```cpp
#ifndef LISTA_H
#define LISTA_H

#include <string>
#include "Exception.h"
#include "node.h"

template <class T, int ARRAYSIZE = 1024>
class Musicbox {
public:
class Node {
private:
T data;
public:
Node(const T&);
T& getData();
void setData(const T&);
};
private:
Node* AllSongs[ARRAYSIZE];
int Last;
void copyAll(const Musicbox<T,ARRAYSIZE>&);
void swapSong(Node*,Node*);
public:
Musicbox();
bool isFull();
bool isEmpty();
bool isValidPos(const int&);
void insertSong(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&);
std::string toString();
void DeleteAllMusic();
int getLastPos();
Musicbox<T,ARRAYSIZE>& operator = (const Musicbox<T,ARRAYSIZE>&);
};
template <class T, int ARRAYSIZE>
Musicbox<T,ARRAYSIZE>::Node::Node(const T& e) : data(e) { }
template <class T, int ARRAYSIZE>
T& Musicbox<T,ARRAYSIZE>::Node::getData() {
return data;
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::Node::setData(const T& e) {
data = e;
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::copyAll(const Musicbox<T,ARRAYSIZE>& AllMusic) {
int i(0);
while ( i <= Last ) {
this->*AllSongs[i] = AllMusic.AllSongs[i];
}
}
template <class T, int ARRAYSIZE>
Musicbox<T,ARRAYSIZE>::Musicbox() {
Last=-1;
```

```cpp
int i=0;
while(i<=1023) {
AllSongs[i] = nullptr;
i++;
}
}
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>::insertSong(const int& pos, const T& Song) {
if( isFull() ) {
throw Exception("Desbordamiento de datos, insertSong");
}
if( pos != -1 and !isValidPos(pos) ) {
throw Exception("Posicion invalida, insertSong");
}
Node* aux(new Node(Song));
int i(Last);
while( i > pos ) {
AllSongs[i] = AllSongs[i-1];
i--;
}
AllSongs[pos + 1] = aux;
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;
delete AllSongs[pos];
while( i < Last ) {
AllSongs[i] = AllSongs[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( AllSongs[i]->getData() == 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( AllSongs[m]->getData() == Element ) {
return m;
}
if( Element < AllSongs[m]->getData() ) {
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 AllSongs[pos]->getData();
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::swapSong(Node* a,Node* b) {
Node* 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( AllSongs[j]->getData() > AllSongs[j+1]->getData() ) {
swapSong( AllSongs[j], AllSongs[i] );
flag = true;
}
}
if( opt == 'B' ) {
if( AllSongs[j]->getData() > AllSongs[j+1]->getData() ) {
swapSong( AllSongs[j], AllSongs[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( AllSongs[i]->getData() > AllSongs[ i+ dif ]->getData() ) {
swapSong( AllSongs[i], AllSongs[i+dif] );
}
}
if(opt == 'B') {
if( AllSongs[i]->getData() > AllSongs[ i+ dif ]->getData() ) {
swapSong( AllSongs[i], AllSongs[i+dif] );
}
}
i++;
}
dif *= fact;
}
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::InsertSortSongs(const char& opt) {
int i(1), j;
Node* aux;
if( opt == 'A') {
while( i <= Last ) {
aux = AllSongs[i];
j=i;
while ( j > 0 and aux->getData() < AllSongs[j-1]->getData() ) {
AllSongs[j] = AllSongs[j-1];
j--;
}
if( i != j ) {
AllSongs[j] = aux;
```

```
}
i++;
}
}
if(opt == 'B') {
while( i <= Last ) {
aux = AllSongs[i];
j=i;
while ( j > 0 and aux->getData() < AllSongs[j-1]->getData() ) {
AllSongs[j] = AllSongs[j-1];
j--;
}
if( i != j ) {
AllSongs[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( AllSongs[j]->getData() < AllSongs[m]->getData() ) {
m = j;
}
j++;
}
}
if( opt == 'B') {
while( j < Last ) {
if( AllSongs[j]->getData() < AllSongs[m]->getData() ) {
m = j;
}
j++;
}
}
if( m!=i ) {
swapSong( AllSongs[i], AllSongs[m] );
}
i++;
}
}
template <class T, int ARRAYSIZE>
std::string Musicbox<T,ARRAYSIZE>::toString() {
std::string AllMusic;
for( int i(0) ; i <= Last ; i++ ) {
AllMusic += AllSongs[i]->toString() + "\n" ;
}
return AllMusic;
}
template <class T, int ARRAYSIZE>
void Musicbox<T,ARRAYSIZE>::DeleteAllMusic() {
int i(0);
while(i <= Last) {
delete AllSongs[i];
i++;
}
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
```