Программа для определения принадлежности точки невыпуклому многоугольнику	
Внутренняя спецификация	
Студент	Година К.М.
Преподаватель	
Сдано	

1. Назначение

Программа предназначена для определения принадлежности точки невыпуклому многоугольнику на основе координат вершин многоугольника и координат проверяемой точки, заданных во входном файле.

2. Описание структур данных

Структура Point содержит поля:
double x - координата точки по оси X
double y - координата точки по оси Y

Структура Polygon содержит поля:
vector<Point> vertices - вектор точек, представляющих вершины многоугольника

Перечисление error_type содержит значения:
FILE_NOT_FOUND - файл не найден
OUTPUT_FILE_ERROR - ошибка выходного файла
WRONG_ARGS_COUNT - неверное количество аргументов командной строки
VERTEXES_SECTION_NOT_FOUND - не найден раздел Vertexes
POINT_SECTION_NOT_FOUND - не найден раздел Point
VERTEX_COUNT_TOO_SMALL - слишком мало вершин многоугольника
VERTEX_COUNT_TOO_BIG - слишком много вершин многоугольника
VERTEX_FORMAT_ERROR - ошибка формата координат вершин
POINT_FORMAT_ERROR - ошибка формата координат точки
COORDINATE_OUT_OF_RANGE - координата вне допустимого диапазона
INCORRECT_VERTICES_ORDER - неверный порядок вершин многоугольника

Структура Error содержит поля:
enum error_type type - тип ошибки
int line_num - номер строки, где обнаружена ошибка
int col_num - номер столбца, где обнаружена ошибка

char symbol - символ, вызвавший ошибку
string str - дополнительная информация об ошибке

Константы:
MIN_VERTICES = 3 - минимальное количество вершин многоугольника
MAX_VERTICES = 100 - максимальное количество вершин многоугольника
MIN_COORDINATE = -1000.0 - минимальное значение координаты
MAX_COORDINATE = 1000.0 - максимальное значение координаты
MAX_DECIMAL_PLACES = 6 - максимальное количество знаков после запятой

3. Описание алгоритмов функций

Главная функция программы:

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

Входные данные:

argc - количество аргументов командной строки

argv - массив аргументов командной строки

argv[1] - путь к входному файлу, argv[2] - путь к выходному файлу

Выходные данные:

0 - функция завершилась успешно

1 - ошибка

Алгоритм функции:

```
// Обработать возможные исключения с помощью блока try-catch
// В блоке try
    // Проверить количество аргументов командной строки
    // Получить путь к входному и выходному файлам
    // Создать объекты polygon и testPoint
    // Прочитать данные из входного файла (readInputFile)
    // Проверить принадлежность точки многоугольнику (isPointInPolygon)
    // Открыть выходной файл
    // Записать результат в выходной файл (writeResult)
// В блоке catch для Error:
    // Вывести сообщение об ошибке (getErrorMessage)
    // Вывести дополнительную информацию об ошибке (строка, позиция)
// В блоке catch для стандартных исключений:
    // Вывести сообщение о непредвиденной ошибке
// В блоке catch для всех остальных исключений:
    // Вывести сообщение о неизвестной ошибке
// Завершить работу программы
```

Функция для формирования сообщений об ошибке:

string getErrorMessage(const Error& error)

Входные данные:

error - объект с информацией об ошибке

Выходные данные:

строка с сообщением об ошибке

Алгоритм функции:

```
// Проверить тип ошибки
// В зависимости от типа ошибки вернуть соответствующее сообщение об ошибке:
    // FILE_NOT_FOUND: "Файл с входными данными указан некорректно. Файл может не существовать."
    // OUTPUT_FILE_ERROR: "Выходной файл указан некорректно. Возможно, указанное расположение не существует или отсутствуют права на запись."
    // WRONG_ARGS_COUNT: "Программа принимает два аргумента: <путь к входному файлу> <путь к выходному файлу>"
    // VERTEXES_SECTION_NOT_FOUND: "Неверная структура входного файла. Раздел Vertexes не найден."
    // POINT_SECTION_NOT_FOUND: "Неверная структура входного файла. Раздел Point не найден."
    // VERTEX_COUNT_TOO_SMALL: "Ошибка в разделе Vertexes: количество вершин должно быть не менее 3."
    // VERTEX_COUNT_TOO_BIG: "Ошибка в разделе Vertexes: количество вершин не должно превышать 100."
    // VERTEX_FORMAT_ERROR: "Ошибка формата координат вершин."
    // POINT_FORMAT_ERROR: "Ошибка формата координат точки."
    // COORDINATE_OUT_OF_RANGE: "Ошибка: значение координаты выходит за допустимый диапазон от -1000 до 1000"
    // INCORRECT_VERTICES_ORDER: "Ошибка: вершины многоугольника должны быть указаны в порядке их обхода (по часовой или против часовой стрелки)."
    // В случае неизвестной ошибки: "Неизвестная ошибка."
```

Функция для работы с ошибками:

void throwError(enum error_type type, int line, int col, char symbol, const string& str)

Входные данные:

type - тип ошибки

line - номер строки

col - номер столбца

symbol - символ ошибки

str - дополнительная информация

Алгоритм функции:

```
// Создать объект ошибки Error с переданными параметрами
```

// Выбросить созданный объект как исключение

Функция для проверки корректности координаты:

bool isValidCoordinate(double coord)

Входные данные:

coord - значение координаты для проверки

Выходные данные:

true - если координата в допустимом диапазоне

false – если не в допустимом диапазоне

Алгоритм функции:

// Проверить, находится ли координата в диапазоне от MIN_COORDINATE до MAX_COORDINATE
// Вернуть результат проверки

Функция для чтения числа с плавающей точкой из строки:

bool readDouble(const string& str, size_t& pos, double& value)

Входные данные:

str - строка, содержащая число

pos - позиция в строке, с которой начинается чтение

Выходные данные:

value - считанное число

true - если чтение выполнено успешно,

false - иначе

Алгоритм функции:

// Запомнить начальную позицию в строке
// Пропустить начальные пробелы в строке, увеличивая позицию
// Если достигнут конец строки, вернуть false
// Проверить наличие знака числа (+ или -), при наличии увеличить позицию
// Считать цифры до десятичной точки, формируя целую часть числа
// При наличии десятичной точки считать цифры после точки (не более MAX_DECIMAL_PLACES знаков)
// Если не было считано ни одной цифры, вернуть false
// Установить значение переменной value и применить знак числа
// Пропустить завершающие пробелы в строке
// Вернуть true в случае успешного чтения числа

Функция чтения данных из входного файла:

void readInputFile(const string& filename, Polygon& polygon, Point& testPoint)

Входные данные:

filename - имя входного файла

Выходные данные:

polygon - многоугольник с вершинами из файла

testPoint - точка для проверки

Алгоритм функции:

```
// Открыть входной файл
// Если файл не открылся, выбросить исключение FILE_NOT_FOUND
// Инициализировать счетчик строк и флаги нахождения разделов Vertexes и Point
// Искать в файле строку "Vertexes:"
    // Если не найдена, выбросить исключение VERTEXES_SECTION_NOT_FOUND
// Прочитать количество вершин многоугольника
    // Если количество меньше MIN_VERTICES, выбросить исключение VERTEX_COUNT_TOO_SMALL
    // Если количество больше MAX_VERTICES, выбросить исключение VERTEX_COUNT_TOO_BIG
// Прочитать координаты всех вершин многоугольника:
    // Для каждой вершины проверить формат координат (используя readDouble)
    // Проверить, находятся ли координаты в допустимом диапазоне (используя isValidCoordinate)
    // Если формат неверный, выбросить исключение VERTEX_FORMAT_ERROR
    // Если координаты вне диапазона, выбросить исключение COORDINATE_OUT_OF_RANGE
    // Добавить вершину в вектор polygon.vertices
// Искать в файле строку "Point:"
    // Если не найдена, выбросить исключение POINT_SECTION_NOT_FOUND
// Прочитать координаты проверяемой точки:
    // Проверить формат координат (используя readDouble)
    // Проверить, находятся ли координаты в допустимом диапазоне (используя isValidCoordinate)
    // Если формат неверный, выбросить исключение POINT_FORMAT_ERROR
    // Если координаты вне диапазона, выбросить исключение COORDINATE_OUT_OF_RANGE
    // Установить значения testPoint.x и testPoint.y
// Закрыть файл
```

