Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»
Дисциплина «Объектно-ориентированное программирование»

Лабораторная работа №3

Тема: Наследование, полиморфизм

Студент: Инютин М. А.
Группа: М8О-207Б-19
Преподаватель: Чернышев Л. Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Изучить механизм работы наследования в C++.

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

1. Вычисление геометрического центра фигуры
2. Вывод в стандартный поток вывода std::cout координат вершин фигуры
3. 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. То есть 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 2. Квадрат, прямоугольник, трапеция.

2. Описание программы

Для вычисления геометрического центра фигуры создадим вспомогательный класс Cord с двумя членами X и Y. В абстрактном классе Figure три виртуальные функции: FigureCenter, FigureSquare, FigurePrint (сделать виртуальную перегрузку оператора вывода не получилось). Унаследуем для каждой из фигур класс Figure и реализуем родительские методы. Каждая фигура описана координата левого нижнего угла и длинами сторон (для квадрата всех, для прямоугольника ширина и высота, для трапеции большее и меньшее основание, боковая сторона). В конструктор добавим проверку, что длины сторон неотрицательны (иначе программа выдаст исключение). Реализуем ввод и вывод данных трёх фигур, их сохранение в std::vector<Figure*> и вызов для них общих методов. Затем удалим одну из фигур и снова вызовем для оставшихся общие методы.

3. Набор тестов

Программа принимает на ввод данные для квадрата, прямоугольника и трапеции. Затем считывает индекс элемента, который нужно удалить.

Тест №1
0 0 64
0 0 21 34
0 0 20 10 13
1

Тест №2
-1 -1 3
0 0 4 5
1 1 10 4 5
2

Тест №3
-1 -1 3
0 0 4 5
1 1 10 4 5
3

Тест №4
1000 1000 10
500 500 10 10
0 0 10 10 10
2

4. Результаты выполнения тестов

Программа выводит результат применения общих методов к каждой фигуре, а так же общую площадь фигур. Затем удаляет по индексу один из элементов вектора и снова печатает результат.

Тест №1

```
Square verticies: [(0, 0), (0, 64), (64, 64), (64, 0)]
Center of figure is (32, 32)
Square of figure is 4096
Rectangle verticies: [(0, 0), (0, 34), (21, 34), (21, 0)]
Center of figure is (10.5, 17)
Square of figure is 714
Trapeze verticies: [(0, 0), (5, 12), (15, 12), (20, 0)]
Center of figure is (10, 6)
Square of figure is 180
Total square: 4990
Rectangle verticies: [(0, 0), (0, 34), (21, 34), (21, 0)]
Center of figure is (10.5, 17)
Square of figure is 714
Trapeze verticies: [(0, 0), (5, 12), (15, 12), (20, 0)]
Center of figure is (10, 6)
Square of figure is 180
Total square after erase: 894
```

Тест №2

```
Square verticies: [(-1, -1), (-1, 2), (2, 2), (2, -1)]
Center of figure is (0.5, 0.5)
Square of figure is 9
Rectangle verticies: [(0, 0), (0, 5), (4, 5), (4, 0)]
Center of figure is (2, 2.5)
Square of figure is 20
Trapeze verticies: [(1, 1), (4, 5), (8, 5), (11, 1)]
Center of figure is (6, 3)
Square of figure is 28
Total square: 57
Square verticies: [(-1, -1), (-1, 2), (2, 2), (2, -1)]
Center of figure is (0.5, 0.5)
Square of figure is 9
Trapeze verticies: [(1, 1), (4, 5), (8, 5), (11, 1)]
Center of figure is (6, 3)
Square of figure is 28
Total square after erase: 37
```

Тест №3
Square verticies: [(-1, -1), (-1, 2), (2, 2), (2, -1)]
Center of figure is (0.5, 0.5)
Square of figure is 9
Rectangle verticies: [(0, 0), (0, 5), (4, 5), (4, 0)]
Center of figure is (2, 2.5)
Square of figure is 20
Trapeze verticies: [(1, 1), (4, 5), (8, 5), (11, 1)]
Center of figure is (6, 3)
Square of figure is 28
Total square: 57
Square verticies: [(-1, -1), (-1, 2), (2, 2), (2, -1)]
Center of figure is (0.5, 0.5)
Square of figure is 9
Rectangle verticies: [(0, 0), (0, 5), (4, 5), (4, 0)]
Center of figure is (2, 2.5)
Square of figure is 20
Total square after erase: 29

