

```

1  #ifndef LISTA_H
2  #define LISTA_H
3
4  #include <string>
5  #include "Exception.h"
6  #include "node.h"
7
8  template <class T, int ARRAYSIZE = 1024>
9  class Musicbox {
10 public:
11     class Node {
12     private:
13         T data;
14     public:
15         Node(const T&);
16         T& getData();
17         void setData(const T&);
18     };
19     private:
20         Node* AllSongs[ARRAYSIZE];
21         int Last;
22         void copyAll(const Musicbox<T,ARRAYSIZE>&);
23         void swapSong(Node*,Node*);
24     public:
25         Musicbox();
26         bool isFull();
27         bool isEmpty();
28         bool isValidPos(const int&);
29         void insertSong(const int&, const T&);
30         void deleteMusic(const int&);
31         int FirstMusic();
32         int LastMusic();
33         int PrevMusic(const int&);
34         int NextMusic(const int&);
35         int FindDataL(const T&);
36         int FindDataB(const T&);
37         T& retrieve(const int&);
38         void BubbleSortSongs(const char&);
39         void ShellSortSongs(const char&);
40         void InsertSortSongs(const char&);
41         void SelectSortSongs(const char&);
42         std::string toString();
43         void DeleteAllMusic();
44         int getLastPos();
45         Musicbox<T,ARRAYSIZE> operator = (const Musicbox<T,ARRAYSIZE>&);
46     };
47     template <class T, int ARRAYSIZE>
48     Musicbox<T,ARRAYSIZE>::Node::Node(const T& e) : data(e) { }
49     template <class T, int ARRAYSIZE>
50     T& Musicbox<T,ARRAYSIZE>::Node::getData() {
51     return data;
52     }
53     template <class T, int ARRAYSIZE>
54     void Musicbox<T,ARRAYSIZE>::Node::setData(const T& e) {
55     data = e;
56     }
57     template <class T, int ARRAYSIZE>
58     void Musicbox<T,ARRAYSIZE>::copyAll(const Musicbox<T,ARRAYSIZE>& AllMusic) {
59     int i(0);
60     while ( i <= Last ) {
61     this->AllSongs[i] = AllMusic.AllSongs[i];
62     }
63     }
64     template <class T, int ARRAYSIZE>
65     Musicbox<T,ARRAYSIZE>::Musicbox() {
66     Last=-1;

```

```

67  int i=0;
68  while(i<=1023) {
69  AllSongs[i] = nullptr;
70  i++;
71  }
72  }
73  template <class T, int ARRAYSIZE>
74  bool Musicbox<T,ARRAYSIZE>::isFull() {
75  return Last == ARRAYSIZE - 1;
76  }
77  template <class T, int ARRAYSIZE>
78  bool Musicbox<T,ARRAYSIZE>::isValidPos(const int& pos) {
79  return pos > -1 and pos <= Last;
80  }
81  template <class T, int ARRAYSIZE>
82  bool Musicbox<T,ARRAYSIZE>::isEmpty() {
83  return Last == -1;
84  }
85  template <class T, int ARRAYSIZE>
86  void Musicbox<T,ARRAYSIZE>::insertSong(const int& pos, const T& Song) {
87  if( isFull() ) {
88  throw Exception("Desbordamiento de datos, insertSong");
89  }
90  if( pos != -1 and !isValidPos(pos) ) {
91  throw Exception("Posicion invalida, insertSong");
92  }
93  Node* aux(new Node(Song));
94  int i(Last);
95  while( i > pos ) {
96  AllSongs[i] = AllSongs[i-1];
97  i--;
98  }
99  AllSongs[pos + 1] = aux;
100  Last++;
101  }
102  template <class T, int ARRAYSIZE>
103  void Musicbox<T,ARRAYSIZE>::deleteMusic(const int& pos) {
104  if(isEmpty()) {
105  throw Exception("Insuficiencia de datos, deleteMusic");
106  }
107  if(!isValidPos(pos)) {
108  throw Exception("Posicion invalida, deleteMusic");
109  }
110  int i = pos;
111  delete AllSongs[pos];
112  while( i < Last ) {
113  AllSongs[i] = AllSongs[i+1];
114  i++;
115  }
116  Last--;
117  }
118  template <class T, int ARRAYSIZE>
119  int Musicbox<T,ARRAYSIZE>::FirstMusic() {
120  if(isEmpty()) {
121  throw Exception("Insuficiencia de datos, FirstMusic");
122  }
123  return 0;
124  }
125  template <class T, int ARRAYSIZE>
126  int Musicbox<T,ARRAYSIZE>::LastMusic() {
127  if(isEmpty()) {
128  throw Exception("Insuficiencia de datos, LastMusic");
129  }
130  return Last;
131  }
132  template <class T, int ARRAYSIZE>

