

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

**Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики**
Факультет Прикладной оптики

Отчет по практике

Выполнил:	Нелюбин Д.А.
Группа:	В3316
Преподаватель:	Кудрявцев А. С.

Санкт-Петербург, 2019

Оглавление

Задание №1. Знакомство с системой контроля версий Git.....	3
Задание №2 Форматирование и стиль.....	5
Задание №3. TDD: Разработка через тестирование.....	12

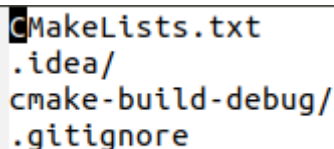
Задание №1. Знакомство с системой контроля версий Git

Цель работы: научиться работать с распределенной системой контроля версий Git.

Для достижения вышеприведенной цели были проделаны следующие шаги

Так как сначала репозиторий был создан на локальном компьютере, а потом привязан к github пришлось выполнить команду «git remote add origin https://github.com/danilka-na/test.git»

Для того, чтобы не индексировать ненужные файлы был создан файл «.gitignore» куда были добавлены следующие строки:



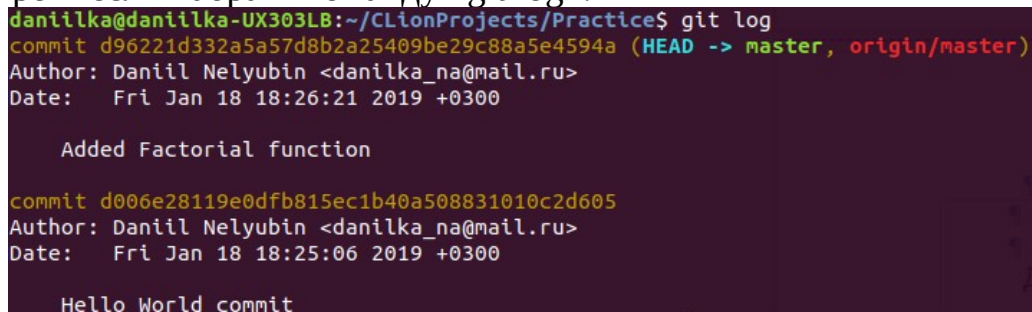
```
CMakeLists.txt  
.idea/  
cmake-build-debug/  
.gitignore
```

Рис.1. Содержимое файла .gitignore

Далее была создана простейшая программа Hello World, все соответствующие результаты были закомичены.

После этого была реализована функция вычисляющая факторал числа.

Все эти реализации находятся в разных комитах, которые можно посмотреть если набрать команду «git log»:



```
danilka@danilka-UX303LB:~/CLionProjects/Practice$ git log  
commit d96221d332a5a57d8b2a25409be29c88a5e4594a (HEAD -> master, origin/master)  
Author: Daniil Nelyubin <danilka_na@mail.ru>  
Date: Fri Jan 18 18:26:21 2019 +0300  
  
    Added Factorial function  
  
commit d006e28119e0dfb815ec1b40a508831010c2d605  
Author: Daniil Nelyubin <danilka_na@mail.ru>  
Date: Fri Jan 18 18:25:06 2019 +0300  
  
    Hello World commit
```

Рис.2. Результат ввода команды git log

Для того, чтобы понять какова разница между этими двумя комитами, нужно ввести команду git diff:

```
daniilka@daniilka-UX303LB:~/CLionProjects/Practice$ git diff d006e28
diff --git a/main.cpp b/main.cpp
index 7fff635..347c1c2 100644
--- a/main.cpp
+++ b/main.cpp
@@ -1,9 +1,12 @@
#include <iostream>

-
+int factorial(int a){
+    if(a==1) return 1;
+    else return a*factorial(a-1);
+}

int main() {
-    std::cout << "Hello world!" << std::endl;
+    std::cout << factorial(4) << std::endl;
    return 0;
}
\ No newline at end of file
```

Рис.3. Результат работы команды git diff

Задание №2 Форматирование и стиль

Условия задания:

1. Разработать программу оформленную в соответствии с правилами в файле CodeRules.pdf. Разработка осуществляется на C++.
2. Сделать скриншот с именем и успешным результатом проверки.
3. Поместить результаты работы — программу (файлы с расширениями
cpp, h, sln, vxcproj, filters), а также скриншот с сайта timus в свой репозиторий.

1780. Код Грея

Ограничение времени: 0.5 секунды

Ограничение памяти: 64 МБ

Денис, Ваня и Федя собрались на свою первую командную тренировку. Федя рассказал, что выучил алгоритм генерации кода Грея:

1. Создадим список из двух элементов: $\{0, 1\}$.
2. Добавим в конец списка все его элементы в обратном порядке: $\{0, 1, 1, 0\}$.
3. К первой половине элементов списка допишем слева 0, ко второй половине элементов списка допишем слева 1: $\{00, 01, 11, 10\}$.
4. Будем повторять шаги 2 и 3 до тех пор, пока длина всех элементов списка не станет равна n .

Число n называется длиной кода Грея. Так, код длины 3 выглядит следующим образом: $\{000, 001, 011, 010, 110, 111, 101, 100\}$.

Когда Денис применил алгоритм Феди, у него получилось, что на k -й позиции в списке (если нумеровать позиции с нуля) стоит двоичное число x . Ваня записал на бумажку числа k и x в двоичной системе счисления. Спустя много лет эта бумажка попала к вам в руки. К сожалению, некоторые цифры на ней стёрлись за эти годы. Сможете ли вы по оставшимся цифрам восстановить числа, которые были на ней записаны?

Исходные данные

В первой строке записано число k в двоичной системе счисления. Стёршиеся цифры обозначены символом «?». Во второй строке в аналогичном формате записано число x . Длины обоих чисел совпадают и не превосходят 10^5 . Числа могут содержать ведущие нули.

Результат

Если по уцелевшим цифрам можно однозначно восстановить числа k и x , выведите их, заменив символы «?» на символы «0» и «1». Если существует несколько способов сделать это, выведите «Ambiguity». Если Денис или Ваня ошиблись, и восстановить числа невозможно, выведите «Impossible».

Примеры

исходные данные	результат
0?1 0?0	011 010
?00 ??0	Ambiguity
100 100	Impossible

Автор задачи: Дмитрий Полетаев

Источник задачи: XV Открытый командный чемпионат УрГУ по программированию

Метки: нет ([скрыть метки для нерешенных задач](#))

Сложность: 780 [Версия для печати](#) [Отправить на проверку](#) [Обсуждение на форуме \(3\)](#)

✓ [Мои попытки](#) [Все попытки \(1704\)](#) [Все успешные попытки \(466\)](#) [Рейтинг решений \(336\)](#)

Рис.4. Задание №1780 с сайта timus

Алгоритм:

Суть алгоритма заключается в том, чтобы попытаться дешифровать числа k и x , последовательно производя «хог» числа k и его сдвинутой влево копией. Если обнаруживается ситуация, когда можно явно задать число, то данное число подставляется и алгоритм идет в обратную сторону для того чтобы попытаться решить ранее пропущенные символы.

