МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ |

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | Е.О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4 |
| Наследование классов, базовый класс, производный класс |
| по дисциплине: Объектно-ориентированное программирование |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | Z1431 |  |  |  | М.Д. Быстров |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 2021/3572 | |  |  |  |

Санкт-Петербург 2023

**Условие**

Цель работы: изучить механизм создания нового класса на основе уже существующего, варианты доступа к элементам базового класса из производного.

Закрепить знания по теме: классы, наследование классов, варианты доступа.

Описание работы: в работе необходимо реализовать базовый класс заданной структуры, на основе него создать производные классы. В нём предусмотреть конструктор для установки начальных значений полей. Создать объекты производных классов. Продемонстрировать работу всех методов, реализуемых в классах.

Индивидуальное задание (вариант 2):

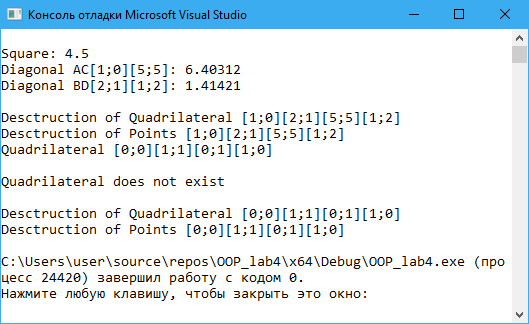
2. Создать класс Points для хранения координат четырёх точек A, B, C и D на плоскости. В классе предусмотреть возможность распечатки координат каждой точки по отдельности и всех разом. На основе класса Points создать класс Quadrilateral для работы с четырёхугольником. Предусмотреть методы для проверки существования четырёхугольника, нахождения площади и диагоналей.

**Полный текст (листинг) программы**

1. Файл «main.cpp»
2. #include <iostream>
3. #include "Quadrilateral.h"
4. #include "main.h"
6. **#define QUADR\_COORDS 1,0, 2,1, 5,5, 1,2**
7. #define INVALID\_COORDS 0,0, 1,1, 0,1, 1,0
9. using namespace std;
11. **void CheckQuadrilateral(Quadrilateral\* quadr)**
12. {
13. cout << "Quadrilateral " << quadr->getAllPointsStr() << endl << endl;
15. if (quadr->isReal())
16. **{**
17. cout << "Square: " << quadr->getSquare() << endl;
18. cout << "Diagonal AC" << quadr->getAStr() << quadr->getCStr() << ": " << quadr->getACLength() << endl;
19. cout << "Diagonal BD" << quadr->getBStr() << quadr->getDStr() << ": " << quadr->getBDLength() << endl;
20. }
21. **else**
22. {
23. cout << "Quadrilateral does not exist" << endl;
24. }
26. **cout << endl;**
28. delete quadr;
29. }
31. **int main()**
32. {
33. Quadrilateral\* quadr1 = new Quadrilateral(QUADR\_COORDS);
34. Quadrilateral\* quadr2 = new Quadrilateral(INVALID\_COORDS);
36. **CheckQuadrilateral(quadr1);**
37. CheckQuadrilateral(quadr2);
38. }
39. Файл «Points.h»
40. #include <string>
42. #pragma once
43. class Points
44. **{**
45. private:
47. int ax;
48. int ay;
50. int bx;
51. int by;
53. int cx;
54. **int cy;**
56. int dx;
57. int dy;
59. **public:**
61. Points(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
62. ~Points();
64. **int getAX() const;**
65. void setAX(int x1);
67. int getAY() const;
68. void setAY(int y1);
70. int getBX() const;
71. void setBX(int x2);
73. int getBY() const;
74. **void setBY(int y2);**
76. int getCX() const;
77. void setCX(int x3);
79. **int getCY() const;**
80. void setCY(int y3);
82. int getDX() const;
83. void setDX(int x4);
85. int getDY() const;
86. void setDY(int y4);
88. std::string getAllPointsStr() const;
90. std::string getAStr() const;
91. std::string getBStr() const;
92. std::string getCStr() const;
93. std::string getDStr() const;
94. **};**
95. Файл «Points.cpp»
96. #include <stdlib.h>
97. #include <utility>
98. #include <sstream>
99. #include <iostream>
100. **#include "Points.h"**
102. using namespace std;
104. Points::Points(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
105. **{**
106. this->ax = x1;
107. this->ay = y1;
109. this->bx = x2;
110. **this->by = y2;**
112. this->cx = x3;
113. this->cy = y3;
115. **this->dx = x4;**
116. this->dy = y4;
117. }
119. Points::~Points()
120. **{**
121. cout << "Desctruction of Points " << this->getAllPointsStr() << endl;
122. }
124. int Points::getAX() const { return this->ax; }
125. **void Points::setAX(int ax) { this->ax = ax; }**
126. int Points::getAY() const { return this->ay; }
127. void Points::setAY(int ay) { this->ay = ay; }
129. int Points::getBX() const { return this->bx; }
130. **void Points::setBX(int bx) { this->bx = bx; }**
131. int Points::getBY() const { return this->by; }
132. void Points::setBY(int by) { this->by = by; }
134. int Points::getCX() const { return this->cx; }
135. **void Points::setCX(int cx) { this->cx = cx; }**
136. int Points::getCY() const { return this->cy; }
137. void Points::setCY(int cy) { this->cy = cy; }
139. int Points::getDX() const { return this->dx; }
140. **void Points::setDX(int dx) { this->dx = dx; }**
141. int Points::getDY() const { return this->dy; }
142. void Points::setDY(int dy) { this->dy = dy; }
144. string Points::getAllPointsStr() const
145. **{**
146. return this->getAStr() + this->getBStr() + this->getCStr() + this->getDStr();
147. }
149. string coordsToStr(int a, int b)
150. **{**
151. std::stringstream ss;
153. ss << "[" << a << ";" << b << "]";
155. **return ss.str();**
156. }
158. string Points::getAStr() const
159. {
160. **return coordsToStr(this->ax, this->ay);**
161. };
163. string Points::getBStr() const
164. {
165. **return coordsToStr(this->bx, by);**
166. };
168. string Points::getCStr() const
169. {
170. **return coordsToStr(this->cx, this->cy);**
171. };
173. string Points::getDStr() const
174. {
175. **return coordsToStr(this->dx, this->dy);**
176. }
177. Файл «Quadrilateral.h»
178. #pragma once
179. #include "Points.h"
180. class Quadrilateral :
181. public Points
182. **{**
183. public:
185. Quadrilateral(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
186. ~Quadrilateral();
188. double getSquare() const;
190. double getACLength() const;
192. **double getBDLength() const;**
194. bool isReal() const;
195. };

