Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Отчёт по Лабораторной работе №7
“Наследование, Полиморфизм“
по курсу “Проектирование структуры классов“
III Семестр

Студент:	Катермин В.С.
Группа:	М8О-208Б-18
Преподаватель:	Журавлёв А.А.
Оценка:	
Дата:	**09.12.19**

1. **Тема**: Проектирование, Структуры классов в C++.
2. **Код программы:**

vertex.h

```
#ifndef D_VERTEX_H
#define D_VERTEX_H

struct vertex {
   int32_t x, y;

};

#endif //D_VERTEX_H
```

figure.h

```
#ifndef D_FIGURE_H
#define D_FIGURE_H

#include <iostream>
#include <memory>

#include "sdl.h"
#include "imgui.h"
#include "vertex.h"

struct figure {
   virtual void render(const sdl::renderer& renderer, const int32_t r, const int32_t g, const int32_t b) const = 0;
   virtual void save(std::ostream& os) const = 0;
   virtual ~figure() = default;
};

double distance(int x1, int y1, int x2, int y2) {
   return sqrt(pow((x1 - x2),2)+ pow((y1 - y2), 2));
}


double distance(vertex a, vertex b) {
   return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));
}

#endif //D_FIGURE_H
```

square.h

```
#ifndef D_SQUARE_H
#define D_SQUARE_H

#include "figure.h"

struct square : figure {
   square(const std::array<vertex, 4>& vertices) : vertices_(vertices) {}

   void render(const sdl::renderer& renderer, const int32_t r, const int32_t g, const int32_t b) const override {
      renderer.set_color(r, g, b);
      for (int32_t i = 0; i < 4; ++i) {
         renderer.draw_line(vertices_[i].x, vertices_[i].y,
            vertices_[(i + 1) % 4].x, vertices_[(i + 1) % 4].y);
      }
   }

   void save(std::ostream& os) const override {
      os << "square\n";
      for (int32_t i = 0; i < 4; ++i) {
```

```
            os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
        }
    }

private:
    std::array<vertex, 4> vertices_;

};
#endif //D_SQUARE_H
```

rectangle.h

```
#ifndef D_RECTANGLE_H
#define D_RECTANGLE_H

#include "figure.h"

struct rectangle : figure {
    rectangle(const std::array<vertex, 4>& vertices) : vertices_(vertices) {}

    void render(const sdl::renderer& renderer, const int32_t r, const int32_t g, const int32_t b) const override {
        renderer.set_color(r, g, b);
        for (int32_t i = 0; i < 4; ++i) {
            renderer.draw_line(vertices_[i].x, vertices_[i].y,
                vertices_[(i + 1) % 4].x, vertices_[(i + 1) % 4].y);
        }
    }

    void save(std::ostream& os) const override {
        os << "rectangle\n";
        for (int32_t i = 0; i < 4; ++i) {
            os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
        }
    }

private:
    std::array<vertex, 4> vertices_;

};
#endif //D_RECTANGLE_H
```

trapezoid.h

```
#ifndef D_TRAPEZOID_H
#define D_TRAPEZOID_H

#include "figure.h"

struct trapezoid : figure {
    trapezoid(const std::array<vertex, 4>& vertices) : vertices_(vertices) {}

    void render(const sdl::renderer& renderer, const int32_t r, const int32_t g, const int32_t b) const override {
        renderer.set_color(r, g, b);
        for (int32_t i = 0; i < 4; ++i) {
            renderer.draw_line(vertices_[i].x, vertices_[i].y,
                vertices_[(i + 1) % 4].x, vertices_[(i + 1) % 4].y);
        }
    }

    void save(std::ostream& os) const override {
        os << "trapezoid\n";
        for (int32_t i = 0; i < 4; ++i) {
            os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
        }
```

```
    }

private:
    std::array<vertex, 4> vertices_;

};
#endif //D_TRAPEZOID_H
```

