```
#ifndef LISTA_H
#define LISTA_H
#include "Musica.h"
#include <string>
#include "Exception.h"
template <class T, int ARRAYSIZE = 1024>
class Musicbox {
private:
T AllMusics[ARRAYSIZE];
int Last;
void copyAll(const Musicbox<T,ARRAYSIZE>&);
void MergeSortSongs(const int&, const int&);
void QuickSortSongs(const int&, const int&);
void swapSong(T&,T&);
public:
Musicbox();
bool isFull();
bool isEmpty();
bool isValidPos(const int&);
void insertMusic(const int&, const T&);
void deleteMusic(const int&);
int FirstMusic();
int LastMusic();
int PrevMusic(const int&);
int NextMusic(const int&);
int FindDataL(const T&);
int FindDataB(const T&);
T& retrieve(const int&);
void BubbleSortSongs(const char&);
void ShellSortSongs(const char&);
void InsertSortSongs(const char&);
void SelectSortSongs(const char&);
void QuickSortSongs();
void MergeSortSongs();
std::string toString();
void DeleteAllMusic();
int getLastPos();
Musicbox<T,ARRAYSIZE>& operator = (const Musicbox<T,ARRAYSIZE>&);
};
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::copyAll(const Musicbox<T,ARRAYSIZE>& AllMusic) {
int i(0);
while ( i <= Last ) {
this->AllMusics[i] = AllMusic.AllMusics[i];
}
}
template <class T, int ARRAYSIZE>
Musicbox<T,ARRAYSIZE>::Musicbox() : Last(-1) { }
template <class T, int ARRAYSIZE>
bool Musicbox<T,ARRAYSIZE>::isFull() {
return Last == ARRAYSIZE - 1;
}
template <class T, int ARRAYSIZE>
bool Musicbox<T,ARRAYSIZE>::isValidPos(const int& pos) {
return pos > -1 and pos <= Last;
}
template <class T, int ARRAYSIZE>
bool Musicbox<T,ARRAYSIZE>::isEmpty() {
return Last == -1;
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::insertMusic(const int& pos, const T& Song) {
if( isFull() ) {
throw Exception("Desbordamiento de datos, insertMusic");
}
if( pos != -1 and !isValidPos(pos) ) {
```

```
throw Exception("Posicion invalida, insertMusic");
}
int i(Last);
while( i > pos ) {
AllMusics[i] = AllMusics[i-1];
i--;
}
AllMusics[pos + 1] = Song;
Last++;
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::deleteMusic(const int& pos) {
if(isEmpty()) {
throw Exception("Insuficiencia de datos, deleteMusic");
}
if(!isValidPos(pos)) {
throw Exception("Posicion invalida, deleteMusic");
}
int i = pos;
while( i < Last ) {
AllMusics[i] = AllMusics[i+1];
i++;
}
Last--;
}
template <class T, int ARRAYSIZE>
int Musicbox<T,ARRAYSIZE>::FirstMusic() {
if(isEmpty()) {
throw Exception("Insuficiencia de datos, FirstMusic");
}
return 0;
}
template <class T, int ARRAYSIZE>
int Musicbox<T,ARRAYSIZE>::LastMusic() {
if(isEmpty()) {
throw Exception("Insuficienca de datos, LastMusic");
}
return Last;
}
template <class T, int ARRAYSIZE>
int Musicbox<T,ARRAYSIZE>::PrevMusic(const int& pos) {
if( isEmpty() ) {
throw Exception("Insuficiencia de datos, PrevMusic");
}
if( !isValidPos(pos) or pos == 0) {
throw Exception("Posicion invalida, PrevMusic");
}
return pos - 1;
}
template <class T, int ARRAYSIZE>
int Musicbox<T,ARRAYSIZE>::NextMusic(const int& pos) {
if( isEmpty() ) {
throw Exception("Insuficiencia de datos");
}
if( !isValidPos(pos) or pos == Last ) {
throw Exception("Posicion invalida, NextPos");
}
return pos + 1;
}
template <class T, int ARRAYSIZE>
int Musicbox<T,ARRAYSIZE>::FindDataL(const T& Element ) {
if( isEmpty() ) {
throw Exception("Insuficiencia de datos, FindDataL");
}
int i(0);
while ( i <= Last ) {
```

```
if( AllMusics[i] == Element ) {
return i;
}
i++;
}
return -1;
}
template <class T, int ARRAYSIZE>
int Musicbox<T,ARRAYSIZE>::FindDataB(const T& Element) {
if( isEmpty() ) {
throw Exception("Insuficiencia de datos, FindDataB");
}
int i(0), j(Last), m;
while ( i <= j ) {
m = ( i + j ) / 2;
if( AllMusics[m] == Element ) {
return m;
}
if( Element < AllMusics[m] ) {
j = m - 1;
}
else {
i = m + 1;
}
}
return -1;
}
template <class T, int ARRAYSIZE>
T& Musicbox<T,ARRAYSIZE>::retrieve(const int& pos) {
if( isEmpty() ) {
throw Exception("Insuficiencia de datos, retrieveSong");
}
if( !isValidPos(pos) ) {
throw Exception("Insuficiencia de datos, retrieveSong");
}
return AllMusics[pos];
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::swapSong(T& a,T& b) {
T aux(a);
a=b;
b=aux;
