Лабораторная работа №5Д (дополнительная)

«Виртуальные функции и абстрактные классы»

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.

Выбор варианта задания

Определить вариант задания, равный порядковому номеру студента в журнале (взять свой порядковый номер по модулю количества вариантов при необходимости).

Описание работы

Производные классы — это простое, гибкое и эффективное средство определения класса с целью повторного использования готового программного кода. Новые возможности добавляются к уже существующему классу, не требуя его перепрограммирования или перекомпиляции. С помощью производных классов можно организовать общий интерфейс с несколькими различными классами так, что в других частях программы можно будет единообразно работать с объектами этих классов. Понятие виртуальной функции позволяет использовать объекты надлежащим образом даже в тех случаях, когда их тип на стадии трансляции неизвестен. Основное назначение производных классов — упростить программисту задачу выражения общности классов.

Любое понятие не существует изолированно, оно существует во взаимосвязи с другими понятиями, и мощность данного понятия во многом определяется наличием таких связей. Раз класс служит для представления понятий, встаёт вопрос, как представить взаимосвязь понятий. Понятие производного класса и поддерживающие его языковые средства служат для представления иерархических связей, иными словами, для выражения общности между классами. Например, понятия окружности и треугольника связаны между собой, так как оба они представляют ещё понятие фигуры, т. е. содержат более общее понятие. Чтобы представлять в программе окружности и треугольники и при этом не упускать из вида, что они являются фигурами, надо явно определять классы окружность и треугольник так, чтобы было видно, что у них есть общий класс — фигура. Эта простая идея по сути является основой того, что обычно называется объектно-ориентированным программированием.

Рассмотрим учебную программу (см. Приложение), использующую некоторые из этих идей. Программа предназначена для вывода на экран картинок, составленных из набора заготовок — «фигур». В программе объявлен абстрактный класс «фигура» (*Shape*). Все конкретные фигуры — линия, прямоугольник и т. п. — являются производными от этого класса. Класс «фигура» поддерживает действия, необходимые для всех фигур: он создаёт из всех объявляемых фигур общий список, который может быть обработан программой рисования при выдаче фигур на экран. Кроме того, в классе «фигура» объявлен набор функций-членов, которые должны поддерживать все фигуры, чтобы из них можно было создавать картинки. Это функции, возвращающие координаты всех крайних точек фигуры, по которым их можно будет стыковать. Эти функции — чисто виртуальные, они должны быть обязательно определены затем отдельно в каждой фигуре. Имеются также два дополнительных класса, уточняющие свойства фигур. Некоторые фигуры можно

поворачивать. Для таких фигур имеется базовый класс *Rotatable*. Для других фигур возможна операция отражения относительно горизонтальной или вертикальной оси. Эти фигуры можно строить на базе класса *Reflectable*. Если фигура имеет оба этих свойства, то она должна быть производной от обоих классов.

Класс «фигура» является ядром библиотеки фигур *snape.hpp*. Имеется также библиотека поддержки работы с экраном *screen.hpp*, в которой определены размеры экрана, введено понятие точки и перечислены утилиты работы с экраном, конкретизированные затем в *shape.hpp*. Для простоты и универсальности работа с экраном реализована как формирование и построчный вывод матрицы символов.

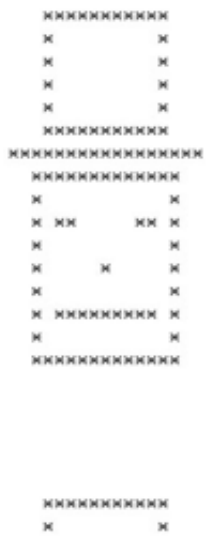
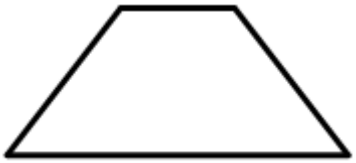
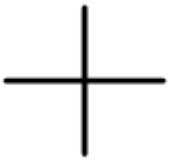
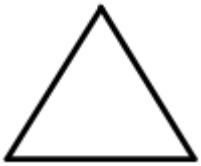
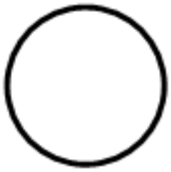
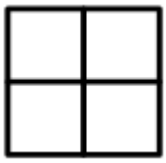


Рис. 1. Результат работы программы

В таблице 1 собрана коллекция фигур, которыми можно дополнить рассмотренную прикладную программу.

