Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Лабораторной работе № 03
по курсу «Объектно-ориентированное программирование»

Тема:
«Наследование, полиморфизм»

Студент:	Пшеницын А. А.
Группа:	М80-208Б-18
Преподаватель:	Журавлев А. А.
Вариант:	17
Оценка:	
Дата:	

Москва
2019

Цель:

- Изучение механизмов работы наследования в C++

Задание(Вариант 17)

Разработать классы треугольник, квадрат, прямоугольник, которые 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 вывода std::cout координат вершин фигуры
- вычисление площади фигуры

Составить 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
- удалять из массива фигуру по индексу

Код программы

point.h

```
#ifndef _POINT_H_
#define _POINT_H_

typedef struct{
   double x, y;
}point;

double scalar_mult(point top1_end, point top_begin, point top2_end);
double segment_length(point top1, point top2);

#endif
```

point.cpp

```
#include<iostream>
#include<vector>
#include<cmath>
#include<string.h>


#include "point.h"

double scalar_mult(point top1_end, point top_begin, point top2_end){
   return (top_begin.x - top1_end.x)*(top_begin.x - top2_end.x) + (top_begin.y -
top1_end.y)*(top_begin.y - top2_end.y);
}

double segment_length(point top1, point top2){
   return sqrt(pow(top1.x - top2.x, 2) + pow(top1.y - top2.y, 2));
}
```

figure.h

```
#ifndef _FIGURE_H_
```

```
#define _FIGURE_H_

#include "point.h"

typedef struct{
   virtual bool correct() const = 0;
   virtual point center() const = 0;
   virtual double square() const = 0;
   virtual void print(std::ostream& os) const = 0;
}fig;

#endif
```

trigon.h

```
#ifndef _TRIGON_H_
#define _TRIGON_H_

#include "figure.h"

struct trigon : public fig{
private:
   point l, r, top;
public:
   trigon(std::istream& is){
      is >> l.x >> l.y >> r.x >> r.y >> top.x >> top.y;
   }

   bool correct() const override;
   point center() const override;
   double square() const override;
   void print(std::ostream& os) const override;
};

#endif
```

trigon.cpp

```
#include<iostream>
#include<vector>
#include<cmath>
#include<string.h>

#include "trigon.h"
#include "figure.h"

bool trigon::correct() const{
   if((top.x - l.x) * (r.y - l.y) == (top.y - l.y) * (r.x - l.x)){
      return false;
   }
   return true;
}
```

```
double trigon::square() const{
   double mult1 = (r.x - l.x) * (top.y - l.y);
   double mult2 = (top.x - l.x) * (r.y - l.y);
   return 0.5 * fabs(mult1 - mult2);
}


point trigon::center() const{
   double mid1 = (r.x + l.x + top.x) / 3;
   double mid2 = (r.y + l.y + top.y) / 3;
   return point{mid1, mid2};
}

void trigon::print(std::ostream& os) const{
   os << "trigon: ";
   os << "(" << l.x << ", " << l.y << ")" << " ";
   os << "(" << r.x << ", " << r.y << ")" << " ";
   os << "(" << top.x << ", " << top.y << ")" << '\n';
}
```

rectangle.h

```
#ifndef _RECTANGLE_H_
#define _RECTANGLE_H_

#include "figure.h"

struct rectangle : public fig{
private:
   point top1, top2, top3, top4;
public:
   rectangle(std::istream& is){
      is >> top1.x >> top1.y >> top2.x >> top2.y >> top3.x >> top3.y >> top4.x >> top4.y;
   }

   bool correct() const override;
   point center() const override;
   double square() const override;
   void print(std::ostream& os) const override;
};

#endif
```

