|  |  |  |
| --- | --- | --- |
| **American University of Sharjah**  **School of Engineering**  Computer Engineering Department  P. O. Box 26666 Sharjah, UAE |  | **Instructor:** Dr. Ghassan Z. Qadah  **Lab Instructor:** Ms.Praveena Kolli  **Office**: EB2-126  **Phone**: 971-6-515-2352  **e-mail**: pkolli@aus.edu  **Semester**: Spring 2018 |

**CMP305L Data Structures and Algorithms**

**Lab #4 –Unsorted List**

Objectives:

* To understand and implement unsorted linear and linked list
* To understand processing of unsorted linear and linked list

**Exercise 1:**

Abstract classes for **Book** and **UnsortedType** are given below. Implement the **Book** and **UnsortedType classes**. **UnsortedType** list **must** be implemented using linear array.

|  |
| --- |
| const int MAX\_ITEMS = 50 ;  enum RelationType { LESS, EQUAL, GREATER } ;  class Book // declares class data type  {  public : // 3 public member functions  Book(int, char\*,char\*,float,int);  int sellBook(int cout); //decrements the number of copies  int buyBook(int count); //increments the number of copies  RelationType ComparedTo ( Book )const; //Compare with ISBN  void print ( ) const ; //prints book info  int getISBN();  void getAuthor(char[]);  void getTitle(char[]);  float getPrice();  int getNumCopies();  void getPublisher(char[]);  private :  int isbn;  char\* author;  char\* title;  float price;  int numCopies;  char\* publisher;  } ;  class UnsortedType // declares a class data type  {  public : // 8 public member functions  UnsortedType ( ) ;  bool IsFull ( ) const ;  int LengthIs ( ) const ; // returns length of list  void RetrieveItem ( Book& item, bool& found ) ;  void InsertItem (Book item ) ;  void DeleteItem (Book item ) ;  void ResetList ( );  void GetNextItem (Book & item ) ;  private : // 3 private data members  int length ;  ItemType info[MAX\_ITEMS] ;  int currentPos ;  } ; |

**#include<iostream>**

**#include<string>**

**using namespace std;**

**const int MAX\_ITEMS = 50 ;**

**enum RelationType { LESS, EQUAL, GREATER } ;**

**class Book // declares class data type**

**{**

**public : // 3 public member functions**

**Book();**

**Book(int, string,string,string,float,int);**

**Book(const Book & b);**

**~Book();**

**Book& operator=(const Book& b);**

**int sellBook(int count); //decrements the number of copies**

**int buyBook(int count); //increments the number of copies**

**RelationType ComparedTo (const Book &)const; //Compare with ISBN**

**void print ( ) const ; //prints book info**

**int getISBN(){return isbn;}**

**string getAuthor(){return author;}**

**string getTitle(){return title;}**

**float getPrice(){return price;}**

**int getNumCopies(){return numCopies;}**

**string getPublisher(){return publisher;}**

**friend ostream& operator<<(ostream& cout, const Book& b);**

**private :**

**int isbn;**

**string author;**

**string title;**

**float price;**

**int numCopies;**

**string publisher;**

**} ;**

**Book::~Book(){**

**}**

**Book::Book(int ISBN, string Title, string Author, string Publisher, float Price, int NumCopies){**

**isbn = ISBN;**

**author = Author;**

**publisher = Publisher;**

**title = Title;**

**price = Price;**

**numCopies = NumCopies;**

**}**

**Book::Book(){**

**}**

**Book::Book(const Book& b){**

**isbn = b.isbn;**

**author = b.author;**

**title = b.title;**

**price = b.price;**

**numCopies = b.numCopies;**

**publisher = b.publisher;**

**}**

**Book& Book::operator=(const Book& b){**

**isbn = b.isbn;**

**author = b.author;**

**title = b.title;**

**price = b.price;**

**numCopies = b.numCopies;**

**publisher = b.publisher;**

**return \*this;**

**}**

**int Book::sellBook(int count){**

**if(numCopies - count < 0){**

**cout << "Cannot sell "<< count << " copies." << endl;**

**return -1;**

**}**

**numCopies -= count;**

**return numCopies;**

**}**

**int Book::buyBook(int count){**

**numCopies += count;**

**return numCopies;**

**}**

**RelationType Book::ComparedTo(const Book & b) const{**

**if(isbn < b.isbn){return LESS;}**

**if(isbn == b.isbn){return EQUAL;}**

**if(isbn > b.isbn){return GREATER;}**

**}**

**void Book::print() const{**

**cout << "ISBN: " << isbn << " Title: " << title << " Author: "<< author << " Publisher: " << publisher << " Price: " << price << " Number of Copies: " << numCopies;**

**}**

**ostream& operator<<(ostream& cout, const Book& b){**

**b.print();**

**return cout;**

**}**

**Exercise 2:**

ExtendUnsortedType with the below functions,

1. Write a function ***Split*** of the class **UnsortedType** to create two lists **A** and **B** of Books. **A** contains all book from publisher and **B** contains the remaining books. You should not change the calling list. The function ***Split*** should have the following format:

***void*  *UnsortedList::Split* ( *UnsortedList* &A, *UnsortedList* &B , *char\* publisher);***

1. Write a function ***order*** of the class **UnsortedType** to create a list **A** of Books whose number of copies is less than 10. You should not change the calling list**.**

