Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Волгоградский государственный технический университет»

УТВЕРЖДАЮ:

Зав. Кафедрой ПОАС

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Дворянкин А.М.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

НАДЕЖНОСТЬ И КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Тема работы: «Определение принадлежности точки невыпуклому многоугольнику»

Программа и методика испытаний

|  |  |
| --- | --- |
| СОГЛАСОВАНО: | РАЗРАБОТЧИК: |
| Руководитель работы доцент кафедры ПОАС | Студент группы ПрИн-268 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Сычев О.А. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Нечта Б.О |
| «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |
|  | НОРМОКОНТРОЛЛЕР: |
|  |  |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сычев О.А. |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |

2025

1. Объект испытаний

Программа называется "Polygon Point Containment" («Программа для проверки принадлежности точки многоугольнику»).  
Программа написана с использованием языков программирования C и C++ без использования сторонних библиотек. Она предназначена для чтения данных о многоугольниках и точках, проверки валидности данных, корректности многоугольника, а также определения принадлежности точки этому многоугольнику.

2. Цель испытаний

Целью испытаний является проверка функциональности программы с точки зрения её способности:

3. Требования к программе

Требования к программе:

1. Программа должна корректно работать с входными данными в формате текстового файла, содержащего координаты точек.
2. Программа должна проверять:  
   Корректность данных (наличие ошибок в формате, дублирующиеся вершины, координаты, выходящие за пределы допустимого диапазона и другие).  
   Корректность многоугольника (минимальное количество вершин, правильность порядка точек).  
   Принадлежность точки многоугольнику (с использованием метода пересечения луча или другого алгоритма).
3. Программа должна корректно записывать результат в выходной файл.
4. Программа должна корректно выводить сообщения об ошибках в консоль, если данные или многоугольник некорректны.

4. Требования к программной документации

В бумажной форме должны быть предоставлены: Техническое задание, описание программы. Программа и методика испытаний. Руководство программиста.

В электронном формате должны быть предоставлены: Копии всех документов бумажной формы. Рабочая документация и текст программы. Вся документация должна быть составлена в соответствии с ГОСТ 19.

5. Средства и порядок испытаний

Для запуска тестов необходимо наличие компилятора C++ и выполнения тестов. Кроме того, необходимо использовать стандартные библиотеки C++ для работы с файлами, строками, контейнерами и математическими вычислениями. Тесты будут выполнены с использованием IDE Visual Studio Code

Приложение 1 Методика тестирования функции validate

/\*!

\* \brief Проверяет семантическую корректность уже считанных данных (многоугольник + точка).

\* \param[out] err – ссылка на объект Error, куда записываются сведения об ошибках валидации.

\* \return true, если данные полностью валидны; false — при обнаружении ошибки.

\* \throw (Не выбрасывает исключений, все ошибки отражаются через err.)

\*/

bool validate(Error &err);

