Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 3

Тема: Механизмы наследования в C++

Студент: Бирюков В. В.
Группа: 80-207
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:

- Вычисление центра фигуры.
- Вывод в стандартный поток вывода std::cout координат вершин фигуры.
- Вычисление площади фигуры.

Создать программу, которая позволяет:

- Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.
- Сохранять созданные фигуры в динамический массив std::vector<Figure*>.
- Вызывать для всего массива общие функции (1-3 см. выше). Т.е. распечатывать для каждой фигуры в массиве геометрический центр, координаты вершин и площадь.
- Необходимо уметь вычислять общую площадь фигур в массиве.
- Удалять из массива фигуру по индексу.

Вариант 17: треугольник, квадрат, прямоугольник.

2. Описание программы

Для удобства работы с координатами реализован класс Coord, в котором координаты хранятся в виде двух вещественных чисел (double) и который содержит перегруженные операции ввода и вывода в стандартные потоки, вычитания (покоординатно) и умножения (модуль векторного произведения).

Класс Figure является абстрактным и содержит виртуальные методы center, square и print. Также в нем перегружен оператор ввода, который вызывает функцию print. Это позволяет выводит в стандартный поток значение указателя на Figure, который на самом деле является указателем на его наследника, при этом будет вызываться переопределенная в этом наследнике функция print.

Класс Triangle является наследником класса Figure и хранит треугольник в виде координат каждой его вершины. За центр берется центр тяжести треугольника, каждая координата центра - среднее арифметическое соответствующих координат сторон. Площадь

вычисляется как половина векторного произведения двух сторон треугольника. Перегружена операция ввода из стандартного потока. При создании треугольника, проверяется, что его вершины не лежат на одной прямой.

Класс Square является наследником класса Figure и хранит квадрат как координаты левого нижнего угла и длины стороны. Из такого способа легко получить координаты всех вершин квадрата, прибавив длину к первой координате вершины, второй и к обеим. Центр находится прибавлением к каждой координате вершины половины длины, площадь вычисляется как квадрат стороны. Перегружена операция ввода из стандартного потока. При создании квадрата, проверяется, что длина его стороны положительная.

Класс Rectangle реализован аналогично квадрату и хранит координаты левого нижнего угла и длины двух сторон. Все функции и проверки аналогичны, с поправкой на вторую сторону.

3. Набор тестов

Программа получает на вход количество фигур, описание каждой фигуры и индекс для удаления. Перед описанием каждой фигурой вводится символ: t - для треугольника, s - для квадрата, r - для прямоугольника. Для треугольника необходимо ввести 6 чисел - координаты каждой вершины. Для квадрата 3 числа - координаты нижней левой вершины и длины стороны. Для прямоугольника 4 числа - координаты левой нижней вершины и длины сторон.

Тест 1 (треугольник, квадрат и прямоугольник, удаляем квадрат):

```
3
t 0 0 1 0 0 1
s 0 0 1
r 0 0 1 2
2
```

Тест 2 (одинаковые центры, удаляем 4 фигуру):

```
4
s 0 0 4
r -2 0 8 4
s -3 -3 10
t 0 1 2 3 4 2
4
```

Тест 3 (одинаковые площади, удаляем 5 фигуру):

```
5
s 0 0 2
r -5 -5 1 4
t 0 4 2 0 0 0
s 8 9 2
r 2 4 8 0.5
5
```

4. Результаты выполнения тестов

Программа выводит координаты всех вершин, координаты центров и площади для каждой фигуры, суммарную площадь и координаты всех фигур после удаления.

Тест 1 (треугольник, квадрат и прямоугольник, удаляем квадрат):

```
Coordinates:
(0, 0) (1, 0) (0, 1)
(0, 0) (0, 1) (1, 1) (1, 0)
(0, 0) (0, 2) (1, 2) (1, 0)
Centers:
(0.333333, 0.333333)
(0.5, 0.5)
(0.5, 1)
Squares:
0.5
1
2
Total square: 3.5
Deleting figure by index 1
Coordinates again:
(0, 0) (1, 0) (0, 1)
(0, 0) (0, 2) (1, 2) (1, 0)
```

