ФИО: Ребдев Павел Александрович
Группа: 5130904/30008
Лабораторная работа: «Геометрические фигуры»

Постановка задачи

Разработать детальные требования и тест план для следующей задачи:

[CONCAVE] и [POLYGON]. Реализовать программу принимающую на вход фигуры в виде строк (Название фигуры и вершины фигуры) и параметры сканирования (точка скалирования, коэффициент скалирования) и возвращающую суммарную площадь и ограничивающие прямоугольники фигур до и после скалирования

Детальные требования

1. фигура RECTANGLE:
   1.1. После ключевого слова RECTANGLE должны идти координаты двух точек. Если хотя бы одна из координат не является вещественным числом, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре
   1.2. Если введённые две точки не являются левым нижним и правым верхнем углами прямоугольника, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре
2. фигура CONCAVE:
   2.1. После ключевого слова CONCAVE должны идти координаты четырёх точек. Если хотя бы одна из координат не является вещественным числом, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре
   2.2. Если среди введённых четырёх точек нет такой последовательности, при которой три идущие подряд удовлетворяют условию вершин треугольника, а четвёртая лежит внутри этого треугольника, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре
3. фигура POLYGON:
   3.1. После ключевого слова POLYGON должны идти координаты не менее трёх точек. Если хотя бы одна из координат не является вещественным числом, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре
   3.2. Если среди введённых точек есть повторяющиеся, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре
4. Ввод неизвестных фигур и пустых строк:
   4.1. Если введена пустая строка, то она игнорируется и программа продолжает свою работу
   4.2. Если строка начинается со слова не являющегося RECTANGLE, CONCAVE, POLYGON или SCALE, то строка игнорируется и программа продолжает свою работу
5. команда SCALE:
   5.1. Если не было введено не одной корректной фигуры, то после скалирования программа завершается с выводом сообщения об ошибке в стандартный поток ошибок и кодом 1
   5.2. Если после ключевого слова SCALE идут данные, не соответствующие формату: вещественное (x) вещественное (y) беззнаковое вещественное (k), то программа завершается с выводом сообщения об ошибке в стандартный поток ошибок и кодом 1
   5.3. Если после ключевого слова SCALE идут данные, соответствующие формату: вещественное (x) вещественное (y) беззнаковое вещественное (k); и скалирование выполняется успешно, то в стандартный поток выводятся: сумма площадей всех корректно заданных фигур до скалилорования, координаты вершин ограничивающих прямоугольников всех корректно заданных фигур до скалирования, сумма площадей всех корректно заданных фигур после скалилорования, координаты вершин ограничивающих прямоугольников всех корректно заданных фигур после скалирования и программа завершается с кодом 0

Тест-план

Проверка детальных требований с помощью тест-плана:

#	Описание	Результат
1.1	После ключевого слова RECTANGLE должны идти координаты двух точек. Если хотя бы	**Стандартный поток ввода:** `RECTANGLE 0.0 a.0 1.0 1.0` **Стандартный поток ошибок:**