Таблица 1. Коллекция дополнительных фигур

№ п/п	Наименование	Вид	Отражение	Поворот
1	Прямоугольник		Нет	Да
2	Квадрат		Нет	Нет

3	Ромб		Нет	Нет
4	Параллелограмм		Да	Да
5	Трапеция		Да	Да
6	Крест		Нет	Нет
7	Косой крест		Нет	Нет
8	Треугольник		Да	Да
9	Кружок		Нет	Нет
10	Квадрат с крестом		Нет	Нет
11	Ромб с крестом		Нет	Нет
12	Кружок с крестом		Нет	Нет

13	Треугольник с крестом		Да	Да
14	Зачёркнутый квадрат		Нет	Нет
15	Зачёркнутый кружок		Нет	Нет
16	Зачёркнутый треугольник		Да	Да

Для некоторых фигур возможны поворот на 90^{o} вправо или влево, или отражение относительно горизонтальной и/или вертикальной оси симметрии, причём для части из них имеются обе возможности. Некоторые фигуры строятся как составные из более простых. Эти идеи можно отобразить показанной на рис. 2 иерархией классов.

ОБЩЕЕ ЗАДАНИЕ!

Доработать учебную программу: добавить в коллекцию ещё одну фигуру, номер которой указан в таблице 1. Для этой фигуры нужно будет определить подходящее место в иерархии классов и написать необходимые функции-члены. Функции-члены, использование которых не предполагается, можно определить так, чтобы они были недоступны. Разработанной фигурой нужно дополнить картинку в указанных в варианте позициях. Позиция *1* обозначает галстук или воротник, *2* и *3* — бакенбарды, *4* и *5* — уши, *6* — кокарду, *7* и *8*— рога, *9* — нос, *10* и *11* — глаза, *12* — шляпу в целом (*см*. Рис. 3). Возможно, некоторые из фигур нужно будет повернуть или отразить. Для примыкания фигур должны использоваться их габаритные точки.

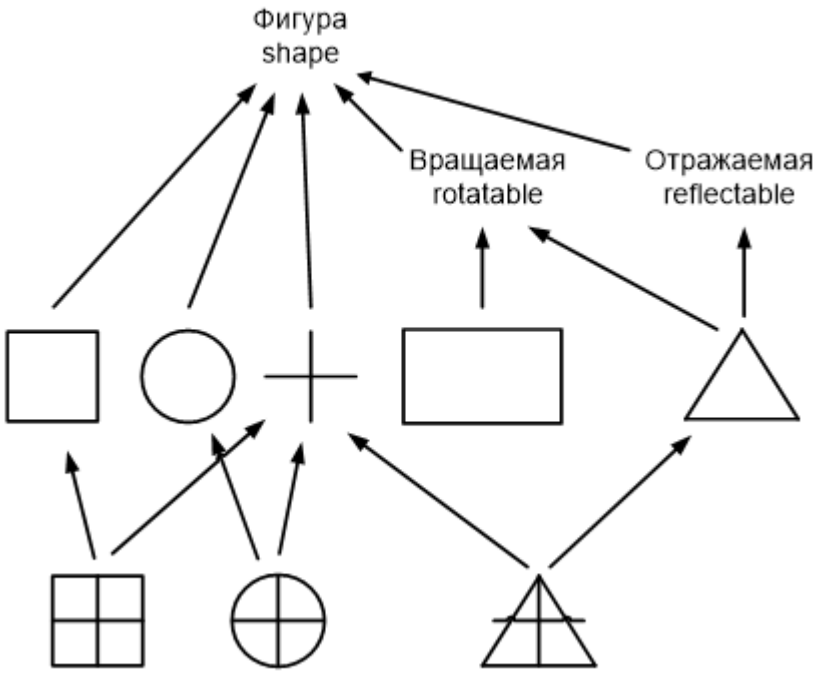


Рис 2. Фрагмент иерархии классов фигур

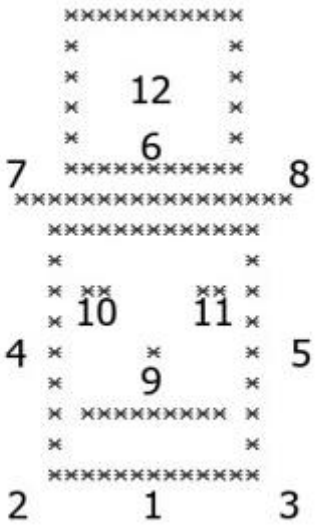


Рис 3. Позиции на результате работы программы для возможной вставки дополнительной фигуры