Тест 2 (одинаковые центры):

```
Coordinates:
(0, 0) (0, 4) (4, 4) (4, 0)
(-2, 0) (-2, 4) (6, 4) (6, 0)
(-3, -3) (-3, 7) (7, 7) (7, -3)
(0, 1) (2, 3) (4, 2)
Centers:
(2, 2)
(2, 2)
(2, 2)
(2, 2)
```

```
Squares:
16
32
100
3
Total square: 151
Deleting figure by index 3
Coordinates again:
(0, 0) (0, 4) (4, 4) (4, 0)
(-2, 0) (-2, 4) (6, 4) (6, 0)
(-3, -3) (-3, 7) (7, 7) (7, -3)
```

Тест 3 (одинаковые площади, удаляем 5 фигуру):

```
Coordinates:
(0, 0) (0, 2) (2, 2) (2, 0)
(-5, -5) (-5, -1) (-4, -1) (-4, -5)
(0, 4) (2, 0) (0, 0)
(8, 9) (8, 11) (10, 11) (10, 9)
(2, 4) (2, 4.5) (10, 4.5) (10, 4)
Centers:
(1, 1)
(-4.5, -3)
(0.666667, 1.33333)
(9, 10)
(6, 4.25)
Squares:
4
4
4
4
4
Total square: 20
Deleting figure by index 4
Coordinates again:
(0, 0) (0, 2) (2, 2) (2, 0)
(-5, -5) (-5, -1) (-4, -1) (-4, -5)
(0, 4) (2, 0) (0, 0)
(8, 9) (8, 11) (10, 11) (10, 9)
```