	одна из координат не является вещественным числом, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре	Some figure is incorrect
1.2	Если введённые две точки не являются левым нижним и правым верхнем углами прямоугольника, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре	**Стандартный поток ввода:** RECTANGLE 0.0 1.0 0.0 0.0 **Стандартный поток ошибок:** Some figure is incorrect
2.1	После ключевого слова CONCAVE должны идти координаты четырёх точек. Если хотя бы одна из координат не является вещественным числом, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре	**Стандартный поток ввода:** CONCAVE 0.0 0.0 1.0 1.0 a.b c.d 2.0 2.0 **Стандартный поток ошибок:** Some figure is incorrect
2.2	Если среди введённых четырёх точек нет такой последовательности, при которой три идущие подряд удовлетворяют условию вершин треугольника, а четвёртая лежит внутри этого треугольника, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре	**Стандартный поток ввода:** CONCAVE 0.0 0.0 1.0 1.0 2.0 2.0 3.0 3.0 **Стандартный поток ошибок:** Some figure is incorrect
3.1	После ключевого слова POLYGON должны идти координаты не менее трёх точек. Если хотя бы одна из координат не является вещественным числом, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре	**Стандартный поток ввода:** POLYGON 0.0 0.0 a.0 1.0 2.0 2.0 **Стандартный поток ошибок:** Some figure is incorrect
3.2	Если среди введённых точек есть повторяющиеся, то после скалирование в стандартный ввод ошибок должно быть выведено сообщение о некорректной фигуре	**Стандартный поток ввода:** POLYGON 0.0 0.0 1.0 1.0 2.0 2.0 0.0 0.0 **Стандартный поток ошибок:** Some figure is incorrect
4.1	Если введена пустая строка, то она игнорируется и программа продолжает свою работу	**Стандартный поток ввода:** RECTANGLE 0.0 0.0 1.0 1.0 CONCAVE -1.0 -1.0 0.0 2.0 2.0 0.0 0.0 0.0 **Expected:** программа работет так же, как при вводе: RECTANGLE 0.0 0.0 1.0 1.0 CONCAVE -1.0 -1.0 0.0 2.0 2.0 0.0 0.0 0.0
4.2	Если строка начинается со слова не являющегося RECTANGLE, CONCAVE, POLYGON или SCALE, то строка игнорируется и программа продолжает свою работу	**Стандартный поток ввода:** RECTANGLE 0.0 0.0 1.0 1.0 CIRCLE 1.0 1.0 2.0 CONCAVE -1.0 -1.0 0.0 2.0 2.0 0.0 0.0 0.0 **Expected:** программа работет так же, как при вводе: RECTANGLE 0.0 0.0 1.0 1.0 CONCAVE -1.0 -1.0 0.0 2.0 2.0 0.0 0.0 0.0
5.1	Если не было введено не одной корректной фигуры, то после скалирования программа завершается с выводом сообщения об ошибке в стандартный поток ошибок и кодом 1	**Стандартный поток ввода:** SCALE 0.0 0.0 2.0 **Стандартный поток ошибок:** Programm end without scale! **Код завершения программы:** 1
5.2	Если после ключевого слова SCALE идут данные, не соответствующие формату: вещественное вещественное беззнаковое вещественное, то программа завершается с выводом сообщения об ошибке в стандартный поток ошибок и кодом 1	**Стандартный поток ввода:** (Some shapes) SCALE 0.0 0.0 -2.0 **Стандартный поток ошибок:** Programm end without scale! **Код завершения программы:** 1

		Стандартный поток ввода: (Some shapes) SCALE 0.0 a.0 2.0 **Стандартный поток ошибок:** Programm end without scale! **Код завершения программы:** 1
5.3	Если после ключевого слова SCALE идут данные, соответствующие формату: вещественное вещественное беззнаковое вещественное; и скалирование выполняется успешно, то в стандартный поток выводятся: сумма площадей всех фигур до скалилорования, координаты вершин ограничивающих прямоугольников всех корректно заданных фигур до скалирования, сумма площадей всех фигур после скалилорования, координаты вершин ограничивающих прямоугольников всех корректно заданных фигур после скалирования и программа завершается с кодом 0	**Стандартный поток ввода:** RECTANGLE 0.0 0.0 1.0 1.0 CONCAVE -1.0 -1.0 0.0 2.0 2.0 0.0 0.0 0.0 CONCAVE -0.5 -1.0 2.5 1.0 0.5 3.0 0.5 1.0 POLYGON 0.0 0.0 2.0 2.0 0.0 1.0 SCALE 0.0 0.0 2.0 **Стандартный поток вывода:** 7.0 0.0 0.0 1.0 1.0 -1.0 -1.0 2.0 2.0 -0.5 -1.0 2.5 3.0 0.0 0.0 2.0 2.0 28.0 0.0 0.0 2.0 2.0 -2.0 -2.0 4.0 4.0 -1.0 -2.0 5.0 6.0 0.0 0.0 4.0 4.0 **Код завершения программы:** 0

Исходные тексты программы

Файлы с исходными текстами лабораторной работы (полагаем <ROOT> для папки в котором располагаются исходные тексты):

