ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕУЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ
ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ

Отчет о прохождении технологической практики на тему: «Автоматизация проверки практических работ»

Дробилко Валерий Александрович, гр. 5130903/10301

Направление подготовки: 09.03.03 Прикладная информатика.
Место прохождения практики: СПбПУ, ИКНК, ВШПИ.
Сроки практики: с 10.06.2024 по 08.07.2024.
Руководитель практики от ФГАОУ ВО «СПбПУ»: Туральчук К.А., к. т. н., доцент ВШПИ.
Консультант практики от ФГАОУ ВО «СПбПУ»: нет.
Оценка: _____

Руководитель практики
от ФГАОУ ВО «СПбПУ» К.А. Туральчук

Обучающийся В.А. Дробилко

Дата: 07.07.2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Автоматизация любого процесса в жизнедеятельности человека всегда будет актуальной задачей, так как потраченное время и силы на решение всегда будет окупать время, которое человек может потратить на работу с неавтоматизированным процессом.

В своей технологической практике мне нужно автоматизировать проверку практических работ для курса «Объектно-ориентированной программирование» (2 учебный курс) на языке C++.

Для того, чтобы достичь данной цели недостаточно использовать только библиотеку Google-test. По мимо неё мне также предстоит изучить методы статического анализа кода, так как большинство работ курса, которые нужно автоматизировать, несут в себе обучаемо-теоретический характер (использование lambda-выражений, открытых и закрытых методов, определенные иерархии классов и прочее) нежели алгоритмический.

После того, как для всех работ будут созданы модульные тесты, мне также предстоит создать приложение для автоматизации проверки различных проектов разных студентов. Это приложение будет автоматически собирать решение для проверки работы студента, исходя из выбранного номера лабораторной работы, варианта и исходных файлов проекта студента.

По итогу, разработанная мной программа должна упростить проверку работ преподавателем по курсу «Объектно-ориентированной программирование», избавить его от рутинного просмотра и проверки кода каждого студента.

ГЛАВА 1. ОПРЕДЕЛЕНИЕ ОБЪЕКТА АВТОМАТИЗАЦИИ

Прежде, чем приступить к написанию модульных тестов для 6-ти работ, нужно определить то, что нужно проверять в каждой из них. Где-то достаточно просто написать в Google-test тест для проверки правильной работы определенной функции, а где-то придется использовать методы статического анализа кода. Также, для каждой работы, следует указать то, что следует изменить преподавателю в задании к лабораторной работе, так как для автоматизации будет проще, чтобы студенты использовали единые имена классов, функций, структуру проекта и другое.

1.1. Цели автоматизации лабораторной работы №1

В данной лабораторной работе от студентов требуется реализовать ряд функций для знакомства с типами и основными конструкциями языка C++. Таким образом, мне требуется создать модульные тесты для проверки функций:
1) Функция belongsToInterval определяет, принадлежит ли значение x заданному интервалу.
2) Функция minValue возвращает меньшее из двух целочисленных аргументов. В реализации задействуйте тернарный оператор.
3) Функция minValue возвращает меньшее из двух аргументов типа float.
4) Функция order возвращает порядок числа (количество знаков). Параметры: x – любое целочисленное значение.
5) Функция getLetterCode возвращает числовое представление для любого символа.

В задании к работе следует указать:
1) Структура проекта состоит из двух файлов – Source.h и Source.cpp. В файле Source.h реализовать функции, в Source.cpp – точка входа и работа с консолью.
2) Для функции order дать пояснение сигнатуры. В тестах учитывается, что аргумент может быть типа long long.
3) Для функции belongsToInterval указать строгость знака для границ. В тестах границы не строгие, то есть bottom<=x<=top.

1.2. Цели автоматизации лабораторной работы №2

В данной лабораторной работе от студентов требуется реализовать работу с динамическими массивами и указателями в C++. Существует 8 вариантов заданий,

где в каждом из них нужно создать различные функции для работы с матрицами. Таким образом, мне требуется написать модульные тесты для таких функций:

1) В-0, матричное умножение;
2) В-1, умножение матрицы на вектор;
3) В-2, сложение матриц;
4) В-3, поэлементное умножение матриц;
5) В-4, вычисление следа квадратной матрицы;
6) В-5, вычитание матриц;
7) В-6, умножение строки на столбец;
8) В-7, умножение матрицы на число.

В задании к работе следует указать такие сигнатуры и имена для функций:

1) int** multipMatrix(int** matrix1, int row1, int column1, int** matrix2, int row2, int column2);
2) int* multipVector(int** matrix, int* vector, int row, int column);
3) int** addMatrix(int** matrix1, int** matrix2, int row, int column);
4) int** elemMultipMatrix(int** matrix1, int** matrix2, int row, int column);
5) int traceMatrix(int** matrix, int row);
6) int** submatrix(int** matrix1, int** matrix2, int row, int column);
7) int multipRowByColumn(int* row, int* column, int length);
8) int** multipMatrixNumber(int** matrix, int row, int column, int number);

Структура проекта для студентов: Source.cpp (main), Source.h (реализация функций).

1.3. Цели автоматизации лабораторной работы №3

В данной лабораторной работе от студентов требуется реализовать пользовательские типы данных в виде структур и классов. В зависимости от варианта необходимо реализовать функции для работы с данными типами. Таким образом, мне требуется написать модульные тесты для таких функций:

1) Принадлежат ли точки одной прямой;
2) Нахождение максимальной группы точек, которые лежат на одной прямой (нет другой группы, состоящей из большего количества точек, которые лежат на одной прямой);
3) Найти все тройки точек массива, через которые можно провести прямую линию;
4) Нахождение двух пар точек, которые определяют параллельные линии. Аргументы (p11, p12) и (p21, p22) соответствуют парам точек, которые задают

параллельные линии;

5) Найти такую точку, что окружность радиуса R с центром в этой точке содержит максимальное число точек заданного множества;

6) Найти такую точку, сумма расстояний от которой до остальных точек множества максимальна;

7) Найти такую точку, сумма расстояний от которой до остальных точек множества минимальна;

8) Найти такую точку, что окружность радиуса R с центром в этой точке содержит минимальное число точек заданного множества;

9) Нахождение всех прямоугольных треугольников

10) Нахождение всех равнобедренных треугольников;

11) Найти три точки, образующие треугольник наибольшего периметра;

12) Найти три точки, образующие треугольник наименьшего периметра. Возвращается периметр найденного треугольника;

13) Нахождение фигуры наибольшего периметра, которую можно сконструировать из 4-х точек массива (в зависимости от варианта вместо Figure использовать Rectangle, Square, Rhomb или Практическая работа 3. Фигуры и точки 4 Trapeze);

14) Создание фигуры по точкам (в зависимости от варианта вместо Figure использовать Rectangle, Square, Rhomb или Trapeze). Передается массив из 4-х точек в произвольном порядке;

15) Для заданного массива фигур найти пару фигур, центры которых наиболее близки к друг другу (в зависимости от варианта вместо Figure использовать Rectangle, Square, Rhomb или Trapeze);

16) Проверка пересечения двух фигур (в зависимости от варианта вместо Figure использовать Rectangle, Square, Rhomb или Trapeze).

В задании к работе следует указать:

1) Структура проекта: Source.cpp, Point.h, Point.cpp (здесь также реализации функций из L и P), Figure.h, Figure.cpp (здесь также реализации функций из T и F);

2) Изменить сигнатуры функций на: bool inLine(Point* points, int size), int getPointsInLine(Point* points, int size), int countLines(Point* points, int size), int getRectTriangles(Triangle* t, int size), int getIsoscelesTriangles(Triangle* t, int size), Point* getMaxLengthTriangle(Point* points, int size), Point* getMinLengthTriangle(Point* points, int size);

3) В классе Point важно, чтобы были реализованы данные методы: void SetPoint(double , double), double GetX(), double GetY();

4) В классе Triangle важно, чтобы были реализованы данные методы: void SetFigure(Point , Point , Point), Point GetP1(), Point GetP2(), Point GetP3(), Point GetP4().

5) Изменить описание функций на: T1 - Нахождение ЧИСЛА всех прямоугольных треугольников, T4 - Найти три точки, образующие треугольник наименьшего периметра;

6) В этих функциях учесть: F1-4 - Важно, чтобы фигуры, которые строились в этих функциях, строились по точкам по часовой стрелки, а также начинались с той точки, которая стояла первее в начальном массиве. Например, передаем в функцию массив из таких точек: [0](0, 0), [1](0, 10), [2](10, 0), [3](10, 10), то ожидается, что x = (0, 0), y = (10, 0), z = (10, 10), t = (0, 10). Таким образом, важен порядок, что упростит проверку в тестах; F3 - В этой функции важно, чтобы f1 и f2 были фигурами в том порядке, в котором передавались из начального массива. То есть, если передается Rect1, Rect2, Rect3 и из этих фигур по результатам работы функции подходят Rect1 и Rect3, то f1 = Rect1, а f2 = Rect3, НО не наоборот.

1.4. Цели автоматизации лабораторной работы №4

В данной лабораторной работе от студентов требуется реализовать класс в соответствии с вариантом. В этих классах нужно использовать определенные языковые конструкции, поэтому здесь придется использовать методы статического анализа. Таким образом, мне требуется написать модульные тесты для проверки:

1) Реализован ли нужный класс;
2) Реализованы ли открытые и закрытые поля;
3) Реализован ли конструктор;
4) Реализован ли деструктор;
5) Реализованы ли операторы (+, =, += и так далее);

В задании к работе следует указать:

1) Структура проекта: Source.cpp, Class.h, Class.cpp;
2) Для всех вариантов в классах функции имеют такие имена: push(), pop(), search(), compareSize();
3) Для B0 дополнительная функция имеет имя - getAllWithPriority();
4) Для B2 дополнительные функции имеют имена: popSeveral(), numberAvailable().

1.5. Цели автоматизации лабораторной работы №5

В данной лабораторной работе от студентов требуется реализовать иерархию классов для базового класса в соответствии с вариантом. Для этой работы также будут использованы методы статического анализа кода для выявления и тестирования иерархии. Таким образом, мне требуется написать модульные тесты для проверки:

1) Реализации базового класса в соответствии с вариантом;
2) Реализации не менее 5 производных классов;
3) Наличия не менее 3-х уровней иерархии;
4) Наличия открытых и закрытых полей в каждом классе;
5) Наличия виртуальных методов, а также переопределение;
6) Наличие статических элементов хотя бы в 3-х классах.

В задании к работе следует указать:

1) Структура проекта: Source.cpp, Class1.h, Class1.cpp, и так далее;
2) Для каждого класса есть свой заголовочный файл (.h) и файл с реализациями (.cpp).

1.6. Цели автоматизации лабораторной работы №6

В данной лабораторной работе от студентов требуется работу с контейнерами и алгоритмами STL. В этой работе также важно использовать различные языковые конструкции (лямбда-выражения, функторы и другие). Значит в этой работе также понадобятся методы статического анализа кода. Таким образом, мне требуется написать модульные тесты для проверки:

1) Реализации базового класса в соответствии с вариантом;
2) Реализации не менее 3-х полей разного типа;
3) Реализации операций, указанных в задании;
4) Реализации лямбда-выражений;
5) Реализации функторов;
6) Реализации работы с файлами с использованием потокового объекта ifstream\ofstream.

В задании к работе следует указать:

1) Структура проекта: Source.cpp, Class.h, Class.cpp (Class – это имя класса);
2) Проверяются только те классы, которые указаны по вариантам (данный тест можно отключить, если это не важно).

ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНЫХ РЕШЕНИЙ ТЕСТИРОВАНИЯ

После того, как были определены задачи на каждую работу, можно приступать к реализации проверки этих самых работ. Для всех 6 лабораторных работ ядром тестирования в моих проектах служит библиотека Google Test. Данная библиотека предоставляет удобный функционал для реализации модульного тестирования на языке C++. Особенности библиотеки:

1) Минимальной единицей тестирования является одиночный тест;
2) Тесты не требуется отдельно регистрировать для запуска;
3) Тесты объединяются в группы;
4) Тесты могут использовать общие данные, что позволяет логически объединять тесты и создавать универсальную среду для тестирования всего модуля или класса;
5) Библиотека является безопасной для многопотокового использования.

Приведу пример реализации такого метода тестирования в своих проектах. В первой лабораторной работе студентам нужно реализовать функцию minValue, которая возвращает меньшее из двух целочисленных аргументов. Код тестирования данной функции представлен на рисунке 2.1:

```
TEST(TestCase_minValueInt, Test2) {
     int x = 20;
     int y = 10;
     int expected;

     expected = y;
     EXPECT_EQ(minValue((int)x, (int)y), expected);
}
```

Рисунок 2.1 – алгоритм реализации тестирования функции minValue.

Таких тестов можно создавать сколько угодно, но важно, чтобы они имели разные имена в пределах одного тест кейса. Тест кейс – это такой элемент, который логически объединяет группу тестов. Например, если я хочу создать еще один тест для проверки правильной реализации функции minValue, то в макрос TEST() первым параметром я передам имя тест кейса, в этом случае это TestCase_minValueInt, а вторым параметром имя теста (Необязательно Test3, но они не должны повторятся).

Все тесты запускаются в точке входа в программу тестирования. Пример запуска тестов представлен на рисунке 2.2:

```
int main(int argc, char** argv) {
     testing::InitGoogleTest(&argc, argv);
     RUN_ALL_TESTS();
     std::cin.get();

     return 0;
}
```

Рисунок 2.2 – алгоритм запуска тестов.

На рисунке видно, что в функцию инициализации тестов передаются аргументы запуска программы. Это важная особенность, так как теперь при помощи параметра запуска программы тестирования можно указать аргумент «--gtest_filter=…», в котором указать только те тесты, которые нужны для текущего запуска. В моих тестах это очень удобная функция, так как не весь функционал лабораторной работы нужно реализовывать студенту, а соответственно и проверять. Обычно у варианта нужно реализовать свой отличный от других вариантов функционал. Пример запуска тестов для лабораторной работы №3 показан на рисунке 2.3:

```
cd ..\..\..\..\project\lab3_plug\x64\Debug
Lab3-test.exe --gtest_filter=TestCase_L.Test2:TestCase_P.Test2:TestCase_Triangle.Test3
@pause
```

Рисунок 2.3 – запуск программы тестирования в командной строке с применением фильтрации.

Используя всего лишь Google Test, я покрыл тестами весь функционал, определенный в первой главе, для лабораторных работ №1, 2 и 3. Для лабораторных работ №4, 5 и 6 только Google Test недостаточно – нужно также задействовать методы статического анализа кода. Для решения данной проблемы в своих проектах я использовал библиотеку clang-c (libclang). Данная библиотека предоставляет API, который реализует средства для синтаксического анализа кода в абстрактное синтаксическое дерево (AST), загрузки уже проанализированного AST, обхода AST, связывания физических исходных местоположений с элементами в AST и других средств. Для чего вообще использовать данную библиотеку? Все просто, данная библиотека строит дерево на основе реального кода, определяя тип и значения каждого элемента этого дерева. Например, если нужно определить, существует ли определенный класс, простым способом было бы просто открыть файл .cpp в коде, перевести его содержимое в строку и найти подстроку с таким именем, но это вовсе неверная реализация, так как это имя может находиться в комментарии, или код может быть вовсе некорректным. При помощи библиотеки же можно точно и легко определить, существует ли данный класс или нет, а также верный ли вообще код, ведь тогда в этом не будет смысла. Есть очень много таких нюансов, поэтому использование библиотеки является нужным.

Таким образом, во всех тестах в лабораторных работах №4, 5 и 6 я создаю AST деревья и работаю с ними, а уже по результату работы с этим деревом (например, реализованы ли лямбда-выражения и прочее) в тесте производится сравнение с ожидаемым результатом.

