**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

**­­ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Работа с иерархией объектов:**

**наследование и полиморфизм**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 5307 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Правиленко М. А. |
|  |  | Сахаров И. В. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Колинько П. Г. |

Санкт-Петербург

2017

Содержание

[Цель работы 2](#_Toc478399986)

[1. Задание 3](#_Toc478399987)

[2. Описание иерархии классов 3](#_Toc478399988)

[3. Результат работы программы 3](#_Toc478399989)

[4. Вывод 5](#_Toc478399990)

[Приложение 6](#_Toc478399991)

[Исходный код программы 6](#_Toc478399992)

­­­­­­

# Цель работы

Получить практические навыки работы с иерархией объектов и изучить основы объектно-ориентированного программирования.

# Задание

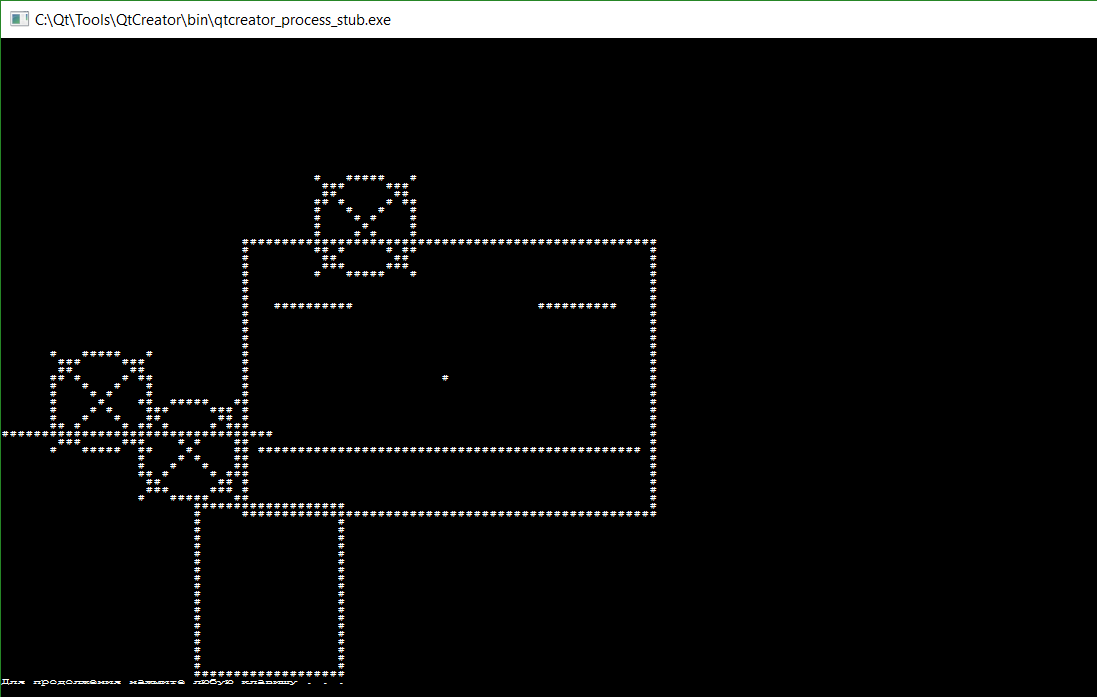
Доработать учебную программу: добавить фигуру «кружок с косым крестом» и определить для неё место в иерархии объектов. Разместить фигуры на картинке в позициях 4, 9, 5, использовав для этого функции, обеспечивающие примыкание очередной фигуры с нужной стороны.

# Описание иерархии классов

В ходе выполнения работы были созданы новые классы circle и cross, которые являются наследниками класса rectangle, и их наследник target – кружок с крестом. circle, cross и target используют переопределённые функции-члены draw(). rotate\_right() и rotate\_left() сделаны недоступными для этих классов при помощи спецификатора private в классе rectangle.

# Результат работы программы

Результат работы программы представлен на рис. 3-4.