./<ROOT>/main.cpp

```
#include <iostream>
#include <string>
#include "shape.hpp"
#include "concave.hpp"
#include "polygon.hpp"
#include "rectangle.hpp"
#include "base-types.hpp"
#include "figureInputFunction.hpp"
#include "outputShapes.hpp"

int main()
{
  std::string figureName;

  rebdev::Shape * shapes[1000] = {};

  size_t numOfShape = 0;
  bool isScale = 0, figureError = 0;

  while (std::getline(std::cin, figureName, ' '))
  {
    if (figureName.find('\n') == 0)
    {
      figureName.erase(0, 1);
    }

    if (figureName.find("SCALE") != std::string::npos)
    {
      rebdev::point_t isoPoint = {0.0, 0.0};
```

```
      double k = 0;
      std::cin >> isoPoint.x >> isoPoint.y >> k;

      if (!std::cin)
      {
        break;
      }

      std::cout << std::fixed;
      std::cout.precision(1);
      rebdev::printShapes(shapes, numOfShape, std::cout);
      std::cout << '\n';
      try
      {
        rebdev::isoScale(shapes, numOfShape, isoPoint, k);
      }
      catch (const std::logic_error & e)
      {
        break;
      }
      isScale = 1;
      break;
    }
    else
    {
      try
      {
        shapes[numOfShape] = rebdev::newFigure(std::cin, figureName);
        numOfShape += (shapes[numOfShape] != nullptr);
      }
      catch (const std::exception & e)
      {
        figureError = 1;
      }
    }

    figureName.clear();
  }

  if (figureError)
  {
    std::cerr << "Some figure is incorrect!\n";
  }
  if (!isScale || (numOfShape == 0))
  {
    std::cerr << "Programm end without scale!\n";
    rebdev::deleteShapes(shapes, numOfShape);
    return 1;
  }

  rebdev::printShapes(shapes, numOfShape, std::cout);
  std::cout << '\n';

  rebdev::deleteShapes(shapes, numOfShape);
```

```
  return 0;
}
```

./<ROOT>/base-types.hpp

```
#ifndef BASE_TYPES_HPP
#define BASE_TYPES_HPP

namespace rebdev
{
  struct point_t
  {
    double x;
    double y;
  };

  struct rectangle_t
  {
    double width;
    double height;
    point_t pos;
  };
}

#endif
```

./<ROOT>/shape.hpp

```
#ifndef SHAPE_HPP
#define SHAPE_HPP

#include <cstddef>
#include "base-types.hpp"

namespace rebdev
{
  class Shape
  {
    public:
    virtual ~Shape() = default;
    virtual double getArea() const = 0;
    virtual rectangle_t getFrameRect() const = 0;
    virtual void move(const point_t & point) = 0;
    virtual void move(double x, double y) = 0;
    virtual void scale(double k) = 0;
  };

    void isoScale(Shape ** arr, size_t numOfShape, const point_t &
scalePoint, double k);
  void deleteShapes(Shape ** shapes, size_t numOfShape);
}

#endif
```

./<ROOT>/shape.cpp

```
#include "shape.hpp"

void rebdev::isoScale(Shape ** shapes, size_t numOfShape, const point_t
& scalePoint, double k)
{
  for (size_t i = 0; i < numOfShape; ++i)
  {
    point_t centerPoint = shapes[i]->getFrameRect().pos;
    shapes[i]->move(scalePoint);
    point_t centerPoint2 = shapes[i]->getFrameRect().pos;
    shapes[i]->scale(k);
    shapes[i]->move((centerPoint.x - centerPoint2.x) * k, (centerPoint.y
- centerPoint2.y) * k);
  }
}

void rebdev::deleteShapes(Shape ** shapes, size_t numOfShape)
{
  for (size_t i = 0; i < numOfShape; ++i)
  {
    delete shapes[i];
  }
}
```

./<ROOT>/rectangle.hpp

```
#ifndef RECTANGLE_HPP
#define RECTANGLE_HPP

#include "shape.hpp"
#include "base-types.hpp"

namespace rebdev
{
  class Rectangle: public Shape
  {
    public:
           Rectangle(const  point_t  &  lowLeftCorner,  const  point_t  &
upRightCorner);

      virtual ~Rectangle() = default;
      virtual double getArea() const;
      virtual rectangle_t getFrameRect() const;
      virtual void move(const point_t & point);
      virtual void move(double x, double y);
      virtual void scale(double k);
    private:
      point_t corners_[2];
  };
}

#endif
```

./<ROOT>/rectangle.cpp

```
#include "rectangle.hpp"
#include <stdexcept>
#include "figureFunction.hpp"

rebdev::Rectangle::Rectangle(const point_t & lowLeftCorner, const point_t & upRightCorner):
  corners_{lowLeftCorner, upRightCorner}
{
     if ((corners_[0].x >= corners_[1].x) || (corners_[0].y >= corners_[1].y))
  {
    throw std::logic_error("rectangle error");
  }
}

double rebdev::Rectangle::getArea() const
{
     return ((corners_[1].x - corners_[0].x) * (corners_[1].y - corners_[0].y));
}

rebdev::rectangle_t rebdev::Rectangle::getFrameRect() const
{
  return getFrameRectangle(corners_, 2);
}

void rebdev::Rectangle::move(const point_t & point)
{
  rectangle_t frameRectangle = getFrameRect();
  move(point.x - frameRectangle.pos.x, point.y - frameRectangle.pos.y);
}

void rebdev::Rectangle::move(double x, double y)
{
  movePoints(x, y, corners_, 2);
}

void rebdev::Rectangle::scale(double k)
{
  scaleFigure(corners_, 2, getFrameRect().pos, k);
}
```

