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 №3 |
| Перегрузка операторов |
| 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 перегрузки операторов для типов, определенных пользователем посредством использования методов класса и дружественных функций.

Закрепить знания по теме: перегрузка операторов.

Описание работы: В работе необходимо реализовать класс в соответствии с вариантом задания и создать приложение. В классе должен быть предусмотрен конструктор для установки начальных значений полей, методы – члены класса и дружественные методы, обеспечивающие перегрузку операций для заданного класса. Часть перегруженных операторов должны быть членами класса, а часть – дружественными функциями. Т.е. в работе должны быть продемонстрированы оба способа перегрузки операторов (не нужно один и тот же оператор перегружать двумя способами).

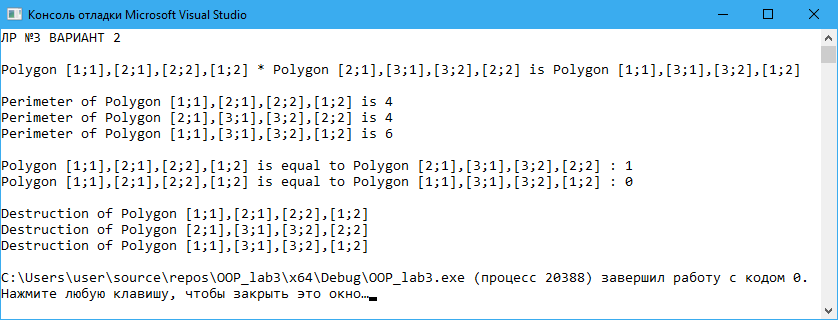
Индивидуальное задание (вариант 2):

2. Разработать класс «Прямоугольник», в котором содержатся поля для хранения 4 вершин прямоугольника. Определить в нем конструкторы и деструктор, перегрузить операцию объединения прямоугольников (операция “\*”) для случая успешного выполнения перегруженной операции проверки совпадения сторон с равной длиной), операцию вычисления периметра прямоугольника, операции сравнения двух прямоугольников (по периметру).