Варианты заданий

Таблица 2. Индивидуальные задания по лабораторной работе

№ варианта	Фигура	Расположение	№ варианта	Фигура	Расположение
1	2	2, 3, 10, 11	26	15	6, 7, 8
2	7	1, 4, 5	27	12	9, 10 11
3	6	4, 5	28	11	7, 8, 12
4	3	10, 11, 12	29	2	4, 5, 9
5	8	4, 5, 6	30	11	1, 7, 8
6	5	1	31	12	1, 2, 3
7	4	7, 8	32	3	1, 4, 5
8	9	10, 11, 12	33	10	4, 5
9	4	2, 3	34	13	1, 7, 8
10	5	4, 5	35	4	7, 8
11	10	1, 12	36	9	1, 4, 5
12	3	4, 5, 9	37	14	4, 5, 6
13	6	1, 12	38	5	4, 5
14	11	10, 11	39	8	1, 6, 9
15	2	1, 7, 8, 12	40	15	1, 2, 3
16	7	6, 7, 8	41	6	2, 3, 9
17	12	2, 3, 7, 8	42	7	10, 11
18	15	10, 11	43	2	1, 2, 3, 7, 8
19	8	2, 3, 7, 8	44	7	2, 3, 12
20	13	1, 12	45	6	10, 11
21	14	2, 3, 12	46	3	2, 3, 6
22	9	2, 3, 9	47	8	9, 10, 11
23	14	7, 8, 9	48	5	12
24	13	1, 4, 5	49	4	4, 5
25	10	7, 8	50	9	4, 5, 7, 8

Содержание отчета:

1. Титульный лист

2. Условие (с указанием номера варианта)

3. Полный текст (листинг) программы с выделением кода, написанного студентом с подробными комментариями!

4. Скриншоты с результатами (скриншоты должны демонстрировать все возможные ветви алгоритма решения).

5. Выводы.

6. Файл с отчетом должен быть в формате doc или pdf. Имя файла задаем так: Группа_ФИО_ЛРномер (Например, 4431_Иванов_ЛР5Д)

Приложение

Учебная программа «Библиотека фигур»

```
#pragma once
//Файл point.h
#ifndef POINT_H
#define POINT_H

#include <cstdint>

class Point //Точка на экране
{
public:
	Point() : _x(0), _y(0) { };
	Point(std::uint32_t x, std::uint32_t y) : _x(x), _y(y) {}

	std::uint32_t getX() const { return _x; }
	std::uint32_t getY() const { return _y; }
	void setX(std::uint32_t val) { _x = val; }
	void setY(std::uint32_t val) { _y = val; }

protected:
	std::uint32_t _x, _y;
};

#endif // POINT_H


#pragma once
//Файл screen.h
#ifndef SCREEN_H
#define SCREEN_H

#include "point.h"
```

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

class Screen
{
public:
	enum class Pixel : char
	{
		WHITE = ' ',
		BLACK = '*'
	};

	Screen(std::uint32_t xSize, std::uint32_t ySize);

	void putLine(const Point& a, const Point& b);

	void putLine(std::uint32_t x0, std::uint32_t y0, std::uint32_t x1, std::uint32_t
y1);

	void putPoint(const Point& p);

	void putPoint(const std::uint32_t x, const std::uint32_t y);

	void clear();

	void draw() const;

private:
	std::uint32_t _xSize;
	std::uint32_t _ySize;
	std::vector<std::vector<Pixel> > _screen;
};