Для того, чтобы подключить библиотеку, нужно сначала скачать Clang к себе на компьютер. Далее в проекте Visual Studio в разделе «Свойства проекта» выбрать: 1) В разделе «Свойства конфигурации»->«C++»->«Дополнительные каталоги» нужно указать папку «\include\» в папке Clang; 2) В разделе «Свойства конфигурации»->«Компановщик»->«Общие»->«Дополнительные каталоги» нужно указать папку «\lib\» в папке Clang; 3) В разделе «Свойства конфигурации»->«Компановщик»->«Ввод»->«Дополнительные зависимости» нужно указать все файлы *.lib папки «/lib/», находящаяся в папке Clang.

Для того, чтобы создать дерево AST нужно сначала подключить файл, на основе кода которого будет происходить разбор. Пример алгоритма создания дерева представлен на рисунке 2.4:

```
CXIndex index = clang_createIndex(0, 1);

// Добавление флага для компиляции c++, а не c
char** new_argv = new char* [gArgc + 2];
for (int i = 0; i < gArgc; i++)
{
     new_argv[i] = gArgv[i];
}
new_argv[gArgc] = "-x";
new_argv[gArgc + 1] = "c++";

// Создание строки каталога исходного файла
string source_filename = "../../Lab5_plug/"; // ИЗМЕНИТЬ
source_filename.append(className[i]);
source_filename.append(".h");

CXTranslationUnit unit = clang_parseTranslationUnit(
     index, // CIdx
     source_filename.c_str(), // source_filename
     new_argv, // command_line_args
     gArgc + 2, // num_command_line_args
     0, // unsave_files
     0, // num_unsaved_files
     CXTranslationUnit_None // options

      );
```

Рисунок 2.4 – создание AST дерева на основе подключаемого файла с кодом.

В данном примере также можно увидеть, что функция создания дерева поддерживает передачу аргументов командной строки. Это очень важно для моих проектов, так как, например, когда мне нужно построить дерево файла заголовка (*.h), по умолчанию библиотека считает, что это язык Си, а не C++, поэтому важно установить параметр «-x c++», указывающий на то, что файл написан на языке C++.

После создания дерева можно приступать к работе с ним. Для начала обычно создается верхушка дерева, а после запускается запуск обхода всего

дерева, начиная с этой верхушки. Пример алгоритма запуска и обхода всего дерева показан на рисунке 2.5:

```
CXCursor root = clang_getTranslationUnitCursor(unit);
clang_visitChildren(root, searchAccessSpecifier, nullptr);
```

Рисунок 2.4 – запуск обхода дерева AST.

На рисунке также видно, что 2-м параметром для запуска обхода дерева передаётся функция. Данная функция выполняется для всех узлов при проходе по дереву. В этих функциях будет заложена основная логика каждого из тестов. Именно в них будет определяться существует ли определенное выражение, класс и другое. Пример алгоритма такой функции представлен на рисунке 2.6:

```
bool condStat = false;

CXChildVisitResult findStatic2(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
CXSourceLocation location = clang_getCursorLocation(cursor);
if (clang_Location_isFromMainFile(location) == 0)
                return CXChildVisit_Continue;


// Основная часть - начало

if (clang_getCursorKind(cursor) == 9)
{
                condStat = true;
}


// Основная часть - конец

return CXChildVisit_Continue;
}
```

Рисунок 2.5 – функция, запускаемая на всех узлах AST дерева.

Таким образом, схема выполнения алгоритма работы с синтаксическим анализом кода можно представить так, как показано на рисунке 2.6:

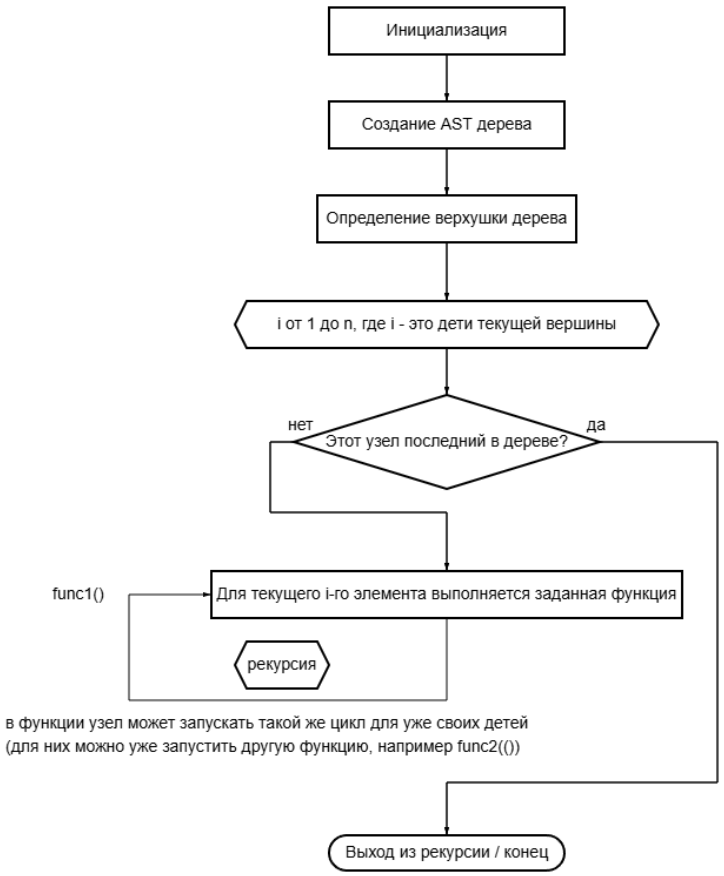

Рисунок 2.6 – алгоритм выполнения синтаксического анализа.

После отработки основных таких функций происходит сравнение полученных результатов с тем, что ожидалось.

Библиотеки Clang-c и Google Test не единственные, которые я использовал. В лабораторной работе № 5 нужно построить иерархию классов. Самая главная трудность заключается в том, что количество классов и их имена могут различаться. Для того, чтобы определять, какие классы существуют, я смотрю в каталог с проектом студента и во всех файлах .h смотрю на объявления классов. Библиотека boost как раз и помогает мне с тем, чтобы открыть каталог и взять оттуда все нужные мне файлы.

ГЛАВА 3. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ АВТОМАТИЗАЦИИ ПРОВЕРКИ РАБОТ

После того, как основная задача выполнена, можно создать оконное приложение, которое упростит работу со всеми разработанными проектами. Приложение будет работать, как командный центр – выбирать вариант, лабораторную работу, загружать выбранные файлы в нужный (выбранный нами в приложении) проект, пересобирать его, а уже только после всего этого запускать в командной строке с нужными аргументами. Схема того, как будет

выглядеть финальный программный продукт представлена на рисунке 3.1:

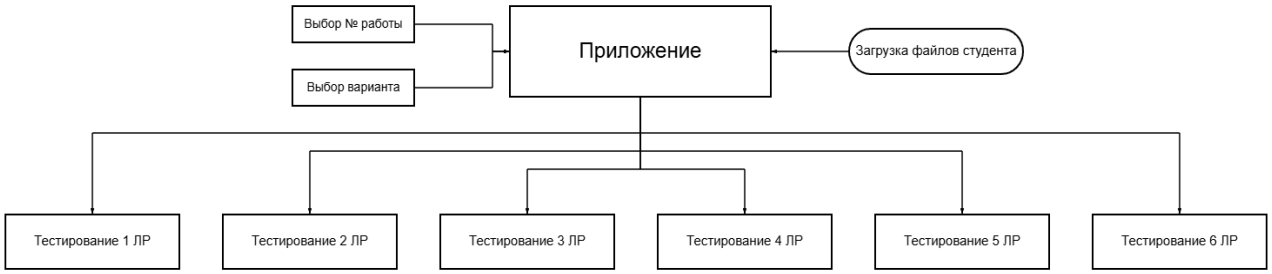

Рисунок 3.1 – схема работы финального программного продукта.

Стоит уточнить, как именно работает взаимодействие между главным приложением и тестирующими проектами. Внутри каждого проекта с тестированием лежит bat-файл, в котором написана команда пересобирающая текущий проект. Это нужно для того, чтобы можно было тестировать работы разных студентов, при этом не заходя в visual studio. Данная команда всегда запускается перед запуском тестирующей программы. Запуск тестирующей программы устроен схожим образом, только она разбита на несколько bat-файлов, где каждый такой файл в качестве аргумента передает номер варианта.

Вернемся к главному приложению. Данное оконное ПО было создано на языке C# с использованием технологии WPF. Интерфейс данного приложения представлен на рисунке 3.2:

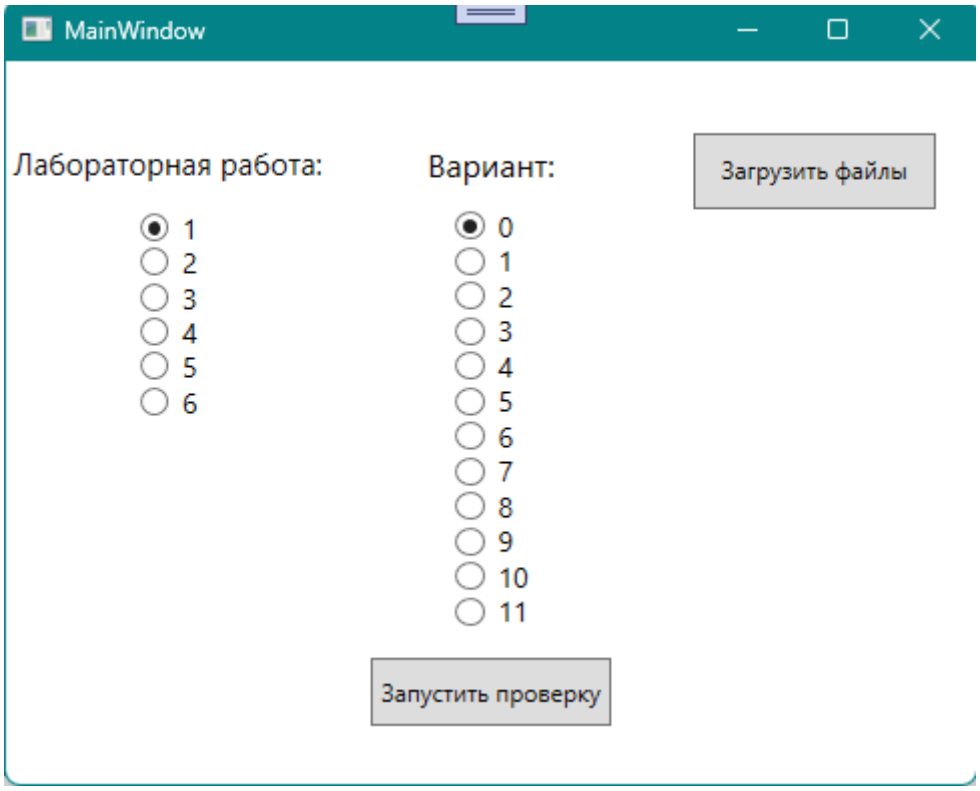

Рисунок 3.2 – главное оконное приложение.

На рисунке видно, что в главном приложении выбирается лабораторная работа, номер варианта, а также загружаются файлы студента, которые будут

тестироваться. По нажатию на кнопку «Запустить проверку» сначала копируются загруженные файлы в выбранный проект, после происходит сборка проекта, и в самом конце запускается проект с учётом варианта. По шаговая схема работы программы представлена на рисунке 3.3:

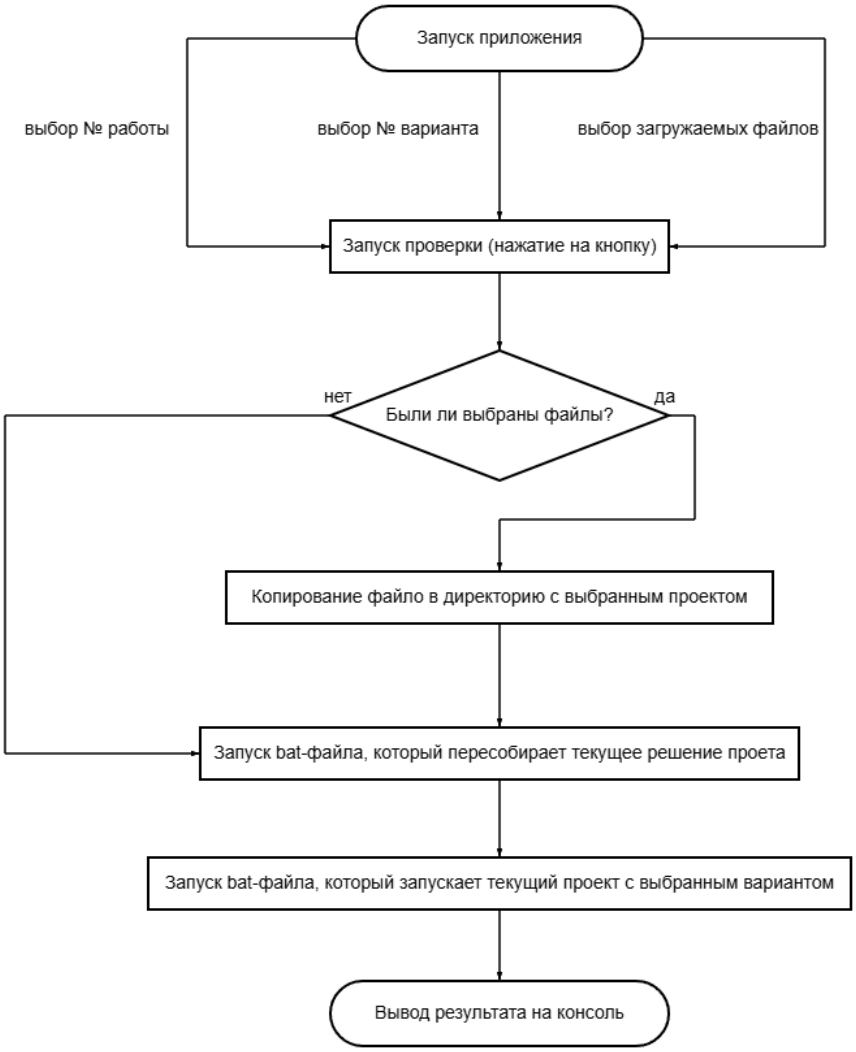

Рисунок 3.3 – по шаговая работа всего приложения.