```

```

133 int Musicbox<T,ARRAYSIZE>::PrevMusic(const int& pos) {
134     if( isEmpty() ) {
135         throw Exception("Insuficiencia de datos, PrevMusic");
136     }
137     if( !isValidPos(pos) or pos == 0 ) {
138         throw Exception("Posicion invalida, PrevMusic");
139     }
140     return pos - 1;
141 }
142 template <class T, int ARRAYSIZE>
143 int Musicbox<T,ARRAYSIZE>::NextMusic(const int& pos) {
144     if( isEmpty() ) {
145         throw Exception("Insuficiencia de datos");
146     }
147     if( !isValidPos(pos) or pos == Last ) {
148         throw Exception("Posicion invalida, NextPos");
149     }
150     return pos + 1;
151 }
152 template <class T, int ARRAYSIZE>
153 int Musicbox<T,ARRAYSIZE>::FindDataL(const T& Element ) {
154     if( isEmpty() ) {
155         throw Exception("Insuficiencia de datos, FindDataL");
156     }
157     int i(0);
158     while ( i <= Last ) {
159         if( AllSongs[i]->getData() == Element ) {
160             return i;
161         }
162         i++;
163     }
164     return -1;
165 }
166 template <class T, int ARRAYSIZE>
167 int Musicbox<T,ARRAYSIZE>::FindDataB(const T& Element) {
168     if( isEmpty() ) {
169         throw Exception("Insuficiencia de datos, FindDataB");
170     }
171     int i(0), j(Last), m;
172     while ( i <= j ) {
173         m = ( i + j ) / 2;
174         if( AllSongs[m]->getData() == Element ) {
175             return m;
176         }
177         if( Element < AllSongs[m]->getData() ) {
178             j = m - 1;
179         }
180         else {
181             i = m + 1;
182         }
183     }
184     return -1;
185 }
186 template <class T, int ARRAYSIZE>
187 T& Musicbox<T,ARRAYSIZE>::retrieve(const int& pos) {
188     if( isEmpty() ) {
189         throw Exception("Insuficiencia de datos, retrieveSong");
190     }
191     if( !isValidPos(pos) ) {
192         throw Exception("Insuficiencia de datos, retrieveSong");
193     }
194     return AllSongs[pos]->getData();
195 }
196 template <class T, int ARRAYSIZE>
197 void Musicbox<T,ARRAYSIZE>::swapSong(Node* a,Node* b) {
198     Node* aux(a);