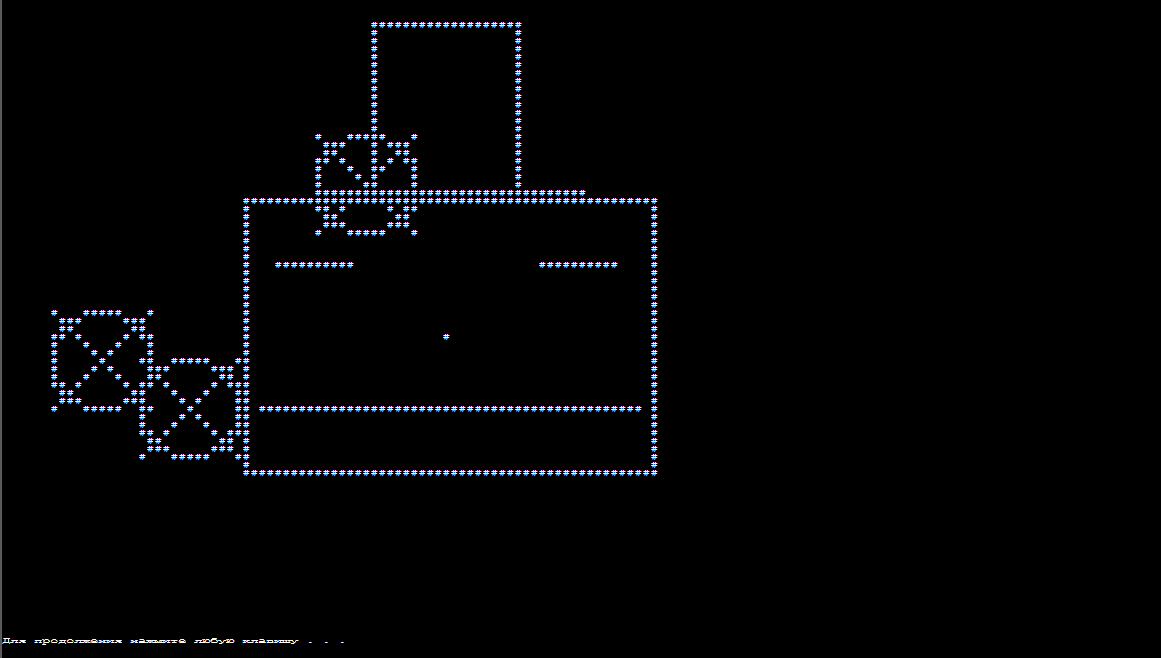


Рис. 3. Результат работы программы

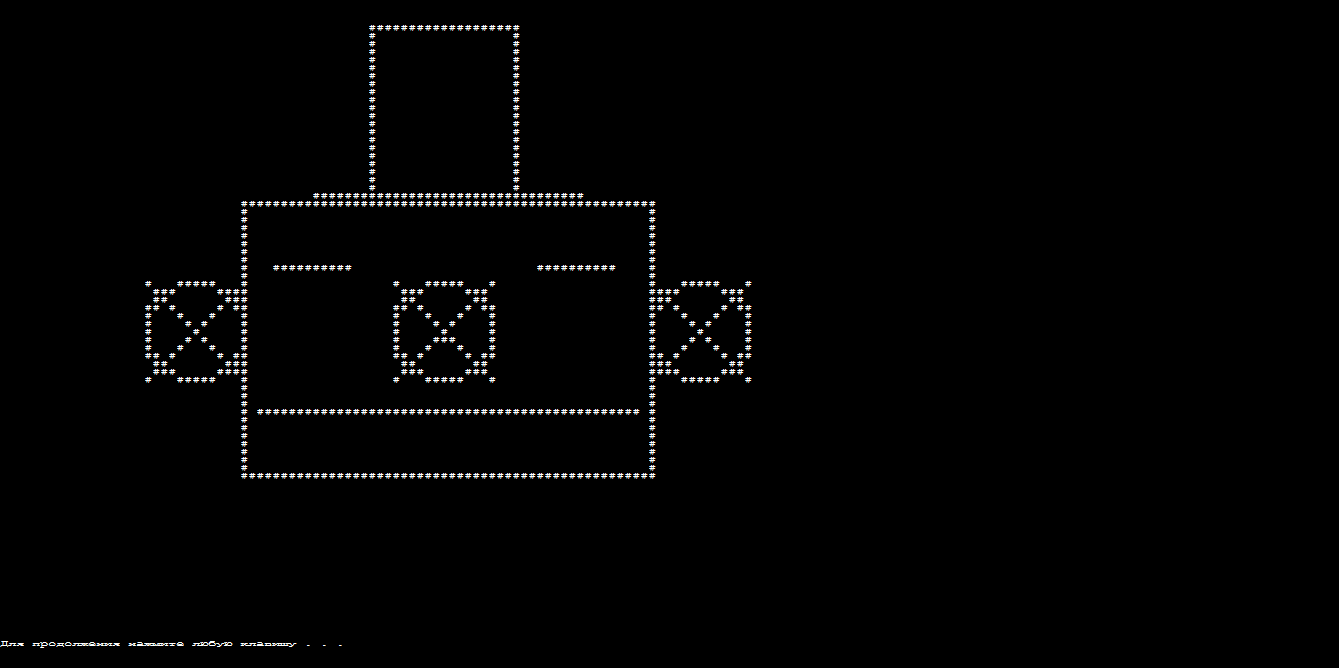


Рис. 4. Результат работы программы (продолжение)

## **Вывод**

В ходе работы были получены практические навыки работы с иерархией объектов. Наследование и полиморфизм позволяют повысить структурированность и наглядность программы.

## **Приложение**

## **Исходный код программы**

**Файл screen.h**

#ifndef SCREEN\_H

#define SCREEN\_H

const int XMAX = 80\*2;

const int YMAX = 40\*2;

const int rad = 6;

class point { //Точка

public:

int x, y;

point() {};

point(int a, int b) :x(a), y(b) {}

};

//Функции работы с экраном

extern void put\_point(int a, int b); //Вывод точки

inline void put\_point(point p) { put\_point(p.x, p.y); }

extern void put\_line(int, int, int, int); //Вывод линии

extern void put\_line(point a, point b);

extern void screen\_init(); //Создание экрана

extern void screen\_destroy(); //Удаление экрана

extern void screen\_refresh(); //Обновление экрана

extern void screen\_clear(); //Очистка экрана

#endif // SCREEN\_H

**Файл** **shape.h**

#ifndef SHAPE\_H

#define SHAPE\_H

#include <iostream>

#include "screen.h"

using namespace std;

char screen[XMAX][YMAX];

enum color { black = '#', white = ' ' };

void screen\_init()

{

for (int y = 0; y < YMAX; ++y)

for (int x = 0; x < XMAX; ++x)

screen[x][y] = white;

}

int on\_screen(int a, int b) //Попадание на экран

{

return (0 <= a && a < XMAX && 0 <= b && b < YMAX);

}

void put\_point(int a, int b)

{

if (on\_screen(a, b)) screen[a][b] = black;

}

void put\_line(int x0, int y0, int x1, int y1)

{

/\*

Рисование отрезка прямой (x0, y0) - (x1, y1)

