**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

**Лабораторная работа №1**

**по дисциплине «Алгоритмы и Структуры Данных»**

**Тема: РАБОТА С ИЕРАРХИЕЙ ОБЪЕКТОВ:  
НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ**

|  |  |  |
| --- | --- | --- |
| **Студент гр. 2307** |  | **Подберёзский А.Д.** |
| **Преподаватель** |  | **Манирагена В.** |

**Санкт-Петербург**

**2024**

Оглавление

[1. Цель работы 1](#_Toc18326)

[2. Задание 1](#_Toc8338)

[3. Результат работы программы 2](#_Toc24678)

[4. Выводы 4](#_Toc18048)

[5. Список используемой литературы 4](#_Toc20897)

[6. Приложение 5](#_Toc11619)

1. **Цель работы**

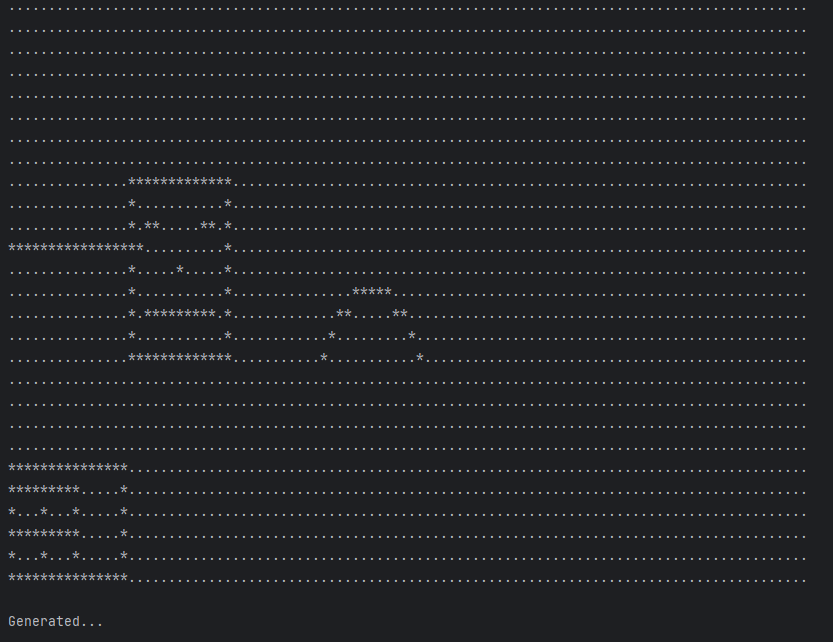
Добавить в коллекцию ещё одну фигуру, согласно выбранному варианту. Для этой фигуры нужно будет оп­ределить подходящее место в иерархии классов и написать недостающие функ­ции-члены.