ГЛАВА 4. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ НА ПРОЕКТАХ СТУДЕНТОВ

В данной главе проведём тестирование разработанного мной приложения. Запустим для тестирования проект 1 лабораторной работы студента S. Результат тестирования представлен на рисунке 4.1:

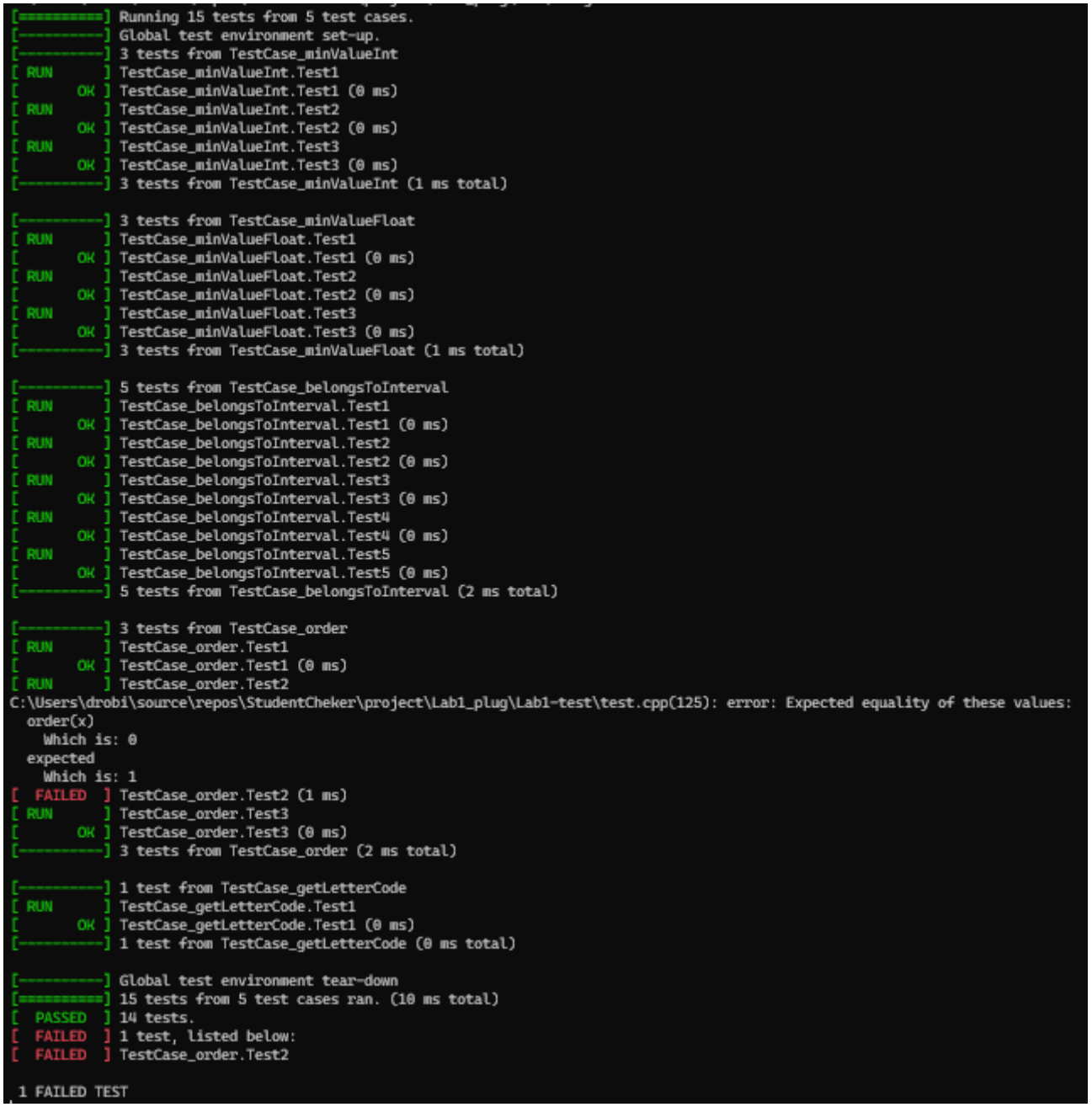

Рисунок 4.1 – тестирование работы №1 студента S.

Как видно на рисунке, у студента не прошел тест с сортировкой. Если посмотреть код его проекта, то можно убедиться, что он действительно расставил строгие знаки для границ, что и привело к ошибке.

Теперь запустим проект 5 лабораторной работы студента S2. Не забудем указать его вариант (3). Результат тестирования представлен на рисунке 4.2:

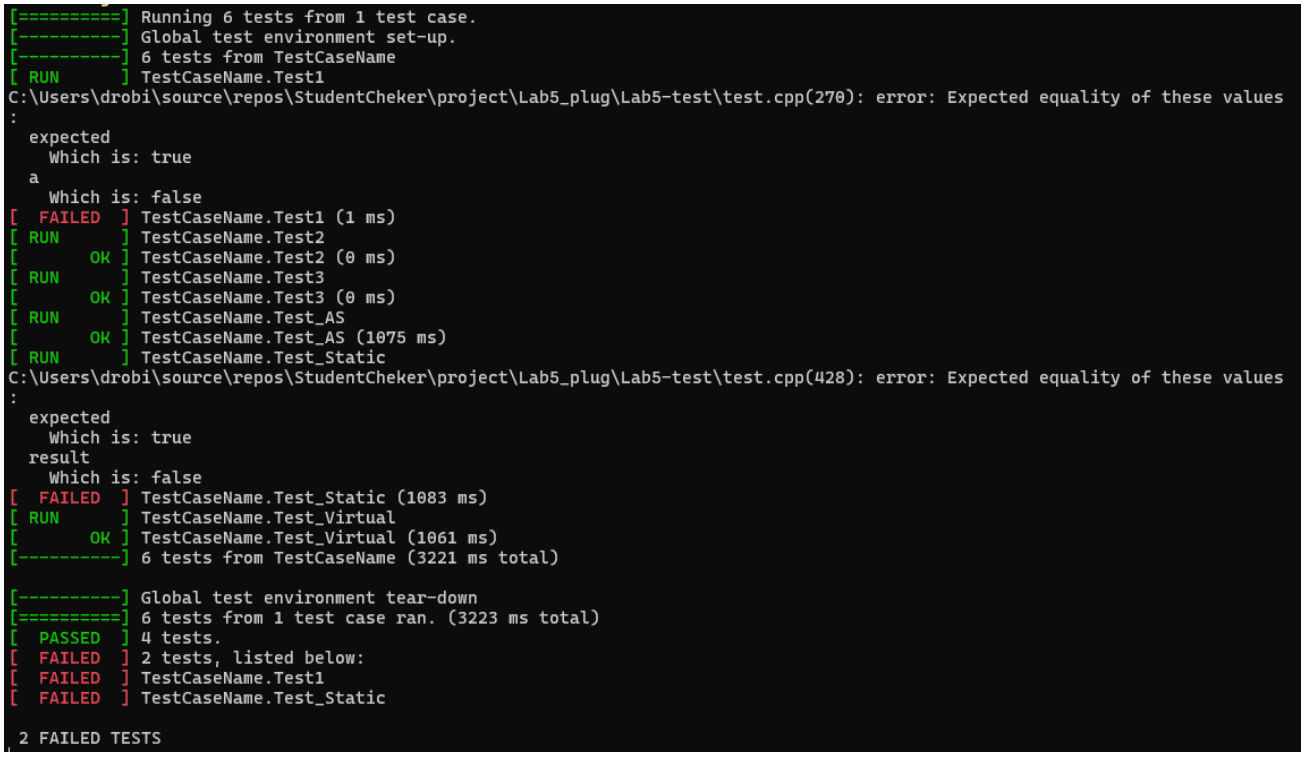

Рисунок 4.2 – тестирование работы №5 студента S2.

Как видно на рисунке, у студента не прошли 2 теста: 1) на минимальное число классов, 2) на статические методы и поля. Проверим у убедимся в том, что число классов студента действительно 5, а не минимум 6 (в работе сказано, главный класс и 5 производных). Также статические элементы присутствуют только лишь в 1-м классе, а в задании сказано минимум в 3-х.

Запустим проект 2 лабораторной работы студента S3 варианта 1. Результаты тестов представлены на рисунке 4.3:

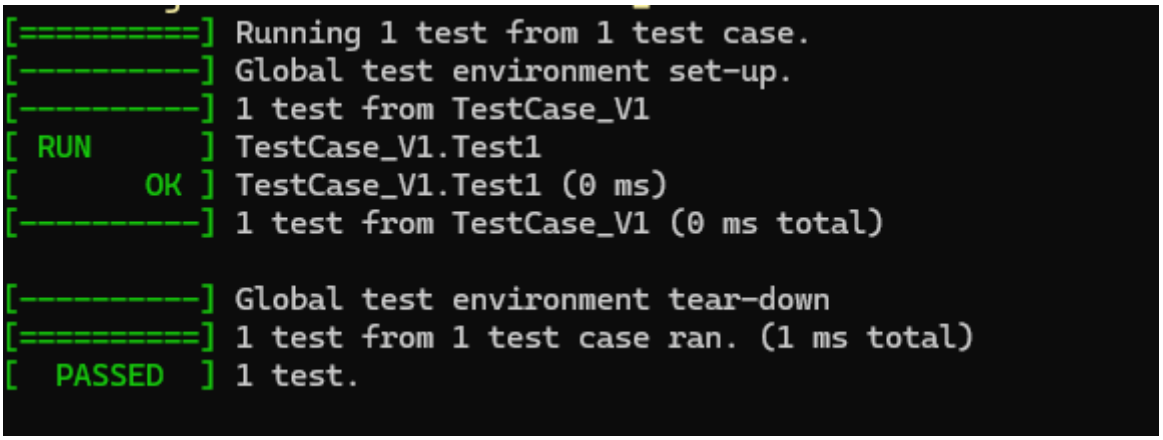

Рисунок 4.3 – тестирование работы №2 студента S3.

Как видно на рисунке, тестирование прошло успешно.

В таблице 4.1 можно увидеть то, как программа тестирования реагирует на уникальные ситуации, которые могут возникнуть при работе с ней:

Таблица 4.1

Тестирование программы на уникальные ситуации

Ситуация	Результат
Запуск приложения без указания файлов.	Приложение отработало корректно, так как в ней заложены заглушки при условии, что функционал еще не загружен.
Запуск приложения с неверным вариантом.	В работах, где проверяется логика работы определенной функции, если указаны неверные сигнатуры или имена, то программа сообщит о том, что компиляция не прошла (что равно сильно провалу тестирования). В работах, где проводится синтаксический анализ, тесты, приведенных специально для вариантов, не будут успешно пройдены.
Загрузка файлов с неверными именами.	Программа сообщит о том, что компиляция не была пройдена.
Последовательная загрузка различных файлов проектов разных студентов.	Файлы заменяются друг другом, а в 5-й работе перед копированием дополнительно удаляются, поэтому запускаться будет для тестирования последний загруженный проект, и это не будет вызывать ошибку.

ЗАКЛЮЧЕНИЕ

В ходе данной технологической практики мною были созданы 6 проектов, каждый из которых тестировал соответствующие лабораторные работы по курсу «Объектно-ориентированное программирование» на языке C++. Для достижения данной цели мною была изучена библиотека Google Test, предназначенная для модульного тестирования проектов. Также была изучена и использована библиотека Clang-c, которая предназначена для статического (в том числе синтаксического) анализа кода.

На основе 6 проектов было создано оконное приложение на языке C# с использованием технологии WPF. Данное приложение позволяет интерактивно выбрать номер лабораторной работы, вариант, загрузить нужные файлы для проверки и автоматически собрать нужное решение для его дальнейшего запуска с учётом варианта.

По результатам тестирования программа показала, что способно верно выполнять заложенную в ней логику. Она может быстро и без проблем находить ошибки в проектах студентов, на что преподавателю потребовалось гораздо больше времени.

Для данной программы возможны дальнейшие пути совершенствования, например, в неё также можно добавить сохранения результатов выполнения работы студента или, например, добавить возможность загрузки сразу нескольких работ разных студентов.

По итогу, разработанная мной программа должна упростить проверку работ преподавателем по курсу «Объектно-ориентированной программирование», избавить его от рутинного просмотра и проверки кода каждого студента.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Libclang. Интерфейс C для Clang — URL: https://clang.llvm.org/doxygen/group__CINDEX.html (дата обращения: 07.07.2024).

2. Использование Google Test для Visual Studio 2022 — URL: https://learn.microsoft.com/ru-ru/visualstudio/test/how-to-use-google-test-for-cpp?view=vs-2022ac.ru/doc/kotelnikovchebotaev2004b.pdf (дата обращения: 07.07.2024).

3. Google C++ Testing Framework — URL: https://ru.wikipedia.org/wiki/Google_C%2B%2B_Testing_Framework (дата обращения: 07.07.2024).

4. Библиотека Boost — URL: https://www.boost.org/ (дата обращения: 07.07.2024).

Приложение 1

Исходный код разработанного приложения

test.cpp (Lab1):

```
#include "pch.h"
#include "..\Lab1_plug\Source.h"

// minValueInt
TEST(TestCase_minValueInt, Test1) {
	int x = 10;
	int y = 20;
	int expected;

	expected = x;
	EXPECT_EQ(minValue((int)x, (int)y), expected);
}

TEST(TestCase_minValueInt, Test2) {
	int x = 20;
	int y = 10;
	int expected;

	expected = y;
	EXPECT_EQ(minValue((int)x, (int)y), expected);
}

TEST(TestCase_minValueInt, Test3) {
	int x = -10;
	int y = 10;
	int expected;

	expected = x;
	EXPECT_EQ(minValue((int)x, (int)y), expected);
}

// minValueFloat
TEST(TestCase_minValueFloat, Test1) {
	float x = 10.3;
	float y = 20.565;
	float expected;

	expected = x;
	EXPECT_EQ(minValue((float)x, (float)y), expected);
}

TEST(TestCase_minValueFloat, Test2) {
	float x = 20.1245;
	float y = 10;
	float expected;

	expected = y;
	EXPECT_EQ(minValue((float)x, (float)y), expected);
}

TEST(TestCase_minValueFloat, Test3) {
	float x = 10.2345;
	float y = 10.23;
	float expected;

	expected = y;
	EXPECT_EQ(minValue((float)x, (float)y), expected);
}

// belongsToInterval
TEST(TestCase_belongsToInterval, Test1) {
	int x = 10;
	int top = 20;
	int bottom = 0;
	bool expected;
```

```
    expected = true;
    EXPECT_EQ(belongsToInterval(x, top, bottom), expected);
}

TEST(TestCase_belongsToInterval, Test2) {
    int x = 10;
    int top = 0;
    int bottom = 20;
    bool expected;

    expected = false;
    EXPECT_EQ(belongsToInterval(x, top, bottom), expected);
}

TEST(TestCase_belongsToInterval, Test3) {
    int x = 10;
    int top = 10;
    int bottom = 10;
    bool expected;

    expected = true;
    EXPECT_EQ(belongsToInterval(x, top, bottom), expected);
}

TEST(TestCase_belongsToInterval, Test4) {
    int x = 10;
    int top = 10;
    int bottom = 0;
    bool expected;

    expected = true;
    EXPECT_EQ(belongsToInterval(x, top, bottom), expected);
}

TEST(TestCase_belongsToInterval, Test5) {
    int x = 10;
    int top = 20;
    int bottom = 10;
    bool expected;

    expected = true;
    EXPECT_EQ(belongsToInterval(x, top, bottom), expected);
}

// order
TEST(TestCase_order, Test1) {
    long long x = 9123495214543563453;
    short expected;

    expected = 19;
    EXPECT_EQ(order(x), expected);
}

TEST(TestCase_order, Test2) {
    long long x = 0;
    short expected;

    expected = 1;
    EXPECT_EQ(order(x), expected);
}

TEST(TestCase_order, Test3) {
    long long x = -9123495214543563453;
    short expected;

    expected = 19;
    EXPECT_EQ(order(x), expected);
}

// getLetterCode
TEST(TestCase_getLetterCode, Test1) {
    char c;
    int expected;

    for (int i = 0; i < 100; i++)
    {
```

```
            c = i;
            expected = (int)c;
            EXPECT_EQ(getLetterCode(c), expected);
            i++;
      }
}

int main(int argc, char** argv) {
      testing::InitGoogleTest(&argc, argv);
      RUN_ALL_TESTS();
      std::cin.get();

      return 0;
}
```

