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 №6 |
| Стандартная библиотека С++. Последовательные и ассоциативные контейнеры. Обобщенные алгоритмы |
| 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 принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Общая часть задания

Реализовать класс, содержащий:

- массив данных (вектор), заполненный случайными числами в диапазоне m1 - m2;

- методы, обеспечивающие выполнение действий (согласно варианту) с использованием обобщенных алгоритмов, объектов-функций и предикатов;

- обеспечить вывод результатов после выполнения каждого действия.

Обобщенные алгоритмы использовать обязательно.

Индивидуальное задание (вариант 2):

Вариант 2.

- m1=-10, m2=10

- найти максимальный элемент массива

- прибавить к каждому элементу массива найденный максимальный элемент

- отсортировать массив по абсолютному значению

**Полный текст (листинг) программы**

1. Файл «main.cpp»
2. #include <iostream>
3. #include "MyVector.h"
5. using namespace std;
7. int main()
8. {
9. cout << "LR#6 VAR2" << endl << endl;
11. **MyVector\* vector = new MyVector(20);**
13. cout << "Vector: " << vector->getVectorStr() << endl << endl;
15. cout << "Max: " << vector->max() << endl;
17. vector->addMax();
19. cout << "Max added to each element: " << vector->getVectorStr() << endl;
21. **vector->sortByAbs();**
23. cout << "Vector sorted by absolute value: " << vector->getVectorStr() << endl;
25. return 0;
26. **}**
27. Файл «MyVector.h»
28. #include <vector>
29. #include <string>
31. #pragma once
32. **class MyVector**
33. {
34. private:
36. std::vector<int>\* vector;
38. public:
40. MyVector(int size);
41. ~MyVector();
43. std::string getVectorStr();
45. int max();
46. void addMax();
47. **void sortByAbs();**
48. };

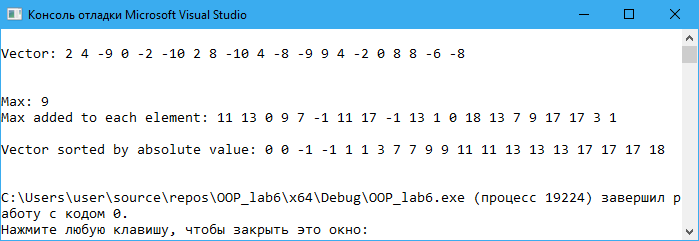
51. Файл «MyVector.cpp»
52. #include <vector>
53. #include <limits>
54. #include <cstdlib>
55. #include <string>
56. **#include <sstream>**
57. #include <ctime>
58. #include "MyVector.h"
59. #include "AddFunctor.h"

62. #define m1 -10
63. #define m2 10
65. MyVector::MyVector(int size)
66. **{**
67. this->vector = new std::vector<int>(size);
69. srand((unsigned)time(NULL));
71. **for (int i = 0; i < vector->size(); i++)**
72. {
73. vector->at(i) = (rand() % (abs(m1 - m2) + 1)) + m1;
74. }
75. }
77. MyVector::~MyVector()
78. {
79. delete this->vector;
80. }
82. std::string MyVector::getVectorStr()
83. {
84. std::stringstream ss;
86. **std::vector<int>::iterator iter = this->vector->begin();**
88. for (iter; iter < this->vector->end(); iter++)
89. {
90. ss << \*iter << " ";
91. **}**
93. ss << std::endl;
95. return ss.str();
96. **}**
98. template <class T> T findMax(std::vector<T>\* vector, T minValue)
99. {
100. T maxValue = minValue;
102. for (int i = 0; i < vector->size(); i++)
103. {
104. if (vector->at(i) > maxValue)
105. {
106. **maxValue = vector->at(i);**
107. }
108. }
110. return maxValue;
111. **}**
113. int MyVector::max()
114. {
115. int max = findMax<int>(vector, std::numeric\_limits<int>::min());
117. return max;
118. }
120. void MyVector::addMax()
121. **{**
122. AddFunctor add{};
124. int max = this->max();
126. **for (int i = 0; i < vector->size(); i++)**
127. {
128. add(&(vector->at(i)), max);
129. }
130. }
132. bool intSortCriterion(const int int1, const int int2)
133. {
134. return abs(int1) <= abs(int2);
135. }
137. void mergeArray(std::vector<int>\* vector, int left, int middle, int right, bool(\*op)(int, int))
138. {
139. int leftArrayLength = middle - left + 1;
140. int rightArrayLength = right - middle;
141. **std::vector<int>\* leftTempArray = new std::vector<int>(leftArrayLength);**
142. std::vector<int>\* rightTempArray = new std::vector<int>(rightArrayLength);
144. int i, j;
146. **for (i = 0; i < leftArrayLength; i++)**
147. {
148. leftTempArray->at(i) = vector->at(left + i);
149. }
150. for (j = 0; j < rightArrayLength; j++)
151. **{**
152. rightTempArray->at(j) = vector->at(middle + 1 + j);
153. }
155. i = 0;
156. **j = 0;**
157. int k = left;
159. while (i < leftArrayLength
160. && j < rightArrayLength)
161. **{**
162. if (op(leftTempArray->at(i), rightTempArray->at(j)))
163. {
164. vector->at(k++) = leftTempArray->at(i++);
165. }
166. **else**
167. {
168. vector->at(k++) = rightTempArray->at(j++);
169. }
170. }
172. while (i < leftArrayLength)
173. {
174. vector->at(k++) = leftTempArray->at(i++);
175. }
177. while (j < rightArrayLength)
178. {
179. vector->at(k++) = rightTempArray->at(j++);
180. }
182. delete leftTempArray;
183. delete rightTempArray;
184. }
186. **std::vector<int>\* sort(std::vector<int>\* vector, int left, int right, bool(\*op)(int, int))**
187. {
188. if (left < right)
189. {
190. int middle = left + (right - left) / 2;
191. **sort(vector, left, middle, op);**
192. sort(vector, middle + 1, right, op);
193. mergeArray(vector, left, middle, right, op);
194. }
196. **return vector;**
197. }
199. std::vector<int>\* mergeSort(std::vector<int>\* vector, bool(\*op)(int, int))
200. {
201. **if (vector->size() == 0**
202. || vector->size() == 1)
203. {
204. return new std::vector<int>(\*(vector));
205. }
207. std::vector<int>\* ret = new std::vector<int>(\*(vector));
209. ret = sort(ret, 0, ret->size() - 1, op);
211. **return ret;**
212. }
214. void MyVector::sortByAbs()
215. {
216. **std::vector<int>\* prevVector = this->vector;**
218. this->vector = mergeSort(this->vector, intSortCriterion);
220. delete prevVector;
221. **}**

224. Файл «AddFunctor.h»
225. #pragma once
226. class AddFunctor
227. {
228. public:
230. void operator()(int\* a, int b);
231. };

234. Файл «AddFunctor.cpp»
235. #include "AddFunctor.h"
237. void AddFunctor::operator()(int\* a, int b)
238. {
239. **\*(a) += b;**
240. }

**Работа программы**



*Рисунок 1 Работа программы*

**Выводы**

В ходе выполнения лабораторной работы №6 изучены принципы построения консольных приложений, применены на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Получен опыт в работе со структурами данных STL, использованы объекты-функции, предикаты.

Создана и использована шаблонизированная функция, позволяющая обрабатывать различные типы данных.