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название теста | Входные данные | Ожидаемый результат |
| 1 | Успешная валидация данных | 3 0 0 1 0 0 1 0.5 0.5 | validate → true; err.type = noError; err.errorLineNumber = 0; err.errorLineContent = ""; |
| 2 | Пустой файл | (файл пустой) | validate → false; err.type = emptyFile; err.errorLineNumber = 1; err.errorLineContent = ""; |
| 3 | Пустые строки между данными | 4 0 0  ---- 1 1 1 0 0.5 0.5 | validate → false; err.type = emptyLineFound; err.errorLineNumber = 3; err.errorLineContent = ""; |
| 4 | Пустые строки в конце | 3 0 0 1 1 0 1 0.5 0.5 --------- | validate → false; err.type = emptyLineFound; err.errorLineNumber = 6; err.errorLineContent = ""; |
| 5 | Некорректное количество элементов | 4 0 0 1 0 1 1 1 0.5 0.5 | validate → false; err.type = wrongElementCountInLine; err.errorLineNumber = 3; err.errorLineContent = "1"; |
| 6 | лишний элемент | 3 0 0 1 0 0 0 1 0.5 0.5 | validate → false; err.type = wrongElementCountInLine; err.errorLineNumber = 3; err.errorLineContent = "1 0 0"; |
| 7 | Дублирующиеся вершины | 3 0 0 1 0 0 0 0.5 0.5 | validate → false; err.type = duplicateVertex; err.errorLineNumber = 4; err.errorLineContent = "0 0"; |
| 8 | Некорректные символы в строках | 3 0 0 a b 1 1 0.5 0.5 | validate → false; err.type = invalidCharacters; err.errorLineNumber = 3; err.errorLineContent = "a b"; |
| 9 | Отрицательные координаты вне диапазона | 3 -1000 0 0 1 1 0 0.5 0.5 | validate → false; err.type = coordinateOutOfRange; err.errorLineNumber = 2; err.errorLineContent = "-1000 0"; |
| 10 | Координаты больше диапазона | 3 0 0 1000 1 1 0 0.5 0.5 | validate → false; err.type = coordinateOutOfRange; err.errorLineNumber = 3; err.errorLineContent = "1000 1"; |
| 11 | Слишком мало вершин (<3) | 2 0 0 1 1 0.5 0.5 | validate → false; err.type = invalidVertexCount; err.errorLineNumber = 1; err.errorLineContent = "2"; |
| 12 | Слишком много вершин (>1000) | 1001 0 0 0 1 ... (999 точек) 1 1 0.5 0.5 | validate → false; err.type = invalidVertexCount; err.errorLineNumber = 1; err.errorLineContent = "1001"; |
| 13 | Неверный порядок обхода (wrongOrder) | 3 0 0 0 1 1 0 0.5 0.3 | validate → false; err.type = wrongOrder; err.errorLineNumber = 0; err.errorLineContent = ""; |
| 14 | Тестовая точка вне диапазона | 3 0 0 1 0 0 1 1000 1000 | validate → false; err.type = pointOutOfRange; err.errorLineNumber = 5; err.errorLineContent = "1000 1000"; |
| 15 | Строка с двойным разделителем | 3 0 0 1,,0 0 1 0.5 0.5 | validate → false; err.type = wrongElementCountInLine; err.errorLineNumber = 3; err.errorLineContent = "1,,0"; |
| 16 | Несоответствие количества вершин и точек | 4 0 0 1 0 0 1 0.5 0.5 | validate → false; err.type = verticesMismatch; err.errorLineNumber = 1; err.errorLineContent = "4"; (например, указано 4, а вершин — меньше/больше) |
| 17 | Пустая строка перед данными | --- 3 0 0 1 0 0 1 0.5 0. | validate → false; err.type = emptyLineFound; err.errorLineNumber = 1; err.errorLineContent = ""; |
| 18 | Многоугольник с точками на одной прямой | 3 0 0 1 1 2 2 0.5 0.5 | validate → false; err.type = invalidPolygon; err.errorLineNumber = 4; err.errorLineContent = "2 2"; |

Приложение 2 Методика тестирования функции isValidPolygon

/\*!

\* \brief Проверяет, является ли набор точек корректным (невыпуклым) многоугольником.

\* \param[in] vertices – ссылка на вектор Point, задающий вершины многоугольника.

\* \param[out] err – ссылка на объект Error, куда записываются сведения об ошибках при проверке.

\* \return true, если многоугольник корректен; false — в противном случае.

\* \throw (Не выбрасывает исключений, все ошибки отражаются через err.)

\*/

bool isValidPolygon(const std::vector<Point> &vertices, Error &err);

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название теста | Входные данные | Ожидаемый результат |
| 1 | Валидный невыпуклый | { (0,0), (1,1), (2,0), (1,-1) } | isValidPolygon → true; err.type = noError. |
| 2 | < 3 вершин | { (0,0), (1,1) } | isValidPolygon → false; err.type = invalidVertexCount; err.errorLineNumber=1; |
| 3 | Коллинеарные точки | { (0,0), (1,1), (2,2), (0,2) } | isValidPolygon → false; err.type = invalidPolygon; err.errorLineNumber=3; |
| 4 | Самопересечение рёбер | { (0,0), (2,2), (0,2), (2,0) } | isValidPolygon → false; err.type = invalidPolygon; err.errorLineNumber=3; |
| 5 | Сложное самопересечение | { (0,0), (2,0), (1,2), (0,2), (1,1) } | isValidPolygon → false; err.type = invalidPolygon; err.errorLineNumber=5; |
| 6 | > 1000 вершин | 1001 случайная точка | isValidPolygon → false; err.type = invalidVertexCount; |
| 7 | Выпуклый многоугольник | { (0,0), (0,2), (2,2), (2,0) } | isValidPolygon → false; err.type = invalidPolygon; |
| 8 | Повторяющиеся точки | { (0,0), (1,1), (1,1), (0,1) } | isValidPolygon → false; err.type = duplicateVertex; |
| 9 | Неправильный порядок обхода | { (0,0), (0,2), (2,0), (2,2) } (по часовой стрелке) | isValidPolygon → false; err.type = wrongOrder; |
| 10 | Точки вне диапазона | { (0,0), (1,1), (1001,0), (0,1) } | isValidPolygon → false; err.type = coordinateOutOfRange; |
| 11 | Первая точка совпадает с последней | { (0,0), (1,1), (1,0), (0,0) } | isValidPolygon → false; err.type = duplicateVertex; |
| 12 | Все точки совпадают | { (1,1), (1,1), (1,1) } | isValidPolygon → false; err.type = duplicateVertex; |
| 13 | Два ребра "касательно" | { (0,0), (1,0), (1,1), (2,0), (0,2) } | isValidPolygon → false; err.type = invalidPolygon; (если реализована проверка касания) |
| 14 | Один и тот же вектор дважды | { (0,0), (2,2), (4,4), (2,2) } | isValidPolygon → false; err.type = duplicateVertex; |