Тест №4
Square verticies: [(1000, 1000), (1000, 1010), (1010, 1010), (1010, 1000)]
Center of figure is (1005, 1005)
Square of figure is 100
Rectangle verticies: [(500, 500), (500, 510), (510, 510), (510, 500)]
Center of figure is (505, 505)
Square of figure is 100
Trapeze verticies: [(0, 0), (0, 10), (10, 10), (10, 0)]
Center of figure is (5, 5)
Square of figure is 100
Total square: 300
Square verticies: [(1000, 1000), (1000, 1010), (1010, 1010), (1010, 1000)]
Center of figure is (1005, 1005)
Square of figure is 100
Trapeze verticies: [(0, 0), (0, 10), (10, 10), (10, 0)]
Center of figure is (5, 5)
Square of figure is 100
Total square after erase: 200

5. Листинг программы

Программа разбита на файлы: cord.hpp, cord.cpp, figure.hpp, square.hpp, square.cpp, rectangle.hpp, rectangle.cpp, trapeze.hpp, trapeze.cpp, main.cpp.

cord.hpp

```
#ifndef CORD_HPP
#define CORD_HPP

#include <bits/stdc++.h>

struct Cord {
protected:
     long double X, Y;
public:
     Cord() : X(NAN), Y(NAN) {};
     Cord(long double x, long double y) : X(x), Y(y) {};
     ~Cord() {};
     friend std::ostream & operator << (std::ostream & out,
                                        const Cord & crd);
};

#endif /* CORD_HPP */
```

cord.cpp

```
#include "cord.hpp"

std::ostream & operator << (std::ostream & out, const Cord & crd)
{
     out << "(" << crd.X << ", " << crd.Y << ")";
     return out;
}
```

figure.hpp

```
#ifndef FIGURE_HPP
#define FIGURE_HPP

#include "cord.hpp"

class Figure {
public:
     virtual long double FigureSquare() = 0;
     virtual Cord FigureCenter() = 0;
     virtual void FigurePrint() = 0;
};

#endif /* FIGURE_HPP */
```

square.hpp

```
#ifndef SQUARE_HPP
#define SQUARE_HPP

#include "figure.hpp"

class Square : public Figure {
private:
     /* Cords of left bottom corner, side */
     long double X, Y, A;
public:
     Square(const long double & x, const long double & y, const
long double & a);
     long double FigureSquare() override;
     Cord FigureCenter() override;
     void FigurePrint() override;
     friend std::ostream & operator << (std::ostream & out, const
Square & sq);
};

#endif /* SQUARE_HPP */
```

square.cpp

```
#include "square.hpp"

Square::Square(const long double & x, const long double & y,
                    const long double & a) : X(x), Y(y), A(a) {
     if (A < 0.0) {
          throw std::invalid_argument("Invalid square
parameters!");
     }
}

long double Square::FigureSquare() {
     return A * A;
}

Cord Square::FigureCenter() {
     return Cord(X + A / 2.0, Y + A / 2.0);
}

void Square::FigurePrint() {
     std::cout << *this << std::endl;
}

std::ostream & operator << (std::ostream & out, const Square &
square) {
     out << "Square verticies: [";
     out << Cord(square.X, square.Y) << ", ";
     out << Cord(square.X, square.Y + square.A) << ", ";
     out << Cord(square.X + square.A, square.Y + square.A) << ", ";
     out << Cord(square.X + square.A, square.Y);
     out << "]";
     return out;
}
```

rectangle.hpp

```
#ifndef RECTANGLE_HPP
#define RECTANGLE_HPP

#include "figure.hpp"

class Rectangle : public Figure {
private:
     /* Cords of left bottom corner, width and height */
     long double X, Y, A, B;
public:
     Rectangle(const long double & x, const long double & y, const
long double & a, const long double & b);
     long double FigureSquare() override;
     Cord FigureCenter() override;
     void FigurePrint() override;
     friend std::ostream & operator << (std::ostream & out, const
Rectangle & rect);
};

#endif /* RECTANGLE_HPP */
```