5. Листинг программы

```
#include <iostream>
#include <vector>
#include <cmath>
```

```
class Coord {
public:
  double x;
  double y;

  Coord() : x(0.0), y(0.0) {}
  Coord(double _x, double _y) : x(_x), y(_y) {}
  Coord(const Coord &coord) : x(coord.x), y(coord.y) {}

  friend std::ostream& operator<<(std::ostream &, const Coord &);
  friend std::istream& operator>>(std::istream &, Coord &);
  friend double operator*(const Coord &, const Coord &);
  friend Coord operator-(const Coord &, const Coord &);
};

std::ostream& operator<<(std::ostream &os, const Coord &coord) {
  os << "(" << coord.x << ", " << coord.y << ")";
  return os;
}

std::istream& operator>>(std::istream &is, Coord &coord) {
  is >> coord.x >> coord.y;
  return is;
}

double operator*(const Coord &a, const Coord &b) {
  // модуль векторного произведения
  return fabs(a.x * b.y - a.y * b.x);
}

Coord operator-(const Coord &a, const Coord &b) {
  return Coord(a.x - b.x, a.y - b.y);
}

class Figure {
protected:
  virtual std::ostream& print(std::ostream &) const = 0;

public:
  virtual Coord center() = 0;
  virtual double square() = 0;
  virtual ~Figure() = default;

  friend std::ostream& operator<<(std::ostream &, const Figure &);
};

std::ostream& operator<<(std::ostream &os, const Figure &f) {
```

```
  return f.print(os);
}

class Triangle : public Figure {
private:
  // Координаты каждой вершины
  Coord a;
  Coord b;
  Coord c;

  std::ostream& print(std::ostream &) const override;

public:
  Triangle() : a(), b(), c() {}
  Triangle(Coord _a, Coord _b, Coord _c) : a(_a), b(_b), c(_c) {
    if (square() == 0) {
      throw std::invalid_argument("Invalid triangle parameters");
    }
  }
  Triangle(const Triangle& tr) : a(tr.a), b(tr.b), c(tr.c) {}
  Coord center() override;
  double square() override;
  friend std::istream& operator>>(std::istream &, Triangle &);
};

double Triangle::square() {
  return (b - a) * (c - a) / 2;
}

Coord Triangle::center() {
  return Coord((a.x + b.x + c.x) / 3, (a.y + b.y + c.y) / 3);
}

std::ostream& Triangle::print(std::ostream &os) const {
  os << a << " " << b << " " << c;
  return os;
}

std::istream& operator>>(std::istream &is, Triangle &tr) {
  is >> tr.a >> tr.b >> tr.c;
  if (tr.square() == 0) {
    throw std::invalid_argument("Invalid triangle parameters");
  }
  return is;
}

class Square : public Figure {
private:
```

```
  // Координаты левого нижнего угла и длина стороны
  Coord x;
  double a;

  std::ostream& print(std::ostream &) const override;

public:
  Square() : x(), a(0.0) {}
  Square(Coord _x, double _a) : x(_x), a(_a) {
    if (a <= 0) {
      throw std::invalid_argument("Invalid square parameters");
    }
  }
  Square(const Square& sq) : x(sq.x), a(sq.a) {}
  Coord center() override;
  double square() override;
  friend std::istream& operator>>(std::istream &, Square &);
};

double Square::square() {
  return a * a;
}

Coord Square::center() {
  return Coord(x.x + a/2, x.y + a/2);
}

std::ostream& Square::print(std::ostream &os) const {
  os << x << " " << Coord(x.x, x.y + a) << " "
     << Coord(x.x + a, x.y + a) << " " << Coord(x.x + a, x.y);
  return os;
}

std::istream& operator>>(std::istream &is, Square &sq) {
  is >> sq.x >> sq.a;
  if (sq.a <= 0) {
    throw std::invalid_argument("Invalid square parameters");
  }
  return is;
}

class Rectangle : public Figure {
private:
  // Координаты левого нижнего угла и длины сторон
  Coord x;
  double a;
  double b;
```

```
  std::ostream& print(std::ostream &) const override;

public:
  Rectangle() : x(), a(), b() {}
  Rectangle(Coord _x, double _a, double _b) : x(_x), a(_a), b(_b) {
    if (a <= 0 || b <= 0) {
      throw std::invalid_argument("Invalid rectangle parameters");
    }
  }
  Rectangle(const Rectangle& rect) :
    x(rect.x), a(rect.a), b(rect.b) {}
  Coord center() override;
  double square() override;
  friend std::istream& operator>>(std::istream &, Rectangle &);
};

double Rectangle::square() {
  return a * b;
}

Coord Rectangle::center() {
  return Coord(x.x + a/2, x.y + b/2);
}

std::ostream& Rectangle::print(std::ostream &os) const {
  os << x << " " << Coord(x.x, x.y + b) << " "
     << Coord(x.x + a, x.y + b) << " " << Coord(x.x + a, x.y);
  return os;
}

std::istream& operator>>(std::istream &is, Rectangle &rect) {
  is >> rect.x >> rect.a >> rect.b;
  if (rect.a <= 0 || rect.b <= 0) {
    throw std::invalid_argument("Invalid rectangle parameters");
  }
  return is;
}

int main() {
  Triangle *tr = nullptr;
  Square *sq = nullptr;
  Rectangle *rect = nullptr;
  char s;
  unsigned int del = -1;
  size_t n;

  std::cin >> n;
```

```
std::vector<Figure*> figures(n);

try {
  for (size_t i = 0; i < n; i++) {
    std::cin >> s;
    if (s == 't') {
      tr = new Triangle();
      std::cin >> *tr;
      figures[i] = (Figure *)tr;
    } else if (s == 's') {
      sq = new Square();
      std::cin >> *sq;
      figures[i] = (Figure *)sq;
    } else if (s == 'r') {
      rect = new Rectangle();
      std::cin >> *rect;
      figures[i] = (Figure *)rect;
    }
  }
} catch (const std::invalid_argument &ex) {
  std::cout << ex.what() << '\n';
  return 1;
}

std::cin >> del;
del--;

if (del < 0 || del >= figures.size()) {
  std::cout << "Invalid index for deletion" << '\n';
  return 1;
}

std::cout << "Coordinates:" << '\n';
for (Figure* f : figures) {
  std::cout << *f << '\n';
}

std::cout << "Centers:" << '\n';
for (Figure* f : figures) {
  std::cout << f->center() << '\n';
}

std::cout << "Squares:" << '\n';
double sum = 0;
for (Figure* f : figures) {
  sum += f->square();
  std::cout << f->square() << '\n';
}
```

```
  std::cout << "Total square: " << sum << '\n';

  std::cout << "Deleting figure by index " << del << '\n';

  auto i = figures.begin();
  for (; i != figures.end() && del > 0; ++i, --del);
  figures.erase(i);

  std::cout << "Coordinates again:" << '\n';
  for (Figure* f : figures) {
    std::cout << *f << '\n';
  }

  for (Figure* f : figures) {
    delete f;
  }

}
```

6. Выводы

В ходе лабораторной работы я ознакомился с механизмом наследования в языке C++, а также с созданием абстрактных классов и переопределением виртуальных методов.

Литература

1. Справочник по языку C++ [Электронный ресурс]. URL: https://ru.cppreference.com