./<ROOT>/concave.hpp

```
#ifndef CONCAVE_HPP
#define CONCAVE_HPP

#include "base-types.hpp"
#include "shape.hpp"

namespace rebdev
{
  class Concave: public Shape
  {
    public:
```

```
      Concave(const point_t & a, const point_t & b, const point_t & c,
const point_t & d);

      virtual ~Concave() = default;
      virtual double getArea() const;
      virtual rectangle_t getFrameRect() const;
      virtual void move(const point_t & point);
      virtual void move(double x, double y);
      virtual void scale(double k);
    private:
      point_t vertexes_[4];
  };

  bool isTriangle(const point_t & f, const point_t & s, const point_t &
t);
}

#endif
```

./<ROOT>/concave.cpp

```
#include "concave.hpp"
#include <cstddef>
#include <stdexcept>
#include "figureFunction.hpp"

rebdev::Concave::Concave(const point_t & a, const point_t & b, const
point_t & c, const point_t & d):
  vertexes_{a, b, c, d}
{
  bool isConcave = false;

  for (size_t i = 0; ((i < 4) && (!isConcave)); ++i)
  {
    if (!isTriangle(vertexes_[i], vertexes_[(i + 1) % 4], vertexes_[(i +
2) % 4]))
    {
      continue;
    }

    double arr[3] = {0.0, 0.0, 0.0};
    for (size_t j = 0; j < 3; ++j)
    {
       point_t vertexesArr[3] = {vertexes_[(i + j) % 4], vertexes_[(i +
3) % 4], vertexes_[(i + (j + 1) % 3) % 4]};

      arr[j] = (vertexesArr[0].x - vertexesArr[1].x) * (vertexesArr[2].y
- vertexesArr[0].y);
                arr[j]   -=   (vertexesArr[2].x   -   vertexesArr[0].x)   *
(vertexesArr[0].y - vertexesArr[1].y);
    }

     bool identicalSigns = ((arr[0] > 0) && (arr[1] > 0) && (arr[2] >
0));
```

```
    identicalSigns = (identicalSigns || ((arr[0] < 0) && (arr[1] < 0) &&
(arr[2] < 0)));

    if (identicalSigns)
    {
       point_t vertexes[4] = {vertexes_[0], vertexes_[1], vertexes_[2],
vertexes_[3]};
      vertexes_[0] = vertexes[i];
      vertexes_[1] = vertexes[(i + 1) % 4];
      vertexes_[2] = vertexes[(i + 3) % 4];
      vertexes_[3] = vertexes[(i + 2) % 4];
      isConcave = true;
    }
  }
  if (!isConcave)
  {
    throw std::logic_error("concave error");
  }
}

double rebdev::Concave::getArea() const
{
  return getFigureArea(vertexes_, 4);
}

rebdev::rectangle_t rebdev::Concave::getFrameRect() const
{
  return getFrameRectangle(vertexes_, 4);
}

void rebdev::Concave::move(const point_t & point)
{
  move(point.x - vertexes_[2].x, point.y - vertexes_[2].y);
}

void rebdev::Concave::move(double x, double y)
{
  movePoints(x, y, vertexes_, 4);
}

void rebdev::Concave::scale(double k)
{
  scaleFigure(vertexes_, 4, vertexes_[2], k);
}

bool  rebdev::isTriangle(const  point_t  &  f,  const  point_t  &  s,  const
point_t & t)
{
  return (((t.x - f.x) / (s.x - f.x)) != ((t.y - f.y) / (s.y - f.y)));
}
```

./<ROOT>/polygon.hpp

```
#ifndef POLYGON_HPP
#define POLYGON_HPP
```

```
#include <cstddef>
#include "base-types.hpp"
#include "shape.hpp"

namespace rebdev
{
  class Polygon: public Shape
  {
    public:
      Polygon(const point_t * vertexes, size_t numOfVertexes);

      virtual ~Polygon();
      virtual double getArea() const;
      virtual rectangle_t getFrameRect() const;
      virtual void move(const point_t & point);
      virtual void move(double x, double y);
      virtual void scale(double k);
    private:
      point_t * vertexes_;
      size_t numOfVertexes_;

      point_t getPolygonCenter();
  };
}

#endif
```