rectangle.cpp

```
#include "rectangle.hpp"

Rectangle::Rectangle(const long double & x, const long double & y,
const long double & a, const long double & b) : X(x), Y(y), A(a),
B(b) {
     if (A < 0.0 or B < 0.0) {
          throw std::invalid_argument("Invalid rectangle
parameters!");
     }
};

long double Rectangle::FigureSquare() {
     return A * B;
}

Cord Rectangle::FigureCenter() {
     return Cord(X + A / 2.0, Y + B / 2.0);
}

void Rectangle::FigurePrint() {
     std::cout << *this << std::endl;
}

std::ostream & operator << (std::ostream & out, const Rectangle &
rectangle) {
     out << "Rectangle verticies: [";
     out << Cord(rectangle.X, rectangle.Y) << ", ";
     out << Cord(rectangle.X, rectangle.Y + rectangle.B) << ", ";
     out << Cord(rectangle.X + rectangle.A, rectangle.Y +
rectangle.B) << ", ";
     out << Cord(rectangle.X + rectangle.A, rectangle.Y);
     out << "]";
     return out;
}
```

trapeze.hpp

```
#ifndef TRAPEZE_HPP
#define TRAPEZE_HPP

#include "figure.hpp"

class Trapeze : public Figure {
private:
     /* Cords of left bottom corner, larger and smaller base, side
*/
     long double X, Y, A, B, C;
public:
     Trapeze(const long double & x, const long double & y, const
long double & a, const long double & b, const long double & c);
     long double FigureSquare() override;
     Cord FigureCenter() override;
     void FigurePrint() override;
     friend std::ostream & operator << (std::ostream & out, const
Trapeze & trapez);
};

#endif /* TRAPEZE_HPP */
```

trapeze.cpp

```
#include "trapeze.hpp"

Trapeze::Trapeze(const long double & x, const long double & y,
const long double & a, const long double & b, const long double &
c) : X(x), Y(y), A(a), B(b), C(c) {
     if (A < 0.0 or B < 0.0 or C < 0.0) {
          throw std::invalid_argument("Invalid trapeze
parameters!");
     }
     if (B > A) {
          std::swap(A, B);
     }
};

long double Trapeze::FigureSquare() {
     long double diff = (A - B) / 2.0;
     long double height = std::sqrt(C * C - diff * diff);
     return height * (A + B) / 2.0;
}

Cord Trapeze::FigureCenter() {
     long double diff = (A - B) / 2.0;
     long double height = std::sqrt(C * C - diff * diff);
     return Cord(X + A / 2.0, Y + height / 2.0);
}

void Trapeze::FigurePrint() {
     std::cout << *this << std::endl;
}

std::ostream & operator << (std::ostream & out, const Trapeze &
trapeze) {
     long double diff = (trapeze.A - trapeze.B) / 2.0;
     long double height = std::sqrt(trapeze.C * trapeze.C - diff *
diff);
     out << "Trapeze verticies: [";
     out << Cord(trapeze.X, trapeze.Y) << ", ";
     out << Cord(trapeze.X + diff, trapeze.Y + height) << ", ";
     out << Cord(trapeze.X + trapeze.A - diff, trapeze.Y + height)
<< ", ";
     out << Cord(trapeze.X + trapeze.A, trapeze.Y);
     out << "]";
     return out;
}
```