painter.h

```
#ifndef D_PAINTER_H
#define D_PAINTER_H

#include <array>
#include <fstream>
#include <memory>
#include <vector>
#include <cmath>

#include "sdl.h"
#include "imgui.h"
#include "figure.h"
#include "triangle.h"
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"
#include "undoer.h"

struct builder {
    virtual std::unique_ptr<figure> add_vertex(const vertex& v) = 0; // Добавление новой вершины в фигуру

    virtual ~builder() = default; // Деструктор (Не нужен, но должен быть)

};

struct triangle_builder : builder {
    std::unique_ptr<figure> add_vertex(const vertex& v) {
        vertices_[n_] = v;
        n_ += 1;
        if (n_ != 3) {
            return nullptr;
        }
        ul.push(ul_add, nullptr);
        return std::make_unique<triangle>(vertices_);
    }

private:
    int32_t n_ = 0;
    std::array<vertex, 3> vertices_; // вершины фигуры

};

struct square_builder : builder {
    std::unique_ptr<figure> add_vertex(const vertex& v) {
        if (n_ == 2) {
            int32_t vx = vertices_[1].x - vertices_[0].x;
            int32_t vy = vertices_[1].y - vertices_[0].y;
            int32_t D = (v.x - vertices_[0].x) * vy - (v.y - vertices_[0].y) * vx;
            if (D < 0) {
                vertices_[n_] = vertex{ vertices_[1].x - vy, vertices_[1].y + vx };
                n_ += 1;
                vertices_[n_] = vertex{ vertices_[0].x - vy, vertices_[0].y + vx };
                n_ += 1;
            }
```

```
            else {
                vertices_[n_] = vertex{ vertices_[1].x + vy, vertices_[1].y - vx };
                n_ += 1;
                vertices_[n_] = vertex{ vertices_[0].x + vy, vertices_[0].y - vx };
                n_ += 1;
            }
        }
        else {
            vertices_[n_] = v;
            n_ += 1;
        }
        if (n_ != 4) {
            return nullptr;
        }
        ul.push(ul_add, nullptr);
        return std::make_unique<square>(vertices_);
    }

private:
    int32_t n_ = 0;
    std::array<vertex, 4> vertices_; // вершины фигуры

};

struct rectangle_builder : builder {
    std::unique_ptr<figure> add_vertex(const vertex& v) {
        if (n_ == 2) {
            int32_t vx1 = vertices_[1].x - vertices_[0].x;
            int32_t vy1 = vertices_[1].y - vertices_[0].y;
            int32_t px = ((vx1 * vy1 * (v.y - vertices_[0].y) + vertices_[0].x * pow(vy1, 2) + v.x * pow(vx1, 2)) /
(pow(vy1, 2) + pow(vx1, 2)));
            int32_t py = (vy1 * (px - vertices_[0].x)) / (vx1)+vertices_[0].y;
            int32_t vx2 = v.x - px;
            int32_t vy2 = v.y - py;

            vertices_[n_] = vertex{ vertices_[1].x + vx2, vertices_[1].y + vy2 };
            n_ += 1;
            vertices_[n_] = vertex{ vertices_[0].x + vx2, vertices_[0].y + vy2 };
            n_ += 1;
        } else {
            vertices_[n_] = v;
            n_ += 1;
        }
        if (n_ != 4) {
            return nullptr;
        }
        ul.push(ul_add, nullptr);
        return std::make_unique<rectangle>(vertices_);
    }

private:
    int32_t n_ = 0;
    std::array<vertex, 4> vertices_; // вершины фигуры

};

struct trapezoid_builder : builder {
    std::unique_ptr<figure> add_vertex(const vertex& v) {
        vertices_[n_] = v;
        n_ += 1;
        if (n_ != 4) {
            return nullptr;
        }
        ul.push(ul_add, nullptr);
```

```
        return std::make_unique<trapezoid>(vertices_);
    }

private:
    int32_t n_ = 0;
    std::array<vertex, 4> vertices_; // вершины фигуры

};


#endif //D_PAINTER_H
```

loader.h

```
#ifndef D_LOADER_H
#define D_LOADER_H

#include<vector>
#include<memory>

#include "figure.h"
#include "undoer.h"

struct loader {
    std::vector<std::unique_ptr<figure>> load(std::ifstream& is) {
        std::string figure_name;
        std::vector<std::unique_ptr<figure>> figures;
        while (is >> figure_name) {
            vertex v;
            if (figure_name == std::string("triangle")) {
                std::array<vertex, 3> vertices;
                for (int32_t i = 0; i < 3; ++i) {
                    is >> v.x >> v.y;
                    vertices[i] = v;
                }
                figures.emplace_back(std::make_unique<triangle>(vertices));
            }
            if (figure_name == std::string("square")) {
                std::array<vertex, 4> vertices;
                for (int32_t i = 0; i < 4; ++i) {
                    is >> v.x >> v.y;
                    vertices[i] = v;
                }
                figures.emplace_back(std::make_unique<square>(vertices));
            }
            else if (figure_name == std::string("rectangle")) {
                std::array<vertex, 4> vertices;
                for (int32_t i = 0; i < 4; ++i) {
                    is >> v.x >> v.y;
                    vertices[i] = v;
                }
                figures.emplace_back(std::make_unique<rectangle>(vertices));
            }
            else if (figure_name == std::string("trapezoid")) {
                std::array<vertex, 4> vertices;
                for (int32_t i = 0; i < 4; ++i) {
                    is >> v.x >> v.y;
                    vertices[i] = v;
                }
                figures.emplace_back(std::make_unique<trapezoid>(vertices));
            }
        }
        return figures;
    }
```

```
    ~loader() = default; // Деструктор (Не нужен, но должен быть)
};

#endif //D_LOADER_H
```

