Министерство науки и образования РФ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

**Отчёт по заданию № 1**

**на тему: “Работа с иерархией объектов: наследование и полиморфизм”**

**по дисциплине**

**“Алгоритмы и структуры данных”**

**Вариант 44**

|  |  |
| --- | --- |
| Выполнили студенты гр.9308: | Дементьев Д.П., Ручкин Д.А. |
| Проверил: | Колинько П.Г. |

Санкт-Петербург, 2021 г.

Оглавление

[Введение 4](#_Toc65424232)

[1. Задание 4](#_Toc65424233)

[2. Формализация задания 4](#_Toc65424234)

[3. Контрольные примеры 5](#_Toc65424240)

[4. Работа с наследованием объектов 5](#_Toc65424242)

[5. Результаты эксперимента 6](#_Toc65424243)

[Вывод 8](#_Toc65424246)

[Список используемых источников 9](#_Toc65424247)

[Приложение 1 (Исходный текст программы) 10](#_Toc65424248)

## Введение

Целью задания является ознакомление с принципами ООП, в частности, с принципами наследования и полиморфизма, использование библиотеки «из пакета» и последующего расширения его функционала.

## 1. Задание

Доработать модуль shape.cpp, добавив в коллекцию еще одну фигуру - окружность. Для этой фигуры нужно определить подходящее место в иерархии классов и написать недостающие функции-члены.

Разработанной фигурой нужно дополнить картинку в указанных в варианте позициях. Необходимо написать функции, обеспечивающие примыкание очередной фигуры p с нужной стороны по отношению к уже размещенной q.

## 2. Формализация задания

## Имеется схема объекта с пометками возможных точек присоединения (рис.1), в нашему случае такими точками будут: 12, 2, 3

## 

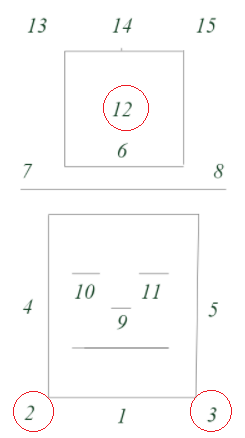
## *Рис.1. Возможные точки стыковки*

## Фигурой для присоединения будет использоваться окружность – примитивная фигура.

## Также для присоединения нашей фигуры к точке 12 потребуется функция центровки двух фигур относительно друг друга, а для точек 2 и 3 функции присоединения по северо-восточной – юго-западной и северо-западной – юго-восточной точкам соответственно.

## 3. Контрольные примеры

В качестве контрольного примера выберем ожидаемый итоговый рисунок, не учитывая конкретные размеры окружности. Добавленные атрибуты выделены красным цветом на Рис. 2: эмблема на шляпе(т.12), бакенбарды(т.2 и т.3):



## *Рис.2. Ожидаемый результат(контрольный пример)*

## 4. Работа с наследованием объектов

**4.1. Добавленные классы**

Для реализации примитивной фигуры окружности был создан класс circle, являющийся производным от класса shape – абстрактного представления фигур. Наследование от классов rotatable и reflectableявлялось бы избыточным т.к. из-за симметричности окружности смысла в повороте или отражении не имеется.

**4.2. Переопределение функций-членов у добавленного класса**

По причине того, что класс-родитель shape является первым и единственным в иерархии наследования для производного класса circle – нам следует переопределить ВСЕ изначально заданные виртуальные(абстрактные) функции-члены в классе shape, т.к. ранее они не могли быть переопределены промежуточными классами.

**4.3. Реализация недоступности необязательных функций-членов класса**

Было решено сделать недоступными четыре функции-члена: конструктор копирования, конструктор перемещения, оператор присваивания копированием и оператор присваивания перемещением.

Способом реализации было выбрано изменение модификаторов доступа функций-членов на приватный. Таким образом, при попытке воспользоваться этими функциями возникнет ошибка на этапе компиляции программы.

## 5. Результаты эксперимента

Для начала убедимся в корректности создания и отрисовки добавленной фигуры и создадим три окружности радиусов 2, 8 и 10 соответственно:



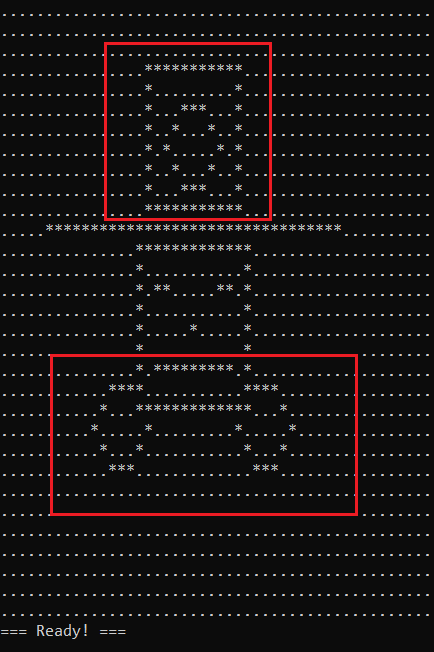
*Рис.3. Отрисовка исходных фигур*

Сформируем бакенбарды из двух наибольших окружностей путём их уменьшения в 4 и 5 раз соответственно:



## *Рис.4. Преобразование фигур*

Уже подготовленные фигуры состыкуем на соответствующие позиции:



## *Рис.5. Преобразование фигур*

При получении результата нужно учитывать неидеальность принятой системы отображений фигур на экране, а потому может сложится ситуация, когда, например, высота или(и) длина имеют четные значения, то точку «идеального центра» выделить не удастся и появится двойственность в выборе этой точке, которая в свою очередь будет решаться округлением арифметических действий в языке С++.

