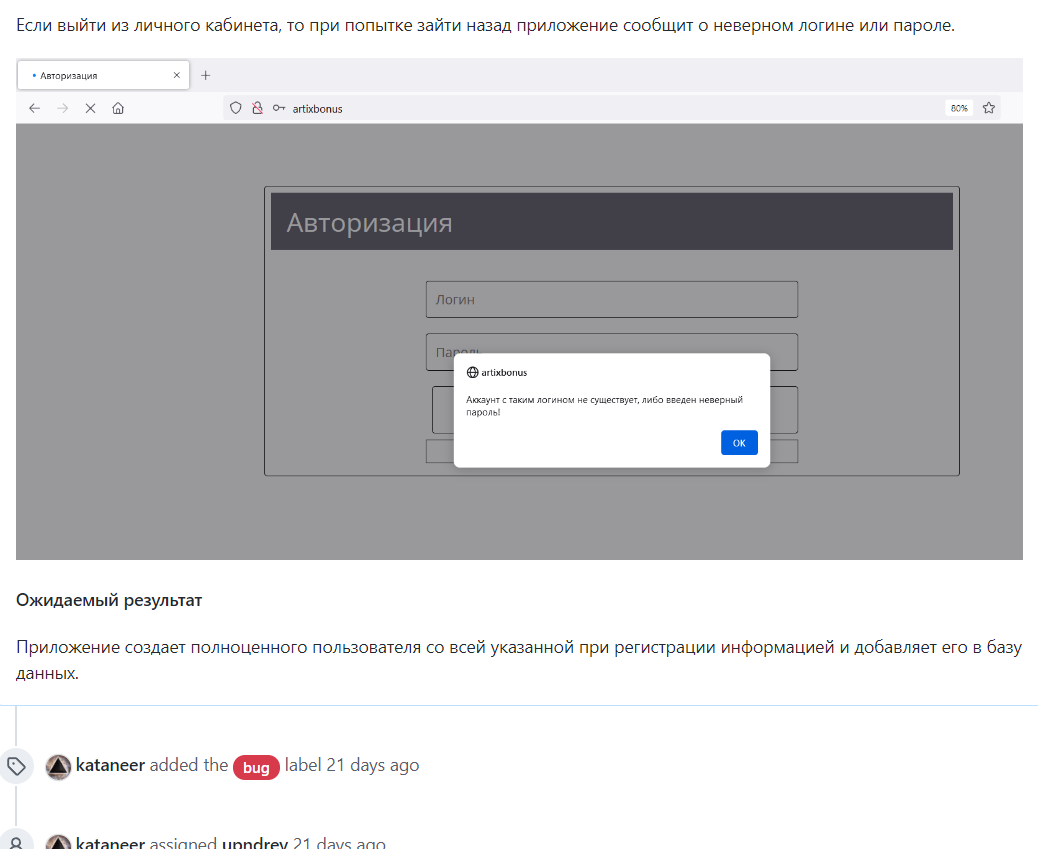












Юнит-тесты:

Код тестируемых методов:

class Facade {

public:

countersFactory \*rivCounts;

rivWatch \*rivParams;

Facade()

{

rivCounts = new countersFactory();

}

void facadeW(float a, float b, int c)

{

rivParams = new rivWatch(a, b, c);

}

void creati(char nameCount[], char type[], int number, int atm, float temp)

{

rivCounts->creati(nameCount, type, number, atm, temp);

}

void Consumption()

{

rivParams->Consumption();

}

void pConsumption()

{

rivParams->pConsumption();

}

};

class metFields {

private:

float time;

float wind;

float tempView;

public:

void setParams(float t, float w, float tw)

{

time = t;

wind = w;

tempView = tw;

}

float getTime()

{

return time;

}

float getWind()

{

return wind;

}

float getTempView()

{

return tempView;

}

float atFieldCount(int atm, float temp)

{

return atm - 0.05 \* ((tempView + wind)/time);

}

float tempFieldCount(int atm, float temp)

{

return temp + 0.05 \* (temp - tempView)/time;

}

};

class metCounter {

public:

char \*name = new char[];

char \*type = new char[];

int number;

public:

metFields \*fieldWay;

metCounter()

{

fieldWay = new metFields();

}

int atm;

int temp;

void initM(char \*nm, char \*tp, int num)

{

name = nm;

type = tp;

number = num;

}

char \*getName()

{

return name;

}

char \*getType()

{

return type;

}

void init(int a, int b)

{

atm = a;

temp = b;

}

void meteoCount();

};

class weatherCounter : public metCounter {

public:

float ets;

void meteoCount()

{

ets = atm \* temp;

}

};

class highWaters {

private:

float time;

public:

void setTime(float t)