Screen::Screen(std::uint32_t xSize, std::uint32_t ySize)
{
	_xSize = xSize;
	_ySize = ySize;
	_screen.reserve(ySize);
	for (size_t i = 0; i < ySize; i++)
		_screen.emplace_back(std::vector<Pixel>(xSize, Pixel::WHITE));
}

void Screen::putLine(const Point& a, const Point& b)
{
	putLine(a.getX(), a.getY(), b.getX(), b.getY());
}

void Screen::putLine(std::uint32_t x0, std::uint32_t y0, std::uint32_t x1, std::uint32_t
y1)
{
	std::int32_t dx = 1;
	std::int32_t a = x1 - x0;
	if (a < 0) { dx = -1; a = -a; }
	std::int32_t dy = 1;
	std::int32_t b = y1 - y0;
	if (b < 0) { dy = -1; b = -b; }
	std::int32_t two_a = 2 * a;
	std::int32_t two_b = 2 * b;
	std::int32_t xcrit = -b + two_a;
	std::int32_t eps = 0;

	while (true)
	{
		putPoint(x0, y0);
```

```
                if (x0 == x1 && y0 == y1) break;
                if (eps <= xcrit)
                {
                        x0 += dx;
                        eps += two_b;
                }

                if (eps >= a || a < b)
                {
                        y0 += dy;
                        eps -= two_a;
                }
        }

}

void Screen::putPoint(const Point& p)
{
        _screen[p.getX()][p.getY()] = Pixel::BLACK;
}

void Screen::putPoint(const std::uint32_t x, const std::uint32_t y)
{
        _screen[x][y] = Pixel::BLACK;
}

void Screen::clear()
{
        for (size_t i = 0; i < _ySize; ++i)
                for (size_t j = 0; j < _xSize; ++j)
                        _screen[i][j] = Pixel::WHITE;
}

void Screen::draw() const
{
        for (size_t i = 0; i < _ySize; ++i)
        {
                for (size_t j = 0; j < _xSize; ++j)
                        std::cout << (char)_screen[i][j];
                std::cout << std::endl;
        }
}

#endif // SCREEN_H


#pragma once
// Файл shape.h
#ifndef SHAPE_H
#define SHAPE_H

#include "screen.h"

class Shape
{
public:
        virtual void draw(Screen* screen) const = 0;

        virtual void move(Point p) = 0;

        virtual Point getLeftTop() const = 0;

        virtual Point getRightTop() const = 0;

        virtual Point getLeftBottom() const = 0;
```

```
        virtual Point getRightBottom() const = 0;

        virtual ~Shape() {}
};

#endif // SHAPE_H
```

```
#pragma once
// Файл rotatable.h
#ifndef ROTATABLE_H
#define ROTATABLE_H

class Rotatable { //Фигуры, пригодные к повороту
public:
        virtual void rotateLeft() = 0;    //Повернуть влево
        virtual void rotateRight() = 0;   //Повернуть вправо
};

#endif // ROTATABLE_H
```

```
#pragma once
// Файл reflectable.h
#ifndef REFLECTABLE_H
#define REFLECTABLE_H
class reflectable { // Фигуры, пригодные к зеркальному отражению

public:
        virtual void flipHorisontally() = 0;     // Отразить горизонтально
        virtual void flipVertically() = 0;       // Отразить вертикально
};

#endif // REFLECTABLE_H
```

```
#pragma once
// Файл line.h
#ifndef LINE_H
#define LINE_H

#include "shape.h"
#include "screen.h"

#include <algorithm>

class Line : public Shape
{
protected:
        Point _a;
        Point _b;

public:
        Line(const Point& a, const Point& b);

        Line(unsigned int x0, unsigned int y0, unsigned int x1, unsigned int y1);

        Line(const Line& source);

        void draw(Screen* screen) const;

        void move(Point p);

        Point getLeftTop() const;
```

```
        Point getRightTop() const;

        Point getLeftBottom() const;

        Point getRightBottom() const;

        Point getFirstPoint() const;

        Point getSecondPoint() const;
};

Line::Line(const Point& a, const Point& b)
{
        _a = a;
        _b = b;
}

Line::Line(unsigned int x0, unsigned int y0, unsigned int x1, unsigned int y1)
{
        _a = Point(x0, y0);
        _b = Point(x1, y1);
}

Line::Line(const Line& source)
{
        _a = source._a;
        _b = source._b;
}

void Line::draw(Screen* screen) const
{
        screen->putLine(_a.getY(), _a.getX(), _b.getY(), _b.getX());
}

void Line::move(Point p)
{
        _a.setX(_a.getX() + p.getX()); _a.setY(_a.getY() + p.getY());
        _b.setX(_b.getX() + p.getX()); _b.setY(_b.getY() + p.getY());
}

Point Line::getLeftTop() const
{
        return Point(std::min(_a.getX(), _b.getX()), std::min(_a.getY(), _b.getY()));
}

