Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ» имени В.И. Ульянова (Ленина)

Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

Зачётная работа №4
по дисциплине «Алгоритмы и структуры данных»
«Работа с иерархией объектов: наследование и полиморфизм»

Вариант№21

Выполнил: студент группы 5005
Зинатуллин Азат Салаватович

Проверил: старший преподаватель
Колинько Павел Георгиевич

Цель работы

Получить практические навыки в работе с классами, научиться наследовать объекты.

1. Какие классы пришлось добавить ?

По заданию моей фигурой являлся треугольник с косым крестом. Данная фигура является составной из двух фигур: треугольника и косого креста. Поэтому было принято решение добавить два класса в библиотеку фигур: "triangle" и "oblique_cross". А в файл с прикладной программой добавить класс с собранной фигурой - треугольник с косым крестом, и назвать его "myshape_2".

2. Какой класс выбран для них в качестве базового?

В качестве базового класса я выбрал - "rectangle" (прямоугольник), так как в нем уже объявлены опорные точки и функции-члены всех необходимых габаритных точек, по которым можно нарисовать треугольник и косой крест. Опорные точки в базовом классе, изначально объявлены со спецификатором доступа "private", и не доступны из производного, поэтому необходимо было изменить спецификатор доступа на "protected", чтобы наследовать эти точки и все функции-члены принимающие их в качестве аргументов.

3. Какие функции-члены пришлось переопределить и почему?

Для добавленных классов пришлось переопределить функцию рисования draw(), потому что в базовом классе данная функция рисует прямоугольник. Мне нужно было изменить аргументы функций, таким образом чтобы рисовались треугольник и косой крест. Треугольник может быть в повернутом и отраженном положении, и для этого так же добавлены необходимые функции-члены. Нужное состояние можно задавать с помощью переменной флага "state", которая объявлена в теле класса. Остальные функции-члены из базового класса наследуются без изменений.
Так же, отдельно от классов фигур, была добавлена функция "imposition", которая накладывает одну фигуру поверх другой, используя для этого габаритные точки.

4. Какие функции-члены сделаны недоступными и каким образом это осуществлено?

Все добавленные мной, функции-члены находятся под спецификатором доступа "public".

5. Результат работы программы.

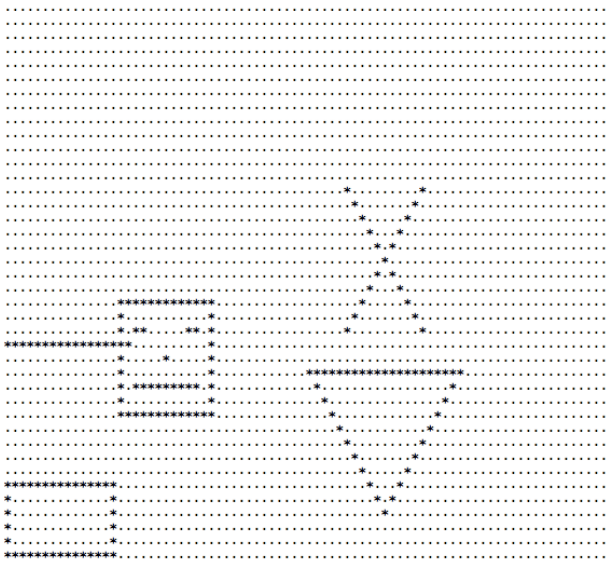

Рис. 1. Косой крест и треугольник в вертикально-отраженном положении.

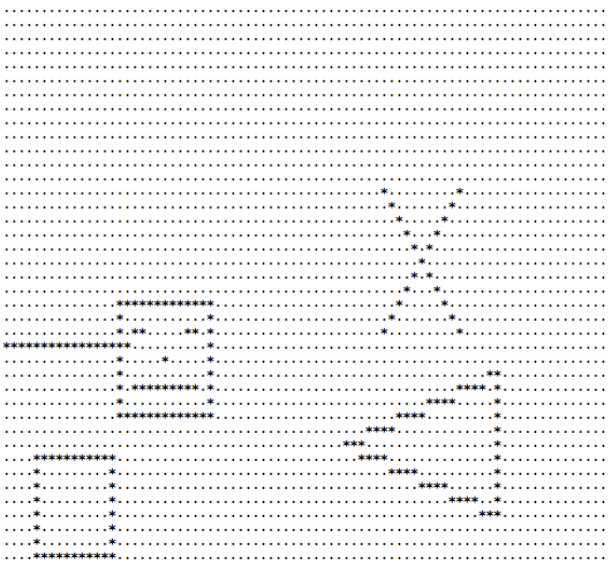

Рис. 2. Косой крест со сдвигом на 5 точек вправо, и треугольник в повернутом положении со сдвигом на 5 точек вправо.

