**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г.ШУХОВА)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

дисциплина: **Объектно - ориентированное программирование**

Лабораторная работа № 8

тема: «**Создание шаблонов классов в С++**»

Выполнил: ст. группы ВТ-22

Воскобойников Илья

Белгород 2020

**Цель работы:** Получение теоретических знаний о шаблонах классов в С++. Получение практических навыков по созданию классов-шаблонов С++.

Задания к лабораторной работе

1. Изучить теоретические сведения о шаблонах классов в С++.
2. Разработать программу в соответствии с заданным вариантом задания.
3. Оформить отчет.

Задание 1

Реализовать шаблон класса в соответствии с указанным вариантом. Предусмотреть необходимые методы для работы со структурой данных, указанной в варианте. Предусмотреть исключительные ситуации, которые могут возникнуть в процессе работы.

Задание для варианта 3

Двусвязный список

Задание 2.

На основе разработанного шаблона решить прикладную задачу в соответствии с выбранным вариантом.

Задание для варианта 3

Дан файл структур с тремя полями: фамилия, возраст, оценка. Реализовать структуру для упорядоченного поиска элемента из файла по возрасту, оценки, фамилии. В качестве метода сравнения двух элементов обязательно использовать template функцию.

1. #include <iostream>
2. #include <locale.h>
3. #include <math.h>
4. #include <fstream>
5. #include <stdio.h>
6. #include <string>
7. using namespace std;
8. struct Person {
9. string name;
10. int age;
11. int marks[4];
12. Person() {};
13. friend bool operator < (const Person p1, const Person p2);
14. friend bool operator > (const Person p1, const Person p2);
15. friend bool operator == (const Person p1, const Person p2);
16. friend bool operator != (const Person p1, const Person p2);
17. friend bool operator <= (const Person p1, const Person p2);
18. friend bool operator >= (const Person p1, const Person p2);
19. friend ostream& operator<<(ostream& out, const Person& p);
20. };
21. class DLSException : public exception {
22. string mesError;
23. public:
24. DLSException(string \_mes) : mesError(\_mes) {};
25. ~DLSException() = default;
26. const char\* what() { return mesError.c\_str(); };
27. };
28. class EmptyDList : public DLSException {
29. public:
30. EmptyDList() : DLSException("Список пуст!") {};
31. };
32. class EndDList : public DLSException {
33. public:
34. EndDList() : DLSException("Переполнение списка!") {};
35. };
36. template<class Type>
37. class DList
38. {
39. size\_t size;
40. struct Node
41. {
42. Type x;
43. Node\* next, \* prev;
44. };
45. Node\* head, \* tail;
46. public:
47. DList()
48. {
49. head = NULL;
50. tail = NULL;
51. }
52. Node\* temp;
53. bool endList() { return (temp == NULL); }
54. bool emptyList() { return (head == NULL); }
55. void addNode(Type x)
56. {
57. temp = new Node;
58. if (endList())
59. throw EndDList();
60. temp->next = NULL;
61. temp->x = x;
62. if (head != NULL)
63. {
64. temp->prev = tail;
65. tail->next = temp;
66. tail = temp;
67. }
68. else
69. {
70. temp->prev = NULL;
71. head = tail = temp;
72. }
73. }
74. void show()
75. {
76. if (emptyList())
77. throw EmptyDList();
78. temp = head;
79. while (!endList())
80. {
81. cout << temp->x << endl;
82. temp = temp->next;
83. }
84. }
85. bool inList(Type el)
86. {
87. temp = head;
88. while (!endList())
89. {
90. if (temp->x == el)
91. return true;
92. temp = temp->Next;
93. }
94. return false;
95. }
96. void getListPost(Type\* E)
97. {
98. if (emptyList())
99. throw EmptyDList();
100. Node\* pntr;
101. pntr = temp->next;
102. pntr->next->prev = temp;