undoer.h

```
#ifndef D_UNDOER_H
#define D_UNDOER_H

#include<string>
#include<vector>
#include<memory>
#include<stack>

#include "figure.h"
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"

enum Indexes {
    ul_clear,
    ul_add,
    ul_rmv,
    ul_load
};

static std::vector<std::unique_ptr<figure>> figures;

struct undo_list {
    struct ul_element {
        int32_t idx;
        std::unique_ptr<figure> fig;
        ul_element(int32_t _idx, std::unique_ptr<figure> _fig) : idx(_idx), fig(std::move(_fig)) {}
    };
    std::stack<ul_element> ul_stack;
    std::vector<std::unique_ptr<figure>> ul_figures;
    void push(int exp, std::unique_ptr<figure> value) {
        this->ul_stack.push(ul_element(exp, std::move(value)));
    }
    void undo() {
        //ul_stack.top();
        if (ul_stack.size() > 0) {
            if (ul_stack.top().idx == ul_add) {
                figures.pop_back();
            }
            else if (ul_stack.top().idx == ul_rmv) {
                figures.emplace_back(std::move(ul_stack.top().fig));
            }
            else if (ul_stack.top().idx == ul_clear) {
                figures = std::move(ul_figures);
            }
            else if (ul_stack.top().idx == ul_load) {
                figures.clear();
            }
            ul_stack.pop();
        }
    }
};
static undo_list ul;

#endif //D_UNDOER_H
```

main.cpp

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

#include "square.h"
#include "rectangle.h"
#include "trapeze.h"

int main(){
   std::vector<figure*> figures;
   for (;;){
      int command;
      std::cin >> command;
      if (command == 0){
         break;
      }else if (command == 1){
         int figure_type;
         std::cin >> figure_type;
         figure* ptr;
         if (figure_type == 0){
            ptr = new square(std::cin);
         }else if (figure_type == 1){
            ptr = new rectangle(std::cin);
         } else {
          ptr = new trapeze(std::cin);
        }
         figures.push_back(ptr);
      }else if (command == 2){
         int id;
         std::cin >> id;
         delete figures[id];
         figures.erase(figures.begin() + id);
      }else if (command == 3){
         std::cout << "Centers:\n";
         for (figure* ptr: figures){
            std::cout << ptr->center() << std::endl;
         }
      }else if (command == 4){
         std::cout << "Areas:\n";
         for (figure* ptr: figures){
            std::cout << ptr->area() << std::endl;
         }
      }else if (command == 5){
         std::cout << "Figures:\n";
         for (figure* ptr: figures){
            ptr->print(std::cout);
            std::cout << std::endl;
         }
      }
   }
   for (figure* ptr: figures){
      delete ptr;
   }
}
```

Makefile

```
cmake_minimum_required(VERSION 3.0)

project(lab7)

set(CMAKE_CXX_STANDARD_REQUIRED YES)
set(CMAKE_CXX_STANDARD 17)
```

```
add_executable(lab7
  main.cpp
  sdl.cpp
)

add_subdirectory(lib/SDL2/)
target_link_libraries(lab7 SDL2-static)
target_include_directories(lab7 PRIVATE ${SDL2_INCLUDE_DIR})

add_subdirectory(lib/imgui/)
target_include_directories(imgui PRIVATE lib/SDL2/include/)
target_link_libraries(lab7 imgui)
```

3. **Ссылка на репозиторий:**
   https://github.com/GitGood2000/oop_exercise_07

4. **Набор testcases:**

test.txt

```
square
319 156
278 288
410 329
451 197
rectangle
603 97
542 259
655 301
716 139
trapezoid
255 530
564 548
480 446
338 449
```

test2.txt

```
trapezoid
322 297
395 240
522 238
582 289
rectangle
297 115
307 214
456 207
446 108
rectangle
683 97
712 106
684 199
655 190
rectangle
521 59
534 136
579 131
566 54
square
235 414
331 486
403 390
```

307 318
square
666 526
622 412
508 456
552 570
square
457 389
467 364
492 374
482 399
trapezoid
67 493
94 460
145 488
150 529

test3.txt
triangle
264 36
241 103
388 111
triangle
385 39
469 111
531 36
triangle
663 57
573 107
685 115
rectangle
427 202
519 203
518 312
426 311
square
285 177
193 265
281 357
373 269
square
639 193
585 245
637 299
691 247
trapezoid
317 388
365 493
425 495
478 388
trapezoid
473 497
542 392
638 397
671 475
trapezoid
325 486
289 396
236 392
172 464
rectangle
325 155
340 200
426 174

411 129
rectangle
528 135
513 162
561 187
576 160

5. **Результаты выполнения тестов:**

Все фигуры из тестовых файлов успешно загружены и нарисованы правильно

6. **Объяснение результатов работы программы:**
   1) Создаётся чёрный экран - “Холст“
   2) Программа выполняет определённые действия в зависимости от нажатой кнопки:
      A) “New canvas“ - стирает все фигуры;
      B) “Save\Open“ - Сохраняет координаты вершин фигур в файл или создаёт фигуры по координатам из файла;
      C) “RGB“ - Изменение цвета линий;
      D) “Triangle/Square/Rectangle/Trapezoid“ - Создаёт фигуру (Треугольник(базовое было дано вместе с GUI, Квадрат, Прямоугольник или Трапецию)), рисует её и добавляет её в вектор
      E) “Remove“ - Удаление фигуры по индексу
      F) “Undo“ - Отменяет последнее совершенное действие

7. **Вывод:** 1) Ознакомились с проектированием и структурами классов в C++.и усвоили навык работы с ними; 2) Написана программа, производящая операции на графическом интерфейсе.