198. Файл «Quadrilateral.cpp»
199. #include <stdlib.h>
200. #include <utility>
201. #include <stdexcept>
202. #include <iostream>
203. **#include "Quadrilateral.h"**
205. using namespace std;
207. inline int area(int ax, int ay, int bx, int by, int cx, int cy)
208. **{**
209. return (bx - ax) \* (cy - ay) - (by - ay) \* (cx - ax);
210. }
212. inline bool intersect\_1(int a, int b, int c, int d)
213. **{**
214. if (a > b)
215. {
216. swap(a, b);
217. }
219. if (c > d)
220. {
221. swap(c, d);
222. }
224. return max(a, c) <= min(b, d);
225. }
227. bool intersect(int ax, int ay, int bx, int by, int cx, int cy, int dx, int dy) {
228. **return intersect\_1(ax, bx, cx, dx)**
229. && intersect\_1(ay, by, cy, dy)
230. && area(ax, ay, bx, by, cx, by) \* area(ax, ay, bx, by, dx, dy) <= 0
231. && area(cx, cy, dx, dy, ax, ay) \* area(cx, cy, dx, dy, bx, by) <= 0;
232. }
234. Quadrilateral::Quadrilateral(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) :
235. Points(x1, y1, x2, y2, x3, y3, x4, y4)
236. {
237. }
239. Quadrilateral::~Quadrilateral()
240. {
241. cout << "Desctruction of Quadrilateral " << this->getAllPointsStr() << endl;
242. }
244. double Quadrilateral::getSquare() const
245. {
246. const int topsCount = 4;
248. **int xArray[topsCount] = {**
249. this->getAX(),
250. this->getBX(),
251. this->getCX(),
252. this->getDX()};
254. int yArray[topsCount] = {
255. this->getAY(),
256. this->getBY(),
257. this->getCY(),
258. **this->getDY() };**
260. int sum = 0;
262. for (int i = 0; i < topsCount; i++)
263. **{**
264. sum += xArray[i] \* yArray[(i + 1) % topsCount];
265. sum -= xArray[(i + 1) % topsCount] \* yArray[i];
266. }
268. **sum = abs(sum);**
270. double square = (double)sum / (double)2;
272. return square;
273. **}**
275. double hypo(int x, int y)
276. {
277. double sumOfCatets = pow(x, 2) + pow(y, 2);
279. double ret = sqrt(sumOfCatets);
281. return ret;
282. }
284. double Quadrilateral::getACLength() const
285. {
286. return hypo(
287. abs(this->getAX() - this->getCX()),
288. **abs(this->getAY() - this->getCY()));**
289. }
291. double Quadrilateral::getBDLength() const
292. {
293. **return hypo(**
294. abs(this->getBX() - this->getDX()),
295. abs(this->getBY() - this->getDY()));
296. }
298. **bool Quadrilateral::isReal() const**
299. {
300. return !intersect(
301. this->getAX(),
302. this->getAY(),
303. **this->getBX(),**
304. this->getBY(),
305. this->getCX(),
306. this->getCY(),
307. this->getDX(),
308. **this->getDY());**
309. }

**Работа программы**



*Рисунок 1 Работа программы*

**Выводы**

В ходе выполнения лабораторной работы №4 получены навыки конструирования классов на основе базовых классов.

Создан производный класс. Продемонстрировано использование конструкторов базового класса в производном. Показано использование функциональности базового класса при работе с объектом производного класса. В консоль выведены сообщения о работе деструкторов, из которых можно определить порядок вызова деструкторов при освобождении памяти от объекта, участвующего в иерархии наследования.

Продемонстрирована работа всех методов, реализованных в классах.