103. temp->next = pntr->next;
104. \*E = pntr->x;
105. pntr->next = NULL;
106. pntr->prev = NULL;
107. free(pntr);
108. }
109. template<class Type>
110. void insertBeforeElement(DList\* list, Node\* elm, Type\* value) {
111. Node\* ins = NULL;
112. if (emptyList())
113. throw EmptyDList();
114. if (!elm->prev) {
115. addNode(value);
116. return;
117. }
118. ins = (Node\*)malloc(sizeof(Node));
119. ins->value = value;
120. ins->prev = elm->prev;
121. elm->prev->next = ins;
122. ins->next = elm;
123. elm->prev = ins;
124. }
125. void insertionSort(DList\*\* list, int (\*cmp)(void\*, void\*)) {
126. DList\* out = DList();
127. Node\* sorted = NULL;
128. Node\* unsorted = NULL;
129. unsorted = (\*list)->head;
130. while (unsorted) {
131. sorted = out->head;
132. while (sorted && !cmp(unsorted->value, sorted->value)) {
133. sorted = sorted->next;
134. }
135. if (sorted) {
136. insertBeforeElement(out, sorted, unsorted->value);
137. }
138. else {
139. addNode(unsorted->value);
140. }
141. unsorted = unsorted->next;
142. }
143. free(\*list);
144. \*list = out;
145. }
146. ~DList()
147. {
148. while (head) {
149. tail = head->next;
150. delete head;
151. head = tail;
152. }
153. }
154. };
155. bool operator < (const Person p1, const Person p2) {
156. if (p1.name != p2.name)
157. return p1.name < p2.name;
158. else {
159. if (p1.age != p2.age)
160. return p1.age < p2.age;
161. for (int i = 0; i < 4; i++)
162. if (p1.marks[i] != p2.marks[i])
163. return p1.marks[i] < p2.marks[i];
164. }
165. return false;
166. }
167. bool operator > (const Person p1, const Person p2) {
168. if (p1.name != p2.name)
169. return p1.name > p2.name;
170. else {
171. if (p1.age != p2.age)
172. return p1.age > p2.age;
173. for (int i = 0; i < 4; i++)
174. if (p1.marks[i] != p2.marks[i])
175. return p1.marks[i] > p2.marks[i];
176. }
177. return false;
178. }
179. bool operator == (const Person p1, const Person p2) {
180. if (p1.name != p2.name)
181. return false;
182. else {
183. if (p1.age != p2.age)
184. return false;
185. for (int i = 0; i < 4; i++)
186. if (p1.marks[i] != p2.marks[i])
187. return false;
188. }
189. return true;
190. }
191. bool operator != (const Person p1, const Person p2) {
192. return !(p1 == p2);
193. }
194. bool operator <= (const Person p1, const Person p2) {
195. return (p1 < p2) || (p1 == p2);
196. }
197. bool operator >= (const Person p1, const Person p2) {
198. return (p1 > p2) || (p1 == p2);
199. }
200. ostream& operator<< (ostream& out, const Person& p) {
201. out << "Last Name: " << p.name << " | Age: " << p.age << " | Marks: ";
202. for (auto mark : p.marks)
203. out << mark << " ";
204. out << endl;
205. return out;
206. }
207. void personsToFile(size\_t n) {
208. FILE\* f = NULL;
209. fopen\_s(&f, "persons.bin", "wb");
210. if (f == NULL) throw EndDList();
211. for (size\_t i = 0; i < n; i++) {
212. Person a;
213. cout << "ENTER Last Name: "; cin >> a.name;
214. cout << "ENTER Age: "; cin >> a.age;
215. cout << "ENTER 4 marks: ";
216. for (int i = 0; i < 4; i++)
217. cin >> a.marks[i];
218. fwrite(&a, sizeof(Person), 1, f);
219. cout << "+=============================================================+" << endl;
220. }
221. fclose(f);
222. }
223. int main()
224. {
225. setlocale(LC\_ALL, "rus");
226. personsToFile(3);
227. try {
228. FILE\* f = NULL;
229. fopen\_s(&f, "persons.bin", "rb");
230. DList<Person> x;
231. Person a;
232. while (fread\_s(&a, sizeof(a), sizeof(Person), 1, f))
233. x.addNode(a);
234. fclose(f);
235. x.show();
236. }
237. catch (DLSException \*e) {
238. e->what();
239. }
240. return 0;
241. }