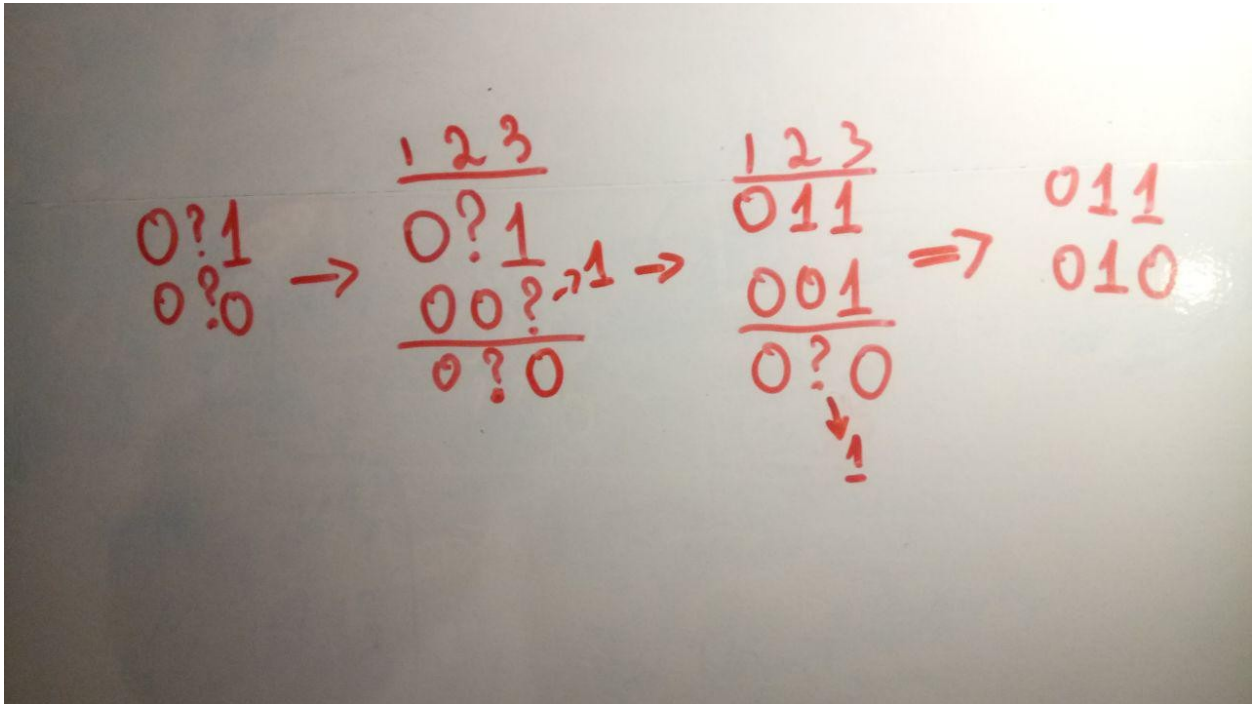


Рис. 5. Пример работы алгоритма

На рисунке 5 показана работа алгоритма описанного выше.

Результат проверки на сайте timus (Рис. 6)

8268941	01:31:36 26 фев 2019	daniika_na	1780. Код Грея	G++ 7.1	Accepted	0.015	640 КБ
-------------------------	-------------------------	----------------------------	--------------------------------	---------	----------	-------	--------

Рис. 6. Результат проверки на timus

Код программы:

```
#include <iostream>

#include <vector>
#include <algorithm>

std::string xorForString(std::string iFirstString,
std::string iSecondString) {
    std::string finishString(iFirstString);
    for (int i = 0; i < iFirstString.size(); ++i) {
        if (iSecondString[i] == iFirstString[i]) {
            finishString[i] = '0';
        } else {
            finishString[i] = '1';
        }
    }
}
```

```

        }
    }
    return finishString;
}

std::string grayenCode(std::string inString) {
    std::string shiftString(inString);
    for (unsigned long j = shiftString.size(); j > 0; j--)
{
        shiftString[j] = shiftString[j - 1];
    }
    shiftString[0] = '0';
    std::string finishString = xorForString(inString,
shiftString);
    return finishString;
}

std::vector<std::string> *generateVariance(std::string
iString) {
    if ((int) iString.find('?') >= 0) {
        std::string copy1(iString);
        std::string copy2(iString);
        copy1.replace(iString.find('?'), 1, "1");
        copy2.replace(iString.find('?'), 1, "0");
        std::vector<std::string> *ret_vec1 =
generateVariance(copy1);
        std::vector<std::string> *ret_vec2 =
generateVariance(copy2);
        ret_vec1->insert(ret_vec1->end(), ret_vec2->begin(),
ret_vec2->end());
        delete ret_vec2;
        return ret_vec1;
    } else {
        std::vector<std::string> *vec = new
std::vector<std::string>;

```



```

        vec->push_back(iString);
        return vec;
    }
}

std::vector<std::string> *smartXor(std::string iNum,
std::string iGrey) {
    std::vector<std::string> *resultVector = new
std::vector<std::string>;
    std::string prevNumChar = "0";
    int countOfDoubleQuestions = 0;
    int countOfSingleQuestions = 0;
    for (int i = 0; i < iNum.size(); ++i) {
        if (std::count(iNum.begin(), iNum.end(), '?') == 0
&&
            std::count(iGrey.begin(), iGrey.end(), '?') ==
0) {
            break;
        }
        if (prevNumChar[0] == '?' && iNum[i] == '?' ||
            iNum[i] == '?' && iGrey[i] == '?' ||
            prevNumChar[0] == '?' && iGrey[i] == '?') {
            countOfDoubleQuestions++;
            if (i == 0) {
                prevNumChar.replace(0, 1, iNum.substr(i, 1));
            } else prevNumChar = iNum[i - 1];
        } else {
            for (int j = i; j > i - (countOfDoubleQuestions
+ 1); j--) {
                if (j == 0) {
                    prevNumChar.replace(0, 1, "0");
                } else prevNumChar = iNum[j - 1];
                if (prevNumChar == "?") {

