**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №1**

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

Тема: «Работа с иерархией объектов: наследование и полиморфизм»

|  |  |  |
| --- | --- | --- |
| Студенты гр. 1307 |  | Лицеванова М.М. |
|  |  | Пименов Г.А. |
| Преподаватель |  | Манирагена В. |

Санкт-Петербург

2023

**Задание (вариант 8)**

**Доработать модуль *shape.cpp***, добавив в коллекцию еще одну фигуру, номер которой указан в табл. 1.2 в строке с вашим вариантом. Для этой фигуры нужно оп­ределить подходящее место в иерархии классов и написать недостающие функ­ции-члены. Конструктор копии и другие генерируемые компилятором функции-члены, использование которых не предполагается, рекомендуется сделать недоступными. Грамотная иерархия наследования позволит сократить количество необходимых функций-членов. Переопределять память, имеющуюся в базовом классе, вообще не следует, но можно изменить ее смысл, если наследуемые функции-члены это позволяют.

Разработанной фигурой нужно дополнить картинку в указанных в варианте позициях (рис. 1.3). 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* — эмблему на шляпе, *13* и *15* — перья, *14* — шишак. Возможно, некоторые из фигур нужно будет повернуть или отразить. Так, в позициях *2*, *4*, *7*, *10* и *13* фигура должна быть повернута (или отражена) влево; *3*, *5*, *8*, *11* и *15* — вправо; *14* — отражена вверх; *1* и *9* — вниз.

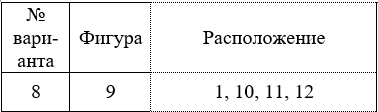
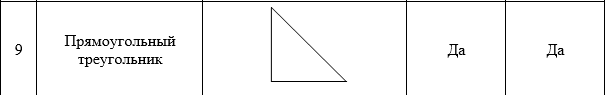
Для фигур, назначенных в позиции *6*, *9*, *10*, *11* и *12*, допускается замена отношения между классами ЯВЛЯЕТСЯ отношением СОДЕР­ЖИТ (см. комментарии в определении класса *myshape*).



*Рис*. *1.3*. Позиции  
для присоединения  
дополнительных фигур

Для примыкания фигур должны использоваться их габаритные точки. Необходимо напи­сать аналоги функции *up* (поместить *p* над *q*), обеспечивающие примыкание очередной фигуры *p* с нужной стороны по отношению к уже размещенной *q*. Это должны быть обычные функции, а не функции-члены класса, чтобы их можно было применять для любых объектов. Категорически запрещается вычислять или подбирать координаты размещения фигуры вручную по готовой картинке и задавать их затем константами в функции move( ).

При проектировании класса фигуры, допускающей поворот и/или отражение, рекомендуется отделить информацию, необходимую для рисования фигуры с учетом ее ориентации, и информацию для вычисления положения точек привязки, не зависящих от ориентации.

** **

**Иерархия классов**

Отражаемая

Вращаемая

Фигура

shape

1. **Какие классы пришлось добавить?**

Был добавлен класс прямоугольного треугольника (r\_triangle), который наследует прямоугольник (rectangle) и класс, работающий с отражениями (reflectable). (Множественное наследование) Была добавлена переменная (rotatable) для определения положения наклона треугольника.

1. **Какие функции-члены пришлось переопределить и почему?**

Были переопределены методы поворота (rotate\_right, rotate\_left) и отрисовки (draw), унаследованные от прямоугольника, а также методы отражения (flip\_horizontally, flip\_vertically) от класса для работы с отражением фигур.

1. **Какие модификаторы доступа были выбраны для функций членов?**

Все методы были сделаны public, так как они вызываются в отдельной функции main и содержат все необходимые проверки, так что сохраняется принцип инкапсуляции (в случае методов поворота и отражения)

1. **Какие дополнительные методы были реализованы?**

Была добавлена функция для присоединения одной фигуры снизу другой (down) и для присоединения одной фигуры в центр другой (center)

#include <iostream>  
using namespace std;  
// Пополнение и использование библиотеки фигур  
#include "screen.h"  
#include "shape.h"  
  
// дополнительная функция присоединения снизу  
void down(shape &p, const shape &q)  
{ point n = q.south( );  
 point s = p.north( );  
 p.move(n.x - s.x, n.y - s.y - 1); }  
  