Функция для проверка порядка обхода вершин многоугольника:

bool checkVerticesOrder(const Polygon& polygon)

Входные данные:

polygon - многоугольник для проверки

Выходные данные:

true - если порядок вершин корректный

false - иначе

Алгоритм функции:

```
// Проверить, что количество вершин многоугольника не менее 3
    // Если меньше, вернуть false
// Вычислить площадь многоугольника по формуле Гаусса:
    // Инициализировать переменную area = 0
    // Для каждой пары последовательных вершин i и j = (i + 1) % n:
        // area += polygon.vertices[i].x * polygon.vertices[j].y
        // area -= polygon.vertices[j].x * polygon.vertices[i].y
    // area = abs(area) / 2.0
// Если площадь близка к нулю (меньше 1e-10), вернуть false
// Иначе вернуть true
```

Функция для проверки принадлежности точки многоугольнику:

bool isPointInPolygon(const Polygon& polygon, const Point& point)

Входные данные:

polygon - многоугольник

point - точка для проверки

Выходные данные:

true - если точка принадлежит многоугольнику

false - иначе

Алгоритм функции:

```
// Проверить корректность порядка вершин многоугольника (checkVerticesOrder)
    // Если порядок некорректный, выбросить исключение INCORRECT_VERTICES_ORDER
// Инициализировать флаг inside = false (точка вне многоугольника)
// Для каждой стороны многоугольника (определяемой последовательными вершинами vi и vj):
    // Проверить, находится ли точка на стороне многоугольника:
        // Вычислить векторное произведение
        // Если близко к нулю, то точка может лежать на стороне
        // Проверить скалярное произведение, чтобы определить, лежит ли точка между вершинами
        // Если да, вернуть true (точка на стороне)
    // Применить алгоритм трассировки лучом (Ray Casting):
        // Если условие ((vi.y > point.y) != (vj.y > point.y)) истинно и точка находится слева от пересечения луча со стороной, инвертировать флаг inside
// Вернуть значение флага inside
```

Функция для записи результата:

void writeResult(const string& outputFile, const Point& testPoint, bool isInside);

Входные данные:

outputFile - имя выходного файла

testPoint - проверяемая точка

isInside - результат проверки: true если точка внутри многоугольника, иначе false

Алгоритм функции:

```
// Открыть выходной файл
// Если не удалось открыть, выбросить исключение OUTPUT_FILE_ERROR
// Записать результат в выходной файл
// Закрыть выходной файл
```

4. Диаграмма вызовов функций и UML-диаграмма классов приведена в приложении №1.

5. Диаграмма потоков данных приведена в приложении №2.