```

```

        iNum.replace(j - 1, 1, myXor(iNum[j],
iGrey[j]));

        } else if (iNum[j] == '?') {
            iNum.replace(j, 1, myXor(prevNumChar[0],
iGrey[j]));

        } else {
            iGrey.replace(j, 1, myXor(prevNumChar[0],
iNum[j]));

        }
        countOfSingleQuestions++;
    }
    countOfDoubleQuestions = 0;
    prevNumChar = iNum[i - 1];
}
}
resultVector->push_back(iNum);
resultVector->push_back(iGrey);
return resultVector;
}

int main() {
    std::string firstNum;
    std::string secondNum;
    std::cin >> firstNum >> secondNum;
    int countFirst = std::count(firstNum.begin(),
firstNum.end(), '?');
    int countSecond = std::count(secondNum.begin(),
secondNum.end(), '?');
    std::vector<std::string> *vecOfVariance;
    if (countFirst == 0 && countSecond == 0) {
        if (grayencode(firstNum) == secondNum) {
            std::cout << firstNum << "\n" << secondNum;
            return 0;
        } else {

```

```

        std::cout << "Impossible";
        return 0;
    }
}

vecOfVariance = smartXor(firstNum, secondNum);
countFirst = std::count(vecOfVariance->at(0).begin(),
                        vecOfVariance->at(0).end(), '?');
countSecond = std::count(vecOfVariance->at(1).begin(),
                        vecOfVariance->at(1).end(), '?');
if (countFirst > 0 || countSecond > 0) {
    std::cout << "Ambiguity";
    return 0;
}
if (grayenCode(vecOfVariance->at(0)) != vecOfVariance-
>at(1)) {
    std::cout << "Impossible";
    return 0;
}

    std::cout << vecOfVariance->at(0) << "\n" <<
vecOfVariance->at(1);
    return 0;
}

```

Задание №3. TDD: Разработка через тестирование

Условия задания:

1. Разработать программу в рамках подхода TDD согласно выбранному заданию.

2. Выбрать задачу на acm.timus.ru.

3. Составить следующие тесты по проверке работоспособности программы:

- Тесты для каждой функции/метода в программе;
- тесты по примерам из задания;
- тесты с данными, приводящими к ошибочным ситуациям;
- тесты с данными – граничными случаями (минимум, максимум и т.п.);
- дополнительные тесты, если покрытие кода для файлов алгоритма меньше 100% по линиям и функциям.

4. делать скриншот с успешным результатом проверки на сайте [timus](https://acm.timus.ru).

5. Сделать скриншот с покрытием кода по линиям/функциям для файлов алгоритма.

6. Поместить результаты работы в свой репозиторий.

Условие задачи см. рис. 7.

1332. Джинн-бомбардировки

Ограничение времени: 1.0 секунды

Ограничение памяти: 64 МБ

Дольше всех задержался там некий Питирим Шварц, бывший монах и изобретатель подпорки для мушкета, беззаветно трудившийся над проектом джинн-бомбардировок. Суть проекта состояла в сбрасывании на города противника бутылок с джиннами, выдержанными в заточении не менее трех тысяч лет. Хорошо известно, что джинны в свободном состоянии способны только либо разрушать города, либо строить дворцы. Основательно выдержанный джинн (рассуждал Питирим Шварц), освободившись из бутылки, не станет строить дворцов, и противнику придется туго. Некоторым препятствием к осуществлению этого замысла являлось недостаточное количество бутылок с джиннами, но Шварц рассчитывал пополнить запасы глубоким трением Красного и Средиземного морей.

Разработка проекта джинн-бомбардировок перешла в экспериментальную стадию. На полигоне силами дублей научных сотрудников было возведено N городов. Город представляет собой круг фиксированного радиуса r , одинакового для всех городов. Так как М. М. Камноедов выдал для эксперимента всего 1 бутылку с джином, экспериментаторы решили, что чем больше городов будет разрушено, тем лучше для науки. Известно, что джинн разрушает все на расстоянии R от места падения бутылки. Город считается разрушенным, если он целиком попадает в зону разрушения. Перед тем, как проводить эксперимент, необходимо найти максимальное количество городов, которое может быть разрушено силами одного джинна.

Исходные данные

В первой строке содержится число городов N ($1 \leq N \leq 100$). Следующие N строк содержат координаты центров городов x_i, y_i - целые числа, $|x_i|, |y_i| \leq 10000$. Центры разных городов не совпадают.

В последней строке содержатся радиус джинн-поражения R и радиус города r — целые числа ($1 \leq R, r \leq 10000$).

Результат

Выведите максимальное количество городов, которое может быть разрушено в результате бомбардировки.

Примеры

исходные данные	результат
3 0 0 0 4 4 0 3 1	2
5 0 0 0 1 0 2 0 3 0 4 1 1	1

Автор задачи: Александр Бикбаев

Источник задачи: Десятый командный чемпионат школьников Свердловской области по программированию (16 октября 2004 года)

Метки: [геометрия](#) ([скрыть метки для нерешенных задач](#))

Сложность: 635 [Версия для печати](#) [Отправить на проверку](#) [Обсуждение на форуме \(19\)](#)

✓ [Мои попытки](#) [Все попытки \(5118\)](#) [Все успешные попытки \(895\)](#) [Рейтинг решений \(700\)](#)

Рис.7 Условие задачи №1332

Алгоритм решения задачи: Во-первых, задача была сведена к другой, уменьшив радиус взрыва на радиус городов и указав размеры городов. Затем, чтобы достичь пары городов размером с точку, помещается джин двумя различными способами так, чтобы его взрыв точно коснулся этих двух точек, и вычисляется, сколько городов в целом уничтожено, и запоминается максимальный результат.

ID	Дата	Автор	Задача	Язык	Результат проверки	№ теста	Время работы	Выделено памяти
8270635	17:32:29 27 фев 2019	daniika_na	1332. Джинн-бомбардировки	G++ 7.1	Accepted		0.873	448 КБ