main.cpp

```
#include "square.hpp"
#include "rectangle.hpp"
#include "trapeze.hpp"

/*
 * Инютин М А М8О-207Б-19
 * Разработать классы согласно варианту задания,
 * классы должны наследоваться от базового класса Figure.
 * Фигуры являются фигурами вращения.
 * Все классы должны поддерживать набор общих методов:
 * - Вычисление геометрического центра фигуры
 * - Вывод в стандартный поток std::cout координат вершин фигуры
 * - Вычисление площади фигуры
 * Создать программу, которая позволяет:
 * - Вводить в стандартный поток std::cin фигуры
 * - Сохранять заданные фигуры в вектор std::vector<Figure*>
 * - Вызывать для всего массива общие функции
 * - Необходимо уметь вычислять общую площадь фигур в массиве.
 * - Удалять из массива фигуру по индексу
 * Квадрат, прямоугольник, трапеция.
 */

long double TotalSquare(std::vector<Figure*> & figures) {
     long double res = 0.0;
     for (auto fig : figures) {
          res = res + fig->FigureSquare();
     }
     return res;
}

signed main() {
     long double x = 0.0, y = 0.0, a = -1.0, b = -1.0, c = -1.0;
     std::vector<Figure*> vec;

     /* Input square */
     std::cout << "Input square as follows: x y a" << std::endl;
     std::cout << "x, y is a left bottom corner cords" <<
std::endl;
     std::cout << "a is square side" << std::endl;
     std::cin >> x >> y >> a;
     Square * square = NULL;
     try {
          square = new Square(x, y, a);
     } catch (std::invalid_argument & ex) {
          std::cout << ex.what() << std::endl;
          return 1;
     }
     x = 0.0, y = 0.0, a = -1.0;
     vec.push_back(square);
```

```
	/* Input rectangle */
	std::cout << "Input rectangle as follows: x y a b" <<
std::endl;
	std::cout << "x, y is a left bottom corner cords" <<
std::endl;
	std::cout << "a and b are width and heigth" << std::endl;
	std::cin >> x >> y >> a >> b;
	Rectangle * rectangle = NULL;
	try {
		rectangle = new Rectangle(x, y, a, b);
	} catch (std::invalid_argument & ex) {
		std::cout << ex.what() << std::endl;
		return 1;
	}
	x = 0.0, y = 0.0, a = -1.0, b = -1.0;
	vec.push_back(rectangle);

	/* Input trapeze */
	std::cout << "Input trapeze as follows: x y a b c" <<
std::endl;
	std::cout << "x, y is a left bottom corner cords" <<
std::endl;
	std::cout << "a, b and c are larger, smaller base and side" <<
std::endl;
	std::cin >> x >> y >> a >> b >> c;
	Trapeze * trapeze = NULL;
	try {
		trapeze = new Trapeze(x, y, a, b, c);
	} catch (std::invalid_argument & ex) {
		std::cout << ex.what() << std::endl;
		return 1;
	}
	x = 0.0, y = 0.0, a = -1.0, b = -1.0, c = -1.0;
	vec.push_back(trapeze);
	for (auto fig : vec) {
		fig->FigurePrint();
		std::cout << "Center of figure is " << fig-
>FigureCenter() << std::endl;
		std::cout << "Square of figure is " << fig-
>FigureSquare() << std::endl;
	}
	std::cout << "Total square: " << TotalSquare(vec) <<
std::endl;
```

```
     int i = 0;
     std::cout << "Input index to remove figure" << std::endl;
     std::cin >> i;
     for (auto fig = vec.begin(); fig != vec.end() and i > 0; ++fig) {
          --i;
          if (i == 0) {
               vec.erase(fig);
          }
     }
     for (auto fig : vec) {
          fig->FigurePrint();
          std::cout << "Center of figure is " << fig->FigureCenter() << std::endl;
          std::cout << "Square of figure is " << fig->FigureSquare() << std::endl;
     }
     std::cout << "Total square after erase: " << TotalSquare(vec) << std::endl;
     return 0;
}
```

6. Выводы

Я научился использовать наследование на C++. Во время выполнения работы возникли проблемы с общим методов вывода: я пытался перегрузить оператор вывода для родительского класса, но оказалось, что делать виртуальные перегрузки нельзя. Так же было проблемой писать названия методов одинаково, тут на помощь пришёл override, который не компилировал программу, если бы я писал метод класса, а не реализацию родительского. Я узнал больше про исключения, в частности std::invalid_argument, которые я использовал в конструкторах дочерних классов для проверки корректности данных. Наследование очень полезно для описания классов со схожими свойствами.

7. Список литературы

1. Наследование в C++: beginner, intermediate, advacned — Habr
   URL: https://habr.com/ru/post/445948/ (дата обращения: 07.10.2020)