## Вывод

Работа реализована на основе базового класса Фигур shape, определенного внутри исходного пакета(shape.h), сама «Фигура» это понятие абстрактное и не имеет смысла без «доопределения» до какой-то существующей геометрической фигуры, но уже имеет некоторый набор функций-членов, которые являются общими для любой из «доопределенных» фигур. Сама абстрактность базового класса достигается путём использования виртуальных методов, которые в обязательном порядке должны быть также доопределены в производных классах.

В этой схеме **принцип наследования** будет заключаться в создании производных классов от базового класса shape, **принцип полиморфизма** же – в обязательном наличии функций-членов у производного класса, которые в свою очередь являются доопределением виртуальных функций-членов базового класса.

## Список используемых источников

1. Колинько П.Г. Пользовательские контейнеры / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020.
2. Понимание lvalue и rvalue в C и С++. URL: <https://habr.com/ru/post/348198/> [дата обращения 25.02]
3. Урок №163. Виртуальные функции и Полиморфизм. URL: <https://ravesli.com/urok-163-virtualnye-funktsii-i-polimorfizm/> [дата обращения 26.02]
4. Урок №114. Спецификаторы доступа public и private. <https://ravesli.com/urok-114-spetsifikatory-dostupa-public-i-private/>

## Приложение 1 (Исходный текст программы)

Добавленный текст программы выделен чёрным цветом:

1. //========== Файл shape.cpp (прикладная программа) ==========
2. // Пополнение и использование библиотеки фигур
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <iostream>
6. #include "screen.h"
7. #include "shape.h"
8. class circle : public shape {
9. protected:
10. point c;
11. int radius;
12. public:
13. circle(point a, int r) : c(a), radius(r) { }
14. void draw();
15. point north() const { return point(c.x,c.y+radius); }
16. point south() const { return point(c.x,c.y-radius); }
17. point east() const { return point(c.x+radius,c.y); }
18. point west() const { return point(c.x-radius,c.y); }
19. point neast() const { return point(c.x+radius\*0.7, c.y+radius\*0.7); }
20. point seast() const { return point(c.x+radius\*0.7, c.y-radius\*0.7); }
21. point nwest() const { return point(c.x-radius\*0.7, c.y+radius\*0.7); }
22. point swest() const { return point(c.x-radius\*0.7, c.y-radius\*0.7); }
23. void move(int a , int b) { c.x += a; c.y += b; } //Перемещение
24. void resize(int d) { //Изменение размера
25. (d>0) ? radius \*= d : radius /= -d; std::cout<<radius<<std::endl;}
26. private: // закрываем доступ к функциям, использование которых не предполагается
27. circle(const circle &x) { std::cout << "copy constructor\n"; } // конструктор копирования
28. circle& operator=(const circle &x) { }// оператор присваивания копированием
29. circle(circle &&x) { std::cout << "move constructor\n"; } // конструктор перемещения
30. circle& operator=(circle &&x) { } // оператор присваивания перемещением
31. };
32. void circle::draw() {
33. int x = 0, y = radius, delta = 1 - 2 \* radius, error = 0;
34. while (y >= 0) {
35. put\_point(c.x + x, c.y + y);
36. put\_point(c.x + x, c.y - y);
37. put\_point(c.x - x, c.y + y);
38. put\_point(c.x - x, c.y - y);
39. error = 2 \* (delta + y) - 1;
40. if (delta < 0 && error <= 0) {
41. ++x;
42. delta += 2 \* x + 1;
43. continue;
44. }
45. error = 2 \* (delta - x) - 1;
46. if (delta > 0 && error > 0) {
47. --y;
48. delta += 1 - 2 \* y;
49. continue;
50. }
51. ++x;
52. delta += 2 \* (x - y);
53. --y;
54. }
55. }
56. // ПРИМЕР ДОБАВКИ: дополнительный фрагмент - полуокружность
57. class h\_circle: public rectangle, public reflectable {
58. bool reflected;
59. public:
60. h\_circle(point a, point b, bool r=true) : rectangle(a, b), reflected(r) { }
61. void draw();
62. void flip\_horisontally( ) { }; // Отразить горизонтально (пустая функция)
63. void flip\_vertically( ) { reflected = !reflected; }; // Отразить вертикально
64. };
65. void h\_circle :: draw() //Алгоритм Брезенхэма для окружностей
66. { //(выдаются два сектора, указываемые значением reflected)
67. int x0 = (sw.x + ne.x)/2, y0 = reflected ? sw.y : ne.y;
68. int radius = (ne.x - sw.x)/2;
69. int x = 0, y = radius, delta = 2 - 2 \* radius, error = 0;
70. while(y >= 0) { // Цикл рисования
71. if(reflected) { put\_point(x0 + x, y0 + y\*0.7); put\_point(x0 - x, y0 + y\*0.7); }
72. else { put\_point(x0 + x, y0 - y\*0.7); put\_point(x0 - x, y0 - y\*0.7); }
73. error = 2 \* (delta + y) - 1;
74. if(delta < 0 && error <= 0) { ++x; delta += 2 \* x + 1; continue; }
75. error = 2 \* (delta - x) - 1;
76. if(delta > 0 && error > 0) { --y; delta += 1 - 2 \* y; continue; }
77. ++x; delta += 2 \* (x - y); --y;
78. }
79. }
80. // ПРИМЕР ДОБАВКИ: дополнительная функция присоединения…
81. void down(shape &p, const shape &q) {
82. point n = q.south( );
83. point s = p.north( );
84. p.move(n.x - s.x, n.y - s.y - 1);
85. }
86. void centering(shape &p, const shape &q) {
87. int delta\_x = (q.east().x + q.west().x)/2 - (p.east().x + p.west().x)/2, // (центр фигуры q) - (центр фигуры p), компонента икс
88. delta\_y = (q.east().y + q.west().y)/2 - (p.east().y + p.west().y)/2; // (центр фигуры q) - (центр фигуры p), компонента игрек
89. p.move(delta\_x, delta\_y);
90. }
91. void rightDown(shape &p, shape&q) {
92. point n = q.seast();
93. point s = p.nwest();
94. p.move(n.x - s.x, n.y - s.y);
95. }
96. void leftDown(shape &p, shape &q){
97. point n = q.swest();
98. point s = p.neast();
99. p.move(n.x - s.x - 1, n.y - s.y);
100. }
101. // Cборная пользовательская фигура - физиономия
102. class myshape : public rectangle { // Моя фигура ЯВЛЯЕТСЯ
103. int w, h; // прямоугольником
104. line l\_eye; // левый глаз – моя фигура СОДЕРЖИТ линию
105. line r\_eye; // правый глаз
106. line mouth; // рот
107. public:
108. myshape(point, point);
109. void draw( );
110. void move(int, int);
111. void resize(int) { }
112. };
113. myshape :: myshape(point a, point b)
114. :rectangle(a, b), //Инициализация базового класса
115. w(neast( ).x - swest( ).x + 1), // Инициализация данных
116. h(neast( ).y - swest( ).y + 1), // - строго в порядке объявления!
117. l\_eye(point(swest( ).x + 2, swest( ).y + h \* 3 / 4), 2),
118. r\_eye(point(swest( ).x + w - 4, swest( ).y + h \* 3 / 4), 2),
119. mouth(point(swest( ).x + 2, swest( ).y + h / 4), w - 4)
120. { }
121. void myshape :: draw( )
122. {
123. rectangle :: draw( ); //Контур лица (глаза и нос рисуются сами!)
124. int a = (swest( ).x + neast( ).x) / 2;
125. int b = (swest( ).y + neast( ).y) / 2;
126. put\_point(point(a, b)); // Нос – существует только на рисунке!
127. }
128. void myshape :: move(int a, int b)
129. {
130. rectangle :: move(a, b);
131. l\_eye.move(a, b); r\_eye.move(a, b);
132. mouth.move(a, b);
133. }
134. int main( )
135. {
136. setlocale(LC\_ALL, "Rus");
137. screen\_init( );
138. //== 1.Объявление набора фигур ==
139. rectangle hat(point(0, 0), point(14, 5));
140. line brim(point(0,15),17);
141. circle emblem(point(60, 15), 2), whisker\_left(point(70,10), 10), whisker\_right(point(70,20), 8);
142. myshape face(point(15,10), point(27,18));
143. h\_circle beard(point(40,10), point(50,20));
144. shape\_refresh( );
145. std::cout << "=== Generated... ===\n";
146. std::cin.get(); //Смотреть исходный набор
147. //== 2.Подготовка к сборке ==
148. hat.rotate\_right( );
149. brim.resize(2);
150. face.resize(2);
151. beard.flip\_vertically();
152. whisker\_left.resize(-5);
153. whisker\_right.resize(-4);
154. shape\_refresh( );
155. std::cout << "=== Prepared... ===\n";
156. std::cin.get(); //Смотреть результат поворотов/отражений
157. //== 3.Сборка изображения ==
158. up(brim, face);
159. up(hat, brim);
160. down(beard, face);
161. centering(emblem, hat);
162. rightDown(whisker\_left, face);
163. leftDown(whisker\_right, face);
164. shape\_refresh( );
165. std::cout << "=== Ready! ===\n";
166. std::cin.get(); //Смотреть результат
167. screen\_destroy( );
168. return 0;
169. }