Рис.8. Результат проверки задачи на timus

Результат проведения тестов:

```
[=====] Running 20 tests from 3 test cases.
[-----] Global test environment set-up.
[-----] 12 tests from structTest
[ RUN      ] structTest.input1
[       OK ] structTest.input1 (0 ms)
[ RUN      ] structTest.operatorMinus1
[       OK ] structTest.operatorMinus1 (0 ms)
[ RUN      ] structTest.operatorMinus2
[       OK ] structTest.operatorMinus2 (0 ms)
[ RUN      ] structTest.operatorPower1
[       OK ] structTest.operatorPower1 (0 ms)
[ RUN      ] structTest.operatorPower2
[       OK ] structTest.operatorPower2 (0 ms)
[ RUN      ] structTest.operatorDiv1
[       OK ] structTest.operatorDiv1 (0 ms)
[ RUN      ] structTest.operatorDiv2
[       OK ] structTest.operatorDiv2 (0 ms)
[ RUN      ] structTest.operatorPlus1
[       OK ] structTest.operatorPlus1 (0 ms)
[ RUN      ] structTest.operatorPlus2
[       OK ] structTest.operatorPlus2 (0 ms)
[ RUN      ] structTest.normalized
[       OK ] structTest.normalized (0 ms)
[ RUN      ] structTest.dist2
[       OK ] structTest.dist2 (0 ms)
[ RUN      ] structTest.dist
[       OK ] structTest.dist (0 ms)
[-----] 12 tests from structTest (3 ms total)
```

```

[-----] 6 tests from inputData
[ RUN      ] inputData.NTest1
[          OK ] inputData.NTest1 (0 ms)
[ RUN      ] inputData.NTest2
[          OK ] inputData.NTest2 (0 ms)
[ RUN      ] inputData.RTest1
[          OK ] inputData.RTest1 (0 ms)
[ RUN      ] inputData.RTest2
[          OK ] inputData.RTest2 (0 ms)
[ RUN      ] inputData.rTest1
[          OK ] inputData.rTest1 (0 ms)
[ RUN      ] inputData.rTest2
[          OK ] inputData.rTest2 (0 ms)
[-----] 6 tests from inputData (2 ms total)

[-----] 2 tests from examples
[ RUN      ] examples.example1
[          OK ] examples.example1 (0 ms)
[ RUN      ] examples.example2
[          OK ] examples.example2 (0 ms)
[-----] 2 tests from examples (0 ms total)

[-----] Global test environment tear-down
[=====] 20 tests from 3 test cases ran. (6 ms total)
[ PASSED ] 20 tests.

```

Код программы:

main.cpp

```

#include <gtest/gtest.h>

#include "task.h"

int main(int argc, char* argv []) {
    testing::InitGoogleTest(&argc, argv);
    int code = RUN_ALL_TESTS();
    task();
    return code;
}

```

task.h

```

#ifndef PRACTICE_3_TASK_H

```

```

#define PRACTICE_3_TASK_H
#include <cmath>
struct point {
    double x, y;
    point(double x = 0, double y = 0) : x(x), y(y) {}
    point operator-(const point &p) const { return point(x
- p.x, y - p.y); }
    point operator*(double t) const { return point(t * x, t
* y); }
    point operator/(double t) const { return point(x / t, y
/ t); }
    point operator+(const point &p) const { return point(x
+ p.x, y + p.y); }
    point normalized() const { return point(x / dist(), y /
dist()); }
    double dist2() const { return x * x + y * y; }
    double dist() const { return std::sqrt(x * x + y *
y); }
};
void task();
#endif //PRACTICE_3_TASK_H

```

task.cpp

```

//
// Created by daniilka on 27.02.19.
//
#include <iostream>
#include "task.h"
#include <algorithm>
#include <cmath>
void task() {
    point *points= new point[100];
    const double EPS = 1e-6;

```



```

int N, x, y, R, r;
std::cin >> N;
if (N < 1) {
    throw std::logic_error("Count of cities <1");
}
if (N > 100) {
    throw std::logic_error("Count of cities >100");
}
for (int i = 0; i < N; i++)
    std::cin >> points[i].x >> points[i].y;
std::cin >> R >> r;
if (r <= 0) {
    throw std::logic_error("City radius <=0");
}
if (R <= 0) {
    throw std::logic_error("Bomb radius <=0");
}
if (r > 10000) {
    throw std::logic_error("City radius >10000");
}
if (R >= 10000) {
    throw std::logic_error("Bomb radius >10000");
}
R -= r;
int max = R >= 0;
for (int i = 0; i < N; i++) {
    for (int j = i + 1; j < N; j++) {
        int A = 0, B = 0;
        point m = (points[j] + points[i]) / 2;
        point v = points[j] - points[i];
        point perp = point(-v.y, v.x);
        double l = std::sqrt(
            R * R - v.dist2() / 4);
    }
}

```

```

        point a = m + perp.normalized() * l, b = m -
perp.normalized() * l;
        for (int k = 0; k < N; k++) {
            if ((points[k] - a).dist() <= R + EPS)
                A++;
            if ((points[k] - b).dist() <= R + EPS)
                B++;
        }
        max = std::max({max, A, B});
    }
    std::cout << max;
}

```

test.cpp

```

#include "task.h"

#include <gtest/gtest.h>
const double EPS = 1e-6;
TEST(structTest, input1) {
    point testPoint = point(1, 1);
    EXPECT_EQ(testPoint.x, 1);
    EXPECT_EQ(testPoint.y, 1);
}
TEST(structTest, operatorMinus1) {
    point testPoint1 = point(2, 2);
    point testPoint2 = point(1, 1);
    point testPointResult = testPoint2 - testPoint1;
    EXPECT_EQ(testPointResult.x, -1);
    EXPECT_EQ(testPointResult.y, -1);
}
TEST(structTest, operatorMinus2) {
    point testPoint1 = point(2, 2);
    point testPoint2 = point(1, 1);

```

```
    point testPointResult = testPoint1 - testPoint2;
    EXPECT_EQ(testPointResult.x, 1);
    EXPECT_EQ(testPointResult.y, 1);
}

TEST(structTest, operatorPower1) {
    point testPoint = point(2, 3) * 2;
    EXPECT_EQ(testPoint.x, 4);
    EXPECT_EQ(testPoint.y, 6);
}

TEST(structTest, operatorPower2) {
    point testPoint = point(2, 3) * -2;
    EXPECT_EQ(testPoint.x, -4);
    EXPECT_EQ(testPoint.y, -6);
}

TEST(structTest, operatorDiv1) {
    point testPoint = point(2, 6) / 2;
    EXPECT_EQ(testPoint.x, 1);
    EXPECT_EQ(testPoint.y, 3);
}

TEST(structTest, operatorDiv2) {
    point testPoint = point(2, 6) / -2;
    EXPECT_EQ(testPoint.x, -1);
    EXPECT_EQ(testPoint.y, -3);
}

TEST(structTest, operatorPlus1) {
    point testPoint1 = point(-2, -2);
    point testPoint2 = point(1, 1);
    point testPointResult = testPoint2 + testPoint1;
    EXPECT_EQ(testPointResult.x, -1);
    EXPECT_EQ(testPointResult.y, -1);
}

TEST(structTest, operatorPlus2) {
    point testPoint1 = point(2, 2);
```

```

    point testPoint2 = point(-1, -1);
    point testPointResult = testPoint1 + testPoint2;
    EXPECT_EQ(testPointResult.x, 1);
    EXPECT_EQ(testPointResult.y, 1);
}

TEST(structTest, normalized) {
    point testPoint = point(1, 1);
    testPoint.normalized();
    EXPECT_GT(testPoint.y, 1/sqrt(2)-EPS);
    EXPECT_GT(testPoint.x, 1/sqrt(2)-EPS);
}

TEST(structTest, dist2) {
    point testPoint = point(5, 5);
    EXPECT_EQ(testPoint.dist2(), 50);
}

TEST(structTest, dist) {
    point testPoint = point(5, 5);
    EXPECT_GT(testPoint.dist(), sqrt(50)-EPS);
}

TEST(inputData, NTest1){
    point *points = new point[2];
    EXPECT_THROW(task(0, 1, 1, points), std::logic_error);
}

TEST(inputData, NTest2){
    point *points = new point[2];
    EXPECT_THROW(task(101, 1, 1, points), std::logic_error);
}

TEST(inputData, RTest1){
    point *points = new point[2];
    EXPECT_THROW(task(1, 0, 1, points), std::logic_error);
}

TEST(inputData, RTest2){
    point *points = new point[2];

