Actividad de Aprendizaje 03. La Lista, implementación estática

Christopher Ceballos Jiménez

219750442

Estructura de datos 1

Al principio intente realizar el programa en base a arreglos de estructuras. Tuve problemas al momento de imprimir los arreglos de estructuras por lo tanto decidí empezando de 0 en base a el ejemplo que vimos en clase .

#include <iostream>  
#include <string>  
#include <random>  
#include <chrono>  
#include <functional>  
  
  
#include "list.h"  
#include "song.h"  
  
**using** **namespace** std**;**  
  
void addSong**();**  
void deleteSong**();**  
  
int data**;**  
string myStr**;**  
List **<**Song**>** myLista**;**  
Song mySong**;**  
  
  
default\_random\_engine generator**(**chrono**::**system\_clock**::**now**().**time\_since\_epoch**().**count**());**  
uniform\_int\_distribution**<**int**>** distribution**(**1**,**100**);**  
**auto** dice **=** bind**(**distribution**,** generator**);**  
  
int main **()** **{**  
       
     int op**;**  
       
       
       
     **do{**  
            
          **if** **(**myLista**.**isEmpty**()){**  
               cout **<<** "\nNo hay canciones en el sistema. "**<<**endl**;**  
          **}**  
          **else{**  
               cout**<<** "\n\tTitulo - Artista - Ranking"**<<** endl **<<** endl**;**  
               cout **<<** myLista**.**toString**();**  
          **}**  
            
          cout **<<** "\n\t1- Agregar cancion." **<<** endl**;**  
          cout **<<** "\t2- Eliminar cancion. " **<<** endl**;**  
          cout **<<** "\t0- Salir" **<<** endl**;**  
          cout **<<** "\nElige una opcion: "**;** cin **>>** op**;**  
            
          **switch(**op**){**  
          **case** 1**:**  
                 
               addSong**();**  
                 
               **break;**  
                 
          **case** 2**:**  
                 
               deleteSong**();**  
                 
               **break;**  
                 
          **}**  
            
     **}while(**op **!=** 0**);**  
       
**}**  
  
  
void addSong**()** **{**  
  
     data **=** dice**();**  
     char op**;**  
       
     **do{**  
          std**::**string s **=** ""**;**  
          s **=** std**::**to\_string**(**dice**());**  
            
          cout**<<** "\n\nNombre de la cancion: "**;**  
          fflush**(**stdin**);**getline**(**cin**,** myStr**);**  
          mySong**.**setSongName**(**myStr**);**  
            
          cout**<<** "Artista: "**;**  
          fflush**(**stdin**);**getline**(**cin**,** myStr**);**  
          mySong**.**setAuthor**(**myStr**);**  
            
            
          mySong**.**setRanking**(**s**);**  
            
          myLista**.**insertData**(**myLista**.**getLastPos**(),** mySong**);**  
            
          cout **<<** "\nDeseas agregar otra cancion? (1=Si, 0= No)"**;** cin **>>** op**;**  
            
     **}while(**op **!=** '0'**);**  
       
**}**  
  
void deleteSong**(){**  
       
     int s**;**  
       
     cout **<<** "\nIngresa la posicion de la cancion a eliminar: "**;** cin **>>** s**;**  
       
     myLista**.**deleteData**(**s**);**  
       
     cout **<<** "\nSe ha eliminado correctamente." **<<** endl**;**  
       
**}**

#ifndef LIST\_H\_INCLUDED  
#define LIST\_H\_INCLUDED  
  
#include <iostream>  
#include <string>  
  
  
**template** **<class** T**,** int ARRAYSIZE **=** 100**>**  
**class** List **{**  
**private:**  
     T data**[**ARRAYSIZE**];**  
     int last**;**  
  
     bool isValidPos**(const** int**&);**  
  
     void copyAll**(const** List**&);**  
**public:**  
  
     **class** ListException **:** **public** std**::**exception**{**  
     **private:**  
          std**::**string msg**;**  
  
     **public:**  
          **explicit** ListException**(const** char**\*** message**)** **:** msg**(**message**){}**  
  
          **explicit** ListException**(const** std**::**string**&** message**)** **:** msg**(**message**)** **{}**  
  
          **virtual** **~**ListException**()** **throw()** **{}**  
  
          **virtual** **const** char**\*** what**()** **const** **throw()** **{**  
               **return** msg**.**c\_str**();**  
          **}**  
  
     **};**  
  
     List**();**  
     List**(const** List**&);**  
       
     bool isEmpty**();**  
     bool isFull**();**  
  
     void insertData**(const** int**&,const** T**&);***//primera la pos y luego el elemento*  
     void deleteData**(const** int**&);**  
  
     int getFirstPos**();**  
     int getLastPos**();**  
     int getPrevPos**(const** int**&);**  
     int getNextPos**(const** int**&);**  
  
     *//int findData(const std::string&);*  
     T retrieve**(const** int**&);**  
  
     void sortData**();**  
  
     std**::**string toString**();**  
  
     void deleteAll**();**  
       
     List**&** **operator** **=** **(const** List**&);**  
**};**  
  
*//implementacion*  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
List**<**T**,**ARRAYSIZE**>::**List**()** **:** last**(-**1**)** **{}**  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
List**<**T**,** ARRAYSIZE**>::**List**(const** List**&** s**)** **{**  
     copyAll**(**s**);**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
void List **<**T**,**ARRAYSIZE**>** **::** copyAll**(const** List**&** s**)** **{**  
       
     int i**(**0**);**  
       
     **while** **(**i**<=** s**.**last**)** **{**  
          **this->**data**[**i**]** **=** s**.**data**[**i**];**  
          i**++;**  
     **}**  
       