test.cpp (Lab2):

```
#include "pch.h"
#include "..\Lab2_plug\Source.h"


// V0
TEST(TestCase_V0, Test1) {
      int** A = new int* [3];
      for (int i = 0; i < 3; i++)
            A[i] = new int[3];

      A[0][0] = 1;
      A[0][1] = 2;
      A[0][2] = 3;
      A[1][0] = 4;
      A[1][1] = 5;
      A[1][2] = 6;
      A[2][0] = 7;
      A[2][1] = 8;
      A[2][02] = 9;

      int** B = new int* [3];
      for (int i = 0; i < 3; i++)
            B[i] = new int[3];

      B[0][0] = 3;
      B[0][0] = 1;
      B[0][0] = 2;
      B[0][0] = 4;
      B[0][0] = 5;
      B[0][0] = 6;

      int** result = multipMatrix(A, 3, 3, B, 3, 2);

      int expend[3][2] =
      {
            {22, 27},
            {52, 60},
            {82, 93}
      };


      for (int i = 0; i < 3; i++)
      {
            for (int y = 0; y < 2; y++)
            {
                  EXPECT_EQ(expend[i][y], result[i][y]);
            }
      }
}

// V1
TEST(TestCase_V1, Test1) {
      int** A = new int* [3];
      for (int i = 0; i < 3; i++)
            A[i] = new int[3];

      A[0][0] = 1;
      A[0][1] = 2;
      A[0][2] = 3;
      A[1][0] = 4;
      A[1][1] = 5;
```

```
	A[1][2] = 6;
	A[2][0] = 7;
	A[2][1] = 8;
	A[2][02] = 9;

	int* B = new int [3];

	B[0] = 3;
	B[1] = 2;
	B[2] = 5;

	int* result = multipVector(A, B, 3, 3);

	int expend[3] = { 22, 52, 82 };


	for (int i = 0; i < 3; i++)
	{
		EXPECT_EQ(expend[i], result[i]);
	}
}

// V2
TEST(TestCase_V2, Test1) {
	int** A = new int* [3];
	for (int i = 0; i < 3; i++)
		A[i] = new int[3];

	A[0][0] = 1;
	A[0][1] = 2;
	A[0][2] = 3;
	A[1][0] = 4;
	A[1][1] = 5;
	A[1][2] = 6;
	A[2][0] = 7;
	A[2][1] = 8;
	A[2][02] = 9;

	int** B = new int* [3];
	for (int i = 0; i < 3; i++)
		B[i] = new int[3];

	B[0][0] = 1;
	B[0][1] = 2;
	B[0][2] = 3;
	B[1][0] = 4;
	B[1][1] = 5;
	B[1][2] = 6;
	B[2][0] = 7;
	B[2][1] = 8;
	B[2][02] = 9;

	int** result = addMatrix(A, B, 3, 3);

	int expend[3][3] =
	{
		{2, 4, 6},
		{8, 10, 12},
		{14, 16, 18}
	};


	for (int i = 0; i < 3; i++)
	{
		for (int y = 0; y < 3; y++)
		{
			EXPECT_EQ(expend[i][y], result[i][y]);
		}
	}
}

// V3
TEST(TestCase_V3, Test1) {
	int** A = new int* [2];
	for (int i = 0; i < 2; i++)
		A[i] = new int[2];
```

```
	A[0][0] = 1;
	A[0][1] = 2;
	A[1][0] = 3;
	A[1][1] = 4;

	int** B = new int* [2];
	for (int i = 0; i < 2; i++)
		B[i] = new int[2];

	B[0][0] = 1;
	B[0][1] = 2;
	B[1][0] = 3;
	B[1][1] = 4;

	int** result = elemMultipMatrix(A, B, 2, 2);

	int expend[3][3] =
	{
		{1, 4},
		{9, 16}
	};


	for (int i = 0; i < 2; i++)
	{
		for (int y = 0; y < 2; y++)
		{
			EXPECT_EQ(expend[i][y], result[i][y]);
		}
	}
}

// V4
TEST(TestCase_V4, Test1) {
	int** A = new int* [3];
	for (int i = 0; i < 3; i++)
		A[i] = new int[3];

	A[0][0] = 1;
	A[0][1] = 2;
	A[0][2] = 3;
	A[1][0] = 4;
	A[1][1] = 5;
	A[1][2] = 6;
	A[2][0] = 7;
	A[2][1] = 8;
	A[2][02] = 9;


	int result = traceMatrix(A, 3);

	int expend = 15;


	EXPECT_EQ(expend, result);
}

// V5
TEST(TestCase_V5, Test1) {
	int** A = new int* [3];
	for (int i = 0; i < 3; i++)
		A[i] = new int[3];

	A[0][0] = 1;
	A[0][1] = 2;
	A[0][2] = 3;
	A[1][0] = 4;
	A[1][1] = 5;
	A[1][2] = 6;
	A[2][0] = 7;
	A[2][1] = 8;
	A[2][02] = 9;

	int** B = new int* [3];
	for (int i = 0; i < 3; i++)
```

```
		B[i] = new int[3];

	B[0][0] = 1;
	B[0][1] = 2;
	B[0][2] = 3;
	B[1][0] = 4;
	B[1][1] = 5;
	B[1][2] = 6;
	B[2][0] = 7;
	B[2][1] = 8;
	B[2][02] = 9;

	int** result = subMatrix(A, B, 3, 3);

	int expend[3][3] =
	{
		{0, 0, 0},
		{0, 0, 0},
		{0, 0, 0}
	};


	for (int i = 0; i < 3; i++)
	{
		for (int y = 0; y < 3; y++)
		{
			EXPECT_EQ(expend[i][y], result[i][y]);
		}
	}
}

// V6
TEST(TestCase_V6, Test1) {
	int* A = new int[3];

	A[0] = 1;
	A[1] = 2;
	A[2] = 3;

	int* B = new int[3];

	B[0] = 1;
	B[1] = 4;
	B[2] = 7;

	int result = multipRowByColumn(A, B, 3);

	int expend = 30;

	EXPECT_EQ(expend, result);
}

// V7
TEST(TestCase_V7, Test1) {
	int** A = new int* [3];
	for (int i = 0; i < 3; i++)
		A[i] = new int[3];

	A[0][0] = 1;
	A[0][1] = 2;
	A[0][2] = 3;
	A[1][0] = 4;
	A[1][1] = 5;
	A[1][2] = 6;
	A[2][0] = 7;
	A[2][1] = 8;
	A[2][02] = 9;


	int** result = multipMatrixNumber(A, 3, 3, 2);

	int expend[3][3] =
	{
		{2, 4, 6},
		{8, 10, 12},
		{14, 16, 18}
```

```
	};


	for (int i = 0; i < 3; i++)
	{
		for (int y = 0; y < 3; y++)
		{
			EXPECT_EQ(expend[i][y], result[i][y]);
		}
	}
}


int main(int argc, char** argv) {
	testing::InitGoogleTest(&argc, argv);
	RUN_ALL_TESTS();
	std::cin.get();

	return 0;
}
```