Уравнение прямой: b(x-x0) + a(y-y0) = 0

Минимизируется величина abs(eps)

где eps = 2\*(b(x-x0))+a(y-y0)

\*/

int dx = 1;

int a = x1 - x0;

if (a < 0) dx = -1, a = -a;

int dy = 1;

int b = y1 - y0;

if (b < 0) dy = -1, b = -b;

int two\_a = 2 \* a;

int two\_b = 2 \* b;

int xcrit = -b + two\_a;

int eps = 0;

for (;;) {

put\_point(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;

}

}

extern void put\_line(point a, point b)

{

put\_line(a.x, a.y, b.x, b.y);

}

void screen\_clear() { screen\_init(); }

void screen\_refresh()

{

for (int y = YMAX - 1; 0 <= y; --y) {

for (int x = 0; x < XMAX; ++x)

cout << screen[x][y];

cout << '\n';

}

}

struct shape {

static shape\* list;

shape\* next;

shape() { next = list; list = this; }

virtual point *north*() const = 0;

virtual point *south*() const = 0;

virtual point *east*() const = 0;

virtual point *west*() const = 0;

virtual point *neast*() const = 0;

virtual point *seast*() const = 0;

virtual point *nwest*() const = 0;

virtual point *swest*() const = 0;

virtual point *center*() const = 0;

virtual void *draw*() = 0;

virtual void *move*(int, int) = 0;

};

shape\* shape::list = nullptr; //Иницализация списка фигур

class rotatable : public virtual shape { //Фигуры, пригодные к повороту

public:

virtual void *rotate\_left*() = 0;

virtual void *rotate\_right*() = 0;

};

class reflectable : public virtual shape //Фигуры пригодные к зеркальному отражению

{

public:

virtual void *flip\_horisontally*() = 0;

virtual void *flip\_vertically*() = 0;

};

class line : public shape {

/\*

Отрезок прямой ["w", "e"]

north() определяет точку "выше центра отрезка и так далеко на север,

как самая его северная точка", и т.п.