Приложение №1 - Диаграмма вызовов функций и UML-диаграмма классов

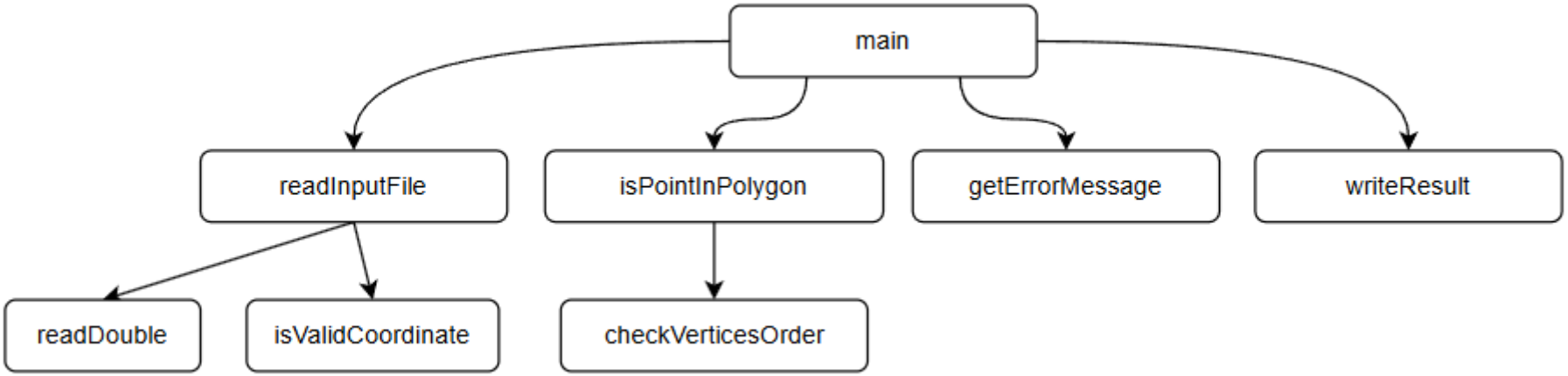

Рис 1. Диаграмма вызовов функций

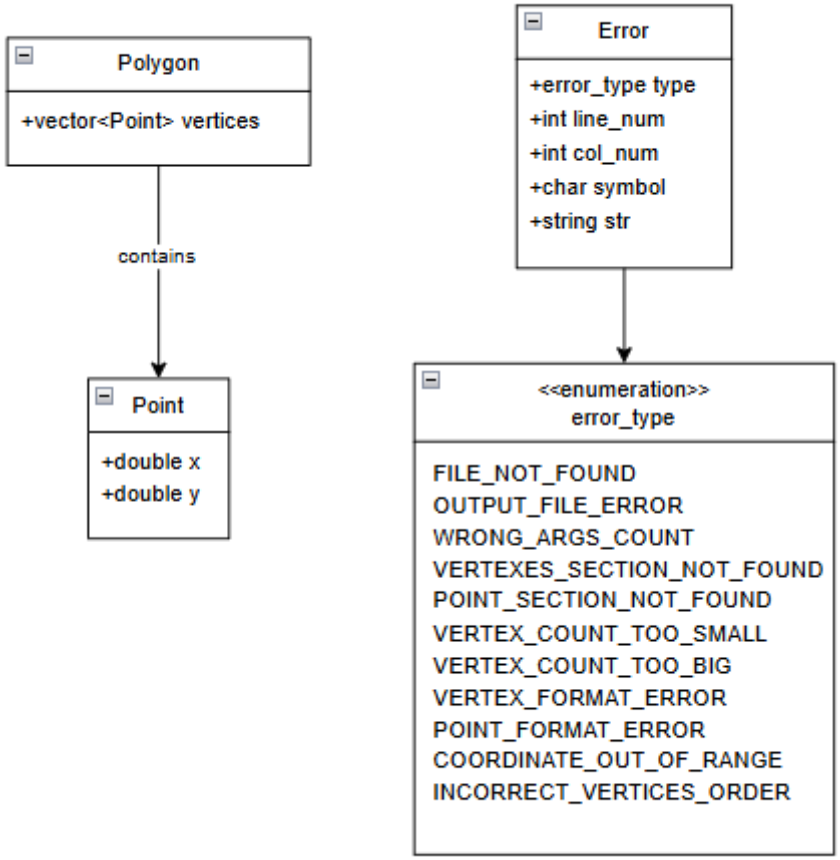