**Полный текст (листинг) программы**

1. Файл «main.cpp»
2. #include <iostream>
3. #include <string>
4. #include <Windows.h>
5. #include "Polygon.h"
7. #define FIRST\_POLYGON\_COORDS 1,1,2,1,2,2,1,2
8. #define SECOND\_POLYGON\_COORDS 2,1,3,1,3,2,2,2
10. using namespace std;
12. int main()
13. {
14. SetConsoleCP(1251);
15. SetConsoleOutputCP(1251);
17. cout << "ЛР №3 ВАРИАНТ 2" << endl << endl;
19. MyPolygon\* polygon1 = new MyPolygon(FIRST\_POLYGON\_COORDS);
20. MyPolygon\* polygon2 = new MyPolygon(SECOND\_POLYGON\_COORDS);
22. MyPolygon\* polygon3 = &((\*polygon1) \* (\*polygon2));
24. cout << polygon1->getDescription() << " \* "
25. << polygon2->getDescription() << " is "
26. **<< polygon3->getDescription() << endl << endl;**
28. cout << "Perimeter of " << polygon1->getDescription() << " is " << polygon1->getPerimeter() << endl;
29. cout << "Perimeter of " << polygon2->getDescription() << " is " << polygon2->getPerimeter() << endl;
30. cout << "Perimeter of " << polygon3->getDescription() << " is " << polygon3->getPerimeter() << endl;
32. cout << endl;
34. bool first\_equal = (\*polygon1) == (\*polygon2);
35. bool second\_equal = (\*polygon1) == (\*polygon3);
37. cout << polygon1->getDescription() << " is equal to "
38. << polygon2->getDescription() << " : "
39. << first\_equal << endl;
41. **cout << polygon1->getDescription() << " is equal to "**
42. << polygon3->getDescription() << " : "
43. << second\_equal << endl;
45. cout << endl;
47. delete polygon1;
48. delete polygon2;
49. delete polygon3;
50. }
51. Файл «Polygon.h»
52. #include <string>
54. using namespace std;
56. **class MyPolygon**
57. {
58. private:
60. int x1;
61. **int y1;**
63. int x2;
64. int y2;
66. **int x3;**
67. int y3;
69. int x4;
70. int y4;
72. public:
74. MyPolygon(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
76. **MyPolygon& operator \*(const MyPolygon& right\_polygon);**
78. friend bool operator ==(const MyPolygon & left\_polygon, const MyPolygon & right\_polygon);
80. bool hasEqualSide(const MyPolygon& right\_polygon);
82. int getX1() const;
83. void setX1(int x1);
85. int getY1() const;
86. **void setY1(int y1);**
88. int getX2() const;
89. void setX2(int x2);
91. **int getY2() const;**
92. void setY2(int y2);
94. int getX3() const;
95. void setX3(int x3);
97. int getY3() const;
98. void setY3(int y3);
100. int getX4() const;
101. **void setX4(int x4);**
103. int getY4() const;
104. void setY4(int y4);
106. **string getDescription() const;**
108. int getPerimeter() const;
110. ~MyPolygon();
111. **};**
112. Файл «Polygon.cpp»
113. #include <iostream>
114. #include <sstream>
115. #include "Polygon.h"
117. **MyPolygon::MyPolygon(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)**
118. {
119. if (x1 != x4
120. || x2 != x3
121. || y1 != y2
122. **|| y3 != y4**
123. || x2 <= x1
124. || y3 <= y2)
125. {
126. throw std::invalid\_argument("Wrong polygon coordinates");
127. **}**
129. this->x1 = x1;
130. this->x2 = x2;
131. this->x3 = x3;
132. **this->x4 = x4;**
134. this->y1 = y1;
135. this->y2 = y2;
136. this->y3 = y3;
137. **this->y4 = y4;**
138. }
140. MyPolygon& MyPolygon::operator\*(const MyPolygon& right\_polygon)
141. {
142. **if (!(this->hasEqualSide(right\_polygon)))**
143. {
144. throw std::invalid\_argument("Polygons has no equal sides");
145. }
147. **int newX1 = this->getX1() < right\_polygon.x1 ? this->getX1() : right\_polygon.x1;**
148. int newY1 = this->getY1() < right\_polygon.y1 ? this->getY1() : right\_polygon.y1;
150. int newX2 = this->getX2() > right\_polygon.x2 ? this->getX2() : right\_polygon.x2;
151. int newY2 = this->getY2() < right\_polygon.x2 ? this->getY2() : right\_polygon.y2;
153. int newX3 = this->getX3() > right\_polygon.x3 ? this->getX3() : right\_polygon.x3;
154. int newY3 = this->getY3() > right\_polygon.x3 ? this->getY3() : right\_polygon.y3;
156. int newX4 = this->getX4() < right\_polygon.x4 ? this->getX4() : right\_polygon.x4;
157. **int newY4 = this->getY4() > right\_polygon.x4 ? this->getY4() : right\_polygon.y4;**
159. MyPolygon\* polygon = new MyPolygon(newX1, newY1, newX2, newY2, newX2, newY3, newX4, newY4);