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::BubbleSortSongs(const char& opt) {
int i(Last), j;
bool flag;
do {
flag = false;
j = 0;
while( j < i ) {
if( opt == 'A' ) {
if( AllMusics[j].getSongName() > AllMusics[j+1].getSongName() ) {
swapSong( AllMusics[j], AllMusics[i] );
flag = true;
}
}
if( opt == 'B' ) {
if( AllMusics[j].getSongAutor() > AllMusics[j+1].getSongAutor() ) {
swapSong( AllMusics[j], AllMusics[i] );
flag = true;
}
}
j++;
}
i--;
```

```
}
while(flag);
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::ShellSortSongs(const char& opt) {
float fact ( 3.0 / 4.0) ;
int dif( (Last + 1) * fact), lim, i;
while(dif > 0) {
lim = Last - dif;
i=0;
while( i <= lim ) {
if(opt == 'A') {
if( AllMusics[i].getSongName() > AllMusics[ i+ dif ].getSongName() ) {
swapSong( AllMusics[i], AllMusics[i+dif] );
}
}
if(opt == 'B') {
if( AllMusics[i].getSongAutor() > AllMusics[ i+ dif ].getSongAutor() ) {
swapSong( AllMusics[i], AllMusics[i+dif] );
}
}
i++;
}
dif *= fact;
}
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::InsertSortSongs(const char& opt) {
int i(1), j;
T aux;
if( opt == 'A') {
while( i <= Last ) {
aux = AllMusics[i];
j=i;
while ( j > 0 and aux.getSongName() < AllMusics[j-1].getSongName() ) {
AllMusics[j] = AllMusics[j-1];
j--;
}
if( i != j ) {
AllMusics[j] = aux;
}
i++;
}
}
if(opt == 'B') {
while( i <= Last ) {
aux = AllMusics[i];
j=i;
while ( j > 0 and aux.getSongAutor() < AllMusics[j-1].getSongAutor() ) {
AllMusics[j] = AllMusics[j-1];
j--;
}
if( i != j ) {
AllMusics[j] = aux;
}
i++;
}
}
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::SelectSortSongs(const char& opt) {
int i(0), j, m;
while( i < Last ) {
m = i;
j = i + 1;
if( opt == 'A') {
```

```
while( j < Last ) {
if( AllMusics[j].getSongName() < AllMusics[m].getSongName() ) {
m = j;
}
j++;
}
}
if( opt == 'B') {
while( j < Last ) {
if( AllMusics[j].getSongAutor() < AllMusics[m].getSongAutor()) {
m = j;
}
j++;
}
}
if( m!=i ) {
swapSong( AllMusics[i], AllMusics[m] );
}
i++;
}
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::MergeSortSongs() {
MergeSortSongs(0,getLastPos());
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::MergeSortSongs(const int& leftEdge, const int& rightEdge) {
if( leftEdge >= rightEdge ) { ///Criterio de paro, si se encuentran, se termina
return;
}
///Dividide y venceras
int m( (leftEdge + rightEdge) / 2 );
MergeSortSongs( leftEdge, m);
MergeSortSongs( m + 1, rightEdge);
///Copiar a temporal
T temp[ARRAYSIZE];
for( int z(leftEdge) ; z <= rightEdge ; z++ ) {
temp[z] = AllMusics[z];
}
///Intercalacion
int i(leftEdge), j( m + 1 ), x(leftEdge);
while ( i <= m and j <= rightEdge) {
while( i <= m and temp[i] <= temp[j] ) {
AllMusics[x++] = temp[i++];
}
if( i <= m) {
while( j <= rightEdge and temp[j] <= temp[i] ) {
AllMusics[x++] = temp [j++];
}
}
while ( i <= m ) {
AllMusics[x++] = temp[i++];
}
while( j <= rightEdge ) {
AllMusics[x++] = temp [j++];
}
}
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::QuickSortSongs() {
QuickSortSongs(0,getLastPos());
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::QuickSortSongs(const int& leftEdge, const int& rightEdge) {
if( leftEdge >= rightEdge ) {
return;
```

```
}
///Separacion
int i(leftEdge), j(rightEdge);
while( i < j ) {
while( i < j and AllMusics[i] <= AllMusics[rightEdge] ) {
i++;
}
while( i < j and AllMusics[j] >= AllMusics[rightEdge] ) {
j--;
}
if( i != j ) {
swapSong(AllMusics[i],AllMusics[j]);
}
}
if( i != rightEdge ) {
swapSong(AllMusics[i],AllMusics[rightEdge]);
}
///Divide y venceras
QuickSortSongs(leftEdge, i - 1);
QuickSortSongs(i + 1, rightEdge);
}
template <class T, int ARRAYSIZE>
std::string Musicbox<T,ARRAYSIZE>::toString() {
std::string AllMusic;
for( int i(0) ; i <= Last ; i++ ) {
AllMusic += AllMusics[i].toString() + "\n" ;
}
return AllMusic;
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::DeleteAllMusic() {
Last = -1;
}
template <class T, int ARRAYSIZE>
int Musicbox<T,ARRAYSIZE>::getLastPos() {
return Last;
}
template <class T, int ARRAYSIZE>
Musicbox<T,ARRAYSIZE>& Musicbox<T,ARRAYSIZE>::operator = (const Musicbox<T,ARRAYSIZE>& Song) {
copyAll(Song);
return *this;
}
#endif // LISTA_H
```