\*/

point w, e;

public:

line(point a, point b) : w(a), e(b) {};

line(point a, int L) : w(point(a.x + L - 1, a.y)), e(a) {};

point *north*() const { return point((w.x + e.x) / 2, e.y<w.y ? w.y : e.y); }

point *south*() const { return point((w.x + e.x) / 2, e.y<w.y ? e.y : w.y); }

point *east*() const { return point((w.x + e.x) / 2, e.y<w.y ? e.y : w.y); }

point *west*() const { return point((w.x + e.x) / 2, e.y<w.y ? e.y : w.y); }

point *neast*() const { return point((w.x + e.x) / 2, e.y<w.y ? e.y : w.y); }

point *seast*() const { return point((w.x + e.x) / 2, e.y<w.y ? e.y : w.y); }

point *nwest*() const { return point((w.x + e.x) / 2, e.y<w.y ? e.y : w.y); }

point *swest*() const { return point((w.x + e.x) / 2, e.y<w.y ? e.y : w.y); }

point *center*() const { return point((w.x + e.x) / 2, e.y<w.y ? e.y : w.y); }

void *move*(int a, int b) { w.x += a; w.y += b;e.x += a; e.y += b; }

void *draw*() {

put\_line(w, e);

}

};

//Прямоугольник

class rectangle : public rotatable

{

void *rotate\_right*()//Поворот относительно se

{

int w = ne.x - sw.x, h = ne.y - sw.y;

sw.x = ne.x - h \* 2;

ne.y = sw.y + w / 2;

}

void *rotate\_left*() //Поворот относительно sw

{

int w = ne.x - sw.x, h = ne.y - sw.y;

ne.x = sw.x + h \* 2;

ne.y = sw.y + w / 2;

}

protected:

/\*

nw-----n-----ne

| |

| |

w c e

| |

| |

sw-----s-----se

\*/

point sw, ne;

public:

rectangle(point, point);

point *north*() const { return point((sw.x + ne.x) / 2, ne.y); }

point *south*() const { return point((sw.x + ne.x) / 2, sw.y); }

point *east*() const { return point(ne.x, (sw.y + ne.y) / 2); }

point *west*() const { return point(sw.x, (sw.y + ne.y) / 2); }

point *neast*() const { return ne; }

point *seast*() const { return point(ne.x, sw.y); }

point *nwest*() const { return point(sw.x, ne.y); }

point *swest*() const { return sw; }

point *center*() const { return point(*west*().x + ((*east*().x - *west*().x) / 2), *east*().y); }

void *move*(int a, int b)

{

sw.x += a;

sw.y += b;

ne.x += a;

ne.y += b;

}

void *draw*();

};

rectangle::rectangle(point a, point b)

{

if (a.x <= b.x)

if (a.y <= b.y) {

sw = a;

ne = b;

}

else {

sw = point(a.x, b.y);

ne = point(b.x, a.y);

}

else

if (a.y <= b.y) {

sw = point(b.x, a.y);

ne = point(a.x, b.y);

}

else {

sw = b;

ne = a;

}

}

void rectangle::*draw*()

{

point nw(sw.x, ne.y);

point se(ne.x, sw.y);

put\_line(nw, ne);

put\_line(ne, se);

put\_line(se, sw);

put\_line(sw, nw);

}

void shape\_refresh()//Перерисовка всех фигур

{

screen\_clear();

for (shape\* p = shape::list; p; p = p->next) p->*draw*();

screen\_refresh();

}

void up(shape\* p, const shape\* q)//Поместить p над q

{

point n = q->*north*();

point s = p->*south*();

p->*move*(n.x - s.x, n.y - s.y + 1);

}

void left(shape\* p, const shape\* q)//Поместить p слева от q

{

point n = q->*west*();

point s = p->*east*();

p->*move*(n.x - s.x, n.y - s.y + 1);

}

void right(shape\* p, const shape\* q)//Поместить p справа от q

{

point n = q->*east*();

point s = p->*west*();

p->*move*(n.x - s.x, n.y - s.y + 1);