Рис. 2. UML-диаграмма классов

Приложение №2 - Диаграмма потоков данных

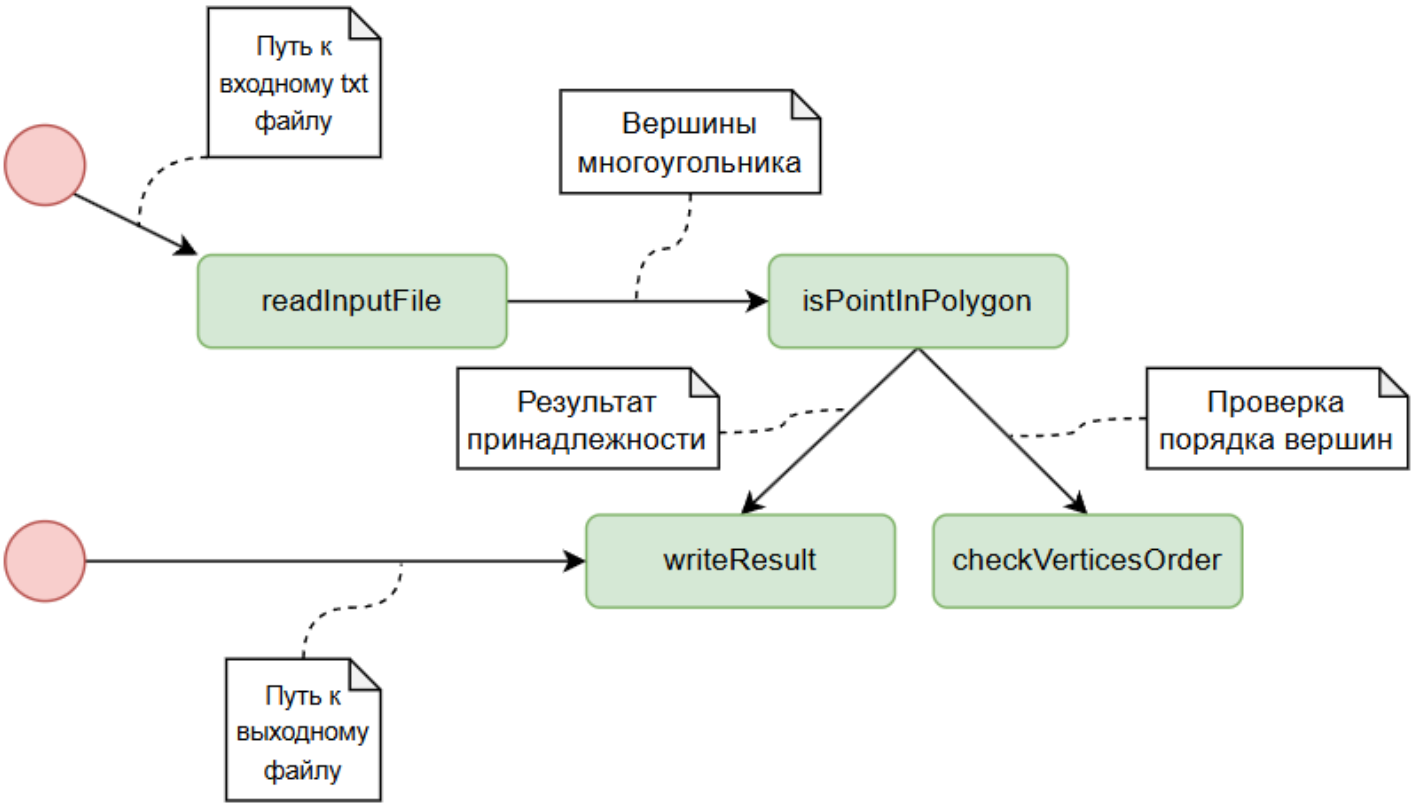

Рис. 3. Диаграмма потоков данных