**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ**  
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

Лабораторная работа №8

Дисциплина: Обьектно-ориентированное программирование

по теме «Создание шаблонов классов в С++»

Выполнил: ст. группы ПВ-21  
Ковалев Павел Александрович

Проверил: Буханов Д.Г.

**Белгород 2020Цель работы**: Получение теоретических знаний о шаблонах классов в С++. Получение практических навыков по созданию классов-шаблонов С++.

Задание.

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.

Вариант 6

ListEl.h

1. #include <iostream>
2. *using* *namespace* std;
3. *template* <*class* Type>
4. *class* **ListEl**{
5. Type data;
6. ListEl\* next\_link;
7. ListEl\* prev\_link;
8. *public*:
9. **ListEl**();
10. **ListEl**(Type data);
11. void **set\_next\_link**(ListEl\* link);
12. void **set\_prev\_link**(ListEl\* link);
13. void **set\_data**(Type data);
14. Type **get\_data**();
15. ListEl \***get\_next**();
16. ListEl \***get\_prev**();
17. void **print**();
18. };
19. **ListEl.cpp**
20. *template* <*class* Type>
21. ListEl<Type>::**ListEl**(){
22. *this*->next\_link=*nullptr*;
23. *this*->prev\_link=*nullptr*;
24. };
25. *template* <*class* Type>
26. ListEl<Type>::**ListEl**(Type data):data(data){
27. *this*->next\_link=*nullptr*;
28. *this*->prev\_link=*nullptr*;
29. };
30. *template* <*class* Type>
31. void ListEl<Type>::**set\_next\_link**(ListEl<Type>\* link){
32. *this*->next\_link=link;
33. };
34. *template* <*class* Type>
35. void ListEl<Type>::**set\_prev\_link**(ListEl<Type>\* link){
36. *this*->prev\_link=link;
37. };
38. *template* <*class* Type>
39. void ListEl<Type>::**set\_data**(Type data){
40. *this*->data=data;
41. };
42. *template* <*class* Type>
43. Type ListEl<Type>::**get\_data**(){
44. *return* *this*->data;
45. };
46. *template* <*class* Type>
47. ListEl<Type> \*ListEl<Type>::**get\_next**(){
48. *return* *this*->next\_link;
49. };
50. *template* <*class* Type>
51. ListEl<Type> \*ListEl<Type>::**get\_prev**(){
52. *return* *this*->prev\_link;
53. };
54. *template* <*class* Type>
55. void ListEl<Type>::**print**(){
56. cout<<"Data: "<<*this*->data<<endl;
57. };
58. **doublelinkedlist.h**
59. #include <listel.h>
60. *template* <*class* Type>
61. *class* **DoubleLinkedList**{
62. ListEl<Type>\* list\_start;
63. ListEl<Type>\* list\_end;
64. ListEl<Type>\* list\_current;
65. *public*:
66. **DoubleLinkedList**();
67. void **add\_el**(Type element);
68. Type **pop\_el**();
69. Type **get\_el**();
70. void **ptr\_to\_start**();
71. void **ptr\_to\_end**();
72. int **is\_end**();
73. void **clear**();
74. void **next\_el**();
75. void **prev\_el**();
76. void **print**();
77. };
78. **doublelinkedlist.cpp**
79. *template* <*class* Type>
80. DoubleLinkedList<Type>::**DoubleLinkedList**(){
81. *this*->list\_start= *new* ListEl<Type>();
82. *this*->list\_end= *new* ListEl<Type>();
83. *this*->list\_current=*this*->list\_start;
84. };
85. *template* <*class* Type>
86. void DoubleLinkedList<Type>::**add\_el**(Type element){
87. ListEl<Type> \*temp\_ptr = *new* ListEl<Type>(element);
88. *if*(*this*->list\_current->get\_next()==*nullptr*){
    1. temp\_ptr->set\_next\_link(*nullptr*);
    2. temp\_ptr->set\_prev\_link(list\_current);
89. }*else*{
    1. temp\_ptr->set\_next\_link(*this*->list\_current->get\_next());
    2. temp\_ptr->set\_prev\_link(list\_current);
    3. *this*->list\_current->get\_next()->set\_prev\_link(temp\_ptr);
90. }
91. *this*->list\_current->set\_next\_link(temp\_ptr);
92. *this*->next\_el();
93. };
94. *template* <*class* Type>
95. Type DoubleLinkedList<Type>::**pop\_el**(){
96. Type temp=*this*->list\_current->get\_data();
97. ListEl<Type> \*pr=*this*->list\_current->get\_prev();
98. ListEl<Type> \*nx=*this*->list\_current->get\_next();
99. pr->set\_next\_link(nx);
100. nx->set\_prev\_link(pr);
101. *return* temp;
102. };
103. *template* <*class* Type>
104. Type DoubleLinkedList<Type>::**get\_el**(){
105. *return* *this*->list\_current->get\_data();
106. };
107. *template* <*class* Type>
108. void DoubleLinkedList<Type>::**ptr\_to\_start**(){
109. *this*->list\_current=*this*->list\_start;
110. *this*->next\_el();
111. };
112. *template* <*class* Type>
113. void DoubleLinkedList<Type>::**ptr\_to\_end**(){
114. *this*->list\_current=*this*->list\_end;
115. *this*->prev\_el();
116. };
117. *template* <*class* Type>
118. void DoubleLinkedList<Type>::**clear**(){
119. *this*->ptr\_to\_start();
120. *while*(*this*->list\_current!=*this*->list\_end){
     1. *this*->pop\_el();
121. }
122. };
123. *template* <*class* Type>
124. int DoubleLinkedList<Type>::**is\_end**(){
125. *return* *this*->list\_current==*this*->list\_end;
126. };
127. *template* <*class* Type>
128. void DoubleLinkedList<Type>::**next\_el**(){
129. *this*->list\_current=*this*->list\_current->get\_next();
130. };
131. *template* <*class* Type>
132. void DoubleLinkedList<Type>::**prev\_el**(){
133. *this*->list\_current=*this*->list\_current->get\_prev();
134. };
135. *template* <*class* Type>
136. void DoubleLinkedList<Type>::**print**(){
137. ListEl<Type> \*temp\_pos=*this*->list\_current;
138. *this*->ptr\_to\_start();
139. int num=1;
140. *while*(*this*->list\_current!=*this*->list\_end){
     1. cout<<"Element : "<<num<<endl;
     2. *this*->list\_current->print();
     3. *this*->next\_el();
     4. num++;
141. };
142. *this*->list\_current=temp\_pos;
143. };
144. **deck.h**
145. #include <doublelinkedlist.h>
146. *template* <*class* Type>
147. *class* **Deck**{
148. DoubleLinkedList<Type> list;
149. unsigned current\_size;
150. unsigned max\_size;
151. *public*:
152. **Deck**(unsigned size);
153. void **push\_back**(Type data);
154. void **push\_front**(Type data);
155. Type **pop\_back**();
156. Type **pop\_front**();
157. int **is\_empty**();
158. void **clear**();
159. };
160. **deck.cpp**
161. *template* <*class* Type>
162. Deck<Type>::**Deck**(unsigned size):max\_size(size),current\_size(0),list(DoubleLinkedList<Type>()){};
163. *template* <*class* Type>
164. void Deck<Type>::**push\_back**(Type data){
165. *this*->list.ptr\_to\_end();
166. *this*->list.add\_el(data);
167. *this*->current\_size++;
168. };
169. *template* <*class* Type>
170. void Deck<Type>::**push\_front**(Type data){
171. *this*->list.ptr\_to\_start();
172. *this*->list.add\_el(data);
173. *this*->current\_size++;
174. };
175. *template* <*class* Type>
176. Type Deck<Type>::**pop\_back**(){
177. *this*->list.ptr\_to\_end();
178. *this*->current\_size--;
179. *return* *this*->list.pop\_el();
180. };
181. *template* <*class* Type>
182. Type Deck<Type>::**pop\_front**(){
183. *this*->list.ptr\_to\_start();
184. *this*->current\_size--;
185. *return* *this*->list.pop\_el();
186. };
187. *template* <*class* Type>
188. int Deck<Type>::**is\_empty**(){
189. *return* *this*->current\_size==0;
190. };
191. *template* <*class* Type>
192. void Deck<Type>::**clear**(){
193. *this*->list.clear();
194. *this*->current\_size=0;
195. };
196. **main.cpp**
197. int **main**(int argc, char \*argv[])
198. {
199. QCoreApplication app(*argc*, argv);
200. DoubleLinkedList<int> a;
201. *for*(int i=1;i<=10;i++){
     1. a.add\_el(i);
202. }
203. a.print();
204. *return* app.exec();
205. }