Приложение 3 Методика тестирования функции containsPoint

/\*!

\* \brief Определяет, принадлежит ли точка многограннику (внутри/на границе/снаружи).

\* \param[in] polygon – ссылка на объект Polygon (или эквивалентный вектор вершин).

\* \param[in] p – проверяемая точка типа Point.

\* \return true, если точка «считается» внутри (или на границе); false — если снаружи.

\* \throw (Не выбрасывает исключений, результат через bool.)

\*/

bool containsPoint(const Polygon &polygon, const Point &p);

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название теста | Входные данные | Ожидаемый результат |
| 1 | Точка внутри выпуклого многоугольника | polygon = { (0,0), (0,2), (2,2), (2,0) }, p = (1,1) | **true** |
| 2 | Точка снаружи выпуклого многоугольника | polygon = { (0,0), (0,2), (2,2), (2,0) }, p = (3,3) | **false** |
| 3 | Точка на ребре | polygon = { (0,0), (0,2), (2,2), (2,0) }, p = (1,2) | **true** |
| 4 | Точка на вершине | polygon = { (0,0), (0,2), (2,2), (2,0) }, p = (2,2) | **true** |
| 5 | Точка на стороне с отрицательными координатами | polygon = { (-1,-1), (-1,1), (1,1), (1,-1) }, p = (0,1) | **true** |
| 6 | Точка строго вне невыпуклого | polygon = { (0,0), (2,1), (1,0), (2,2), (0,2) }, p = (1.5,1.7) | **false** |
| 7 | Точка внутри невыпуклого | polygon = { (0,0), (2,1), (1,0), (2,2), (0,2) }, p = (1,1) | **true** |
| 8 | Точка совпадает с первой вершиной | polygon = { (0,0), (0,2), (2,2), (2,0) }, p = (0,0) | **true** |
| 9 | Точка совпадает с последней вершиной | polygon = { (0,0), (0,2), (2,2), (2,0) }, p = (2,0) | **true** |
| 10 | Точка с очень близким попаданием на грань | polygon = { (0,0), (0,2), (2,2), (2,0) }, p = (2,0.1) | **true** |
| 11 | Точка очень далеко от полигона | polygon = { (0,0), (0,2), (2,2), (2,0) }, p = (-9999,9999) | **false** |
| 12 | Многоугольник с пересекающимися рёбрами, точка внутри | polygon = { (0,0), (2,2), (0,2), (2,0) }, p = (1,1) | **true** |
| 13 | Многоугольник с точкой "на пересечении" рёбер | polygon = { (0,0), (2,2), (0,2), (2,0) }, p = (1,1) | **true** |
| 14 | Большой многоугольник (999 точек), точка вне | polygon = многоугольник на 999 вершин по окружности, p = (999,999) | **false** |
| 15 | Большой многоугольник, точка внутри | polygon = многоугольник на 999 вершин по окружности, p = (0,0) | **true** |
| 16 | Точка внутри сложного 10-угольника | polygon = { (0,0), (2,0), (3,1), (4,3), (4,5), (2,6), (0,5), (-1,3), (0,2), (1,1) },p = (2,3) | true. |
| 17 | Точка вне сложного 15-угольника | polygon = { (0,0), (1,2), (2,4), (3,6), (4,8), (5,10), (6,12), (7,14), (8,16), (9,18), (10,20), (11,22), (12,24), (13,26), (14,28) },p = (20,30) | false. |
| 18 | Точка на границе 20-угольника | polygon = { (0,0), (1,2), (2,4), (3,6), (4,8), (5,10), (6,12), (7,14), (8,16), (9,18), (10,20), (11,22), (12,24), (13,26), (14,28), (15,30), (16,32), (17,34), (18,36), (19,38) },p = (0,0) | true. |