rectangle.cpp

```
#include<iostream>
#include<vector>
#include<cmath>
#include<string.h>

#include "rectangle.h"
```

```
#include "figure.h"

bool rectangle::correct() const{
    double scalar1 = scalar_mult(top2, top1, top3);
    double scalar2 = scalar_mult(top2, top1, top4);
    double scalar3 = scalar_mult(top3, top1, top4);
    double scalar01, scalar02, scalar03;
    if(scalar1 == 0){
        scalar01 = scalar_mult(top4, top2, top1);
        scalar02 = scalar_mult(top2, top4, top3);
        scalar03 = scalar_mult(top1, top3, top4);
        if(scalar01 == 0 && scalar02 == 0 && scalar03 == 0){
            return true;
        }
        return false;
    } else if(scalar2 == 0){
        scalar01 = scalar_mult(top1, top2, top3);
        scalar02 = scalar_mult(top1, top4, top3);
        scalar03 = scalar_mult(top2, top3, top4);
        if(scalar01 == 0 && scalar02 == 0 && scalar03 == 0){
            return true;
        }
        return false;
    } else if(scalar3 == 0){
        scalar01 = scalar_mult(top3, top2, top4);
        scalar02 = scalar_mult(top1, top4, top2);
        scalar03 = scalar_mult(top2, top3, top1);
        if(scalar01 == 0 && scalar02 == 0 && scalar03 == 0){
            return true;
        }
        return false;
    } else {
        return  false;
    }
}

void rectangle::print(std::ostream& os) const{
    os << "rectangle: ";
    os << "(" << top1.x << ", " << top1.y << ")" << " ";
    os << "(" << top2.x << ", " << top2.y << ")" << " ";
    os << "(" << top3.x << ", " << top3.y << ")" << " ";
    os << "(" << top4.x << ", " << top4.y << ")" << " ";
    os << '\n';
}

double rectangle::square() const{
    double scalar1 = scalar_mult(top2, top1, top3);
    double scalar2 = scalar_mult(top2, top1, top4);
    double scalar3 = scalar_mult(top3, top1, top4);
    double mid1, mid2;
    if(scalar1 == 0){
        mid1 = segment_length(top1, top2);
```

```
        mid2 = segment_length(top1, top3);
    } else if(scalar2 == 0){
        mid1 = segment_length(top1, top2);
        mid2 = segment_length(top1, top4);
    } else if(scalar3 == 0){
        mid1 = segment_length(top1, top3);
        mid2 = segment_length(top1, top4);
    }
    return mid1 * mid2;
}


point rectangle::center() const{
    double scalar1 = scalar_mult(top2, top1, top3);
    double scalar2 = scalar_mult(top2, top1, top4);
    double scalar3 = scalar_mult(top3, top1, top4);
    double midx, midy;
    if(scalar1 == 0){
        midx = (top3.x + top2.x) * 0.5;
        midy = (top3.y + top2.y) * 0.5;
    } else if(scalar2 == 0){
        midx = (top4.x + top2.x) * 0.5;
        midy = (top4.y + top2.y) * 0.5;
    } else if(scalar3 == 0){
        midx = (top3.x + top4.x) * 0.5;
        midy = (top3.y + top4.y) * 0.5;
    }
    return point{midx, midy};
}
```

quadrate.h

```
#ifndef _QUADRATE_H_
#define _QUADRATE_H_

#include "figure.h"

struct quadrate : public fig{
private:
    point top1, top2, top3, top4;
public:
    quadrate(std::istream& is){
        is >> top1.x >> top1.y >> top2.x >> top2.y >> top3.x >> top3.y >> top4.x >> top4.y;
    }

    bool correct() const override;
    point center() const override;
    double square() const override;
    void print(std::ostream& os) const override;
};

#endif
```