test.cpp (Lab3):

```
#include "pch.h"
#include "..\Lab3_plug\Point.h"
#include "..\Lab3_plug\Point.cpp"
#include "..\Lab3_plug\Triangle.h"
#include "..\Lab3_plug\Triangle.cpp"
#include "..\Lab3_plug\Rectangle.h"
#include "..\Lab3_plug\Rectangle.cpp"
#include "..\Lab3_plug\Rhomb.h"
#include "..\Lab3_plug\Rhomb.cpp"
#include "..\Lab3_plug\Square.h"
#include "..\Lab3_plug\Square.cpp"
#include "..\Lab3_plug\Trapeze.h"
#include "..\Lab3_plug\Trapeze.cpp"


// L
TEST(TestCase_L, Test1) {
	Point points[3];
	int size = 3;

	for (int i = 0; i < 3; i++)
	{
		points[i].SetPoint(i, i);
	}


	bool result = inLine(points, size);
	bool expected = true;

	EXPECT_EQ(expected, result);
}

TEST(TestCase_L, Test2) {
	Point points[5];
	int size = 5;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(1, 0);
	points[2].SetPoint(2, 0);
	points[3].SetPoint(3, 0);
	points[4].SetPoint(1, 0);

	int result = getPointsInLine(points, size);
	int expected = 4;

	EXPECT_EQ(expected, result);
}

TEST(TestCase_L, Test3) {
	Point points[5];
	int size = 5;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(1, 0);
	points[2].SetPoint(2, 0);
```

```
    points[3].SetPoint(3, 0);
    points[4].SetPoint(1, 0);

    int result = countLines(points, size);
    int expected = 4;

    EXPECT_EQ(expected, result);
}

TEST(TestCase_L, Test4) {
    Point points[6];
    int size = 6;
    Point p11;
    Point p12;
    Point p21;
    Point p22;

    points[0].SetPoint(0, 0);
    points[1].SetPoint(1, 0);
    points[2].SetPoint(2, 0);
    points[3].SetPoint(3, 0);
    points[4].SetPoint(0, 1);
    points[5].SetPoint(3, 1);
    p11.SetPoint(0, 0);
    p12.SetPoint(0, 1);
    p21.SetPoint(3, 0);
    p22.SetPoint(3, 1);

    bool result = getParallelLines(points, size, p11, p12, p21, p22);
    bool expected = true;

    EXPECT_EQ(expected, result);
}


// P
TEST(TestCase_P, Test1) {
    Point points[5];
    int size = 5;
    int R = 1;

    points[0].SetPoint(0, 0);
    points[1].SetPoint(0, 2);
    points[2].SetPoint(0, 3);
    points[3].SetPoint(0, 4);
    points[4].SetPoint(0, 6);

    Point result = getMaxCirclePoint(points, size, R);
    Point expected;
    expected.SetPoint(0, 3);

    EXPECT_EQ(expected.GetX(), result.GetX());
    EXPECT_EQ(expected.GetY(), result.GetY());
}

TEST(TestCase_P, Test2) {
    Point points[5];
    int size = 5;

    points[0].SetPoint(0, 0);
    points[1].SetPoint(0, 1);
    points[2].SetPoint(0, 2);
    points[3].SetPoint(0, 3);
    points[4].SetPoint(0, 15);

    Point result = getFarestPoint(points, size);
    Point expected;
    expected.SetPoint(0, 15);

    EXPECT_EQ(expected.GetX(), result.GetX());
    EXPECT_EQ(expected.GetY(), result.GetY());
}

TEST(TestCase_P, Test3) {
    Point points[3];
    int size = 3;
```

```
	points[0].SetPoint(0, 0);
	points[1].SetPoint(0, 2);
	points[2].SetPoint(0, 3);

	Point result = getClosestPoint(points, size);
	Point expected;
	expected.SetPoint(0, 2);

	EXPECT_EQ(expected.GetX(), result.GetX());
	EXPECT_EQ(expected.GetY(), result.GetY());
}

TEST(TestCase_P, Test4) {
	Point points[5];
	int size = 5;
	int R = 2;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(0, 1);
	points[2].SetPoint(0, 2);
	points[3].SetPoint(0, 3);
	points[4].SetPoint(0, 5);

	Point result = getMinCirclePoint(points, size, R);
	Point expected;
	expected.SetPoint(0, 5);

	EXPECT_EQ(expected.GetX(), result.GetX());
	EXPECT_EQ(expected.GetY(), result.GetY());
}

// Triangle
TEST(TestCase_Triangle, Test1) {
	Triangle triangles[3];
	int size = 3;

	triangles[0].SetTriangle(Point(0, 0), Point(0, 1), Point(1, 0));
	triangles[0].SetTriangle(Point(10, 0), Point(12, 0), Point(11, 6));
	triangles[0].SetTriangle(Point(3, 0), Point(3, 1), Point(4, 0));

	int result = getRectTriangles(triangles, size);
	int expected = 2;

	EXPECT_EQ(expected, result);
}

TEST(TestCase_Triangle, Test2) {
	Triangle triangles[3];
	int size = 3;

	triangles[0].SetTriangle(Point(0, 0), Point(0, 1), Point(1, 0));
	triangles[0].SetTriangle(Point(10, 0), Point(12, 0), Point(11, 6));
	triangles[0].SetTriangle(Point(3, 0), Point(3, 1), Point(4, 0));

	int result = getIsoscelesTriangles(triangles, size);
	int expected = 3;

	EXPECT_EQ(expected, result);
}

TEST(TestCase_Triangle, Test3) {
	Point points[4];
	int size = 4;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(0, 5);
	points[2].SetPoint(9, 0);
	points[3].SetPoint(10, 0);

	Point* result = getMaxLengthTriangle(points, size);


	Point expected[3];
	expected[0].SetPoint(0, 0);
	expected[1].SetPoint(0, 5);
```

```
	expected[2].SetPoint(10, 0);

	EXPECT_EQ(expected[0].GetX(), result[0].GetX());
	EXPECT_EQ(expected[0].GetY(), result[0].GetY());

	EXPECT_EQ(expected[1].GetX(), result[1].GetX());
	EXPECT_EQ(expected[1].GetY(), result[1].GetY());

	EXPECT_EQ(expected[2].GetX(), result[2].GetX());
	EXPECT_EQ(expected[2].GetY(), result[2].GetY());
}

TEST(TestCase_Triangle, Test4) {
	Point points[4];
	int size = 4;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(0, 5);
	points[2].SetPoint(9, 0);
	points[3].SetPoint(100, 0);

	Point* result = getMinLengthTriangle(points, size);


	Point expected[3];
	expected[0].SetPoint(0, 0);
	expected[1].SetPoint(0, 5);
	expected[2].SetPoint(9, 0);

	EXPECT_EQ(expected[0].GetX(), result[0].GetX());
	EXPECT_EQ(expected[0].GetY(), result[0].GetY());

	EXPECT_EQ(expected[1].GetX(), result[1].GetX());
	EXPECT_EQ(expected[1].GetY(), result[1].GetY());

	EXPECT_EQ(expected[2].GetX(), result[2].GetX());
	EXPECT_EQ(expected[2].GetY(), result[2].GetY());
}

// Rect

TEST(TestCase_Rect, Test1) {
	Point points[9];
	int size = 9;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(0, 5);
	points[2].SetPoint(0, 10);
	points[3].SetPoint(5, 0);
	points[4].SetPoint(5, 5);
	points[5].SetPoint(5, 10);
	points[6].SetPoint(10, 0);
	points[7].SetPoint(10, 5);
	points[8].SetPoint(10, 10);

	Rectangle* result = getMaxRectangle(points, size);

	Rectangle expected[1];
	expected[0].SetRectangle(Point(0, 0), Point(0, 10), Point(10, 10), Point(0, 10));


	EXPECT_EQ(expected[0].GetP1().GetX(), result[0].GetP1().GetX());
	EXPECT_EQ(expected[0].GetP1().GetY(), result[0].GetP1().GetY());

	EXPECT_EQ(expected[0].GetP2().GetX(), result[0].GetP2().GetX());
	EXPECT_EQ(expected[0].GetP2().GetY(), result[0].GetP2().GetY());

	EXPECT_EQ(expected[0].GetP3().GetX(), result[0].GetP3().GetX());
	EXPECT_EQ(expected[0].GetP3().GetY(), result[0].GetP3().GetY());

	EXPECT_EQ(expected[0].GetP4().GetX(), result[0].GetP4().GetX());
	EXPECT_EQ(expected[0].GetP4().GetY(), result[0].GetP4().GetY());
}

TEST(TestCase_Rect, Test2) {
	Point points[4];
```

```
	int size = 4;
	Rectangle rect;
	Rectangle &rectRef = rect;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(0, 10);
	points[2].SetPoint(10, 0);
	points[3].SetPoint(10, 10);

	bool result = getRectangle(points, rectRef);

	bool expected = true;
	Rectangle rectExpected = Rectangle(Point(0, 0), Point(10, 0), Point(10, 10),
Point(0, 10));


	EXPECT_EQ(expected, result);

	EXPECT_EQ(rectExpected.GetP1().GetX(), rect.GetP1().GetX());
	EXPECT_EQ(rectExpected.GetP1().GetY(), rect.GetP1().GetY());

	EXPECT_EQ(rectExpected.GetP2().GetX(), rect.GetP2().GetX());
	EXPECT_EQ(rectExpected.GetP2().GetY(), rect.GetP2().GetY());

	EXPECT_EQ(rectExpected.GetP3().GetX(), rect.GetP3().GetX());
	EXPECT_EQ(rectExpected.GetP3().GetY(), rect.GetP3().GetY());

	EXPECT_EQ(rectExpected.GetP4().GetX(), rect.GetP4().GetX());
	EXPECT_EQ(rectExpected.GetP4().GetY(), rect.GetP4().GetY());
}

TEST(TestCase_Rect, Test3) {
	Rectangle rects[3];
	int size = 3;
	Rectangle r1;
	Rectangle r2;

	Rectangle& r1Ref = r1;
	Rectangle& r2Ref = r2;

	rects[0].SetRectangle(Point(0, 0), Point(3, 0), Point(3, 3), Point(3, 0));
	rects[1].SetRectangle(Point(10, 0), Point(13, 0), Point(13, 3), Point(13, 0));
	rects[2].SetRectangle(Point(100, 0), Point(103, 0), Point(103, 3), Point(103, 0));

	getClosestRectangles(rects,size, r1Ref, r2Ref);

	Rectangle expectedR1 = Rectangle(Point(0, 0), Point(3, 0), Point(3, 3), Point(3,
0));
	Rectangle expectedR2 = Rectangle(Point(10, 0), Point(13, 0), Point(13, 3), Point(13,
0));


	EXPECT_EQ(expectedR1.GetP1().GetX(), r1.GetP1().GetX());
	EXPECT_EQ(expectedR1.GetP1().GetY(), r1.GetP1().GetY());

	EXPECT_EQ(expectedR1.GetP2().GetX(), r1.GetP2().GetX());
	EXPECT_EQ(expectedR1.GetP2().GetY(), r1.GetP2().GetY());

	EXPECT_EQ(expectedR1.GetP3().GetX(), r1.GetP3().GetX());
	EXPECT_EQ(expectedR1.GetP3().GetY(), r1.GetP3().GetY());

	EXPECT_EQ(expectedR1.GetP4().GetX(), r1.GetP4().GetX());
	EXPECT_EQ(expectedR1.GetP4().GetY(), r1.GetP4().GetY());

	EXPECT_EQ(expectedR2.GetP1().GetX(), r2.GetP1().GetX());
	EXPECT_EQ(expectedR2.GetP1().GetY(), r2.GetP1().GetY());

	EXPECT_EQ(expectedR2.GetP2().GetX(), r2.GetP2().GetX());
	EXPECT_EQ(expectedR2.GetP2().GetY(), r2.GetP2().GetY());

	EXPECT_EQ(expectedR2.GetP3().GetX(), r2.GetP3().GetX());
	EXPECT_EQ(expectedR2.GetP3().GetY(), r2.GetP3().GetY());

	EXPECT_EQ(expectedR2.GetP4().GetX(), r2.GetP4().GetX());
	EXPECT_EQ(expectedR2.GetP4().GetY(), r2.GetP4().GetY());
}
```

```
TEST(TestCase_Rect, Test4) {
     Rectangle f1 = Rectangle(Point(0, 0), Point(10, 0), Point(10, 10), Point(10, 0));
     Rectangle f2 = Rectangle(Point(5, 5), Point(15, 5), Point(15, 15), Point(15, 5));

     Rectangle& f1Ref = f1;
     Rectangle& f2Ref = f2;

     bool result = checkRectangleCrossing(f1Ref, f2Ref);

     bool expected = true;

     EXPECT_EQ(expected, result);

}

// Rhomb

TEST(TestCase_Rhomb, Test1) {
     Point points[9];
     int size = 9;

     points[0].SetPoint(0, 0);
     points[1].SetPoint(0, 5);
     points[2].SetPoint(0, 10);
     points[3].SetPoint(5, 0);
     points[4].SetPoint(5, 5);
     points[5].SetPoint(5, 10);
     points[6].SetPoint(10, 0);
     points[7].SetPoint(10, 5);
     points[8].SetPoint(10, 10);

     Rhomb* result = getMaxRhomb(points, size);

     Rhomb expected[1];
     expected[0].SetRhomb(Point(0, 0), Point(0, 10), Point(10, 10), Point(0, 10));


     EXPECT_EQ(expected[0].GetP1().GetX(), result[0].GetP1().GetX());
     EXPECT_EQ(expected[0].GetP1().GetY(), result[0].GetP1().GetY());

     EXPECT_EQ(expected[0].GetP2().GetX(), result[0].GetP2().GetX());
     EXPECT_EQ(expected[0].GetP2().GetY(), result[0].GetP2().GetY());

     EXPECT_EQ(expected[0].GetP3().GetX(), result[0].GetP3().GetX());
     EXPECT_EQ(expected[0].GetP3().GetY(), result[0].GetP3().GetY());

     EXPECT_EQ(expected[0].GetP4().GetX(), result[0].GetP4().GetX());
     EXPECT_EQ(expected[0].GetP4().GetY(), result[0].GetP4().GetY());
}

TEST(TestCase_Rhomb, Test2) {
     Point points[4];
     int size = 4;
     Rhomb rect;
     Rhomb& rectRef = rect;

     points[0].SetPoint(0, 0);
     points[1].SetPoint(0, 10);
     points[2].SetPoint(10, 0);
     points[3].SetPoint(10, 10);

     bool result = getRhomb(points, rectRef);

     bool expected = true;
     Rhomb rectExpected = Rhomb(Point(0, 0), Point(10, 0), Point(10, 10), Point(0, 10));


     EXPECT_EQ(expected, result);

     EXPECT_EQ(rectExpected.GetP1().GetX(), rect.GetP1().GetX());
     EXPECT_EQ(rectExpected.GetP1().GetY(), rect.GetP1().GetY());

     EXPECT_EQ(rectExpected.GetP2().GetX(), rect.GetP2().GetX());
     EXPECT_EQ(rectExpected.GetP2().GetY(), rect.GetP2().GetY());
```

```
	EXPECT_EQ(rectExpected.GetP3().GetX(), rect.GetP3().GetX());
	EXPECT_EQ(rectExpected.GetP3().GetY(), rect.GetP3().GetY());

	EXPECT_EQ(rectExpected.GetP4().GetX(), rect.GetP4().GetX());
	EXPECT_EQ(rectExpected.GetP4().GetY(), rect.GetP4().GetY());
}

TEST(TestCase_Rhomb, Test3) {
	Rhomb rects[3];
	int size = 3;
	Rhomb r1;
	Rhomb r2;

	Rhomb& r1Ref = r1;
	Rhomb& r2Ref = r2;

	rects[0].SetRhomb(Point(0, 0), Point(3, 0), Point(3, 3), Point(3, 0));
	rects[1].SetRhomb(Point(10, 0), Point(13, 0), Point(13, 3), Point(13, 0));
	rects[2].SetRhomb(Point(100, 0), Point(103, 0), Point(103, 3), Point(103, 0));

	getClosestRhombs(rects, size, r1Ref, r2Ref);

	Rhomb expectedR1 = Rhomb(Point(0, 0), Point(3, 0), Point(3, 3), Point(3, 0));
	Rhomb expectedR2 = Rhomb(Point(10, 0), Point(13, 0), Point(13, 3), Point(13, 0));


	EXPECT_EQ(expectedR1.GetP1().GetX(), r1.GetP1().GetX());
	EXPECT_EQ(expectedR1.GetP1().GetY(), r1.GetP1().GetY());

	EXPECT_EQ(expectedR1.GetP2().GetX(), r1.GetP2().GetX());
	EXPECT_EQ(expectedR1.GetP2().GetY(), r1.GetP2().GetY());

	EXPECT_EQ(expectedR1.GetP3().GetX(), r1.GetP3().GetX());
	EXPECT_EQ(expectedR1.GetP3().GetY(), r1.GetP3().GetY());

	EXPECT_EQ(expectedR1.GetP4().GetX(), r1.GetP4().GetX());
	EXPECT_EQ(expectedR1.GetP4().GetY(), r1.GetP4().GetY());

	EXPECT_EQ(expectedR2.GetP1().GetX(), r2.GetP1().GetX());
	EXPECT_EQ(expectedR2.GetP1().GetY(), r2.GetP1().GetY());

	EXPECT_EQ(expectedR2.GetP2().GetX(), r2.GetP2().GetX());
	EXPECT_EQ(expectedR2.GetP2().GetY(), r2.GetP2().GetY());

	EXPECT_EQ(expectedR2.GetP3().GetX(), r2.GetP3().GetX());
	EXPECT_EQ(expectedR2.GetP3().GetY(), r2.GetP3().GetY());

	EXPECT_EQ(expectedR2.GetP4().GetX(), r2.GetP4().GetX());
	EXPECT_EQ(expectedR2.GetP4().GetY(), r2.GetP4().GetY());
}

TEST(TestCase_Rhomb, Test4) {
	Rhomb f1 = Rhomb(Point(0, 0), Point(10, 0), Point(10, 10), Point(10, 0));
	Rhomb f2 = Rhomb(Point(5, 5), Point(15, 5), Point(15, 15), Point(15, 5));

	Rhomb& f1Ref = f1;
	Rhomb& f2Ref = f2;

	bool result = checkRhombCrossing(f1Ref, f2Ref);

	bool expected = true;

	EXPECT_EQ(expected, result);

}

// Square

TEST(TestCase_Square, Test1) {
	Point points[9];
	int size = 9;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(0, 5);
	points[2].SetPoint(0, 10);
	points[3].SetPoint(5, 0);
```

```
	points[4].SetPoint(5, 5);
	points[5].SetPoint(5, 10);
	points[6].SetPoint(10, 0);
	points[7].SetPoint(10, 5);
	points[8].SetPoint(10, 10);

	Square* result = getMaxSquare(points, size);

	Square expected[1];
	expected[0].SetSquare(Point(0, 0), Point(0, 10), Point(10, 10), Point(0, 10));


	EXPECT_EQ(expected[0].GetP1().GetX(), result[0].GetP1().GetX());
	EXPECT_EQ(expected[0].GetP1().GetY(), result[0].GetP1().GetY());

	EXPECT_EQ(expected[0].GetP2().GetX(), result[0].GetP2().GetX());
	EXPECT_EQ(expected[0].GetP2().GetY(), result[0].GetP2().GetY());

	EXPECT_EQ(expected[0].GetP3().GetX(), result[0].GetP3().GetX());
	EXPECT_EQ(expected[0].GetP3().GetY(), result[0].GetP3().GetY());

	EXPECT_EQ(expected[0].GetP4().GetX(), result[0].GetP4().GetX());
	EXPECT_EQ(expected[0].GetP4().GetY(), result[0].GetP4().GetY());
}

TEST(TestCase_Square, Test2) {
	Point points[4];
	int size = 4;
	Square rect;
	Square& rectRef = rect;

	points[0].SetPoint(0, 0);
	points[1].SetPoint(0, 10);
	points[2].SetPoint(10, 0);
	points[3].SetPoint(10, 10);

	bool result = getSquare(points, rectRef);

	bool expected = true;
	Square rectExpected = Square(Point(0, 0), Point(10, 0), Point(10, 10), Point(0,
10));


	EXPECT_EQ(expected, result);

	EXPECT_EQ(rectExpected.GetP1().GetX(), rect.GetP1().GetX());
	EXPECT_EQ(rectExpected.GetP1().GetY(), rect.GetP1().GetY());

	EXPECT_EQ(rectExpected.GetP2().GetX(), rect.GetP2().GetX());
	EXPECT_EQ(rectExpected.GetP2().GetY(), rect.GetP2().GetY());

	EXPECT_EQ(rectExpected.GetP3().GetX(), rect.GetP3().GetX());
	EXPECT_EQ(rectExpected.GetP3().GetY(), rect.GetP3().GetY());

	EXPECT_EQ(rectExpected.GetP4().GetX(), rect.GetP4().GetX());
	EXPECT_EQ(rectExpected.GetP4().GetY(), rect.GetP4().GetY());
}

TEST(TestCase_Square, Test3) {
	Square rects[3];
	int size = 3;
	Square r1;
	Square r2;

	Square& r1Ref = r1;
	Square& r2Ref = r2;

	rects[0].SetSquare(Point(0, 0), Point(3, 0), Point(3, 3), Point(3, 0));
	rects[1].SetSquare(Point(10, 0), Point(13, 0), Point(13, 3), Point(13, 0));
	rects[2].SetSquare(Point(100, 0), Point(103, 0), Point(103, 3), Point(103, 0));

	getClosestSquares(rects, size, r1Ref, r2Ref);

	Square expectedR1 = Square(Point(0, 0), Point(3, 0), Point(3, 3), Point(3, 0));
	Square expectedR2 = Square(Point(10, 0), Point(13, 0), Point(13, 3), Point(13, 0));
```

```
    EXPECT_EQ(expectedR1.GetP1().GetX(), r1.GetP1().GetX());
    EXPECT_EQ(expectedR1.GetP1().GetY(), r1.GetP1().GetY());

    EXPECT_EQ(expectedR1.GetP2().GetX(), r1.GetP2().GetX());
    EXPECT_EQ(expectedR1.GetP2().GetY(), r1.GetP2().GetY());

    EXPECT_EQ(expectedR1.GetP3().GetX(), r1.GetP3().GetX());
    EXPECT_EQ(expectedR1.GetP3().GetY(), r1.GetP3().GetY());

    EXPECT_EQ(expectedR1.GetP4().GetX(), r1.GetP4().GetX());
    EXPECT_EQ(expectedR1.GetP4().GetY(), r1.GetP4().GetY());

    EXPECT_EQ(expectedR2.GetP1().GetX(), r2.GetP1().GetX());
    EXPECT_EQ(expectedR2.GetP1().GetY(), r2.GetP1().GetY());

    EXPECT_EQ(expectedR2.GetP2().GetX(), r2.GetP2().GetX());
    EXPECT_EQ(expectedR2.GetP2().GetY(), r2.GetP2().GetY());

    EXPECT_EQ(expectedR2.GetP3().GetX(), r2.GetP3().GetX());
    EXPECT_EQ(expectedR2.GetP3().GetY(), r2.GetP3().GetY());

    EXPECT_EQ(expectedR2.GetP4().GetX(), r2.GetP4().GetX());
    EXPECT_EQ(expectedR2.GetP4().GetY(), r2.GetP4().GetY());
}

TEST(TestCase_Square, Test4) {
    Square f1 = Square(Point(0, 0), Point(10, 0), Point(10, 10), Point(10, 0));
    Square f2 = Square(Point(5, 5), Point(15, 5), Point(15, 15), Point(15, 5));

    Square& f1Ref = f1;
    Square& f2Ref = f2;

    bool result = checkSquareCrossing(f1Ref, f2Ref);

    bool expected = true;

    EXPECT_EQ(expected, result);

}

// Trapeze

TEST(TestCase_Trapeze, Test1) {
    Point points[9];
    int size = 9;

    points[0].SetPoint(0, 0);
    points[1].SetPoint(0, 5);
    points[2].SetPoint(0, 10);
    points[3].SetPoint(5, 0);
    points[4].SetPoint(5, 5);
    points[5].SetPoint(5, 10);
    points[6].SetPoint(10, 0);
    points[7].SetPoint(10, 5);
    points[8].SetPoint(10, 10);

    Trapeze* result = getMaxTrapeze(points, size);

    Trapeze expected[1];
    expected[0].SetTrapeze(Point(0, 0), Point(0, 10), Point(10, 10), Point(0, 10));


    EXPECT_EQ(expected[0].GetP1().GetX(), result[0].GetP1().GetX());
    EXPECT_EQ(expected[0].GetP1().GetY(), result[0].GetP1().GetY());

    EXPECT_EQ(expected[0].GetP2().GetX(), result[0].GetP2().GetX());
    EXPECT_EQ(expected[0].GetP2().GetY(), result[0].GetP2().GetY());

    EXPECT_EQ(expected[0].GetP3().GetX(), result[0].GetP3().GetX());
    EXPECT_EQ(expected[0].GetP3().GetY(), result[0].GetP3().GetY());

    EXPECT_EQ(expected[0].GetP4().GetX(), result[0].GetP4().GetX());
    EXPECT_EQ(expected[0].GetP4().GetY(), result[0].GetP4().GetY());
}
```

```
TEST(TestCase_Trapeze, Test2) {
     Point points[4];
     int size = 4;
     Trapeze rect;
     Trapeze& rectRef = rect;

     points[0].SetPoint(0, 0);
     points[1].SetPoint(0, 10);
     points[2].SetPoint(10, 0);
     points[3].SetPoint(10, 10);

     bool result = getTrapeze(points, rectRef);

     bool expected = true;
     Trapeze rectExpected = Trapeze(Point(0, 0), Point(10, 0), Point(10, 10), Point(0,
10));


     EXPECT_EQ(expected, result);

     EXPECT_EQ(rectExpected.GetP1().GetX(), rect.GetP1().GetX());
     EXPECT_EQ(rectExpected.GetP1().GetY(), rect.GetP1().GetY());

     EXPECT_EQ(rectExpected.GetP2().GetX(), rect.GetP2().GetX());
     EXPECT_EQ(rectExpected.GetP2().GetY(), rect.GetP2().GetY());

     EXPECT_EQ(rectExpected.GetP3().GetX(), rect.GetP3().GetX());
     EXPECT_EQ(rectExpected.GetP3().GetY(), rect.GetP3().GetY());

     EXPECT_EQ(rectExpected.GetP4().GetX(), rect.GetP4().GetX());
     EXPECT_EQ(rectExpected.GetP4().GetY(), rect.GetP4().GetY());
}

TEST(TestCase_Trapeze, Test3) {
     Trapeze rects[3];
     int size = 3;
     Trapeze r1;
     Trapeze r2;

     Trapeze& r1Ref = r1;
     Trapeze& r2Ref = r2;

     rects[0].SetTrapeze(Point(0, 0), Point(3, 0), Point(3, 3), Point(3, 0));
     rects[1].SetTrapeze(Point(10, 0), Point(13, 0), Point(13, 3), Point(13, 0));
     rects[2].SetTrapeze(Point(100, 0), Point(103, 0), Point(103, 3), Point(103, 0));

     getClosestTrapezes(rects, size, r1Ref, r2Ref);

     Trapeze expectedR1 = Trapeze(Point(0, 0), Point(3, 0), Point(3, 3), Point(3, 0));
     Trapeze expectedR2 = Trapeze(Point(10, 0), Point(13, 0), Point(13, 3), Point(13,
0));


     EXPECT_EQ(expectedR1.GetP1().GetX(), r1.GetP1().GetX());
     EXPECT_EQ(expectedR1.GetP1().GetY(), r1.GetP1().GetY());

     EXPECT_EQ(expectedR1.GetP2().GetX(), r1.GetP2().GetX());
     EXPECT_EQ(expectedR1.GetP2().GetY(), r1.GetP2().GetY());

     EXPECT_EQ(expectedR1.GetP3().GetX(), r1.GetP3().GetX());
     EXPECT_EQ(expectedR1.GetP3().GetY(), r1.GetP3().GetY());

     EXPECT_EQ(expectedR1.GetP4().GetX(), r1.GetP4().GetX());
     EXPECT_EQ(expectedR1.GetP4().GetY(), r1.GetP4().GetY());

     EXPECT_EQ(expectedR2.GetP1().GetX(), r2.GetP1().GetX());
     EXPECT_EQ(expectedR2.GetP1().GetY(), r2.GetP1().GetY());

     EXPECT_EQ(expectedR2.GetP2().GetX(), r2.GetP2().GetX());
     EXPECT_EQ(expectedR2.GetP2().GetY(), r2.GetP2().GetY());

     EXPECT_EQ(expectedR2.GetP3().GetX(), r2.GetP3().GetX());
     EXPECT_EQ(expectedR2.GetP3().GetY(), r2.GetP3().GetY());

     EXPECT_EQ(expectedR2.GetP4().GetX(), r2.GetP4().GetX());
     EXPECT_EQ(expectedR2.GetP4().GetY(), r2.GetP4().GetY());
```

```
}

TEST(TestCase_Trapeze, Test4) {
     Trapeze f1 = Trapeze(Point(0, 0), Point(10, 0), Point(10, 10), Point(10, 0));
     Trapeze f2 = Trapeze(Point(5, 5), Point(15, 5), Point(15, 15), Point(15, 5));

     Trapeze& f1Ref = f1;
     Trapeze& f2Ref = f2;

     bool result = checkTrapezeCrossing(f1Ref, f2Ref);

     bool expected = true;

     EXPECT_EQ(expected, result);

}

int main(int argc, char** argv) {
     testing::InitGoogleTest(&argc, argv);
     RUN_ALL_TESTS();
     std::cin.get();

     return 0;
}
```