```

```

    EXPECT_THROW(task(1,10001,1,points),std::logic_error);
}

TEST(inputData,rTest1){
    point *points = new point[2];
    EXPECT_THROW(task(1,1,0,points),std::logic_error);
}

TEST(inputData,rTest2){
    point *points = new point[2];
    EXPECT_THROW(task(1,1,10001,points),std::logic_error);
}

TEST(examples,example1){
    point *points = new point[3];
    points[0].x=0;
    points[0].y=0;
    points[1].x=0;
    points[1].y=4;
    points[2].x=4;
    points[2].y=0;
    EXPECT_EQ(task(3,3,1,points),2);
}

TEST(examples,example2){
    point *points = new point[5];
    points[0].x=0;
    points[0].y=0;
    points[1].x=0;
    points[1].y=1;
    points[2].x=0;
    points[2].y=2;
    points[3].x=0;
    points[3].y=3;
    points[4].x=0;
    points[4].y=4;
    EXPECT_EQ(task(5,1,1,points),1);}

```

Основное задание

Требуется подсчитать количество зерен и измерить их характеристики по фотографии.

Для решения данной задачи была выбрана библиотека openCV, а так же алгоритм Watershed.

Ссылка на репозиторий:

<https://github.com/danilka-na/Practice>

Ссылка на фотографии:

https://yadi.sk/d/JjZf4gK9dWie_Q

Описание работы алгоритма:

Изображение загружается BGR, поэтому переводим в формат HSV. Из берем только Saturation. Далее применяем «Фильтр Лапласиана», для того, чтобы сделать границы объектов изображения более четкими. После чего производим размытие и эрозию с расширением изображения. Теперь изображение готово к началу обработки методом Watershed.

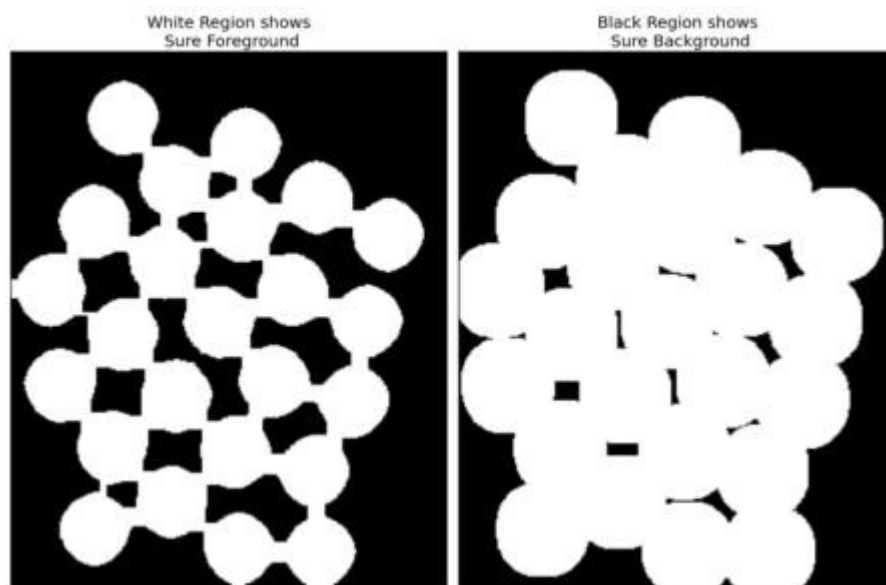


Рис. 9. Пример работы алгоритма Watershed.

Далее проводим сегментацию изображения, пре вращая его в бинарное. Следует провести открытое трансформирование для того, чтобы убирать мелкие объекты. Последним этапом выделяем задний и передний планы,

потому что при вычитании планов можно выделить тонкие контуры объектов. Остается создать маркеры и провести watershed. Места для закрашивания будут помечены -1. Далее мы считаем и подписываем все зерна.

Данным алгоритмом было обработано более 500 фотографий с зернами риса. Средняя погрешность при подсчете зерен приблизительно «-27» ~16% зерен. Это связано с тем, что некоторые зерна лежали очень плотно друг к другу и алгоритму не удалось их разделить.

Пример работы программы:

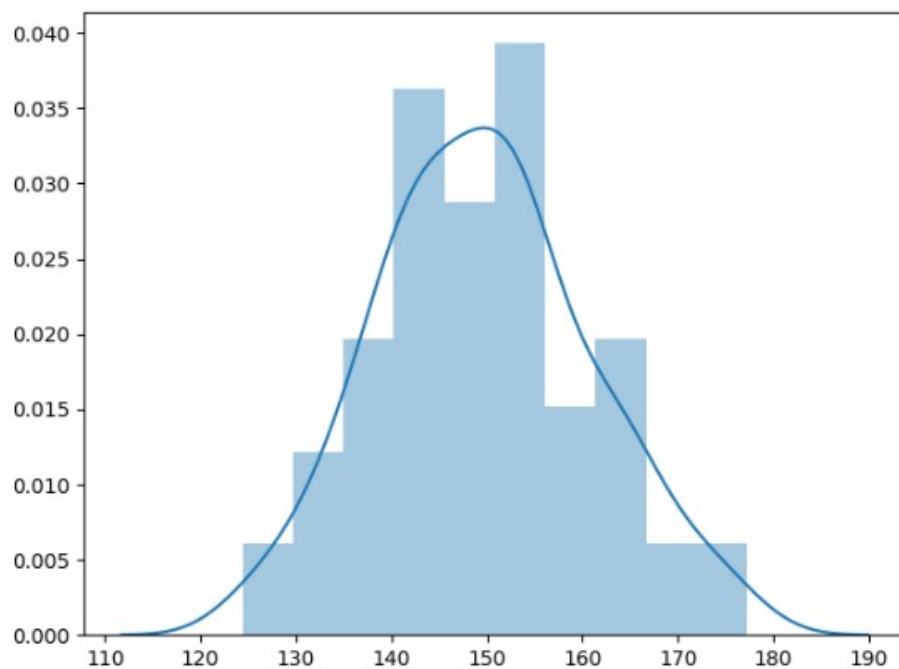


Рис. 10.
Результат
работы на 170
семенах

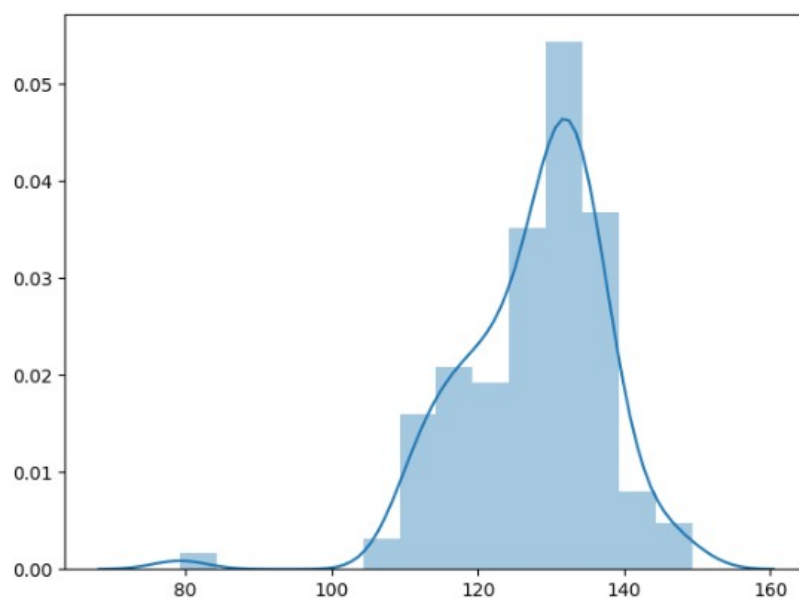


Рис. 11. Результат работы на 150 семенах

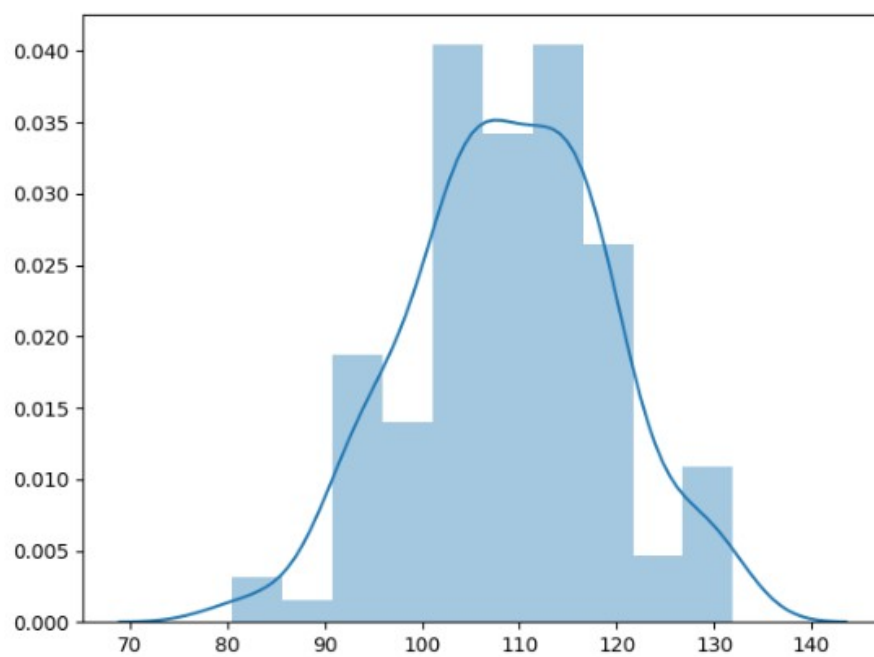


Рис. 12. Результат работы на 130 семенах

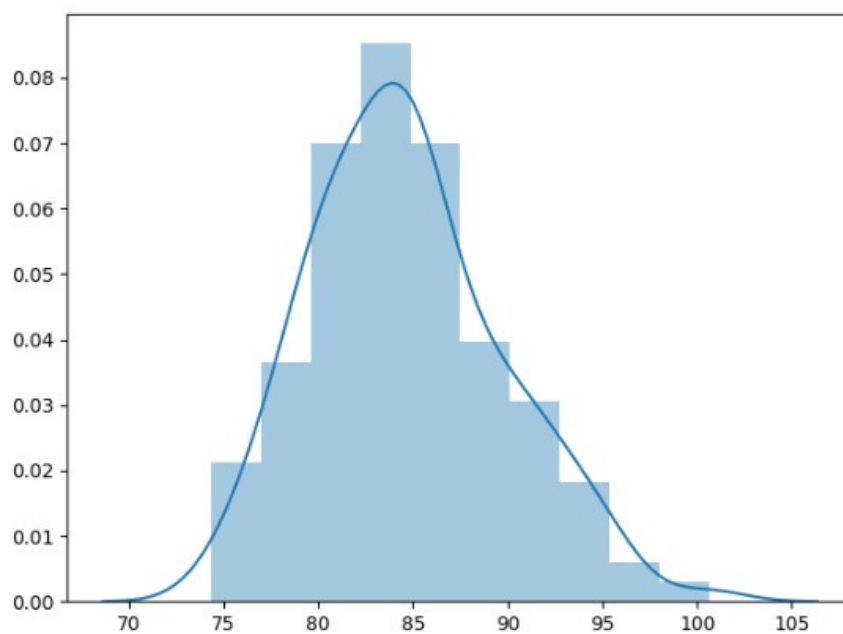


Рис. 13. Результат работы на 130 семенах

Данные графики можно обобщить в одной таблице:

Количество семян	Среднее	Стандартн ое отклонение	Количеств о элементов
170	138.9	12.3	125
150	127.2	11.1	125
130	115.3	7.5	125
100	85.1	4.2	125

Код программы:

main.cpp

```
#include <gtest/gtest.h>
```

```
#include "computerVision.hpp"
```

```
using namespace cv;
```

```
using namespace std;
```

```
int main(int argc, char *argv[]) {
```

```

testing::InitGoogleTest(&argc, argv);
int code = RUN_ALL_TESTS();
std::string dirName =
"/home/daniilka/CLionProjects/Grains_2/Photo/";
std::vector<std::string> files= getFileList(dirName);

ofstream myfile;
myfile.open ("res.csv");
for (int i = 0; i < files.size(); ++i) {
    vector<double> length;
    vector<double> width;
    vector<double> area;
    Mat image = imread(string(dirName+files.at(i)));
    //    int count = mainAlg(image, length, width, area);

    //    myfile << i-2<<","<<count << "\n";

}

myfile.close();

return 0;
}