./<ROOT>/polygon.cpp

```
#include "polygon.hpp"
#include <stdexcept>
#include "figureFunction.hpp"

rebdev::Polygon::Polygon(const point_t * vertexes, size_t
numOfVertexes):
  vertexes_(nullptr),
  numOfVertexes_(numOfVertexes)
{
  if (numOfVertexes < 3)
  {
    throw std::logic_error("polygon error");
  }
  for (size_t i = 0; i < (numOfVertexes - 1); ++i)
  {
    for (size_t j = (i + 1); j < numOfVertexes; ++j)
    {
          if ((vertexes[i].x == vertexes[j].x) && (vertexes[i].y ==
vertexes[j].y))
      {
        throw std::logic_error("polygon error");
      }
    }
  }
```

```
  vertexes_ = new point_t[numOfVertexes_];

  for (size_t i = 0; i < numOfVertexes_; ++i)
  {
    vertexes_[i] = vertexes[i];
  }
}

rebdev::Polygon::~Polygon()
{
  delete[] vertexes_;
}

double rebdev::Polygon::getArea() const
{
  return getFigureArea(vertexes_, numOfVertexes_);
}

rebdev::rectangle_t rebdev::Polygon::getFrameRect() const
{
  return getFrameRectangle(vertexes_, numOfVertexes_);
}

void rebdev::Polygon::move(const point_t & point)
{
  point_t center = getPolygonCenter();
  move(point.x - center.x, point.y - center.y);
}

void rebdev::Polygon::move(double x, double y)
{
  movePoints(x, y, vertexes_, numOfVertexes_);
}

void rebdev::Polygon::scale(double k)
{
  scaleFigure(vertexes_, numOfVertexes_, getPolygonCenter(), k);
}

rebdev::point_t rebdev::Polygon::getPolygonCenter()
{
  point_t center{0.0, 0.0};

  for (size_t i = 0; i < numOfVertexes_; ++i)
  {
    center.x += vertexes_[i].x;
    center.y += vertexes_[i].y;
  }

  center.x /= numOfVertexes_;
  center.y /= numOfVertexes_;

  return center;
}
```

./<ROOT>/figureInputFunction.hpp

```
#ifndef FIGUREINPUTFUNCTION_HPP
#define FIGUREINPUTFUNCTION_HPP
#include <iostream>
#include <cstddef>
#include <string>
#include "base-types.hpp"
#include "shape.hpp"

namespace rebdev
{
  Shape * newFigure(std::istream & input, const std::string & name);
  point_t * inputPoints(std::istream & input, size_t & numOfPoints);
}
#endif
```

./<ROOT>/figureInputFunction.cpp

```
#include "figureInputFunction.hpp"
#include "rectangle.hpp"
#include "concave.hpp"
#include "polygon.hpp"


rebdev::Shape * rebdev::newFigure(std::istream & input, const
std::string & name)
{
  Shape * pointerToFigure = nullptr;
  point_t * points = nullptr;

  size_t numOfPoints = 0;
  points = inputPoints(input, numOfPoints);

  try
  {
    if (name.find("RECTANGLE") != std::string::npos)
    {
      pointerToFigure = new Rectangle(points[0], points[1]);
    }
    else if (name.find("CONCAVE") != std::string::npos)
    {
        pointerToFigure = new Concave(points[0], points[1], points[2],
points[3]);
    }
    else if (name.find("POLYGON") != std::string::npos)
    {
      pointerToFigure = new Polygon(points, numOfPoints);
    }
  }
  catch (const std::logic_error & e)
  {
    delete[] points;
    throw;
  }
```

```
  delete[] points;
  return pointerToFigure;
}

rebdev::point_t * rebdev::inputPoints(std::istream & input, size_t &
numOfPoints)
{
    point_t * points = new point_t[1];
    point_t * bufferArr = nullptr;
    size_t bufferSize = 0;

    while (input >> points[numOfPoints].x >> points[numOfPoints].y)
    {
      if (numOfPoints == bufferSize)
      {
        bufferSize += 10;
        try
        {
          bufferArr = new point_t[bufferSize];
        }
        catch (const std::bad_alloc & e)
        {
          delete[] points;
          throw;
        }

        for (size_t i = 0; i <= numOfPoints; ++i)
        {
          bufferArr[i] = points[i];
        }

        delete[] points;
        points = bufferArr;
        bufferArr = nullptr;
      }

      numOfPoints += 1;
    }

    input.clear();

    try
    {
      bufferArr = new point_t[numOfPoints];
    }
    catch (const std::bad_alloc & e)
    {
      delete[] points;
      throw;
    }

    for (size_t i = 0; i < numOfPoints; ++i)
    {
```

```
      bufferArr[i] = points[i];
    }

    delete[] points;
    points = bufferArr;
    bufferArr = nullptr;

    return points;
}
```