161. return \*polygon;
162. **}**
164. MyPolygon::~MyPolygon()
165. {
166. std::cout << "Destruction of "
167. **<< this->getDescription()**
168. << std::endl;
169. }
171. string MyPolygon::getDescription() const
172. **{**
173. std::stringstream buffer;
175. buffer << "Polygon "
176. << "[" << x1 << ";" << y1 << "],"
177. **<< "[" << x2 << ";" << y2 << "],"**
178. << "[" << x3 << ";" << y3 << "],"
179. << "[" << x4 << ";" << y4 << "]";
181. return buffer.str();
182. **}**
184. int MyPolygon::getPerimeter() const
185. {
186. const int coordNum = 8;
188. int polygonTops[coordNum] = {
189. this->getX1(),
190. this->getY1(),
191. this->getX2(),
192. **this->getY2(),**
193. this->getX3(),
194. this->getY3(),
195. this->getX4(),
196. this->getY4() };
198. int perimeter = 0;
200. for (int i = 0; i < coordNum; i += 2)
201. {
202. **int x1 = polygonTops[i];**
203. int y1 = polygonTops[i + 1];
204. int x2 = polygonTops[(i + 2) % coordNum];
205. int y2 = polygonTops[(i + 3) % coordNum];
207. **if (x1 != x2)**
208. {
209. perimeter += x1 > x2 ? x1 - x2 : x2 - x1;
210. }
211. else
212. **{**
213. perimeter += y1 > y2 ? y1 - y2 : y2 - y1;
214. }
215. }
217. **return perimeter;**
218. }
220. int MyPolygon::getX1() const { return this->x1; }
221. void MyPolygon::setX1(int x1) { this->x1 = x1; }
222. **int MyPolygon::getY1() const { return this->y1; }**
223. void MyPolygon::setY1(int y1) { this->y1 = y1; }
225. int MyPolygon::getX2() const { return this->x2; }
226. void MyPolygon::setX2(int x2) { this->x2 = x2; }
227. **int MyPolygon::getY2() const { return this->y2; }**
228. void MyPolygon::setY2(int y2) { this->y2 = y2; }
230. int MyPolygon::getX3() const { return this->x3; }
231. void MyPolygon::setX3(int x3) { this->x3 = x3; }
232. **int MyPolygon::getY3() const { return this->y3; }**
233. void MyPolygon::setY3(int y3) { this->y3 = y3; }
235. int MyPolygon::getX4() const { return this->x4; }
236. void MyPolygon::setX4(int x4) { this->x4 = x4; }
237. **int MyPolygon::getY4() const { return this->y4; }**
238. void MyPolygon::setY4(int y4) { this->y4 = y4; }
240. bool MyPolygon::hasEqualSide(const MyPolygon& right\_polygon)
241. {
242. **const int coordNum = 8;**
244. int leftPolygonTops[coordNum] = {
245. this->getX1(),
246. this->getY1(),
247. **this->getX2(),**
248. this->getY2(),
249. this->getX3(),
250. this->getY3(),
251. this->getX4(),
252. **this->getY4() };**
254. int rightPolygonTops[coordNum] = {
255. right\_polygon.getX1(),
256. right\_polygon.getY2(),
257. **right\_polygon.getX2(),**
258. right\_polygon.getY2(),
259. right\_polygon.getX3(),
260. right\_polygon.getY3(),
261. right\_polygon.getX4(),
262. **right\_polygon.getY4()};**
264. for (int i = 0; i < coordNum; i += 2)
265. {
266. int leftX1 = leftPolygonTops[i];
267. **int leftY1 = leftPolygonTops[i + 1];**
268. int leftX2 = leftPolygonTops[(i + 2) % coordNum];
269. int leftY2 = leftPolygonTops[(i + 3) % coordNum];
271. for (int j = 0; j < coordNum; j += 2)
272. **{**
273. int rightX1 = rightPolygonTops[j];
274. int rightY1 = rightPolygonTops[j + 1];
275. int rightX2 = rightPolygonTops[(j + 2) % coordNum];
276. int rightY2 = rightPolygonTops[(j + 3) % coordNum];
278. if ((leftX1 == rightX1
279. && leftY1 == rightY1
280. && leftX2 == rightX2
281. && leftY2 == rightY2)
282. **|| (leftX1 == rightX2**
283. && leftY1 == rightY2
284. && leftX2 == rightX1
285. && leftY2 == rightY1))
286. {
287. **return true;**
288. }
289. }
290. }
292. **return false;**
293. }
295. bool operator==(const MyPolygon& left\_polygon, const MyPolygon& right\_polygon)
296. {
297. **return left\_polygon.getPerimeter() == right\_polygon.getPerimeter();**
298. }

**Работа программы**



*Рисунок 1 Работа программы*

**Выводы**

В ходе выполнения лабораторной работы №3 были получены навыки по перегрузке операторов и работе с дружественными функциями класса.

Переопределены операторы в соответствии с вариантом. Один оператор переопределен с помощью метода-члена класса, другой оператор переопределен с помощью дружественной функции. Таким образом, продемонстрированы разные способы перегрузки операторов в C++.