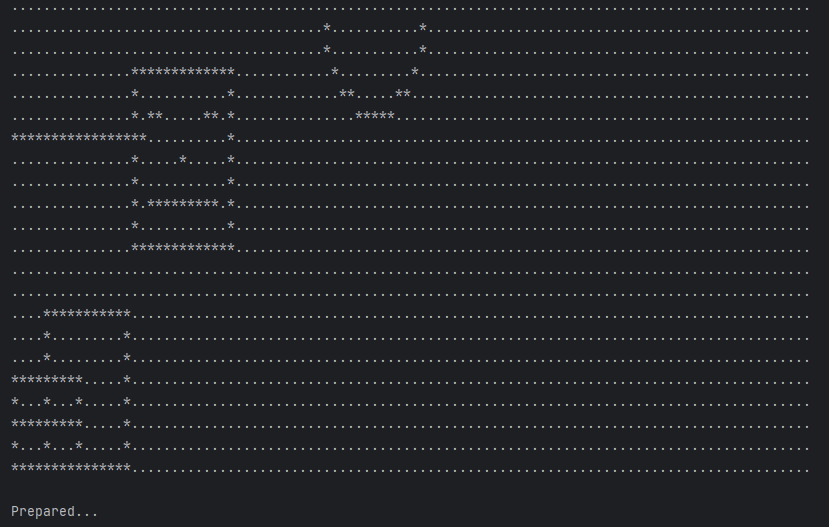
1. **Задание**

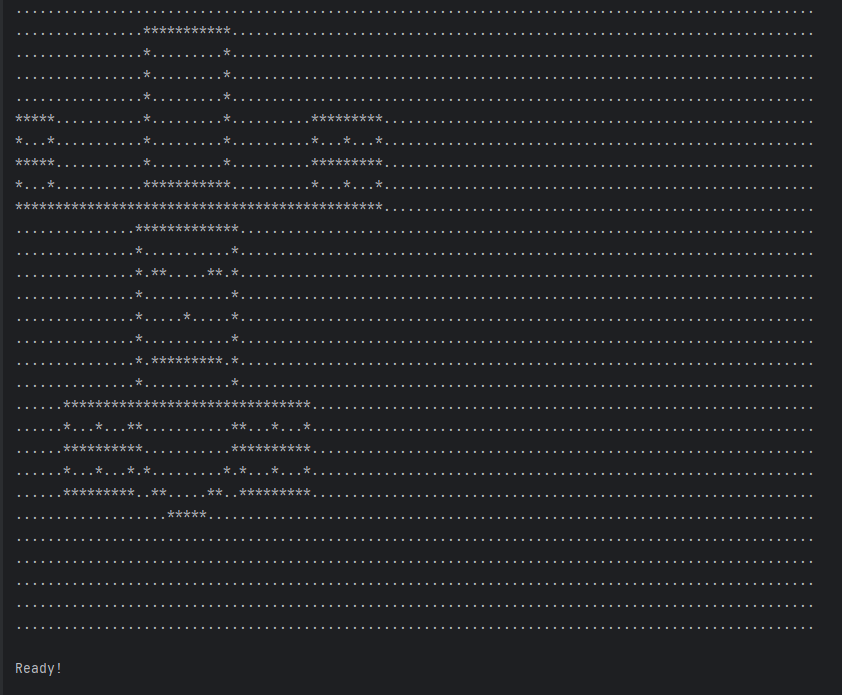
|  |  |  |
| --- | --- | --- |
| № вари- анта | Фигура | Расположение |
| 17 | 12 | 2,3,7,8 |

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование | Отражение | Поворот |
| 12 | Прямоугольник с крестом | Нет | Да |

2 и 3 — бакенбарды,7 и 8 — рога.

1. **Результат работы программы**





1. **Выводы**

В ходе лабораторной работы необходимо было добавить фигуры, предложенные в варианте и наложить её поверх представленной «физиономии», в нескольких позициях. В рамках работы с предоставленным кодом был добавлен класс под названием rectangleWithPlus позволяющий отрисовать данную фигуру в консольном отображении программы и наложить с учётом предоставленных позиции. Для должного позиционирования относительно участков «физиономии» были написаны соответствующие функции.

**5. Список используемой литературы**

1. Пользовательские контейнеры: учебно-метод. пособие. СПб.: СПбГЭТУ «ЛЭТИ», 2024. 64 с. (вып.2402). –– СПб., 2024. — 64 с.

**6. Приложение**

**Файл screen.h**

**const int** XMAX = 100; *//Размер экрана***const int** YMAX = 50;  
  
**class** point { *// Точка на экране***public**:  
 **int** xo, yo;  
 **explicit** point(**int** x = 0, **int** y = 0) : xo(x), yo(y) { }  
};  
*// Набор утилит для работы с экраном***void** put\_point(**int** a, **int** b); *// Вывод точки (2 варианта)***void** put\_point(point p) {  
 put\_point(p.xo, p.yo);  
}  
**void** put\_line(**int**, **int**, **int**, **int**); *// Вывод линии (2 варианта)***void** put\_line(point a, point b) {  
 put\_line(a.xo, a.yo, b.xo, b.yo);  
}  
**void** screen\_init(); *// Создание экрана***void** screen\_destroy(); *// Удаление***void** screen\_refresh(); *// Обновление***void** screen\_clear(); *// Очистка*

**Файл shape.h**

#include **<list>***//==1. Поддержка экрана в форме матрицы символов ==***using namespace** std;  
  
**char** screen[YMAX] [XMAX];  
  
**enum** color {  
 ***black*** = **'\*'**,  
 ***white*** = **'.'**};  
  
**void** screen\_init() {  
 **for** (**auto** & y : screen)  
 **for** (**auto** &x : y) x = ***white***;  
}  
  
**void** screen\_destroy() {  
 **for** (**auto** & y : screen)  
 **for** (**auto** &x : y) x = ***black***;  
}  
  
**bool** on\_screen(**int** a, **int** b) { *// проверка попадания точки на экран* **return** 0 <= a && a < XMAX && 0 <= b && b < YMAX;  
}  
  
