ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ­­

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | А. В. Устинов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |
| ВИРТУАЛЬНЫЕ ФУНКЦИИ И АБСТРАКТНЫЕ КЛАССЫ |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2018

**1. Цель работы**

Изучить принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Закрепить знания по теме: Виртуальные функции и абстрактные классы.

|  |  |  |
| --- | --- | --- |
| **№ варианта** | **Фигура** | **Расположение** |
| 8 | 9 | 10,11,12 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ п/п** | **Наименование** | **Вид** | **Отражение** | **Поворот** |
| 9 | Кружок |  | Нет | Нет |

**2. Листинг программы**

#include "circle.hpp"

#include "line.hpp"

#include "screen.hpp"

#include "square.hpp"

#include <memory>

#include <vector>

int main() {

auto screen = std::make\_unique<Screen>(50, 35);

std::vector<std::shared\_ptr<Shape>> shapes;

shapes.emplace\_back(

std::make\_shared<Square>(Point(15, 3), Point(32, 12))); // Шляпа

Point p1 = shapes[shapes.size() - 1]->getLeftBottom();

p1.setX(p1.getX() - 2);

p1.setY(p1.getY() + 1);

Point p2 = shapes[shapes.size() - 1]->getRightBottom();

p2.setX(p2.getX() + 2);

p2.setY(p2.getY() + 1);

shapes.emplace\_back(std::make\_shared<Line>(p1, p2)); // Линия под шляпой

p1 = shapes[shapes.size() - 1]->getLeftBottom();

p1.setX(p1.getX() + 1);

p1.setY(p1.getY() + 1);

p2 = shapes[shapes.size() - 1]->getRightBottom();

p2.setX(p2.getX() - 1);

p2.setY(p2.getY() + 10);

shapes.emplace\_back(std::make\_shared<Square>(p1, p2)); // Голова

Point eyeLeft = shapes[shapes.size() - 1]->getLeftTop();

eyeLeft.setX(eyeLeft.getX() + 2);

eyeLeft.setY(eyeLeft.getY() + 2);

Point eyeRight = Point(eyeLeft.getX() + 2, eyeLeft.getY());

shapes.emplace\_back(std::make\_shared<Line>(eyeLeft, eyeRight)); // Левый глаз

eyeRight = shapes[shapes.size() - 2]->getRightTop();

eyeRight.setX(eyeRight.getX() - 2);

eyeRight.setY(eyeRight.getY() + 2);

eyeLeft = Point(eyeRight.getX() - 2, eyeRight.getY());

shapes.emplace\_back(std::make\_shared<Line>(eyeLeft, eyeRight)); // Правый глаз

std::shared\_ptr<Shape> leftEye = shapes[shapes.size() - 2];

std::shared\_ptr<Shape> rightEye = shapes[shapes.size() - 1];

Point nose = Point(

leftEye->getRightTop().getX() +

(rightEye->getLeftTop().getX() - leftEye->getRightTop().getX()) / 2,

leftEye->getRightTop().getY() + 2);

shapes.emplace\_back(std::make\_shared<Line>(nose, nose)); // Нос

std::shared\_ptr<Shape> head = shapes[shapes.size() - 4];

p1 = head->getLeftBottom();

p1.setX(p1.getX() + 2);

p1.setY(p1.getY() - 2);

p2 = head->getRightBottom();

p2.setX(p2.getX() - 2);

p2.setY(p2.getY() - 2);

shapes.emplace\_back(std::make\_shared<Line>(p1, p2)); // Рот

auto costume = std::make\_shared<Line>(Point(p1.getX(), p1.getY() + 10),

Point(p2.getX(), p2.getY() + 10));

shapes.emplace\_back(costume); //Линия костюма

p1 = Point(costume->getLeftBottom().getX(),

costume->getLeftBottom().getY() + 1);

auto leftDot = std::make\_shared<Line>(p1, p1);

shapes.emplace\_back(leftDot); //Левая точка

p1 = Point(costume->getRightBottom().getX(),

costume->getRightBottom().getY() + 1);

auto rightDot = std::make\_shared<Line>(p1, p1);

shapes.emplace\_back(rightDot); //Правая точка

// 10

shapes.emplace\_back(std::make\_shared<Circle>(17, 17, 1));

// 11

shapes.emplace\_back(std::make\_shared<Circle>(17, 30, 1));

// 12

shapes.emplace\_back(std::make\_shared<Circle>(7, 24, 2));

for (auto shape : shapes)

shape->draw(screen.get());

screen->draw();

std::cin.get();

return 0;

}

#pragma once

#include "shape.hpp"

class Circle : public virtual Shape {

public:

Circle(uint32\_t x, uint32\_t y, uint32\_t radius) : \_x(x), \_y(y), \_r(radius) {}

virtual void draw(Screen \*screen) const {

int x = 0;

int y = \_r;

int delta = 1 - 2 \* \_r;

int error = 0;

while (y >= 0) {

screen->putPoint(\_x + x, \_y + y);

screen->putPoint(\_x + x, \_y - y);

screen->putPoint(\_x - x, \_y + y);

screen->putPoint(\_x - x, \_y - y);

error = 2 \* (delta + y) - 1;

if (delta < 0 && error <= 0) {

++x;

delta += 2 \* x + 1;

continue;

}

error = 2 \* (delta - x) - 1;

if (delta > 0 && error > 0) {

--y;

delta += 1 - 2 \* y;

continue;

}

++x;

delta += 2 \* (x - y);

--y;

}

}

virtual void move(Point p) {

\_x = \_x + p.getX();

\_y = \_y + p.getY();

}

Point getLeftTop() const { return Point(\_x - \_r, \_y - \_r); }

Point getRightTop() const { return Point(\_x + \_r, \_y - \_r); }

Point getLeftBottom() const { return Point(\_x - \_r, \_y + \_r); }

Point getRightBottom() const { return Point(\_x + \_r, \_y + \_r); }

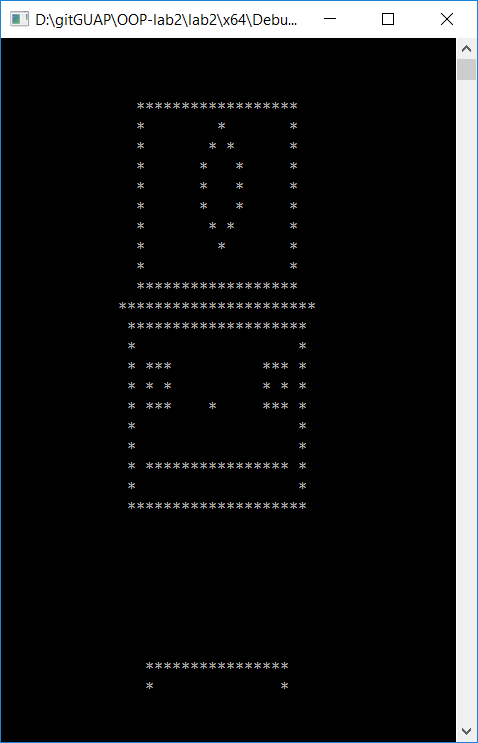
virtual ~Circle() {}

protected:

uint32\_t \_x, \_y, \_r;

};

**3. Пример работы**



**4. Вывод**

Изучил принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Закрепить знания по теме: Виртуальные функции и абстрактные классы.