test.cpp (Lab4):

```
#include "pch.h"

#include <map>
#include <vector>
#include <fstream>

#include <clang-c/Index.h>


using namespace std;

map <string, string> classVariant = {
     {"V0", "Queue"},
     {"V1", "Set"},
     {"V2", "Bufer"},
     {"V3", "Heap"},
     {"V4", "Stack"}
};

string currentVariant;
char** gArgv;
int gArgc;

void printCursor(CXCursor cursor) {
    CXString displayName = clang_getCursorDisplayName(cursor);
    std::cout << clang_getCString(displayName) << "\n";
    clang_disposeString(displayName);
}

CXCursor AccessSpecifier[2];
int countAS = 0;

CXChildVisitResult searchAccessSpecifier(CXCursor cursor, CXCursor /* parent */,
CXClientData /*clientData*/)
{
    CXSourceLocation location = clang_getCursorLocation(cursor);
    if (clang_Location_isFromMainFile(location) == 0)
        return CXChildVisit_Continue;


    // Основная часть - начало

    if (clang_getCursorKind(cursor) == 39 && countAS < 2)
    {
        AccessSpecifier[countAS++] = cursor;
    }

    // Основная часть - конец

    clang_visitChildren(cursor, searchAccessSpecifier, nullptr);
    return CXChildVisit_Continue;
```

```
}


// 0 - not met
// 1 - 1 met
// 2 - 2 met
int checkerPaM = 0;

// условие, есть ли cursor элементы внутри public и private
bool privPaM = false;
bool publPaM = false;

CXChildVisitResult propertiesAndMethods(CXCursor cursor, CXCursor /* parent */,
CXClientData /*clientData*/)
{
    CXSourceLocation location = clang_getCursorLocation(cursor);
    if (clang_Location_isFromMainFile(location) == 0)
        return CXChildVisit_Continue;


    // Основная часть - начало

    if (countAS > 0 && clang_equalCursors(cursor, AccessSpecifier[0]))
    {
        checkerPaM = 1;

        clang_visitChildren(cursor, propertiesAndMethods, nullptr);
        return CXChildVisit_Continue;
    }

    if (countAS > 1 && clang_equalCursors(cursor, AccessSpecifier[1]))
    {
        checkerPaM = 2;

        clang_visitChildren(cursor, propertiesAndMethods, nullptr);
        return CXChildVisit_Continue;
    }

    if (checkerPaM == 1)
    {
        privPaM = true;
    }

    if (checkerPaM == 2)
    {
        publPaM = true;
    }


    // Основная часть - конец

    clang_visitChildren(cursor, propertiesAndMethods, nullptr);
    return CXChildVisit_Continue;
}

bool classCheck = false;

CXChildVisitResult searchClass(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
    CXSourceLocation location = clang_getCursorLocation(cursor);
    if (clang_Location_isFromMainFile(location) == 0)
        return CXChildVisit_Continue;


    // Основная часть - начало

    if (clang_isDeclaration(clang_getCursorKind(cursor)) &&
clang_getCString(clang_getCursorDisplayName(cursor)) == currentVariant)
    {
        classCheck = true;
    }


    // Основная часть - конец
```

```
    clang_visitChildren(cursor, searchClass, nullptr);
    return CXChildVisit_Continue;
}

bool opPlusCheck = false;
bool opMinCheck = false;
bool opPlusRavnoCheck = false;
bool opMinRavnoCheck = false;
bool opRavnoCheck = false;

CXChildVisitResult searchAO(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
    CXSourceLocation location = clang_getCursorLocation(cursor);
    if (clang_Location_isFromMainFile(location) == 0)
        return CXChildVisit_Continue;


    // Основная часть - начало

    string name = clang_getCString(clang_getCursorDisplayName(cursor));

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("operator+(") !=
std::string::npos)
    {
        opPlusCheck = true;
    }

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("operator-(") !=
std::string::npos)
    {
        opMinCheck = true;
    }

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("operator+=(") !=
std::string::npos)
    {
        opPlusRavnoCheck = true;
    }

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("operator-=(") !=
std::string::npos)
    {
        opMinRavnoCheck = true;
    }

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("operator=(") !=
std::string::npos)
    {
        opRavnoCheck = true;
    }


    // Основная часть - конец

    clang_visitChildren(cursor, searchAO, nullptr);
    return CXChildVisit_Continue;
}

bool constrChecker = false;
bool destrChecker = false;

CXChildVisitResult searchCD(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
    CXSourceLocation location = clang_getCursorLocation(cursor);
    if (clang_Location_isFromMainFile(location) == 0)
        return CXChildVisit_Continue;


    // Основная часть - начало

    string name = clang_getCString(clang_getCursorDisplayName(cursor));
    string nameConstr = currentVariant;
    nameConstr.append("(");
    string nameDestr = "~";
```

```
    nameDestr.append(currentVariant);
    nameDestr.append("(");

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find(nameConstr) !=
std::string::npos)
    {
        constrChecker = true;
    }

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find(nameDestr) !=
std::string::npos)
    {
        destrChecker = true;
    }


    // Основная часть - конец

    clang_visitChildren(cursor, searchCD, nullptr);
    return CXChildVisit_Continue;
}

bool funcPushChecker = false;
bool funcPopChecker = false;
bool funcSearchChecker = false;
bool funcCompChecker = false;

bool funcGetPrChecker = false;
bool funcPopSeveralChecker = false;
bool funcNumAvChecker = false;

CXChildVisitResult searchMfunc(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
    CXSourceLocation location = clang_getCursorLocation(cursor);
    if (clang_Location_isFromMainFile(location) == 0)
        return CXChildVisit_Continue;


    // Основная часть - начало

    string name = clang_getCString(clang_getCursorDisplayName(cursor));

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("push(") !=
std::string::npos)
    {
        funcPushChecker = true;
    }

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("pop(") !=
std::string::npos)
    {
        funcPopChecker = true;
    }

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("search(") !=
std::string::npos)
    {
        funcSearchChecker = true;
    }

    if (clang_isDeclaration(clang_getCursorKind(cursor)) && name.find("compareSize(") !=
std::string::npos)
    {
        funcCompChecker = true;
    }

    // для некоторых вариантов
    if (currentVariant == "Queue" && clang_isDeclaration(clang_getCursorKind(cursor)) &&
name.find("getAllWithPriority(") != std::string::npos)
    {
        funcGetPrChecker = true;
    }

    if (currentVariant == "Bufer" && clang_isDeclaration(clang_getCursorKind(cursor)) &&
name.find("popSeveral(") != std::string::npos)
```

```
    {
        funcPopSeveralChecker = true;
    }

    if (currentVariant == "Bufer" && clang_isDeclaration(clang_getCursorKind(cursor)) &&
name.find("numberAvailable(") != std::string::npos)
    {
        funcNumAvChecker = true;
    }


    // Основная часть - конец

    clang_visitChildren(cursor, searchMfunc, nullptr);
    return CXChildVisit_Continue;
}

// проверка класса
TEST(TestCaseName, Test_Class) {
    classCheck = false;

    CXIndex index = clang_createIndex(0, 1);

    // Добавление флага для компиляции c++, а не c
    char** new_argv = new char* [gArgc + 2];
    for (int i = 0; i < gArgc; i++)
    {
        new_argv[i] = gArgv[i];
    }
    new_argv[gArgc] = "-x";
    new_argv[gArgc + 1] = "c++";

    // Создание строки каталога исходного файла
    string source_filename = "../../Lab4_plug/";
    source_filename.append(currentVariant);
    source_filename.append(".h");

    CXTranslationUnit unit = clang_parseTranslationUnit(
        index, // CIdx
        source_filename.c_str(), // source_filename
        new_argv, // command_line_args
        gArgc + 2, // num_command_line_args
        0, // unsave_files
        0, // num_unsaved_files
        CXTranslationUnit_None // options
    );
    if (!unit) {
        std::cout << "Translation unit was not created\n";
    }
    else {

        CXCursor root = clang_getTranslationUnitCursor(unit);

        // Находим нужный класс
        clang_visitChildren(root, searchClass, nullptr);
    }
    clang_disposeTranslationUnit(unit);
    clang_disposeIndex(index);

    bool expected = true;

    EXPECT_EQ(classCheck, expected);
}

// проверка AS
TEST(TestCaseName, Test_AS) {
    bool privRes = false;
    bool publRes = false;
    countAS = 0;
    privPaM = false;
    publPaM = false;

    CXIndex index = clang_createIndex(0,1);

    // Добавление флага для компиляции c++, а не c
    char** new_argv = new char* [gArgc + 2];
```

```
    for (int i = 0; i < gArgc; i++)
    {
        new_argv[i] = gArgv[i];
    }
    new_argv[gArgc] = "-x";
    new_argv[gArgc + 1] = "c++";

    // Создание строки каталога исходного файла
    string source_filename = "../../Lab4_plug/";
    source_filename.append(currentVariant);
    source_filename.append(".h");

    CXTranslationUnit unit = clang_parseTranslationUnit(
        index, // CIdx
        source_filename.c_str(), // source_filename
        new_argv, // command_line_args
        gArgc + 2, // num_command_line_args
        0, // unsave_files
        0, // num_unsaved_files
        CXTranslationUnit_None // options
    );
    if (!unit) {
        std::cout << "Translation unit was not created\n";
    }
    else {
        // content source file
        ifstream file(source_filename);
        istreambuf_iterator<char> begin(file), end;
        vector<char> v(begin, end);
        string contentOfTheFile = &v[0];

        if (contentOfTheFile.find("private:"))
        {
            privRes = true;
        }
        if (contentOfTheFile.find("private:"))
        {
            publRes = true;
        }


        CXCursor root = clang_getTranslationUnitCursor(unit);

        // Находим спецификаторы
        clang_visitChildren(root, searchAccessSpecifier, nullptr);

        // Определяем, существуют ли свойства и методы под этими спецификаторами
        clang_visitChildren(root, propertiesAndMethods, nullptr);
    }
    clang_disposeTranslationUnit(unit);
    clang_disposeIndex(index);


    bool privExpected = true;
    bool publExpected = true;

    // private:
     EXPECT_EQ(privExpected, privRes);
    // public:
    EXPECT_EQ(publExpected, publRes);

    // проверяет, есть ли в этих модификаторах доступа элементы
    EXPECT_EQ(privPaM, privExpected);
    EXPECT_EQ(publPaM, publExpected);
}

// проверка реализации арифметических операций
TEST(TestCaseName, Test_Arithmetic_operations) {
    opPlusCheck = false;
    opMinCheck = false;
    opPlusRavnoCheck = false;
    opMinRavnoCheck = false;
    opRavnoCheck = false;

    CXIndex index = clang_createIndex(0, 1);
```

```
    // Добавление флага для компиляции c++, а не c
    char** new_argv = new char* [gArgc + 2];
    for (int i = 0; i < gArgc; i++)
    {
        new_argv[i] = gArgv[i];
    }
    new_argv[gArgc] = "-x";
    new_argv[gArgc + 1] = "c++";

    // Создание строки каталога исходного файла
    string source_filename = "../../Lab4_plug/";
    source_filename.append(currentVariant);
    source_filename.append(".h");

    CXTranslationUnit unit = clang_parseTranslationUnit(
        index, // CIdx
        source_filename.c_str(), // source_filename
        new_argv, // command_line_args
        gArgc + 2, // num_command_line_args
        0, // unsave_files
        0, // num_unsaved_files
        CXTranslationUnit_None // options
    );
    if (!unit) {
        std::cout << "Translation unit was not created\n";
    }
    else {

        CXCursor root = clang_getTranslationUnitCursor(unit);

        // Находим нужный класс
        clang_visitChildren(root, searchAO, nullptr);
    }
    clang_disposeTranslationUnit(unit);
    clang_disposeIndex(index);


    bool expected = true;

    EXPECT_EQ(opPlusCheck, expected);
    EXPECT_EQ(opMinCheck, expected);
    EXPECT_EQ(opPlusRavnoCheck, expected);
    EXPECT_EQ(opMinRavnoCheck, expected);
    EXPECT_EQ(opRavnoCheck, expected);
}

// проверка реализации конструкторов и деструкторов
TEST(TestCaseName, Test_CD) {
    constrChecker = false;
    destrChecker = false;

    CXIndex index = clang_createIndex(0, 1);

    // Добавление флага для компиляции c++, а не c
    char** new_argv = new char* [gArgc + 2];
    for (int i = 0; i < gArgc; i++)
    {
        new_argv[i] = gArgv[i];
    }
    new_argv[gArgc] = "-x";
    new_argv[gArgc + 1] = "c++";

    // Создание строки каталога исходного файла
    string source_filename = "../../Lab4_plug/";
    source_filename.append(currentVariant);
    source_filename.append(".h");

    CXTranslationUnit unit = clang_parseTranslationUnit(
        index, // CIdx
        source_filename.c_str(), // source_filename
        new_argv, // command_line_args
        gArgc + 2, // num_command_line_args
        0, // unsave_files
        0, // num_unsaved_files
        CXTranslationUnit_None // options
    );
```

```
    if (!unit) {
        std::cout << "Translation unit was not created\n";
    }
    else {

        CXCursor root = clang_getTranslationUnitCursor(unit);

        // Находим нужный класс
        clang_visitChildren(root, searchCD, nullptr);
    }
    clang_disposeTranslationUnit(unit);
    clang_disposeIndex(index);


    bool expected = true;

    EXPECT_EQ(constrChecker, expected);
    EXPECT_EQ(destrChecker, expected);
}

// проверка основных функций
TEST(TestCaseName, Test_Mfunc) {
    funcPushChecker = false;
    funcPopChecker = false;
    funcSearchChecker = false;
    funcCompChecker = false;

    funcGetPrChecker = false;
    funcPopSeveralChecker = false;
    funcNumAvChecker = false;

    CXIndex index = clang_createIndex(0, 1);

    // Добавление флага для компиляции c++, а не c
    char** new_argv = new char* [gArgc + 2];
    for (int i = 0; i < gArgc; i++)
    {
        new_argv[i] = gArgv[i];
    }
    new_argv[gArgc] = "-x";
    new_argv[gArgc + 1] = "c++";

    // Создание строки каталога исходного файла
    string source_filename = "../../Lab4_plug/";
    source_filename.append(currentVariant);
    source_filename.append(".h");

    CXTranslationUnit unit = clang_parseTranslationUnit(
        index, // CIdx
        source_filename.c_str(), // source_filename
        new_argv, // command_line_args
        gArgc + 2, // num_command_line_args
        0, // unsave_files
        0, // num_unsaved_files
        CXTranslationUnit_None // options
    );
    if (!unit) {
        std::cout << "Translation unit was not created\n";
    }
    else {

        CXCursor root = clang_getTranslationUnitCursor(unit);

        // Находим нужный класс
        clang_visitChildren(root, searchMfunc, nullptr);
    }
    clang_disposeTranslationUnit(unit);
    clang_disposeIndex(index);


    bool expected = true;

    EXPECT_EQ(funcPushChecker, expected);
    EXPECT_EQ(funcPopChecker, expected);
    EXPECT_EQ(funcSearchChecker, expected);
    EXPECT_EQ(funcCompChecker, expected);
```

```
    if (currentVariant == "Queue")
    {
        EXPECT_EQ(funcGetPrChecker, expected);
    }

    if (currentVariant == "Bufer")
    {
        EXPECT_EQ(funcPopSeveralChecker, expected);
        EXPECT_EQ(funcNumAvChecker, expected);
    }
}

int main(int argc, char** argv) {
     testing::InitGoogleTest(&argc, argv);

     currentVariant = classVariant[argv[1]];
    gArgv = argv;
    gArgc = argc;


     RUN_ALL_TESTS();
     std::cin.get();

     return 0;
}
```