***void*  *UnsortedList::Order* ( *UnsortedList* &A*);***

***#include"Book.cpp"***

***#define ItemType Book***

***class UnsortedType // declares a class data type***

***{***

***public : // 8 public member functions***

***UnsortedType ( ) ;***

***UnsortedType ( const UnsortedType & l ) ;***

***~UnsortedType();***

***bool IsFull ( ) const ;***

***int LengthIs ( ) const ; // returns length of list***

***void RetrieveItem (const ItemType& item, bool& found ) ;***

***void InsertItem (ItemType item ) ;***

***void DeleteItem (ItemType item ) ;***

***void ResetList ( );***

***void GetNextItem (ItemType & item ) ;***

***void Split ( UnsortedType &A, UnsortedType &B , string publisher);***

***void Order ( UnsortedType &A);***

***private : // 3 private data members***

***int length ;***

***ItemType info[MAX\_ITEMS] ;***

***int currentPos ;***

***} ;***

***UnsortedType::UnsortedType(){***

***currentPos = length = 0;***

***}***

***UnsortedType::~UnsortedType(){***

***}***

***UnsortedType::UnsortedType(const UnsortedType & l){***

***length = l.length;***

***currentPos = l.currentPos;***

***for(int i = 0; i < length; ++i){***

***info[i] = l.info[i];***

***}***

***}***

***void UnsortedType::Split ( UnsortedType &A, UnsortedType &B , string publisher){***

***A.ResetList(); B.ResetList();***

***for(int i = 0; i< currentPos; ++i){***

***if(publisher == info[i].getPublisher()){***

***A.InsertItem(info[i]);***

***}else{***

***B.InsertItem(info[i]);***

***}***

***}***

***}***

***void UnsortedType::Order ( UnsortedType &A){***

***A.ResetList();***

***for(int i = 0; i < length; ++i){***

***if(info[i].getNumCopies() < 10){***

***A.InsertItem(info[i]);***

***}***

***}***

***}***

***bool UnsortedType::IsFull() const{return length == MAX\_ITEMS;}***

***int UnsortedType::LengthIs() const{return length;}***

***void UnsortedType::RetrieveItem(const ItemType & item, bool& found){***

***found = false;***

***for(int i = 0; i <= currentPos; ++i){***

***if(info[i].ComparedTo(item) == EQUAL){***

***found = true;***

***break;***

***}***

***}***

***}***

***void UnsortedType::InsertItem(ItemType item){***

***if(!IsFull()){***

***info[currentPos++] = item;***

***if(currentPos > length){***

***length++;***

***}***

***}else{***

***cout << "List is full! \n";***

***}***

***}***

***void UnsortedType::DeleteItem(ItemType item){***

***bool exists;***

***RetrieveItem(item, exists);***

***if(exists){***

***for(int i = 0; i < length; ++i){***

***if(info[i].ComparedTo(item) == EQUAL){***

***info[i] = info[length - 1];***

***break;***

***}***

***}***

***length--;***

***currentPos--;***

***}***

***}***

***void UnsortedType::ResetList(){***

***currentPos = 0;***

***}***

***void UnsortedType::GetNextItem(ItemType & item){***

***if(!IsFull()){***

***item = info[currentPos];***

***currentPos++;***

***}***

***}***

***void printLib(const UnsortedType& lib){***

***ItemType item;***

***UnsortedType p(lib);***

***p.ResetList();***

***int i = 0;***

***while(i < p.LengthIs()){***

***p.GetNextItem(item);***

***cout << item << endl;***

***i++;***

***}***

***}***

**Exercise 3:** (APPLICATION FUNCTIONS))

Write a function that prints all books information stored in UnsortedType.

void printLib(const **UnsortedType & lib)**

Note: Write one single main for all exercises to test the functionality.

#include"UnsortedType.cpp"

int main(){

//This feels like a waste of time. What am I learning?

Book b(1234,"Title","Author", "Publisher", 5.0, 2);

cout << b << endl; //tests Book::print()

b.sellBook(1);

cout << "Copies after selling one book: " << b.getNumCopies() << endl;

b.buyBook(5);

cout << "Copies after buying five books: " << b.getNumCopies() << endl;

cout << b.getISBN() << " " << b.getAuthor() << " " << b.getTitle() << " " << b.getPublisher() << " " << b.getPrice() << endl;

UnsortedType lib;

Book a(1000, "T", "A", "P", 4.2, 6);

lib.InsertItem(a);

lib.InsertItem(b);

if(lib.IsFull()){

cout << "List is full" << endl;

}else{

cout << "List is not full" << endl;

}

cout << "The length of the list is currently " << lib.LengthIs() << endl;

cout << "The library before removing TAP is: " << endl;

printLib(lib); //also tests ResetList, GetNextItem

lib.DeleteItem(a); //also tests RetrieveItem

cout << "The Library after removing TAP is: " << endl;

printLib(lib);

lib.InsertItem(a);

cout << "Now we'll enter a book! What is the ISBN?" << endl;

int isbn;

cin >> isbn;

cin.ignore(1000, '\n');

string title, author, publisher;

cout << "Title?"; getline(cin, title);

cout << "Author?"; getline(cin, author);

cout << "Publisher?"; getline(cin, publisher);

float price;

cout << "Price?"; cin >> price;

int numcopies;

cout << "Number of Copies?"; cin >> numcopies;

//I know it's a bad practice to not validate user input.

//But if the user inputs something that breaks the program

//And the user is me

//Then the user is dumb and should run the program again

Book c(isbn, title, author, publisher, price, numcopies);

lib.InsertItem(c);

UnsortedType x, y;

lib.Split(x,y,"P");

cout << "The books published by the illustrious and prestigious P are: " << endl;

printLib(x);

cout << "The books published by other people are: " << endl;

printLib(y);

UnsortedType z;

lib.Order(z);

cout << "The books which we need to order more copies of are: " << endl;

printLib(z);

return 0;

}