```

computerVision.hpp

```
#include <opencv2/opencv.hpp>
#include "dirent.h"

using namespace cv;
using namespace std;

Mat imageRead(std::string path);
Mat laplacian(Mat &sharp);

Mat erode(Mat &subPlans, Mat &imgLaplacian);

Mat getMarkers(Mat subPlans, Mat &sureFigure, Mat &unknown) ;
vector<vector<Point> > drawContours(Mat markers, Mat
sureFigure);

int mainAlg(Mat resultImage, std::vector<double> &length,
            std::vector<double> &width, std::vector<double>
&area);

std::vector<std::string> getFileList(std::string path);
```

computerVision.cpp

```
#include "computerVision.hpp"
```

```
using namespace cv;
```

```
using namespace std;
```

```
const double PIXEL_SIZE = 0.102;
```

```
Mat imageRead(std::string path) {
```

```
    Mat img = imread(path.c_str());
```

```
    if(!img.data){
```

```
        throw std::invalid_argument("This file doesn't exist");
```

```
    }
```

```
    return imread(path.c_str());
```

```
}
```

```
Mat laplacian(Mat &sharp) {
```

```
    Mat kernel = (Mat_<float>(3, 3) << 1, 1, 1, 1, -8, 1, 1, 1, 1);
```

```
    Mat imgLaplacian;
```

```
    filter2D(sharp, imgLaplacian, CV_32F, kernel);
```

```
    return imgLaplacian;
```

```
}
```

```
Mat erode(Mat &subPlans, Mat &imgLaplacian) {
```

```
    subPlans.convertTo(subPlans, CV_8UC3);
```

```
    imgLaplacian.convertTo(imgLaplacian, CV_8UC3);
```

```

    blur(subPlans, subPlans, Size(3, 3));

    int erodeSz = 4;

    Mat structuringElement =
cv::getStructuringElement(MORPH_ELLIPSE,
                                                                    Size(2 *
erodeSz + 1,
                                                                    2 *
erodeSz + 1),
                                                                    Point(erodeSz, erodeSz));

    erode(subPlans, subPlans, structuringElement);
    dilate(subPlans, subPlans, structuringElement);
    bitwise_not(subPlans, subPlans);

    threshold(subPlans, subPlans, 40, 255,
              CV_THRESH_BINARY | CV_THRESH_OTSU);

    return subPlans;
}

Mat getMarkers(Mat subPlans, Mat &sureFigure, Mat &unknown) {
    Mat opening;
    Mat kernel_1(3, 3, CV_8U, Scalar(1));
    morphologyEx(subPlans, opening, MORPH_OPEN, kernel_1,
                Point(-1, -1), 1);

```

```

Mat sure_bg;
dilate(opening, sure_bg, kernel_1, Point(-1, -1), 3);

Mat distTransform;
distanceTransform(opening, distTransform, CV_DIST_L2, 5);

double minVal, maxVal;
Point minLoc, maxLoc;
minMaxLoc(distTransform, &minVal, &maxVal, &minLoc, &maxLoc);

threshold(distTransform, sureFigure, 0, 255, 0);

Mat sureFigure1;
sureFigure.convertTo(sureFigure1, CV_8UC1);
subtract(sure_bg, sureFigure1, unknown);

sureFigure.convertTo(sureFigure, CV_32SC1, 1.0);

return Mat::zeros(sureFigure.rows, sureFigure.cols, CV_32SC1);
}

vector<vector<Point> > drawContours(Mat markers, Mat sureFigure)
{
    vector<vector<Point> > contours;
    vector<Vec4i> hierarchy;
    findContours(sureFigure, contours, hierarchy, RETR_CCOMP,

```

```
CHAIN_APPROX_SIMPLE);
```

```
int compCount = 0;
```

```
for (int index = 0; index >= 0; index = hierarchy[index][0],  
compCount++)
```

```
drawContours(markers, contours, index, Scalar::all(compCount  
+ 1),
```

```
-1, 8, hierarchy, INT_MAX);
```

```
return contours;
```

```
}
```

```
int mainAlg(Mat resultImage, std::vector<double> &length,
```

```
std::vector<double> &width, std::vector<double>  
&area) {
```

```
Mat hsv;
```

```
cvtColor(resultImage, hsv, CV_BGR2HSV);
```

```
vector<Mat> channels;
```

```
split(hsv, channels);
```

```
Mat sharp = channels[1];
```

```
// imwrite("result1.png", sharp);
```

```
Mat imgLaplacian = laplacian(sharp);
```

```
imgLaplacian = laplacian(imgLaplacian);
```

```

//  imwrite("result2.png", imgLaplacian);

channels[1].convertTo(sharp, CV_32F);

Mat subPlans = sharp - imgLaplacian;
subPlans = erode(subPlans, imgLaplacian);

Mat sureFigure, contour;

Mat markers = getMarkers(subPlans, sureFigure, contour);

vector<vector<Point> > contours = drawContours(markers,
sureFigure);

markers = markers + 1;
for (int i = 0; i < markers.rows; i++) {
    for (int j = 0; j < markers.cols; j++) {
        unsigned char &v = contour.at<unsigned char>(i, j);
        if (v == 255) {
            markers.at<int>(i, j) = 0;

        }
    }
}

watershed(resultImage, markers);

```



```

for (int i = 0; i < markers.rows; i++) {
    for (int j = 0; j < markers.cols; j++) {
        int index = markers.at<int>(i, j);
        if (index == -1)
            resultImage.at<Vec3b>(i, j) = Vec3b(0, 255, 0);

    }
}

int count = 0;
for (unsigned int i = 0; i < contours.size(); i = i + 2) {
    RotatedRect rect = minAreaRect(contours[i]);
    putText(resultImage, to_string(count + 1), rect.center,
            FONT_ITALIC, 2, Scalar(0, 0, 255), 1);
    length.push_back(rect.size.height * PIXEL_SIZE);
    width.push_back(rect.size.width * PIXEL_SIZE);
    area.push_back(contourArea(contours[i]) * PIXEL_SIZE *
PIXEL_SIZE);
    count++;
}

// imwrite("result.png", resultImage);

return count;
}

```

```

std::vector<std::string> getFileList(std::string path){
    DIR *dir;
    struct dirent *ent;
    std::vector<std::string> files;
    if ((dir = opendir(path.c_str())) != NULL) {
        while ((ent = readdir(dir)) != NULL) {
            if (ent->d_name!= std::string("..") and ent->d_name!=
std::string(".")){
                files.push_back(ent->d_name);
            }
        }
        closedir(dir);
    } else {

        throw std::invalid_argument("This directory doesn't exist");

    }
    return files;
}

```

test.cpp

```

#include <gtest/gtest.h>
#include "computerVision.hpp"

const double EPS = 5.0;