}

void mid(shape\* p, const shape\* q)//Поместить p в центр q

{

point n = q->*center*();

point s = p->*center*();

p->*move*(n.x - s.x, n.y - s.y + 1);

}

#endif // SHAPE\_H

**Файл** **main.cpp**

#include "shape.h"

class myshape : public rectangle {

line\* l\_eye; //левый глаз

line\* r\_eye; //правый глаз

line\* mouth; //рот

public:

myshape(point, point);

void *draw*();

void *move*(int, int);

};

myshape::myshape(point a, point b) : rectangle(a, b)

{

int ll = *neast*().x - *swest*().x + 1;

int hh = *neast*().y - *swest*().y + 1;

l\_eye = new line(point(*swest*().x + 4, *swest*().y + hh \* 3 / 4), ll/5);

r\_eye = new line(point(*swest*().x + ll\*4/5 - 4, *swest*().y + hh \* 3 / 4), ll/5);

mouth = new line(point(*swest*().x + 2, *swest*().y + hh / 4), ll - 4);

}

void myshape::*draw*()

{

rectangle::*draw*();

int a = (*swest*().x + *neast*().x) / 2;

int b = (*swest*().y + *neast*().y) / 2;

put\_point(point(a, b));

}

void myshape::*move*(int a, int b)

{

rectangle::*move*(a, b);

l\_eye->*move*(a, b);

r\_eye->*move*(a, b);

mouth->*move*(a, b);

}

class circle : public virtual shape {

protected:

point mid;

int radius;

public:

circle(point, int);

void *draw*();

};

class cross : public rectangle

{

public:

cross(point, int);

void *draw*();

};

class target : public cross, public circle

{

public:

target(point, int);

void *draw*();

};

circle::circle(point centre, int rad)

{

mid = centre;

radius = rad;

}

void circle::*draw*()

{

int x = 0,

y = radius,

gap = 0,

delta = (2 - 2 \* radius);

while (y >= 0)

{

put\_point(point(*center*().x + x, *center*().y + y));

put\_point(point(*center*().x + x, *center*().y - y));

put\_point(point(*center*().x - x, *center*().y - y));

put\_point(point(*center*().x - x, *center*().y + y));

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

if (delta < 0 && gap <= 0)

{

x++;

delta += 2 \* x + 1;

continue;

}

if (delta > 0 && gap > 0)

{

y--;

delta -= 2 \* y + 1;

continue;

}

x++;

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

y--;

}

// int rad = (ne.x - sw.x) / 2;

// int hrw = 0;

// point nw(sw.x, ne.y);

// for (int y = -rad + 1; y < rad; y++)

// {

// hrw = sqrt(rad\*(rad - 3) - y\*(y));

// //for (int x = -hrw; x < hrw; x++)

// put\_point(point(center().x + hrw, center().y + y));

// put\_point(point(center().x - hrw, center().y + y));

// put\_point(point(center().x + y, center().y + y));

// put\_point(point(center().x - y, center().y + y));

// }

}

cross::cross(point centre, int rad) : rectangle(point(centre.x - rad, centre.y - rad),

point(centre.x + rad, centre.y + rad)){}

void cross::*draw*()

{

put\_line(*nwest*(), *seast*());

put\_line(*swest*(), *neast*());

}

target::target(point centre, int rad) : circle(centre, rad),

cross(centre, rad){}

void target::*draw*()

{

circle::*draw*();

cross::*draw*();

}

int main()

{

screen\_init();

rotatable\* p1 = new rectangle(point(0, 0), point(14\*3, 3\*3));

shape\* p2 = new line(point(0 \* XMAX, 0.375\*YMAX), 0.425\*YMAX);

shape\* p3 = new myshape(point(0.1875 \* XMAX, 0.25 \* YMAX),

point(0.3375 \* XMAX \* 1.5, 0.45 \* YMAX \* 1.5));

shape\* l\_ear = new target(point(45, 56), rad);

shape\* nose = new target(point(12, 34), rad);

shape\* r\_ear = new target(point(23, 28), rad);

p1->*rotate\_right*();

shape\_refresh();

system("pause");

up(p2, p3);

up(p1, p3);

shape\_refresh();

system("pause");

left(l\_ear, p3);

mid(nose, p3);

right(r\_ear, p3);

shape\_refresh();

system("pause");

return 0;

}