// дополнительная функция размещения в центре  
void center(shape &p, const shape &q)  
{  
 point cq;  
 point cp;  
 cq.x = q.north().x;  
 cq.y = q.west().y;  
 cp = p.south();  
 p.move(cq.x - cp.x, cq.y - cp.y - 1);  
}  
  
//Дополнительный фрагмент - прямоугольный треугольник  
class r\_triangle: public rectangle, public reflectable {  
 /\*  
 nw ------ n ------ ne  
 | \* \* |  
 | \* \* |  
 w \* \* e  
 | \* \* |  
 | \* \* |  
 sw------- s ------ se  
 \*/  
 bool reflected;  
 int rotated = 1; // переменная для определения положения наклона треугольника  
public:  
 r\_triangle(point a, point b, bool r=true) : rectangle(a, b), reflected(r) { }  
 void draw();  
 void flip\_horisontally( ); // Отразить горизонтально (пустая функция)  
 void flip\_vertically( ); //{ reflected = !reflected; }; // Отразить вертикально  
 void rotate\_right();  
 void rotate\_left();  
};  
// переопределение функции-члена для рисования фигуры  
void r\_triangle :: draw()  
{  
 if (rotated == 1) {  
 put\_line(seast(), sw);  
 put\_line(sw, nwest());  
 put\_line(nwest(), seast());  
 } else if (rotated == 2) {  
 put\_line(nwest(), ne);  
 put\_line(sw, ne);  
 put\_line(nwest(), sw);  
 } else if (rotated == 3) {  
 put\_line(nwest(), ne);  
 put\_line(seast(), ne);  
 put\_line(seast(), nwest());  
 } else if (rotated == 4) {  
 put\_line(sw, ne);  
 put\_line(seast(), ne);  
 put\_line(seast(), sw);  
 }  
}  
// переопределение функции-члена для поворота направо  
void r\_triangle :: rotate\_right() {  
 int w = ne.x - sw.x, h = ne.y - sw.y; //(учитывается масштаб по осям)  
 sw.x = ne.x - h \* 2; ne.y = sw.y + w / 2;  
  
 rotated += 1;  
 if (rotated == 5) {  
 rotated = 1;  
 }  
}  
// переопределение функции-члена для поворота налево  
void r\_triangle :: rotate\_left() {  
 int w = ne.x - sw.x, h = ne.y - sw.y;  
 ne.x = sw.x + h \* 2; ne.y = sw.y + w / 2;  
  
 rotated -= 1;  
 if (rotated == 0) {  
 rotated = 4;  
 }  
}  
// переопределение функции-члена для отражения по-горизонтали  
void r\_triangle :: flip\_horisontally( ) {  
 point new\_sw;  
 point new\_ne;  
 new\_sw.y = sw.y + (sw.y - nwest().y);  
 new\_sw.x = sw.x;  
 new\_ne = seast();  
  
 this->ne = new\_ne;  
 this->sw = new\_sw;  
  
 if (rotated == 1) {  
 this->rotated = 2;  
 } else if (rotated == 2) {  
 this->rotated = 1;  
 } else if (rotated == 3) {  
 this->rotated = 4;  
 } else if (rotated == 4) {  
 this->rotated = 3;  
 }  
}  
// переопределение функции-члена для отражения по-вертикали  
void r\_triangle :: flip\_vertically() {  
 point new\_sw;  
 point new\_ne;  
 new\_ne.x = ne.x + (ne.x - nwest().x);  
 new\_ne.y = ne.y;  
 new\_sw = seast();  
  
 this->ne = new\_ne;  
 this->sw = new\_sw;  
  
  
 if (rotated == 1) {  
 this->rotated = 4;  
 } else if (rotated == 2) {  
 this->rotated = 3;  
 } else if (rotated == 3) {  
 this->rotated = 2;  
 } else if (rotated == 4) {  
 this->rotated = 1;  
 }  
}  
  