quadrate.cpp

```
#include<iostream>
#include<vector>
#include<cmath>
#include<string.h>

#include "quadrate.h"
#include "figure.h"

bool quadrate::correct() const{
   double scalar1 = scalar_mult(top2, top1, top3);
   double scalar2 = scalar_mult(top2, top1, top4);
   double scalar3 = scalar_mult(top3, top1, top4);
   double scalar01, scalar02, scalar03, scalar_diag;
   if(scalar1 == 0){
      scalar01 = scalar_mult(top4, top2, top1);
      scalar02 = scalar_mult(top2, top4, top3);
      scalar03 = scalar_mult(top1, top3, top4);
      scalar_diag = (top3.x - top2.x)*(top4.x - top1.x) + (top3.y - top2.y)*(top4.y - top1.y);
      if(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0){
         return true;
      }
      return false;
   } else if(scalar2 == 0){
      scalar01 = scalar_mult(top1, top2, top3);
      scalar02 = scalar_mult(top1, top4, top3);
      scalar03 = scalar_mult(top2, top3, top4);
      scalar_diag = (top4.x - top2.x)*(top3.x - top1.x) + (top4.y - top2.y)*(top3.y - top1.y);
      if(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0){
         return true;
      }
      return false;
   } else if(scalar3 == 0){
      scalar01 = scalar_mult(top3, top2, top4);
      scalar02 = scalar_mult(top1, top4, top2);
      scalar03 = scalar_mult(top2, top3, top1);
      scalar_diag = (top4.x - top3.x)*(top2.x - top1.x) + (top4.y - top3.y)*(top2.y - top1.y);
      if(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0){
         return true;
      }
      return false;
   } else {
      return  false;
   }
}

void quadrate::print(std::ostream& os) const{
   os << "quadrate: ";
   os << "(" << top1.x << ", " << top1.y << ")" << " ";
   os << "(" << top2.x << ", " << top2.y << ")" << " ";
   os << "(" << top3.x << ", " << top3.y << ")" << " ";
```

```
    os << "(" << top4.x << ", " << top4.y << ")" << " ";
    os << '\n';
}

double quadrate::square() const{
    double scalar1 = scalar_mult(top2, top1, top3);
    double scalar2 = scalar_mult(top2, top1, top4);
    double scalar3 = scalar_mult(top3, top1, top4);
    double mid;
    if(scalar1 == 0){
        mid = segment_length(top1, top2);
    } else if(scalar2 == 0){
        mid = segment_length(top1, top2);
    } else if(scalar3 == 0){
        mid = segment_length(top1, top3);
    }
    return mid * mid;
}


point quadrate::center() const{
    double scalar1 = scalar_mult(top2, top1, top3);
    double scalar2 = scalar_mult(top2, top1, top4);
    double scalar3 = scalar_mult(top3, top1, top4);
    double midx, midy;
    if(scalar1 == 0){
        midx = (top3.x + top2.x) * 0.5;
        midy = (top3.y + top2.y) * 0.5;
    } else if(scalar2 == 0){
        midx = (top4.x + top2.x) * 0.5;
        midy = (top4.y + top2.y) * 0.5;
    } else if(scalar3 == 0){
        midx = (top3.x + top4.x) * 0.5;
        midy = (top3.y + top4.y) * 0.5;
    }
    return point{midx, midy};
}
```

main.cpp

```
#include<iostream>
#include<vector>
#include<cmath>
#include<string.h>

#include "figure.h"
#include "trigon.h"
#include "rectangle.h"
#include "quadrate.h"


void print_ERROR(int error_code){
```

```
    if(error_code == 1){
        std::cout << "Incorrect command\n";
    }
    else if(error_code == 2){
        std::cout << "incorrect coordinates for a figure\n";
    }
    else{
        std::cout << "There is no item with the given index\n";
    }
    char c;
    while(c != '\n' && c != EOF){
        c = getchar();
    }
}

int main(){
    std::vector<fig*> figs;
    for( ; ; ){
        std::cout << ">";
        char com1[40];
        std::cin >> com1;

        if(strcmp(com1, "add") == 0){
            char com2[40];
            std::cin >> com2;
            fig* new_fig;
            if(strcmp(com2, "trigon") == 0){
                new_fig = new trigon(std::cin);
                if(!new_fig -> correct()){
                    print_ERROR(2);
                    delete new_fig;
                    continue;
                }
            } else if(strcmp(com2, "quadrate") == 0){
                new_fig = new quadrate(std::cin);
                if(!new_fig -> correct()){
                    print_ERROR(2);
                    delete new_fig;
                    continue;
                }
            } else if(strcmp(com2, "rectangle") == 0){
                new_fig = new rectangle(std::cin);
                if(!new_fig -> correct()){
                    print_ERROR(2);
                    delete new_fig;
                    continue;
                }
            } else{
                print_ERROR(1);
            }
            figs.push_back(new_fig);
```

```
        } else if(strcmp(com1, "print") == 0){
            char com2[40];
            std::cin >> com2;
            if(strcmp(com2, "tops") == 0){
                for(fig* cur_fig: figs){
                    cur_fig -> print(std::cout);
                }
            }
            else if(strcmp(com2, "square") == 0){
                for(fig* cur_fig: figs){
                    std::cout << cur_fig -> square() << "\n";
                }
            }
            else if(strcmp(com2, "center") == 0){
                for(fig* cur_fig: figs){
                    point tmp = cur_fig -> center();
                    std::cout << "(" << tmp.x << ", " << tmp.y << ")\n";
                }
            }
            else{
                std::cout << "Incorrect command\n";
            }
        } else if(strcmp(com1, "delete") == 0){
            int id;
            std::cin >> id;
            if(id >= figs.size()){
                print_ERROR(3);
                continue;
            }
            delete figs[id];
            figs.erase(figs.begin() + id);
        } else if(strcmp(com1, "exit") == 0){
            break;
        } else {
            print_ERROR(1);
        }
    }
    for(size_t i = 0; i < figs.size(); ++i){
        delete figs[i];
    }
}
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 3.5)

project(oop_exercise_03)

add_executable(oop_exercise_03
            main.cpp
            rectangle.cpp
            point.cpp
```

```
        trigon.cpp
        quadrate.cpp
)

set_property(TARGET oop_exercise_03 PROPERTY CXX_STANDARD 11)
```