```

```

TEST(structTest, imageRead) {
    EXPECT_THROW(imageRead("kjsdfljs"), std::invalid_argument);
}

TEST(structTest, imageRead2) {
    EXPECT_NO_FATAL_FAILURE(

imageRead("/home/daniilka/CLionProjects/Grains_2/1233.JPG"));
}

TEST(structTest, imageRead3) {
    EXPECT_THROW(

imageRead("/home/daniilka/CLionProjects/Grains_2/1233.jpg"),
            std::invalid_argument);
}

TEST(structTest, imageRead4) {
    EXPECT_THROW(

imageRead("/home/daniilka/CLionProjects/Grains_2/"),std::excepti
on);
}

TEST(laplacianTest, test1) {
    Mat kernel = (Mat_<float>(5, 5) <<
                                0, 0, 0, 0, 0,
                                0, 255, 255, 255, 0,
                                0, 255, 255, 255, 0,

```

```

        0, 255, 255, 255, 0,
        0, 0, 0, 0, 0);

Mat image = laplacian(kernel);

EXPECT_GT (image.at<double>(3, 3) + EPS, 0.0);

}

TEST(laplacianTest, test2) {
    Mat kernel = (Mat_<float>(5, 5) <<
                  255, 255, 255, 255, 255,
                  255, 0, 0, 0, 255,
                  255, 0, 0, 0, 255,
                  255, 0, 0, 0, 255,
                  255, 255, 255, 255, 255);

    Mat image = laplacian(kernel);

    EXPECT_GT (0.0 + EPS, image.at<double>(3, 3));
}

TEST(erodeTest, test1) {
    Mat kernel = (Mat_<float>(5, 5) <<
                  255, 255, 255, 255, 255,
                  255, 0, 0, 0, 255,
                  255, 0, 100, 0, 255,
                  255, 0, 0, 0, 255,

```

```

        255, 255, 255, 255, 255);
Mat lap = laplacian(kernel);
Mat image = erode(kernel, lap);

EXPECT_GT (0.0 + EPS, image.at<double>(3, 3));
}

```

```

TEST(erodeTest, test2) {
    Mat kernel = (Mat_<float>(5, 5) <<
        255, 255, 255, 255, 255,
        255, 255, 255, 255, 255,
        255, 255, 255, 255, 255,
        255, 255, 255, 255, 255,
        255, 255, 255, 255, 255);
    Mat lap = laplacian(kernel);
    Mat image = erode(kernel, lap);

    EXPECT_GT (0.0 + EPS, image.at<double>(3, 3));
}

```

```

TEST(erodeTest, test3) {
    Mat kernel = (Mat_<float>(5, 5) <<
        0, 0, 0, 0, 0,
        0, 0, 255, 0, 0,
        0, 255, 255, 255, 0,
        0, 0, 255, 0, 0,

```

```

        0, 0, 0, 0, 0);
Mat lap = laplacian(kernel);
Mat image = erode(kernel, lap);

EXPECT_GT (0.0 + EPS, image.at<double>(3, 3));
}

TEST(erodeTest, test4) {
    Mat
    kernel = (Mat_<float>(6, 6) <<
        255, 255, 255, 255, 255, 255,
        255, 0, 0, 0, 255, 255,
        255, 0, 0, 0, 255, 255,
        255, 0, 0, 0, 255, 255,
        255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255);
    Mat lap = laplacian(kernel);
    Mat image = erode(kernel, lap);

    EXPECT_GT (0.0 + EPS, image.at<double>(3, 3));
}

TEST(markerTest, centerTest1) {
    Mat img = imread("rice.png");
    Mat hsv;

```

```

cvtColor(img, hsv, CV_BGR2HSV);

vector<Mat> channels;
split(hsv, channels);
Mat sharp = channels[2];

Mat imgLaplacian = laplacian(sharp);
imgLaplacian = laplacian(imgLaplacian);

channels[2].convertTo(sharp, CV_32F);

Mat subPlans = sharp - imgLaplacian;
subPlans = erode(subPlans, imgLaplacian);

Mat sureFigure, unknown;
Mat markers = getMarkers(subPlans, sureFigure, unknown);

EXPECT_EQ(0, unknown.at<double>(30, 60));
EXPECT_GT(0, unknown.at<double>(25, 5)-EPS);
}

TEST(markerTest, centerTest2) {
    Mat img = imread("rice.png");
    Mat hsv;

```

```

    cvtColor(img, hsv, CV_BGR2HSV);

    vector<Mat> channels;
    split(hsv, channels);
    Mat sharp = channels[2];

    Mat imgLaplacian = laplacian(sharp);
    imgLaplacian = laplacian(imgLaplacian);

    channels[2].convertTo(sharp, CV_32F);

    Mat subPlans = sharp - imgLaplacian;
    subPlans = erode(subPlans, imgLaplacian);

    Mat sureFigure, unknown;
    Mat markers = getMarkers(subPlans, sureFigure, unknown);

    EXPECT_EQ(0, unknown.at<double>(100, 60));

}

TEST(markerTest, grainTest1) {
    Mat img = imread("rice.png");
    Mat hsv;

```



```

    cvtColor(img, hsv, CV_BGR2HSV);

    vector<Mat> channels;
    split(hsv, channels);
    Mat sharp = channels[2];

    Mat imgLaplacian = laplacian(sharp);
    imgLaplacian = laplacian(imgLaplacian);

    channels[2].convertTo(sharp, CV_32F);

    Mat subPlans = sharp - imgLaplacian;
    subPlans = erode(subPlans, imgLaplacian);

    Mat sureFigure, unknown;
    Mat markers = getMarkers(subPlans, sureFigure, unknown);

    EXPECT_GT(0, unknown.at<double>(25, 5)-EPS);

}

TEST(markerTest, grainTest2) {
    Mat img = imread("rice.png");

```

```
Mat hsv;
cvtColor(img, hsv, CV_BGR2HSV);

vector<Mat> channels;
split(hsv, channels);
Mat sharp = channels[2];

Mat imgLaplacian = laplacian(sharp);
imgLaplacian = laplacian(imgLaplacian);

channels[2].convertTo(sharp, CV_32F);

Mat subPlans = sharp - imgLaplacian;
subPlans = erode(subPlans, imgLaplacian);

Mat sureFigure, unknown;
Mat markers = getMarkers(subPlans, sureFigure, unknown);

EXPECT_GT(0, unknown.at<double>(120, 5) - EPS);

}

TEST(getFileListTest, listTest1) {
```

```
    std::string path = "dfjakl";
    EXPECT_THROW(getFileList(path), std::invalid_argument);
}

TEST(getFileListTest, listTest2) {
    std::string path = "..";
    EXPECT_NO_THROW(getFileList(path));
}

TEST(getFileListTest, listTest3) {
    std::string path = ".";
    EXPECT_NO_THROW(getFileList(path));
}

TEST(getFileListTest, listTest4) {
    std::string path =
"/home/daniilka/CLionProjects/Grains_2/Photo/";
    EXPECT_NO_THROW(getFileList(path));
}
```