```

```

199 a=b;
200 b=aux;
201 }
202 template <class T, int ARRAYSIZE>
203 void Musicbox<T,ARRAYSIZE>::BubbleSortSongs(const char& opt) {
204     int i(Last), j;
205     bool flag;
206     do {
207         flag = false;
208         j = 0;
209         while( j < i ) {
210             if( opt == 'A' ) {
211                 if( AllSongs[j]->getData() > AllSongs[j+1]->getData() ) {
212                     swapSong( AllSongs[j], AllSongs[i] );
213                     flag = true;
214                 }
215             }
216             if( opt == 'B' ) {
217                 if( AllSongs[j]->getData() > AllSongs[j+1]->getData() ) {
218                     swapSong( AllSongs[j], AllSongs[i] );
219                     flag = true;
220                 }
221             }
222             j++;
223         }
224         i--;
225     }
226     while(flag);
227 }
228 template <class T, int ARRAYSIZE>
229 void Musicbox<T,ARRAYSIZE>::ShellSortSongs(const char& opt) {
230     float fact ( 3.0 / 4.0 );
231     int dif( (Last + 1) * fact), lim, i;
232     while(dif > 0) {
233         lim = Last - dif;
234         i=0;
235         while( i <= lim ) {
236             if(opt == 'A') {
237                 if( AllSongs[i]->getData() > AllSongs[ i+ dif ]->getData() ) {
238                     swapSong( AllSongs[i], AllSongs[i+dif] );
239                 }
240             }
241             if(opt == 'B') {
242                 if( AllSongs[i]->getData() > AllSongs[ i+ dif ]->getData() ) {
243                     swapSong( AllSongs[i], AllSongs[i+dif] );
244                 }
245             }
246             i++;
247         }
248         dif *= fact;
249     }
250 }
251 template <class T, int ARRAYSIZE>
252 void Musicbox<T,ARRAYSIZE>::InsertSortSongs(const char& opt) {
253     int i(1), j;
254     Node* aux;
255     if( opt == 'A' ) {
256         while( i <= Last ) {
257             aux = AllSongs[i];
258             j=i;
259             while ( j > 0 and aux->getData() < AllSongs[j-1]->getData() ) {
260                 AllSongs[j] = AllSongs[j-1];
261                 j--;
262             }
263             if( i != j ) {
264                 AllSongs[j] = aux;

```

```

265 }
266 i++;
267 }
268 }
269 if(opt == 'B') {
270 while( i <= Last ) {
271 aux = AllSongs[i];
272 j=i;
273 while ( j > 0 and aux->getData() < AllSongs[j-1]->getData() ) {
274 AllSongs[j] = AllSongs[j-1];
275 j--;
276 }
277 if( i != j ) {
278 AllSongs[j] = aux;
279 }
280 i++;
281 }
282 }
283 }
284 template <class T, int ARRAYSIZE>
285 void Musicbox<T,ARRAYSIZE>::SelectSortSongs(const char& opt) {
286 int i(0), j, m;
287 while( i < Last ) {
288 m = i;
289 j = i + 1;
290 if( opt == 'A' ) {
291 while( j < Last ) {
292 if( AllSongs[j]->getData() < AllSongs[m]->getData() ) {
293 m = j;
294 }
295 j++;
296 }
297 }
298 if( opt == 'B' ) {
299 while( j < Last ) {
300 if( AllSongs[j]->getData() < AllSongs[m]->getData() ) {
301 m = j;
302 }
303 j++;
304 }
305 }
306 if( m!=i ) {
307 swapSong( AllSongs[i], AllSongs[m] );
308 }
309 i++;
310 }
311 }
312 template <class T, int ARRAYSIZE>
313 std::string Musicbox<T,ARRAYSIZE>::toString() {
314 std::string AllMusic;
315 for( int i(0) ; i <= Last ; i++ ) {
316 AllMusic += AllSongs[i]->toString() + "\n" ;
317 }
318 return AllMusic;
319 }
320 template <class T, int ARRAYSIZE>
321 void Musicbox<T,ARRAYSIZE>::DeleteAllMusic() {
322 int i(0);
323 while(i <= Last) {
324 delete AllSongs[i];
325 i++;
326 }
327 Last = -1;
328 }
329 template <class T, int ARRAYSIZE>
330 int Musicbox<T,ARRAYSIZE>::getLastPos() {

```

```
331 return Last;
332 }
333 template <class T, int ARRAYSIZE>
334 Musicbox<T,ARRAYSIZE>& Musicbox<T,ARRAYSIZE>::operator = (const Musicbox<T,ARRAYSIZE>& Song) {
335 copyAll(Song);
336 return *this;
337 }
338 #endif // LISTA_H
```