Ссылка на репозиторий на GitHub

https://github.com/AlexPshen/oop_exercise_03.git

Набор тестов

est_01.txt

```
add trigon 1 1 5 1 3 2
add trigon 1 1 1 5 2 1
print square
print tops
print center
exit
```

test_02.txt

```
add quadrate
1 1 3 3 1 3 3 1
add quadrate
2 0 1 1 2 2 3 1
print square
print tops
print center
exit
```

test_03.txt

```
add rectangle
1 1 3 1 3 4 1 4
add rectangle
1 1 2 0 3 3 4 2
print square
print tops
print center
>exit
```

test_04.txt

```
add trigon
1 1 5 1 3 3
add quadrate
1 1 4 1 4 4 1 4
add rectangle
1 1 1 5 2 1 2 5
print square
print tops
print center
>exit
```

Результаты

test_01.txt

```
>add trigon 1 1 5 1 3 2
>add trigon 1 1 1 5 2 1
>print square
2
2
>print tops
trigon: (1, 1) (5, 1) (3, 2)
trigon: (1, 1) (1, 5) (2, 1)
>print center
(3, 1.33333)
(1.33333, 2.33333)
>exit
```

test_02.txt

```
>add quadrate
1 1 3 3 1 3 3 1
>add quadrate
2 0 1 1 2 2 3 1
>print square
4
2
>print tops
quadrate: (1, 1) (3, 3) (1, 3) (3, 1)
quadrate: (2, 0) (1, 1) (2, 2) (3, 1)
>print center
(2, 2)
(2, 1)
>exit
```

test_03.txt

```
>add rectangle
1 1 3 1 3 4 1 4
>add rectangle
1 1 2 0 3 3 4 2
>print square
6
4
>print tops
rectangle: (1, 1) (3, 1) (3, 4) (1, 4)
rectangle: (1, 1) (2, 0) (3, 3) (4, 2)
>print center
(2, 2.5)
(2.5, 1.5)
>exit
```

test_04.txt

```
>add trigon
```

1 1 5 1 3 3
>add quadrate
1 1 4 1 4 4 1 4
>add rectangle
1 1 1 5 2 1 2 5
>print square
4
9
4
>print tops
trigon: (1, 1) (5, 1) (3, 3)
quadrate: (1, 1) (4, 1) (4, 4) (1, 4)
rectangle: (1, 1) (1, 5) (2, 1) (2, 5)
>print center
(3, 1.66667)
(2.5, 2.5)
(1.5, 3)
>exit

Объяснение работы программы

На ввод подаются команда com1. Если com1 является:

- add, то вводим вторую команду com2, которая может быть trigon, rectangle, quadrate. Данная команда добавляет соответственно треугольник, прямоугольник, квадрат.
- print, то вводим вторую команду com2, которая может быть tops, center, square. Данная команда печатает соответственно вершины, центр, площадь фигур.
- delete, тогда вводим целое число id и удаляем фигуру по данному индексу
- exit, тогда выходим из программы

bool correct() const — проверка корректности вводимой фигуры
point center() const — вывод центра фигуры
double square() const — вывод площади фигуры
void print(std::ostream& os) const — вывод вершин фигуры

Вывод

В данной лабораторной работе были рассмотрены механизмы работы с наследованием в C++. Наследование позволяет избежать дублирования лишнего кода при написании классов.