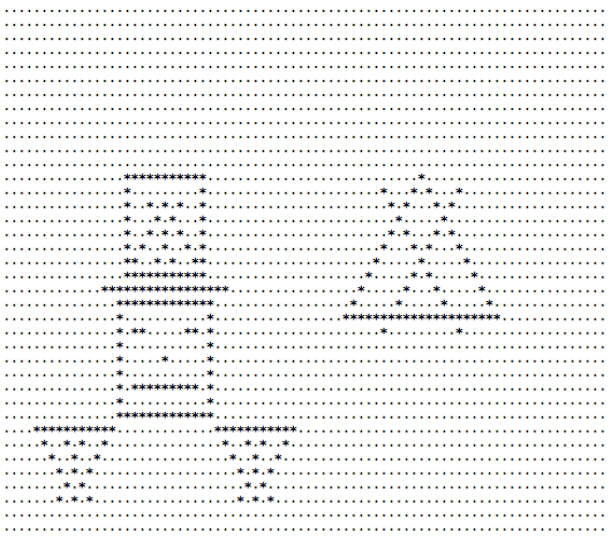

Рис. 3. Наложение треугольника на крест и сборка итоговой фигуры по точкам 2,3 и 12.

6. Список использованных источников.

1. Алгоритмы и структуры данных: методические указания к лабораторным работам, практическим занятиям и курсовому проектированию. Ч.2 Вып. 1707 (для заочников) / сост.: П. Г. Колинько. — СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2017. — 52 с.

2. Язык программирования С++. Вводный курс. Стенли Б, Липпман, Жози Лажойе. 3-е издание. 1185 стр.

3. Курс видео уроков по С++. https://www.youtube.com/playlist?list=PLbmlzoDQrXVFC13GjpPrJxl6mzTiX65gs

6. Приложение

main.cpp

```
//  main.cpp

#include <iostream>
#include "screen.h"
#include "shape.h"
using namespace std;
// Дополнительная "сборная" фигура

class myshape : public rectangle
{ // Моя фигура ЯВЛЯЕТСЯ прямоугольником
    line* l_eye; // левый глаз – моя фигура СОДЕРЖИТ линию
    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)
// myshape запускает конструктор базового класса rectangle !
{
    int ll = neast().x - swest().x + 1;
    int hh = neast().y - swest().y + 1;
    l_eye = new line(point(swest().x + 2, swest().y + hh * 3/4), 2);
    r_eye = new line(point(swest().x + ll - 4, swest().y + hh * 3/4), 2);
    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 myshape_2 : public oblique_cross
{ // Моя фигура ЯВЛЯЕТСЯ треугольником с косым крестом
public:
    myshape_2(point a, point b);
    void draw();
};

// myshape запускает конструктор класса oblique_cross,
// а тот в свою очередь констуркторы классов треугольник и прямоугольник!
myshape_2::myshape_2(point a, point b) : oblique_cross(a, b){};


void myshape_2::draw()
{
```

```
    triangle::draw();
    oblique_cross::draw();
}


int main(int argc, const char * argv[])
{
    screen_init();

    //== 1.Объявление набора фигур ==
    rotatable* p1 = new rectangle(point(0,0), point(14,5));
    shape* p2 = new line(point(0,15),17);
    shape* p3 = new myshape(point(15,10), point(27,18));


    triangle* p4 = new triangle(point(40,3), point(60,13));
    triangle* p5 = new oblique_cross(point(42,26), point(58,16));


    shape_refresh();

    cin.get(); //Смотреть исходный набор

    //== 2.Ориентация ==

    p4->state=2;    // Поворот треугольника влево
    p4->move(5,0);  // Сдвинуть треугольник на 5 точек вправо по оси X
    p5->move(5,0);  // Сдвинуть косой крест на 5 точек вправо по оси X

    p1->rotate_right();
    shape_refresh();

    cin.get(); //Смотреть ориентацию

    //== 3.Сборка изображения ==

    p4->state=0;
    imposition(p4, p5); // Наложить треугольник на крест

    triangle* p6 = new myshape_2(point(38,0), point(48,5));
    p6->state=3;

    triangle* p7 = new myshape_2(point(38,0), point(48,5));
    p7->state=0;

    triangle* p8 = new myshape_2(point(38,0), point(48,5));
    p8->state=3;


    up(p2,p3);
    up(p1,p2);
    imposition(p7,p2);
    imposition_2(p8,p3);
    imposition_1(p6,p3);
    shape_refresh();

    cin.get(); //Смотреть результат
    screen_destroy();
    return 0;
}


//=== Файл shape.h -- библиотека фигур

#ifndef Nasledovanie_shape_h
#define Nasledovanie_shape_h

#include <iostream>
using namespace std;

//==1. Поддержка экрана в форме матрицы символов ==

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) // проверка попадания 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;
    }
}

void screen_clear() { screen_init(); } //Очистка экрана

void screen_refresh() // Обновление экрана
{
    for (int y = YMAX-1; 0 <= y; --y)  // с верхней строки до нижней
    {
        for (int x = 0; x < XMAX; ++x) // от левого столбца до правого
```

```
            cout << screen[x][y];
        cout << '\n';
    }
}
```