Point Line::getRightTop() const
{
        return Point(std::max(_a.getX(), _b.getX()), std::min(_a.getY(), _b.getY()));
}

Point Line::getLeftBottom() const
{
        return Point(std::min(_a.getX(), _b.getX()), std::max(_a.getY(), _b.getY()));
}

Point Line::getRightBottom() const
{
        return Point(std::max(_a.getX(), _b.getX()), std::max(_a.getY(), _b.getY()));
}

Point Line::getFirstPoint() const
{
        return _a;
}
```

```
Point Line::getSecondPoint() const
{
	return _b;
}
#endif // LINE_H


#pragma once
/// Файл cross.h
#ifndef CROSS_H
#define CROSS_H

#include "shape.h"
#include "line.h"

//Крест
class Cross : public virtual Shape
{
protected:
	Line* _first;
	Line* _second;
public:
	Cross(const Point& left, const Point& top);

	virtual void draw(Screen* screen) const;

	virtual void move(Point p);

	Point getLeftTop() const;

	Point getRightTop() const;

	Point getLeftBottom() const;

	Point getRightBottom() const;

	virtual ~Cross();
};

Cross::Cross(const Point& left, const Point& top)
{
	Point p1, p2;
	Point p3, p4;

	p1 = left;
	p2 = Point(top.getX() + top.getX() - left.getX(), left.getY());
	p3 = top;
	p4 = Point(top.getX(), left.getY() - top.getY() + left.getY());

	_first = new Line(p1, p2);
	_second = new Line(p3, p4);
}

void Cross::draw(Screen* screen) const
{
	_first->draw(screen);
	_second->draw(screen);
}

void Cross::move(Point p)
{
	_first->move(p);
	_second->move(p);
}
```

```
Point Cross::getLeftTop() const
{
	return _first->getLeftTop();
}

Point Cross::getRightTop() const
{
	return _second->getRightTop();
}

Point Cross::getLeftBottom() const
{
	return _second->getLeftBottom();
}

Point Cross::getRightBottom() const
{
	return _first->getRightBottom();
}

Cross:: ~Cross()
{
	delete _first;
	delete _second;
}

#endif // CROSS_H
```

```
#pragma once
//Файл square.h
#ifndef SQUARE_H
#define SQUARE_H

#include "line.h"

class Square : public virtual Shape
{
public:
	Square(const Point& leftTop, const Point& rightBottom);

	virtual void draw(Screen* screen) const;

	virtual void move(Point p);

	Point getLeftTop() const;

	Point getRightTop() const;

	Point getLeftBottom() const;

	Point getRightBottom() const;

	virtual ~Square();

protected:
	Line *_left, *_top, *_right, *_bottom;
};

Square::Square(const Point& leftTop, const Point& rightBottom)
{
	_left = new Line(leftTop, Point(leftTop.getX(), rightBottom.getY()));
	_top = new Line(leftTop, Point(rightBottom.getX(), leftTop.getY()));
```

```
	_right = new Line(Point(rightBottom.getX(), leftTop.getY()), rightBottom);
	_bottom = new Line(Point(leftTop.getX(), rightBottom.getY()), rightBottom);
}

void Square::draw(Screen* screen) const
{
	_left->draw(screen);
	_top->draw(screen);
	_right->draw(screen);
	_bottom->draw(screen);
}

void Square::move(Point p)
{
	_left->move(p);
	_top->move(p);
	_right->move(p);
	_bottom->move(p);
}

Point Square::getLeftTop() const
{
	return _left->getLeftTop();
}

Point Square::getRightTop() const
{
	return _right->getRightTop();
}

Point Square::getLeftBottom() const
{
	return _left->getLeftBottom();
}

Point Square::getRightBottom() const
{
	return _right->getRightBottom();
}

Square::~Square()
{
	delete _left;
	delete _top;
	delete _right;
	delete _bottom;
}

#endif //SQUARE_H


// Labor_5D.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается
выполнение программы.
//
// Файл main.cpp
#include <iostream>

#include "screen.h"
#include "line.h"
#include "square.h"

#include <vector>
#include <memory>
```

```
int main()
{
	system("color F1");
	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

	for (auto shape : shapes)
		shape->draw(screen.get());
	screen->draw();
```

```
        std::cin.get();
        return 0;
}
```