**void** put\_point(**int** a, **int** b) {  
 **if** (on\_screen(a,b)) screen[b] [a] = ***black***;  
}  
  
**void** put\_line(**int** x0, **int** y0, **int** x1, **int** y1) {  
 */\*  
 Алгоритм Брезенхэма для прямой:  
 рисование отрезка прямой от (x0,y0) до (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** (;;) { *// Формирование прямой линии по точкам* 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** (**auto** x : screen[y]) *// от левого столбца до правого* cout << x;  
 cout << **'\n'**;  
 }  
}  
*//== 2. Библиотека фигур ==***struct** shape { *// Виртуальный базовый класс "фигура"* **static** list<shape\*> shapes;*// Список фигур (один на все фигуры!)* shape() {  
 shapes.push\_back(**this**);  
 } *//Фигура присоединяется к списку* **virtual** point north() **const** = 0; *//Точки для привязки* **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; *//Рисование* **virtual void** move(**int**, **int**) = 0; *//Перемещение* **virtual void** resize(**int**) = 0; *//Изменение размера*};  
  
list<shape\*> shape::shapes; *// Размещение списка фигур***void** shape\_refresh() { *// Перерисовка всех фигур на экране* screen\_clear();  
 **for** (**auto** p:shape::shapes) p->draw(); *//Динамическое связывание!!!* screen\_refresh();  
}  
  
**class** rotatable:**virtual public** shape { *//Фигуры, пригодные к повороту***public**:  
 **virtual void** rotate\_left() = 0; *//Повернуть влево* **virtual void** rotate\_right() = 0; *//Повернуть вправо*};  
  
**class** reflectable:**virtual public** shape { *// Фигуры, пригодные  
 // к зеркальному отражению***public**:  
 **virtual void** flip\_horizontally() = 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.  
\*/***protected**:  
 point w, e;  
**public**:  
 line (point a, point b) : w(a), e(b) { };  
 line (point a, **int** L) : w(point(a.xo + L - 1, a.yo)), e(a) { };  
 point north() **const override** { **return** point((w.xo + e.xo) / 2, e.yo < w.yo ? w.yo : e.yo); }  
 point south() **const override** { **return** point((w.xo + e.xo) / 2, e.yo < w.yo ? e.yo : w.yo); }  
 point east() **const override** { **return** point(e.xo < w.xo ? w.xo : e.xo, (w.yo + e.yo) / 2); }  
 point west() **const override** { **return** point(e.xo < w.xo ? e.xo : w.xo, (w.yo + e.yo) / 2); }  
 point neast() **const override** { **return** point(w.xo < e.xo ? e.xo : w.xo, e.yo < w.yo ? w.yo : e.yo); }  
 point seast() **const override** { **return** point(w.xo < e.xo ? e.xo : w.xo, e.yo < w.yo ? e.yo : w.yo); }  
 point nwest() **const override** { **return** point(w.xo < e.xo ? w.xo : e.xo, e.yo < w.yo ? w.yo : e.yo); }  
 point swest() **const override** { **return** point(w.xo < e.xo ? w.xo : e.xo, e.yo <w.yo ? e.yo : w.yo); }  
 **void** move(**int** a, **int** b) **override** { w.xo += a; w.yo += b; e.xo += a; e.yo += b; }  
 **void** draw() **override** { put\_line(w, e); }  
 **void** resize(**int** d) **override** { e.xo += (e.xo - w.xo) \* (d - 1); e.yo += (e.yo - w.yo) \* (d - 1); } *// Увеличение длины линии в (d) раз*};  
  