     **this->**last **=** s**.**last**;**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::**isValidPos**(const** int**&** p**){**  
     **return** p **>=** 0 **and** p**<=** last**;**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::**isEmpty **(** **)** **{**  
     **return** last **==** **-**1**;**  
  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::**isFull **(** **)** **{**  
     **return** last **==** ARRAYSIZE **-** 1**;**  
  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**insertData **(const** int **&** p**,** **const** T **&** e**)** **{**  
  
     **if(**isFull**()){**  
          **throw** ListException**(**"Desbordamiento de datos, insertData"**);**  
     **}**  
  
     **if(**p **!=** **-**1 **and** **!**isValidPos**(**p**)){**  
          **throw** ListException**(**"Posicion invalida, insertData"**);**  
     **}**  
  
     int i**(**last**);**  
  
     **while(**i**>**p**){**  
          data**[**i**+**1**]** **=** data**[**i**];**  
            
          i**--;**  
     **}**  
  
     data**[**p**+**1**]** **=** e**;**  
  
     last**++;**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**deleteData**(const** int**&** p**){**  
       
     **if(!**isValidPos**(**p**)){**  
          **throw** ListException**(**"Posicion invalida, deleteData"**);**  
     **}**  
  
     int i**(**p**);**  
  
     **while(**i **<** last**){**  
            
          data**[**i**]=** data**[**i**+**1**];**  
          i**++;**  
     **}**  
  
     last**--;**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**getFirstPos**(){**  
     **if** **(**isEmpty**()){**  
          **return** **-**1**;**  
     **}**  
  
     **return** 0**;**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**getLastPos **(** **)** **{**  
     **return** last**;**  
  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**getPrevPos **(const** int **&** p**)** **{**  
     **if(**p**==** 0 **or** isValidPos**(**p**)){**  
          **return** **-**1**;**  
     **}**  
  
     **return** p**-**1**;**  
  
  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**getNextPos**(const** int**&** p**){**  
     **if(**p **==** 0 **or** **!**isValidPos**(**p**)){**  
          *//problema*     **}**  
  
     **return** 0**+**1**;**  
**}**  
  
*/\*  
template <class T,int ARRAYSIZE>  
int List<T, ARRAYSIZE>::findData(const std::string& e){  
       
     pos = ;  
     lisdo aux;  
     aux=lista;  
     while(aux->sgte!=NULL)  
     {  
          if(aux->nro==elem)  
          {  
               return 1;  
          }  
          aux=aux->sgte;  
          pos++;  
     }  
     return 0;  
}  
\*/*  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
T List**<**T**,** ARRAYSIZE**>::**retrieve**(const** int**&** p**){**  
     **if(!**isValidPos**(**p**)){**  
          **throw** ListException**(**"Posicion invalida, retrieve"**);**  
     **}**  
  
     **return** data**[**p**];**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortData**(){**  
     *//pendiente***}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
std**::**string List**<**T**,** ARRAYSIZE**>::**toString **(** **)** **{**  
  
     std**::**string result**;**  
     int i**(**0**);**  
  
     **while(**i**<=** last**){**  
            
          result **+=** data**[**i**].**toString**();**  
            
          i**++;**  
     **}**  
       
     **return** result**;**  
       
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**deleteAll **(** **)** **{**  
     last **=** **-**1**;**  
**}**  
  
  
#endif

#ifndef SONG\_H  
#define SONG\_H  
  
#include <string>  
  
**class** Song**{**  
**private:**  
       
     std**::**string songName**;**  
     std**::**string author**;**  
     std**::**string ranking**;**  
       
**public:**  
       
     Song**();**  
     Song**(const** Song**&);**  
       
     Song**&** **operator** **=** **(const** Song**&);**  
       
     std**::**string getSongName**();**  
     std**::**string getaAuthor**();**  
     std**::**string getRanking**();**  
       
     std**::**string toString**();**  
       
     void setSongName**(const** std**::**string**&** **);**  
     void setAuthor**(const** std**::**string**&** **);**  
     void setRanking**(const** std**::**string**&** **);**  
       
**};**  
  
#endif

#include "song.h"  
  
  
  
Song**::**Song **(** **)** **{** **}**  
  
Song**::**Song **(const** Song**&** s**)** **:** songName**(**s**.**songName**),** author**(**s**.**author**),** ranking**(**s**.**ranking**){**  
       
**}**  
  
Song **&** Song**::operator=** **(const** Song**&** s**)** **{**  
       
     songName **=** s**.**songName**;**  
     author **=** s**.**author**;**  
     ranking **=** s**.**ranking**;**  
       
     **return** **\*this;**  
       
**}**  
  
  
std**::**string Song**::**toString **(** **)** **{**  
       
     std**::**string result**;**  
       
       
     result **=** "\t"**;**  
     result **+=** songName**;**  
     result **+=** " - "**;**  
     result **+=** author**;**  
     result **+=** " #"**;**  
     result **+=** ranking**;**  
     result **+=** "\n"**;**  
  
       
     **return** result**;**  
       
**}**  
  
  
  
std**::**string Song**::**getSongName **(** **)** **{**  
     **return** songName**;**  
**}**  
  
std**::**string Song**::**getaAuthor **(** **)** **{**  
     **return** author**;**  
**}**  
  
std**::**string Song**::**getRanking **(** **)** **{**  
     **return** ranking**;**  
**}**  
  
void Song**::**setSongName **(const** std**::**string**&** n**)** **{**  
     songName **=** n**;**  
**}**  
  
void Song**::**setAuthor **(const** std**::**string**&** a**)** **{**  
     author **=** a**;**  
**}**  
  
void Song**::**setRanking **(const** std**::**string**&** r**)** **{**  
     ranking **=** r**;**  
**}**