test.cpp (Lab5):

```
#define BOOST_FILESYSTEM_VERSION 3
#define BOOST_FILESYSTEM_NO_DEPRECATED
#include "pch.h"

#include <fstream>
#include <iostream>
#include <filesystem>
#include <boost/filesystem.hpp>

#include <clang-c\Index.h>
#include <map>


using namespace std;
namespace fs = boost::filesystem;

struct TreeClass
{
     CXCursor cursorClass;
     string name;
     int numberParent;
     TreeClass** parents;
};

int maxLevel = 0;



void addAllClassInArray();
void addClassMap();
int searchLevelA(int level, TreeClass* curItem);



map <string, string> classVariant = {
     {"V0", "Person"},
     {"V1", "Animal"},
     {"V2", "Vehicle"},
     {"V3", "Instrument"},
     {"V4", "Figure"},
     {"V5", "ComputeUnit"},
     {"V6", "FlyingObject"},
     {"V7", "SwimmingObject"}
};

string* className;
int coutnClass = 0;

map <string, TreeClass*> allClassMap;
```

```
string currentVariant;
char** gArgv;
int gArgc;

void printCursor(CXCursor cursor) {
     CXString displayName = clang_getCursorDisplayName(cursor);
     std::cout << clang_getCString(displayName) << "\n";
     clang_disposeString(displayName);
}

int numberParent = 0;
CXCursor cursorClass;

CXChildVisitResult visitClass(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
     CXSourceLocation location = clang_getCursorLocation(cursor);
     if (clang_Location_isFromMainFile(location) == 0)
            return CXChildVisit_Continue;


     //                -

     if (clang_getCursorKind(cursor) == 4)
     {
            cursorClass = cursor;
     }

     if (clang_getCursorKind(cursor) == 44)
     {
            numberParent++;
     }

     //                -

     clang_visitChildren(cursor, visitClass, nullptr);
     return CXChildVisit_Continue;
}

//
map <string, string*> mapAppend;
//    -
map <string, int> mapAppendSize;
string curClassname;
int chechParentsCounter = 0;

CXChildVisitResult chechParents(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
     CXSourceLocation location = clang_getCursorLocation(cursor);
     if (clang_Location_isFromMainFile(location) == 0)
            return CXChildVisit_Continue;


     //                -


     if (clang_getCursorKind(cursor) == 44)
     {
            mapAppend[curClassname][chechParentsCounter++] =
(string)clang_getCString(clang_getCursorDisplayName(cursor));
     }

     //                -

     clang_visitChildren(cursor, chechParents, nullptr);
     return CXChildVisit_Continue;
}

CXCursor AccessSpecifier[2];
int countAS = 0;

CXChildVisitResult searchAccessSpecifier(CXCursor cursor, CXCursor /* parent */,
CXClientData /*clientData*/)
{
```

```
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	//		-

	if (clang_getCursorKind(cursor) == 39 && countAS < 2)
	{
		AccessSpecifier[countAS++] = cursor;
	}

	//		-

	clang_visitChildren(cursor, searchAccessSpecifier, nullptr);
	return CXChildVisit_Continue;
}

// 0 - not met
// 1 - 1 met
// 2 - 2 met
int checkerPaM = 0;

//	,	cursor		public	private
bool privPaM = false;
bool publPaM = false;

CXChildVisitResult propertiesAndMethods(CXCursor cursor, CXCursor /* parent */,
CXClientData /*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	//		-

	if (countAS > 0 && clang_equalCursors(cursor, AccessSpecifier[0]))
	{
		checkerPaM = 1;

		clang_visitChildren(cursor, propertiesAndMethods, nullptr);
		return CXChildVisit_Continue;
	}

	if (countAS > 1 && clang_equalCursors(cursor, AccessSpecifier[1]))
	{
		checkerPaM = 2;

		clang_visitChildren(cursor, propertiesAndMethods, nullptr);
		return CXChildVisit_Continue;
	}

	if (checkerPaM == 1)
	{
		privPaM = true;
	}

	if (checkerPaM == 2)
	{
		publPaM = true;
	}


	//		-

	clang_visitChildren(cursor, propertiesAndMethods, nullptr);
	return CXChildVisit_Continue;
}

bool condStat = false;

CXChildVisitResult findStatic2(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
```

```
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	//		-

	if (clang_getCursorKind(cursor) == 9)
	{
		condStat = true;
	}


	//		-

	return CXChildVisit_Continue;
}

CXChildVisitResult findStatic1(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	//		-

	//		-

	clang_visitChildren(cursor, findStatic2, nullptr);
	return CXChildVisit_Continue;
}

bool condVirt = false;

CXChildVisitResult findVirtual2(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	//		-

	if (clang_CXXMethod_isVirtual(cursor) &&
clang_getCString(clang_getCursorDisplayName(cursor))[0] != '~')
	{
		condVirt = true;
	}


	//		-

	return CXChildVisit_Continue;
}

CXChildVisitResult findVirtual1(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	//		-

	//		-

	clang_visitChildren(cursor, findVirtual2, nullptr);
	return CXChildVisit_Continue;
}

//
TEST(TestCaseName, Test1) {
	bool a = coutnClass >= 6;
```

```
	bool expected = true;
	EXPECT_EQ(expected, a);
}

//
TEST(TestCaseName, Test2) {
	bool a = maxLevel >= 3;

	bool expected = true;
	EXPECT_EQ(expected, a);
}

//
TEST(TestCaseName, Test3) {
	bool a = false;

	for (int i = 0; i < coutnClass; i++)
	{
		if (className[i] == currentVariant)
			a = true;
	}

	bool expected = true;
	EXPECT_EQ(expected, a);
}

//		AS
TEST(TestCaseName, Test_AS) {
	for (int i = 0; i < coutnClass; i++)
	{
		bool privRes = false;
		bool publRes = false;
		countAS = 0;
		privPaM = false;
		publPaM = false;

		CXIndex index = clang_createIndex(0, 1);

		//                                    c++,      c
		char** new_argv = new char* [gArgc + 2];
		for (int i = 0; i < gArgc; i++)
		{
			new_argv[i] = gArgv[i];
		}
		new_argv[gArgc] = "-x";
		new_argv[gArgc + 1] = "c++";

		//
		string source_filename = "../../Lab5_plug/"; //
		source_filename.append(className[i]);
		source_filename.append(".h");

		CXTranslationUnit unit = clang_parseTranslationUnit(
			index, // CIdx
			source_filename.c_str(), // source_filename
			new_argv, // command_line_args
			gArgc + 2, // num_command_line_args
			0, // unsave_files
			0, // num_unsaved_files
			CXTranslationUnit_None // options
		);
		if (!unit) {
			std::cout << "Translation unit was not created\n";
		}
		else {
			// content source file
			ifstream file(source_filename);
			istreambuf_iterator<char> begin(file), end;
			vector<char> v(begin, end);
			string contentOfTheFile = &v[0];

			if (contentOfTheFile.find("private:"))
			{
				privRes = true;
			}
```

```
                if (contentOfTheFile.find("private:"))
                {
                        publRes = true;
                }


                CXCursor root = clang_getTranslationUnitCursor(unit);

                //
                clang_visitChildren(root, searchAccessSpecifier, nullptr);

                //           ,
                clang_visitChildren(root, propertiesAndMethods, nullptr);
        }
        clang_disposeTranslationUnit(unit);
        clang_disposeIndex(index);


        bool privExpected = true;
        bool publExpected = true;

        // private:
        EXPECT_EQ(privExpected, privRes);
        // public:
        EXPECT_EQ(publExpected, publRes);

        //          ,
        EXPECT_EQ(privPaM, privExpected);
        EXPECT_EQ(publPaM, publExpected);
    }
}

//
TEST(TestCaseName, Test_Static) {
    int countRes = 0;
    for (int i = 0; i < coutnClass; i++)
    {
        condStat = false;
        CXIndex index = clang_createIndex(0, 1);

        //                                  c++,      c
        char** new_argv = new char* [gArgc + 2];
        for (int i = 0; i < gArgc; i++)
        {
            new_argv[i] = gArgv[i];
        }
        new_argv[gArgc] = "-x";
        new_argv[gArgc + 1] = "c++";

        //
        string source_filename = "../../Lab5_plug/"; //
        source_filename.append(className[i]);
        source_filename.append(".h");

        CXTranslationUnit unit = clang_parseTranslationUnit(
            index, // CIdx
            source_filename.c_str(), // source_filename
            new_argv, // command_line_args
            gArgc + 2, // num_command_line_args
            0, // unsave_files
            0, // num_unsaved_files
            CXTranslationUnit_None // options
        );
        if (!unit) {
            std::cout << "Translation unit was not created\n";
        }
        else {

            CXCursor root = clang_getTranslationUnitCursor(unit);

            //
            clang_visitChildren(root, findStatic1, nullptr);

            if (condStat)
                countRes++;
        }
```

```
            clang_disposeTranslationUnit(unit);
            clang_disposeIndex(index);
      }

      bool result = countRes >= 3;

      bool expected = true;

      EXPECT_EQ(expected, result);
}

//
TEST(TestCaseName, Test_Virtual) {
      condVirt = false;
      for (int i = 0; i < coutnClass; i++)
      {
            condStat = false;
            CXIndex index = clang_createIndex(0, 1);

            //                                    c++,      c
            char** new_argv = new char* [gArgc + 2];
            for (int i = 0; i < gArgc; i++)
            {
                  new_argv[i] = gArgv[i];
            }
            new_argv[gArgc] = "-x";
            new_argv[gArgc + 1] = "c++";

            //
            string source_filename = "../../Lab5_plug/"; //
            source_filename.append(className[i]);
            source_filename.append(".h");

            CXTranslationUnit unit = clang_parseTranslationUnit(
                  index, // CIdx
                  source_filename.c_str(), // source_filename
                  new_argv, // command_line_args
                  gArgc + 2, // num_command_line_args
                  0, // unsave_files
                  0, // num_unsaved_files
                  CXTranslationUnit_None // options
            );
            if (!unit) {
                  std::cout << "Translation unit was not created\n";
            }
            else {

                  CXCursor root = clang_getTranslationUnitCursor(unit);

                  //
                  clang_visitChildren(root, findVirtual1, nullptr);

            }
            clang_disposeTranslationUnit(unit);
            clang_disposeIndex(index);

      }

      bool expected = true;
      EXPECT_EQ(expected, condVirt);
}

int main(int argc, char** argv) {
      testing::InitGoogleTest(&argc, argv);

      currentVariant = classVariant[argv[1]];
      gArgv = argv;
      gArgc = argc;

      //
      addAllClassInArray();
      //
      addClassMap();
      //
      for (int i = 0; i < coutnClass; i++)
      {
```

```
		searchLevelA(0, allClassMap[className[i]]);
	}

	RUN_ALL_TESTS();
	std::cin.get();

	return 0;
}

void addAllClassInArray()
{
	//
	std::string path("../../Lab5_plug/");
	std::string ext(".h");
	for (auto& p : fs::recursive_directory_iterator(path))
	{
		if (p.path().extension() == ext)
		{
			coutnClass++;
		}
	}

	className = new string[coutnClass];
	int i = 0;

	for (auto& p : fs::recursive_directory_iterator(path))
	{
		if (p.path().extension() == ext)
		{
			className[i++] = p.path().stem().string();
		}
	}
}

void addClassMap()
{

	for (int i = 0; i < coutnClass; i++)
	{
		numberParent = 0;

		CXIndex index = clang_createIndex(0, 1);

		//                                    c++,      c
		char** new_argv = new char* [gArgc + 2];
		for (int i = 0; i < gArgc; i++)
		{
			new_argv[i] = gArgv[i];
		}
		new_argv[gArgc] = "-x";
		new_argv[gArgc + 1] = "c++";

		//
		string source_filename = "../../Lab5_plug/";  //              !!!
		source_filename.append(className[i]);
		source_filename.append(".h");

		CXTranslationUnit unit = clang_parseTranslationUnit(
			index, // CIdx
			source_filename.c_str(), // source_filename
			new_argv, // command_line_args
			gArgc + 2, // num_command_line_args
			0, // unsave_files
			0, // num_unsaved_files
			CXTranslationUnit_None // options
		);
		if (!unit) {
			std::cout << "Translation unit was not created\n";
		}
		else {

			CXCursor root = clang_getTranslationUnitCursor(unit);

			//
			clang_visitChildren(root, visitClass, nullptr);
```

```
                mapAppend[className[i]] = new string[numberParent];
                curClassname = className[i];
                mapAppendSize[className[i]] = numberParent;
                chechParentsCounter = 0;

                //                              (        )
                clang_visitChildren(root, chechParents, nullptr);

                TreeClass* newItem = new TreeClass;
                newItem->cursorClass = cursorClass;
                newItem->name = className[i];
                newItem->numberParent = numberParent;

                allClassMap[className[i]] = newItem;

        }
        clang_disposeTranslationUnit(unit);
        clang_disposeIndex(index);
    }

    for (int i = 0; i < coutnClass; i++)
    {
        allClassMap[className[i]]->parents = new
TreeClass*[mapAppendSize[className[i]]];

        for (int z = 0; z < mapAppendSize[className[i]]; z++)
        {
            allClassMap[className[i]]->parents[z] =
allClassMap[mapAppend[className[i]][z]];
        }
    }
}

//
int searchLevelA(int level, TreeClass* curItem)
{
    level++;

    if (level > maxLevel)
        maxLevel = level;

    for (int i = 0; i < mapAppendSize[curItem->name]; i++)
    {
        searchLevelA(level, curItem->parents[i]);
    }
    return 0;
}
```