*// Прямоугольник***class** rectangle:**public** rotatable {  
*/\* nw ------ n ------ ne  
 | |  
 | |  
 w c e  
 | |  
 | |  
 sw ------ s ------ se \*/***protected**:  
 point sw, ne;  
**public**:  
 rectangle(point a, point b) : sw(a), ne(b) { }  
 point north() **const override** {  
 **return** point((sw.xo + ne.xo) / 2, ne.yo);  
 }  
 point south() **const override** {  
 **return** point((sw.xo + ne.xo) / 2, sw.yo);  
 }  
 point east() **const override** {  
 **return** point(ne.xo, (sw.yo + ne.yo) / 2);  
 }  
 point west() **const override** {  
 **return** point(sw.xo, (sw.yo + ne.yo) / 2);  
 }  
 point neast() **const override** {  
 **return** ne;  
 }  
 point seast() **const override** {  
 **return** point(ne.xo, sw.yo);  
 }  
 point nwest() **const override** {  
 **return** point(sw.xo, ne.yo);  
 }  
 point swest() **const override** {  
 **return** sw;  
 }  
 **void** rotate\_right() **override** { *// Поворот вправо относительно se* **int** w = ne.xo - sw.xo,  
 h = ne.yo - sw.yo; *// (учитывается масштаб по осям)* sw.xo = ne.xo - h \* 2;  
 ne.yo = sw.yo + w / 2;  
 }  
 **void** rotate\_left() **override** { *// Поворот влево относительно sw* **int** w = ne.xo - sw.xo,  
 h = ne.yo - sw.yo;  
 ne.xo = sw.xo + h \* 2;  
 ne.yo = sw.yo + w / 2;  
 }  
 **void** move(**int** a, **int** b) **override** {  
 sw.xo += a;  
 sw.yo += b;  
 ne.xo += a;  
 ne.yo += b;  
 }  
 **void** resize(**int** d) **override** {  
 ne.xo += (ne.xo - sw.xo) \* (d - 1);  
 ne.yo += (ne.yo - sw.yo) \* (d - 1);  
 }  
 **void** draw() **override** {  
 put\_line(nwest(), ne);  
 put\_line(ne, seast( ));  
  
 put\_line(seast(), sw);  
 put\_line(sw, nwest());  
 }  
};  
  
**void** up(shape& p, **const** shape& q) { *// поместить p над q  
 // Это ОБЫЧНАЯ функция, не член класса! Динамическое связывание!!* point n = q.north();  
 point s = p.south();  
 p.move(n.xo - s.xo, n.yo - s.yo + 1);  
}  
  
**void** pos2(shape& p, **const** shape& q) {  
 point n = q.nwest();  
 point s = p.swest();  
 p.move(n.xo + s.xo - 9, n.yo + s.yo - 12);  
}  
  
**void** pos3(shape& p, **const** shape& q) {  
 point n = q.east();  
 point s = p.seast();  
 p.move(n.xo + s.xo - 7, s.yo + 6);  
}  
  
**void** pos7(shape& p, **const** shape& q) {  
 point n = q.nwest();  
 point s = p.swest();  
 p.move(s.xo - n.xo + 1, n.yo + s.yo);  
}  
  
**void** pos8(shape& p, **const** shape& q) {  
 point n = q.neast();  
 point s = p.seast();  
 p.move(n.xo, n.yo + s.yo);  
}

**Файл shape.cpp**

#include **<iostream>**#include **"screen.h"**#include **"shape.h"  
  
using namespace** std;  
  
*// Пример добавки: дополнительный фрагмент - полуокружность***class** h\_circle:**public** rectangle, **public** reflectable {  
 **bool** reflected;  
**public**:  
 h\_circle(point a, point b, **bool** r=**true**) : rectangle(a, b), reflected(r) { }  
 **void** draw () **override**;  
 **void** flip\_horizontally ( ) **override** { }; *// Отразить горизонтально (пустая функция)* **void** flip\_vertically ( ) **override** {  
 reflected = !reflected;  
 }; *// Отразить вертикально*};  
  
**void** h\_circle::draw() { *// Алгоритм Брезенхэма для окружностей  
 // (выдаются два сектора, указываемые значением reflected)* **int** x0 = (sw.xo + ne.xo) / 2,  
 y0 = reflected ? sw.yo : ne.yo;  
  
 **int** radius = (ne.xo - sw.xo) / 2;  
 **int** x = 0,  
 y = radius,  
 delta = 2 - 2 \* radius,  
 error = 0;  
  
 **while** (y >= 0) { *// Цикл рисования* **if** (reflected) {  
 put\_point(x0 + x, y0 + y \* 0.7);  
 put\_point(x0 - x, y0 + y \* 0.7);  
 } **else** {  
 put\_point(x0 + x, y0 - y\*0.7);  
 put\_point(x0 - x, y0 - y\*0.7);  
 }  
  
 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;  
 }  
}  
  
*// Пример добавки: дополнительная функция присоединения…***void** down(shape &p, **const** shape &q) {  
 point n = q.south();  
 point s = p.north();  
 p.move(n.xo - s.xo, n.yo - s.yo - 1);  
}  
  