// Cборная пользовательская фигура - физиономия  
class myshape : public rectangle { // Моя фигура ЯВЛЯЕТСЯ  
 int w, h; // прямоугольником  
   
 line mouth; // рот  
 public:  
 //глаза перемещены в public для поворота и отражения треугольников  
 r\_triangle l\_eye; // левый глаз   
 r\_triangle r\_eye; // правый глаз  
 myshape(point, point);  
 void draw( );  
 void move(int, int);  
 void resize(int) { }  
};  
myshape :: myshape(point a, point b)  
 : rectangle(a, b), //Инициализация базового класса  
 w(neast( ).x - swest( ).x + 1), // Инициализация данных  
 h(neast( ).y - swest( ).y + 1), // - строго в порядке объявления!  
 l\_eye(point(swest( ).x + w/8 + 1, swest( ).y + h \* 3 / 4), point(swest( ).x + w/8 + 1 + w/4, swest( ).y + h \* 3 / 4 + h/6)),  
 r\_eye(point(swest( ).x + w - w/8 - 1 - 1 - w/4, swest( ).y + h \* 3 / 4), point(swest( ).x + w - w/8 - 1 - 1, swest( ).y + h \* 3 / 4 + h/6)),  
 mouth(point(swest( ).x + 2, swest( ).y + h / 4), w - 4)   
{ }  
void myshape :: draw( )  
{  
 rectangle :: draw( ); //Контур лица (глаза и нос рисуются сами!)   
 int a = (swest( ).x + neast( ).x) / 2;  
 int b = (swest( ).y + neast( ).y) / 2;  
 put\_point(point(a, b)); // Нос – существует только на рисунке!  
}  
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);  
}  
int main( )   
{ setlocale(LC\_ALL, "Rus");  
 screen\_init( );  
//== 1.Объявление набора фигур ==  
 rectangle hat(point(0, 0), point(14, 5)); // прямоугольная шляпа  
 line brim(point(0,15),17); // линия-край шляпы  
 myshape face(point(15,10), point(27,18)); // класс-лицо  
 r\_triangle beard(point(40,10), point(50,20)); // прямоугольный треугольник - борода  
 r\_triangle pizza(point(60,20), point(65,23)); // прямоугольный треугольник - эмблема на шляпе  
 shape\_refresh( );  
 std::cout << "=== Generated... ===\n";  
 std::cin.get(); //Смотреть исходный набор  
//== 2.Подготовка к сборке ==  
 hat.rotate\_right( );  
 brim.resize(2);  
 face.resize(2);  
 beard.flip\_horisontally(); // бородп отражается по горизонтали  
 face.l\_eye.rotate\_left(); // левый глаз поворачивается налево  
 face.r\_eye.rotate\_right(); // правый глаз поворачивается направо  
 shape\_refresh( );  
 std::cout << "=== Prepared... ===\n";  
 std::cin.get(); //Смотреть результат поворотов/отражений  
//== 3.Сборка изображения ==  
// face.move(0, -10); // Лицо - в исходное положение  
 up(brim, face);  
 up(hat, brim);  
 down(beard, face); // борода присоединяется к лицу снизу  
 center(pizza, hat); // эмблема присоединяется в центр шляпы  
 shape\_refresh( );  
 std::cout << "=== Ready! ===\n";  
 std::cin.get(); //Смотреть результат  
 screen\_destroy( );  
 return 0;  
}

**Результат:**

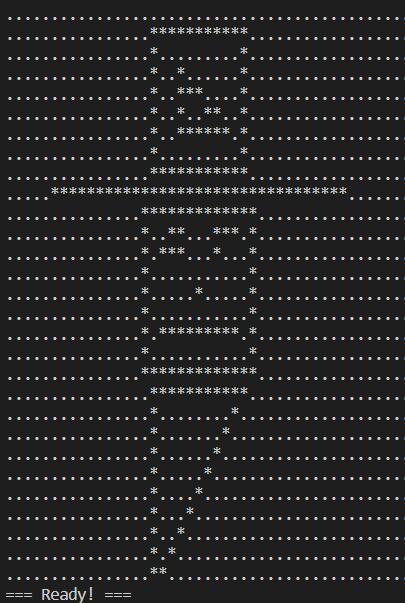
**Фигуры, необходимые для рисунка**



**Фигуры, подготовлены для рисунка**



**Фигуры, собраны в рисунок**



**Выводы**

В результате проделанной работы были изучены классы, разобраны методы исследования, переопределения функций, был добавлен класс, который был определён в иерархии классов, добавлены новые методы и в конченом итоге создано новое изображение согласно заданию.