test.cpp (Lab6):

```
#include "pch.h"

#include <map>
#include <vector>
#include <fstream>

#include <clang-c/Index.h>

using namespace std;


map <string, string> classVariant = {
    {"V0", "Author"},
    {"V1", "Book"},
    {"V2", "Country"},
    {"V3", "City"},
    {"V4", "Island"},
    {"V5", "Language"},
    {"V6", "Sea"},
    {"V7", "Football_Team"}
};

string currentVariant;
char** gArgv;
int gArgc;

void printCursor(CXCursor cursor) {
    CXString displayName = clang_getCursorDisplayName(cursor);
```

```
	std::cout << clang_getCString(displayName) << "\n";
	clang_disposeString(displayName);
}

bool condBaseClass = false;
CXChildVisitResult searchBase(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало

	if (clang_getCString(clang_getCursorDisplayName(cursor)) == currentVariant &&
clang_isDeclaration(clang_getCursorKind(cursor)))
	{
		condBaseClass = true;
	}


	// Основная часть - конец

	clang_visitChildren(cursor, searchBase, nullptr);
	return CXChildVisit_Continue;
}

CXTypeKind diffTypeArray[3] = { CXType_Invalid, CXType_Invalid , CXType_Invalid };

CXChildVisitResult diffType2(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало

	if (clang_getCursorKind(cursor) == 6)
	{
		if (diffTypeArray[0] == CXType_Invalid)
		{
			diffTypeArray[0] = clang_getCursorType(cursor).kind;
		}
		else
		{
			if (diffTypeArray[1] == CXType_Invalid)
			{
				if (clang_getCursorType(cursor).kind != diffTypeArray[0])
				{
					diffTypeArray[1] = clang_getCursorType(cursor).kind;
				}
			}
			else
			{
				if (diffTypeArray[2] == CXType_Invalid)
				{
					if (clang_getCursorType(cursor).kind != diffTypeArray[0]
&& clang_getCursorType(cursor).kind != diffTypeArray[1])
					{
						diffTypeArray[2] =
clang_getCursorType(cursor).kind;
					}
				}
			}
		}
	}

	//cout << clang_getCString(clang_getCursorDisplayName(cursor)) << " == " <<
clang_getCursorKind(cursor) << endl;


	// Основная часть - конец
```

```
	return CXChildVisit_Continue;
}

CXChildVisitResult diffType1(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало

	// Основная часть - конец

	clang_visitChildren(cursor, diffType2, nullptr);
	return CXChildVisit_Continue;
}

bool condSort = false;
CXChildVisitResult searchSort(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало


	if (((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("sort") !=
std::string::npos)
	{
		condSort = true;
	}

	//printCursor(cursor);

	// Основная часть - конец

	clang_visitChildren(cursor, searchSort, nullptr);
	return CXChildVisit_Continue;
}

bool condMin = false;
CXChildVisitResult searchMin(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало


	if
(((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("min_element") !=
std::string::npos)
	{
		condMin = true;
	}

	//printCursor(cursor);

	// Основная часть - конец

	clang_visitChildren(cursor, searchMin, nullptr);
	return CXChildVisit_Continue;
}

bool condTrans = false;
CXChildVisitResult searchTrans(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
```

```
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало


	if (((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("transform")
!= std::string::npos)
	{
		condTrans = true;
	}

	//printCursor(cursor);

	// Основная часть - конец

	clang_visitChildren(cursor, searchTrans, nullptr);
	return CXChildVisit_Continue;
}

bool condAcc = false;
CXChildVisitResult searchAcc(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало


	if
(((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("accumulate") !=
std::string::npos)
	{
		condAcc = true;
	}

	//printCursor(cursor);

	// Основная часть - конец

	clang_visitChildren(cursor, searchAcc, nullptr);
	return CXChildVisit_Continue;
}

bool condFor = false;
CXChildVisitResult searchFor(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало


	if (((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("for_each")
!= std::string::npos)
	{
		condFor = true;
	}

	//printCursor(cursor);

	// Основная часть - конец

	clang_visitChildren(cursor, searchFor, nullptr);
	return CXChildVisit_Continue;
}

bool condFind = false;
```

```
CXChildVisitResult searchFind(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало


	if (((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("find_if")
!= std::string::npos)
	{
		condFind = true;
	}

	//printCursor(cursor);

	// Основная часть - конец

	clang_visitChildren(cursor, searchFind, nullptr);
	return CXChildVisit_Continue;
}

bool condStream = false;
CXChildVisitResult searchStream(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало


	if (((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("ifstream")
!= std::string::npos ||
((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("ofstream") !=
std::string::npos)
	{
		condStream = true;
	}

	//printCursor(cursor);

	// Основная часть - конец

	clang_visitChildren(cursor, searchStream, nullptr);
	return CXChildVisit_Continue;
}

bool condLamda = false;
CXChildVisitResult searchLamda(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало

	if (clang_getCursorKind(cursor) == 144)
	{
		condLamda = true;
	}

	// Основная часть - конец

	clang_visitChildren(cursor, searchLamda, nullptr);
	return CXChildVisit_Continue;
}

bool condFunctor = false;
```

```
CXChildVisitResult searchFunctor(CXCursor cursor, CXCursor /* parent */, CXClientData
/*clientData*/)
{
	CXSourceLocation location = clang_getCursorLocation(cursor);
	if (clang_Location_isFromMainFile(location) == 0)
		return CXChildVisit_Continue;


	// Основная часть - начало

	if
(((string)clang_getCString(clang_getCursorDisplayName(cursor))).find("operator()") !=
std::string::npos)
	{
		condFunctor = true;
	}

	// Основная часть - конец

	clang_visitChildren(cursor, searchFunctor, nullptr);
	return CXChildVisit_Continue;
}

// тест на присутствие базового класса
TEST(TestCaseName, Test_BC) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append(currentVariant);
	source_filename.append(".h");


	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, searchBase, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condBaseClass, expected);
}

// тест на присутствие минимум 3х полей разного типа
TEST(TestCaseName, Test_DT) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
```

```
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append(currentVariant);
	source_filename.append(".h");


	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, diffType1, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool result = true;
	bool expected = true;

	for (int i = 0; i < 3; i++)
	{
		if (diffTypeArray[i] == CXType_Invalid)
			result = false;
	}

	EXPECT_EQ(result, expected);
}

// Тест на sort
TEST(TestCaseName, Test_Sort) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append("Source.cpp");


	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
```

```
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, searchSort, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condSort, expected);
}

// Тест на min_element
TEST(TestCaseName, Test_Min_element) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append("Source.cpp");


	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, searchMin, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condMin, expected);
}

// Тест на transform
TEST(TestCaseName, Test_Transform) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append("Source.cpp");
```

```
	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, searchTrans, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condTrans, expected);
}

// Тест на acc
TEST(TestCaseName, Test_Acc) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append("Source.cpp");


	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, searchAcc, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condAcc, expected);
}

// Тест на for_each
TEST(TestCaseName, Test_For) {
```

```
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append("Source.cpp");


	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, searchFor, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condFor, expected);
}

// Тест на find_if
TEST(TestCaseName, Test_Find) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append("Source.cpp");


	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);
```

```
		// Находим спецификаторы
		clang_visitChildren(root, searchFind, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condFind, expected);
}

// Тест на if/ofstream
TEST(TestCaseName, Test_Stream) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append("Source.cpp");


	CXTranslationUnit unit = clang_parseTranslationUnit(
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, searchStream, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condStream, expected);
}

// Тест на лямда-выражение
TEST(TestCaseName, Test_Lamda) {
	CXIndex index = clang_createIndex(0, 1);

	// Добавление флага для компиляции c++, а не c
	char** new_argv = new char* [gArgc + 2];
	for (int i = 0; i < gArgc; i++)
	{
		new_argv[i] = gArgv[i];
	}
	new_argv[gArgc] = "-x";
	new_argv[gArgc + 1] = "c++";

	// Создание строки каталога исходного файла
	string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
	source_filename.append("Source.cpp");


	CXTranslationUnit unit = clang_parseTranslationUnit(
```

```
		index, // CIdx
		source_filename.c_str(), // source_filename
		new_argv, // command_line_args
		gArgc + 2, // num_command_line_args
		0, // unsave_files
		0, // num_unsaved_files
		CXTranslationUnit_None // options
	);
	if (!unit) {
		std::cout << "Translation unit was not created\n";
	}
	else {

		CXCursor root = clang_getTranslationUnitCursor(unit);

		// Находим спецификаторы
		clang_visitChildren(root, searchLamda, nullptr);
	}
	clang_disposeTranslationUnit(unit);
	clang_disposeIndex(index);

	bool expected = true;

	EXPECT_EQ(condLamda, expected);
}

// поиск функторов
TEST(TestCaseName, Test_Functor) {
	for (int i = 0; i < 2; i++)
	{
		CXIndex index = clang_createIndex(0, 1);

		// Добавление флага для компиляции c++, а не c
		char** new_argv = new char* [gArgc + 2];
		for (int i = 0; i < gArgc; i++)
		{
			new_argv[i] = gArgv[i];
		}
		new_argv[gArgc] = "-x";
		new_argv[gArgc + 1] = "c++";

		// Создание строки каталога исходного файла
		string source_filename = "../../Lab6_plug/"; // ИЗМЕНИТЬ
		if (i == 0)
		{
			source_filename.append("Source.cpp");
		}
		else
		{
			source_filename.append(currentVariant);
			source_filename.append(".h");
		}



		CXTranslationUnit unit = clang_parseTranslationUnit(
			index, // CIdx
			source_filename.c_str(), // source_filename
			new_argv, // command_line_args
			gArgc + 2, // num_command_line_args
			0, // unsave_files
			0, // num_unsaved_files
			CXTranslationUnit_None // options
		);
		if (!unit) {
			std::cout << "Translation unit was not created\n";
		}
		else {

			CXCursor root = clang_getTranslationUnitCursor(unit);

			// Находим спецификаторы
			clang_visitChildren(root, searchFunctor, nullptr);
		}
		clang_disposeTranslationUnit(unit);
		clang_disposeIndex(index);
```

```
    }

    bool expected = true;
    EXPECT_EQ(condFunctor, expected);
}

int main(int argc, char** argv) {
    testing::InitGoogleTest(&argc, argv);

    currentVariant = classVariant[argv[1]];
    gArgv = argv;
    gArgc = argc;


    RUN_ALL_TESTS();
    std::cin.get();

    return 0;
}
```

MainWindow.xaml.cs:

```
using Microsoft.Win32;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System;
using System.IO;
using System.Windows;
using Microsoft.Win32;
using System.Diagnostics;

namespace StudentCheker
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        string currentLab;
        string currentV;
        string[] currentfilenames = new string[0];

        public MainWindow()
        {
            InitializeComponent();
        }

        private void btnOpenFile_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "CPP files (*.cpp;*.h)|*.cpp;*.h";
            openFileDialog.Multiselect = true;
            TB.Text = "";

            if (openFileDialog.ShowDialog() == true)
            {
                currentfilenames = new string[openFileDialog.FileNames.Length];

                int i = 0;
                foreach (string filename in openFileDialog.FileNames)
                {
                    TB.Text += System.IO.Path.GetFileName(filename) + " ";
                    currentfilenames[i++] = filename;
                }
            }
        }

        private void RadioButton1_Checked(object sender, RoutedEventArgs e)
        {
            RadioButton pressed = (RadioButton)sender;
```

```
            currentLab = pressed.Content.ToString();
        }

        private void RadioButton2_Checked(object sender, RoutedEventArgs e)
        {
            RadioButton pressed = (RadioButton)sender;
            currentV = pressed.Content.ToString();
        }

        private void startCheck(object sender, RoutedEventArgs e)
        {
            if(currentLab == "5")
            {
                Process.Start("../../../../project/lab5_plug/delete.bat").WaitForExit();
            }
            if(currentfilenames.Length > 0)
            {
                for(int i = 0; i < currentfilenames.Length; i++)
                {
                    string path = @currentfilenames[i];
                    string newPath =
"C:\\Users\\drobi\\source\\repos\\StudentCheker\\project\\Lab"+ currentLab + "_plug\\Lab"+
currentLab + "_plug\\" + System.IO.Path.GetFileName(currentfilenames[i]);
                    FileInfo fileInf = new FileInfo(path);
                    if (fileInf.Exists)
                    {
                        fileInf.CopyTo(newPath, true);
                        // альтернатива с помощью класса File
                        // File.Move(path, newPath, true);
                    }
                }
            }
            Process.Start("../../../../project/lab"+ currentLab +
"_plug/refresh.bat").WaitForExit();
            Process.Start("../../../../project/lab"+ currentLab + "_plug/x64/Debug/V" +
currentV + ".bat");
        }
    }
}
```