{

time = t;

}

float getTime()

{

return time;

}

float highWatersEvents(float square, float width, float density, float drain, float atm, float temp)

{

float a = (atm + temp + width) \* time / 2 \* (square + density + drain);

return a;

}

};

class snowCounter : public metCounter {

private:

float square;

float width;

float density;

float drainSquare;

public:

highWaters \*higher;

float res;

snowCounter()

{

higher = new highWaters();

}

void setParams(float sq, float wi, float de, float dr)

{

square = sq;

width = wi;

density = de;

drainSquare = dr;

}

void meteoCount()

{

res = (square + width + density) \* (drainSquare + atm - temp);

}

};

class highIcers {

private:

float square;

float density;

public:

void setIceParams(float sq, float de)

{

square = sq;

density = de;

}

float setJamEvents(float atm, float temp, float width)

{

float a = (atm + temp + width) / (square + density);

return a;

}

};

class iceCounter : public metCounter {

private:

float width;

public:

highIcers \*icer;

float res;

iceCounter()

{

icer = new highIcers();

}

void setWidth(float w)

{

width = w;

}

float getWidth()

{

return width;

}

void meteoCount()

{

res = width \* (atm / abs(temp)) / 10;

}

};

class rivWatcher {

public:

float square;

float speed;

float time;

float cont;

void initWatcher (float a, float b, float c)

{

square = a;

speed = b;

time = c;

}

void Consumption()

{

cont = square \* speed \* time;

}

};

class proxyWatcher {

public:

rivWatcher \*ras;

void Consumption()

{

ras->Consumption();

}

proxyWatcher(float a, float b, float c)

{

ras = new rivWatcher();

ras->initWatcher(a, b, c);

}

};

class rivWatch {

public:

rivWatcher \*watchi;

proxyWatcher \*pwatchi;

rivWatch(float a, float b, float c)

{

watchi = new rivWatcher();

watchi->initWatcher(a, b, c);

pwatchi = new proxyWatcher(a, b, c);

}

/\*!

void Consumption()

{

watchi->Consumption();

}

void pConsumption()

{

pwatchi->Consumption();

}

};

Код юнит-тестов:

class LabTest : public ::testing::Test {

protected:

virtual void SetUp()

{}

virtual void TearDown()

{}

Facade \*wet;

metCounter \*met;

iceCounter \*ice;

snowCounter \*snow;

weatherCounter \*wea;

};

TEST\_F(LabTest, Test1) {

wet = new Facade();

wet->facadeW(3, 4, 5);

wet->Consumption();

EXPECT\_EQ(wet->rivParams->watchi->cont, 60);

}

TEST\_F(LabTest, Test2) {

wet = new Facade();

wet->facadeW(2, 5, 12);

wet->pConsumption();

EXPECT\_EQ(wet->rivParams->pwatchi->ras->cont, 90);

}

TEST\_F(LabTest, Test3) {

wet = new Facade();

wet->pConsumption();

EXPECT\_NO\_THROW(wet->pConsumption());

}

TEST\_F(LabTest, Test4) {

met = new metCounter();

met->init(720, 90);

met->fieldWay->setParams(1, 1, 1);

ASSERT\_NEAR(719.9, met->fieldWay->atFieldCount(720, 90), 0.001);

}

TEST\_F(LabTest, Test5) {

met = new metCounter();

met->initM("Bobrovka", "Ice", 1);

ASSERT\_EQ(met->getType(), "Snow");

ASSERT\_EQ(met->getName(), "Bobrovka");

}

TEST\_F(LabTest, Test6) {

wea = new weatherCounter();

wea->init(720, 90);

wea->meteoCount();

EXPECT\_LE(10000, wea->ets);

}

TEST\_F(LabTest, Test7) {

ice = new iceCounter();

ice->init(720, 90);

ice->setWidth(5);

EXPECT\_FALSE(4 > ice->getWidth());

}

TEST\_F(LabTest, Test8) {

ice = new iceCounter();

ice->init(720, 90);

ice->icer->setIceParams(12, 24);

EXPECT\_TRUE(5 < ice->icer->setJamEvents(720, 90, 5));

}

TEST\_F(LabTest, Test9) {

snow = new snowCounter();

snow->init(720, 90);

snow->setParams(1, 2, 3, 4);

EXPECT\_NO\_FATAL\_FAILURE(snow->meteoCount());

}

TEST\_F(LabTest, Test10) {

snow = new snowCounter();

snow->init(720, 90);

snow->higher->setTime(50);

EXPECT\_DOUBLE\_EQ(snow->higher->getTime(), 50);

}

TEST\_F(LabTest, Getcher) {

\_getch();

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

Результат тестирования:

