

```

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

```

```

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

```

```

133     if( AllMusics[i] == Element ) {
134         return i;
135     }
136     i++;
137 }
138 return -1;
139 }
140 template <class T, int ARRAYSIZE>
141 int Musicbox<T,ARRAYSIZE>::FindDataB(const T& Element) {
142     if( isEmpty() ) {
143         throw Exception("Insuficiencia de datos, FindDataB");
144     }
145     int i(0), j(Last), m;
146     while ( i <= j ) {
147         m = ( i + j ) / 2;
148         if( AllMusics[m] == Element ) {
149             return m;
150         }
151         if( Element < AllMusics[m] ) {
152             j = m - 1;
153         }
154         else {
155             i = m + 1;
156         }
157     }
158     return -1;
159 }
160 template <class T, int ARRAYSIZE>
161 T& Musicbox<T,ARRAYSIZE>::retrieve(const int& pos) {
162     if( isEmpty() ) {
163         throw Exception("Insuficiencia de datos, retrieveSong");
164     }
165     if( !isValidPos(pos) ) {
166         throw Exception("Insuficiencia de datos, retrieveSong");
167     }
168     return AllMusics[pos];
169 }
170 template <class T, int ARRAYSIZE>
171 void Musicbox<T,ARRAYSIZE>::swapSong(T& a,T& b) {
172     T aux(a);
173     a=b;
174     b=aux;
175 }
176 template <class T, int ARRAYSIZE>
177 void Musicbox<T,ARRAYSIZE>::BubbleSortSongs(const char& opt) {
178     int i(Last), j;
179     bool flag;
180     do {
181         flag = false;
182         j = 0;
183         while( j < i ) {
184             if( opt == 'A' ) {
185                 if( AllMusics[j].getSongName() > AllMusics[j+1].getSongName() ) {
186                     swapSong( AllMusics[j], AllMusics[j+1] );
187                     flag = true;
188                 }
189             }
190             if( opt == 'B' ) {
191                 if( AllMusics[j].getSongAutor() > AllMusics[j+1].getSongAutor() ) {
192                     swapSong( AllMusics[j], AllMusics[j+1] );
193                     flag = true;
194                 }
195             }
196             j++;
197         }
198         i--;

```

```

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

```

```

265 while( j < Last ) {
266     if( AllMusics[j].getSongName() < AllMusics[m].getSongName() ) {
267         m = j;
268     }
269     j++;
270 }
271 }
272 if( opt == 'B' ) {
273     while( j < Last ) {
274         if( AllMusics[j].getSongAutor() < AllMusics[m].getSongAutor() ) {
275             m = j;
276         }
277         j++;
278     }
279 }
280 if( m!=i ) {
281     swapSong( AllMusics[i], AllMusics[m] );
282 }
283 i++;
284 }
285 }
286 template <class T, int ARRAYSIZE>
287 void Musicbox<T,ARRAYSIZE>::MergeSortSongs() {
288     MergeSortSongs(0,getLastPos());
289 }
290 template <class T, int ARRAYSIZE>
291 void Musicbox<T,ARRAYSIZE>::MergeSortSongs(const int& leftEdge, const int& rightEdge) {
292     if( leftEdge >= rightEdge ) { ///Criterio de paro, si se encuentran, se termina
293         return;
294     }
295     ///Dividide y venceras
296     int m( (leftEdge + rightEdge) / 2 );
297     MergeSortSongs( leftEdge, m);
298     MergeSortSongs( m + 1, rightEdge);
299     ///Copiar a temporal
300     T temp[ARRAYSIZE];
301     for( int z(leftEdge) ; z <= rightEdge ; z++ ) {
302         temp[z] = AllMusics[z];
303     }
304     ///Intercalacion
305     int i(leftEdge), j( m + 1 ), x(leftEdge);
306     while ( i <= m and j <= rightEdge) {
307         while( i <= m and temp[i] <= temp[j] ) {
308             AllMusics[x++] = temp[i++];
309         }
310         if( i <= m ) {
311             while( j <= rightEdge and temp[j] <= temp[i] ) {
312                 AllMusics[x++] = temp [j++];
313             }
314         }
315         while ( i <= m ) {
316             AllMusics[x++] = temp[i++];
317         }
318         while( j <= rightEdge ) {
319             AllMusics[x++] = temp [j++];
320         }
321     }
322 }
323 template <class T, int ARRAYSIZE>
324 void Musicbox<T,ARRAYSIZE>::QuickSortSongs() {
325     QuickSortSongs(0,getLastPos());
326 }
327 template <class T, int ARRAYSIZE>
328 void Musicbox<T,ARRAYSIZE>::QuickSortSongs(const int& leftEdge, const int& rightEdge) {
329     if( leftEdge >= rightEdge ) {
330         return;

```

```

331 }
332 ///Separacion
333 int i(leftEdge), j(rightEdge);
334 while( i < j ) {
335     while( i < j and AllMusics[i] <= AllMusics[rightEdge] ) {
336         i++;
337     }
338     while( i < j and AllMusics[j] >= AllMusics[rightEdge] ) {
339         j--;
340     }
341     if( i != j ) {
342         swapSong(AllMusics[i],AllMusics[j]);
343     }
344 }
345 if( i != rightEdge ) {
346     swapSong(AllMusics[i],AllMusics[rightEdge]);
347 }
348 ///Divide y venceras
349 QuickSortSongs(leftEdge, i - 1);
350 QuickSortSongs(i + 1, rightEdge);
351 }
352 template <class T, int ARRAYSIZE>
353 std::string Musicbox<T,ARRAYSIZE>::toString() {
354     std::string AllMusic;
355     for( int i(0) ; i <= Last ; i++ ) {
356         AllMusic += AllMusics[i].toString() + "\n" ;
357     }
358     return AllMusic;
359 }
360 template <class T, int ARRAYSIZE>
361 void Musicbox<T,ARRAYSIZE>::DeleteAllMusic() {
362     Last = -1;
363 }
364 template <class T, int ARRAYSIZE>
365 int Musicbox<T,ARRAYSIZE>::getLastPos() {
366     return Last;
367 }
368 template <class T, int ARRAYSIZE>
369 Musicbox<T,ARRAYSIZE>& Musicbox<T,ARRAYSIZE>::operator = (const Musicbox<T,ARRAYSIZE>& Song) {
370     copyAll(Song);
371     return *this;
372 }
373 #endif // LISTA_H

```