*// Cборная пользовательская фигура - физиономия***class** face: **public** rectangle { *// Моя фигура ЯВЛЯЕТСЯ* **int** w, h; *// прямоугольником* line l\_eye; *// левый глаз – моя фигура СОДЕРЖИТ линию* line r\_eye; *// правый глаз* line mouth; *// рот***public**:  
 face(point, point);  
 **void** draw() **override**;  
 **void** move(**int**, **int**) **override**;  
 **void** resize(**int**) **override** { }  
};  
  
face::face(point a, point b) : rectangle(a, b), *// Инициализация базового класса* w(neast().xo - swest().xo + 1), *// Инициализация данных* h(neast().yo - swest().yo + 1), *// строго в порядке объявления!* l\_eye(point(swest().xo + 2, swest().yo + h \* 3 / 4), 2),  
 r\_eye(point(swest().xo + w - 4, swest().yo + h \* 3 / 4), 2),  
 mouth(point(swest().xo + 2, swest().yo + h / 4), w - 4) { }  
  
**void** face::draw() {  
 rectangle::draw(); *// Контур лица (глаза и нос рисуются сами!)* **int** a = (swest().xo + neast().xo) / 2;  
 **int** b = (swest().yo + neast().yo) / 2;  
 put\_point(point(a, b)); *// Нос – существует только на рисунке!*}  
  
**void** face::move(**int** a, **int** b) {  
 rectangle::move(a, b);  
 l\_eye.move(a, b);  
 r\_eye.move(a, b);  
 mouth.move(a, b);  
}  
  
**class** rectangleWithPlus: **public** rectangle {  
 **int** w, h;  
 line verticalLine;  
 line horizontalLine;  
**public**:  
 rectangleWithPlus(point, point);  
 **void** draw() **override**;  
 **void** move(**int**, **int**) **override**;  
 **void** resize(**int**) **override** { }  
};  
  
rectangleWithPlus::rectangleWithPlus(point a, point b) : rectangle(a, b), *// Инициализация базового класса* w(neast().xo - swest().xo + 1),  
 h(neast().yo - swest().yo + 1),  
 verticalLine(point(north().xo, north().yo), point(south().xo, south().yo)),  
 horizontalLine(point(west().xo, west().yo), point(east().xo, east().yo)) { }  
  
**void** rectangleWithPlus::draw() {  
 rectangle::draw();  
 **int** a = (swest().xo + neast().xo) / 2;  
 **int** b = (swest().yo + neast().yo) / 2;  
 put\_point(point(a, b));  
}  
  
**void** rectangleWithPlus::move(**int** a, **int** b) {  
 rectangle::move(a, b);  
 verticalLine.move(a, b);  
 horizontalLine.move(a, b);  
}  
  
**int** main () {  
 screen\_init( );  
  
 *// 1. Объявление набора фигур* rectangle hat(point(0, 0), point(14, 5));  
 line brim(point(0,15),17);  
 face face(point(15, 10), point(27, 18));  
 h\_circle beard(point(40,10), point(50,20));  
 rectangleWithPlus rectangleWithPlus2(point(0, 0), point(8, 4));  
 rectangleWithPlus rectangleWithPlus3(point(0, 0), point(8, 4));  
 rectangleWithPlus rectangleWithPlus7(point(0, 0), point(8, 4));  
 rectangleWithPlus rectangleWithPlus8(point(0, 0), point(8, 4));  
 shape\_refresh();  
 cout << endl << **"Generated..."** << endl << endl;  
 *//cin.get();  
  
 // 2. Подготовка к сборке* hat.rotate\_right();  
 brim.resize(2);  
 face.resize(2);  
 beard.flip\_vertically();  
 shape\_refresh();  
 cout << endl << **"Prepared..."** << endl << endl;  
 *// cin.get(); // Смотреть результат поворотов/отражений  
  
 // 3.Сборка изображения  
 // face.move(0, -10); // Лицо - в исходное положение* up(brim, face);  
 up(hat, brim);  
 pos2(rectangleWithPlus2, face);  
 pos3(rectangleWithPlus3, face);  
 pos7(rectangleWithPlus7, brim);  
 pos8(rectangleWithPlus8, brim);  
 down(beard, face);  
 shape\_refresh();  
 cout << endl << **"Ready!"** << endl << endl;  
 *// cin.get(); // Смотреть результат* screen\_destroy( );  
 **return** 0;  
}