./<ROOT>/outputShapes.hpp

```
#ifndef OUTPUTSHAPES_HPP
#define OUTPUTSHAPES_HPP

#include <iostream>
#include "shape.hpp"
#include "base-types.hpp"

namespace rebdev
{
   void printShapes(const Shape * const * shapes, size_t numOfShapes,
std::ostream & out);
}

#endif
```

./<ROOT>/outputShapes.cpp

```
#include "outputShapes.hpp"

void    printVertexes(const    rebdev::point_t    &    upRight,    const
rebdev::point_t & lowLeft, std::ostream & out)
{
  out << ' ' << lowLeft.x << ' ' << lowLeft.y;
  out << ' ' << upRight.x << ' ' << upRight.y;
}


void   rebdev::printShapes(const   Shape   *   const   *   shapes,   size_t
numOfShapes, std::ostream & out)
{
  double sum = 0;
  for (size_t i = 0; i < numOfShapes; ++i)
  {
    sum += shapes[i]->getArea();
  }

  out << sum;

  for (size_t i = 0; i < numOfShapes; ++i)
  {
    rectangle_t rect = shapes[i]->getFrameRect();
      point_t lowLeft = {rect.pos.x - (rect.width / 2), rect.pos.y -
(rect.height / 2)};
```

```
      point_t upRight = {rect.pos.x + (rect.width / 2), rect.pos.y +
(rect.height / 2)};
    printVertexes(upRight, lowLeft, out);
  }
}
```

./<ROOT>/figureFunction.hpp

```
#ifndef FIGUREFUNCTION_HPP
#define FIGUREFUNCTION_HPP

#include <cstddef>
#include "base-types.hpp"

namespace rebdev
{
     rectangle_t getFrameRectangle(const point_t vertexes[], size_t
numOfVertexes);
    void scaleFigure(point_t vertexes[], size_t numOfVertexes, const
point_t & center, double k);
    void movePoints(double x, double y, point_t * points, size_t
numOfPoint);
  double getFigureArea(const point_t * points, size_t numOfPoint);
}
#endif
```

./<ROOT>/figureFunction.cpp

```
#include "figureFunction.hpp"
#include <algorithm>
#include <stdexcept>

rebdev::rectangle_t rebdev::getFrameRectangle(const point_t vertexes[],
size_t numOfVertexes)
{
  double xMin = vertexes[0].x, xMax = vertexes[0].x;
  double yMin = vertexes[0].y, yMax = vertexes[0].y;
  for (size_t i = 0; i < numOfVertexes; ++i)
  {
    xMin = std::min(xMin, vertexes[i].x);
    yMin = std::min(yMin, vertexes[i].y);
    xMax = std::max(xMax, vertexes[i].x);
    yMax = std::max(yMax, vertexes[i].y);
  }

  return rectangle_t{(xMax - xMin), (yMax - yMin), point_t{(xMax + xMin)
/ 2, (yMax + yMin) / 2}};
}

void rebdev::scaleFigure(point_t vertexes[], size_t numOfVertexes, const
point_t & center, double k)
{
  if (k <= 0)
  {
    throw std::logic_error("Bad koeff");
  }
```

```
  for (size_t i = 0; i < numOfVertexes; ++i)
  {
    vertexes[i].x = (vertexes[i].x - center.x) * k + center.x;
    vertexes[i].y = (vertexes[i].y - center.y) * k + center.y;
  }
}

void rebdev::movePoints(double x, double y, point_t * points, size_t
numOfPoint)
{
  for (size_t i = 0; i < numOfPoint; ++i)
  {
    points[i].x += x;
    points[i].y += y;
  }
}

double rebdev::getFigureArea(const point_t * points, size_t numOfPoint)
{
  double sum = 0;

  for (size_t i = 0; i < (numOfPoint - 1); ++i)
  {
     sum += (points[i].x - points[i + 1].x) * (points[i].y + points[i +
1].y);
  }

  sum += (points[numOfPoint - 1].x - points[0].x) * (points[numOfPoint -
1].y + points[0].y);
  sum /= 2;
  return std::abs(sum);
}
```