shape.h

```
//==2. Библиотека фигур ==
struct shape // Виртуальный базовый класс "фигура"
{
    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 void draw() = 0;
    virtual void move(int, int) = 0;
};


shape *shape::list = nullptr; //Инициализация списка фигур

class rotatable : public shape  //Фигуры, пригодные к повороту
{
public:
    virtual void rotate_left() = 0; //Повернуть влево
    virtual void rotate_right() = 0; //Повернуть вправо

};

class reflectable : public shape // Фигуры, пригодные к зеркальному отражению
{
public:
    virtual void flip_horisontally() = 0; // Отразить горизонтально
    virtual void flip_vertically() = 0; // Отразить вертикально
};

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); }

    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
  /*
     nw ------ n ------ ne
     |                  |
     |                  |
     w         c        e
     |                  |
     |                  |
     sw ------ s ------ se
  */
{
protected:
    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; }

    void rotate_right() // Поворот вправо 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 влево 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;
    }

    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);
}


//Треугольник
class triangle : public rectangle
/*

 nw     n     ne
       / \
      /   \
 w   /  c  \  e
    /       \
   /         \
 sw --- s --- se

 */

{
public:
    int state=3;    //Переменная флаг, указывает положение треугольника в пространстве.
    triangle(point a, point b);
    void draw();
};

//Конструктор класса "треугольник" вызывает конструктор класса "прямоугольник"
triangle::triangle(point a, point b) : rectangle(a,b){};

void triangle::draw()
{
    switch (state)
    {
        case 0: //Стандартное положение
            put_line(north(), seast());
            put_line(seast(), sw);
            put_line(sw, north());
            break;
        case 1: //Поворот вправо
            put_line(nwest(), east());
            put_line(east(), sw);
            put_line(sw, nwest());
            break;
        case 2: //Поворот влево
            put_line(ne, west());
            put_line(west(), seast());
            put_line(seast(), ne);
            break;
        case 3: //Вертикальное отражение
            put_line(nwest(), ne);
            put_line(ne, south());
            put_line(south(), nwest());
            break;
    }
}


//Косой крест
class oblique_cross : public triangle
/*

 nw nn  n  ee ne
      \   /
       \ /
 w      c     e
       / \
      /   \
 sw ww  s  ss se
 */

{
public:
    oblique_cross(point a, point b);
    void draw();
};

oblique_cross::oblique_cross(point a, point b) : triangle(a, b){};

void oblique_cross::draw()
{
    point nn(sw.x+3,ne.y);
    point ee(ne.x-3,ne.y);
    point ss(ne.x-3,sw.y);
    point ww(sw.x+3,sw.y);
    put_line(nn, ss);
    put_line(ww, ee);
}


void shape_refresh() // Перерисовка всех фигур
// Здесь используется динамическое связывание!!!
{
    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
// Здесь используется динамическое связывание!!!
{
    point n = q->north();
    point s = p->south();
    p->move(n.x - s.x, n.y - s.y + 1);
}


void imposition(shape* p, const shape* q) // поместить p поверх q
// Здесь используется динамическое связывание!!!
{
    point n = q->south();
    point s = p->south();
    p->move(n.x - s.x , n.y - s.y + 1);
}

void imposition_1(shape* p, const shape* q) // поместить p поверх q
// Здесь используется динамическое связывание!!!
{
    point n = q->swest();
    point s = p->neast();
    p->move(n.x - s.x - 1, n.y - s.y - 1);
}

void imposition_2(shape* p, const shape* q) // поместить p поверх q
// Здесь используется динамическое связывание!!!
{
```

```
    point n = q->seast();
    point s = p->nwest();
    p->move(n.x - s.x + 1, n.y - s.y - 1);
}

//========================================================
#endif
```

screen.h

```
//  screen.h

#ifndef Nasledovanie_screen_h
#define Nasledovanie_screen_h

#include <iostream>

//=== Файл screen.h - поддержка работы с экраном

const int XMAX=80; //Размер экрана
const int YMAX=40;

class point { //Точка на экране
public:
    int x, y;
    point() { };
    point(int a, int b) : x(a), y(b){ }
};

// Набор утилит для работы с экраном
void put_point(int a, int b); // Вывод точки
void put_point(point p) { put_point(p.x, p.y); }
void put_line(int, int, int, int); // Вывод линии
void put_line(point a, point b)
{ put_line(a.x, a.y, b.x, b.y); }
extern void screen_init(); // Создание экрана
extern void screen_destroy(); // Удаление экрана
extern void screen_refresh(); // Обновление
extern void screen_clear(); // Очистка


#endif
```