HW03

소프트웨어융합학과 2019102106 예다정

1-1.

a. 기능: 메소드를 호출한 UnsortedType타입의 리스트 안에 ItemType타입의 객체가 있는지 확인

precondition: item(입력 파라미터)의 멤버 데이터 value와 UnsortedType타입의 리스트가 초기화돼야 한다.

Postcondition: unsorted list안에 해당 item이 존재하면 true를, 그렇지 않으면 false를 반환한다.

b. bool UnsortedType::isThere(const ItemType& item);

c. bool UnsortedType::isThere(const ItemType& item) {

for (int location = 0; location < length; location++) {

if (item.ComparedTo(info[location]) == EQUAL)

return true;

}

return false;

}

🡪item의 value값이 this->info배열 안에 존재하는지 확인하기 위해 this->info의 원소들과 item의 value를 비교한다. this->info배열의 인덱스를 하나씩 키워가며 총 this->length개의 원소까지 확인한다. Item.value와 this->info안의 ItemType원소들이 같은지 비교하기위해 item.ComparedTo 메소드를 사용한다. ComparedTo의 return값이 EQUAL인 경우 isThere함수는 true를, LESS나 GREATER인 경우 false를 반환한다.

d. bool UnsortedType::isThere(const ItemType& item) {

ItemType \*location = this->info;

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

if (item.ComparedTo(\*location) == EQUAL)

return true;

else

location++;

}

return false;

}

🡪this->info배열의 시작 주솟값을 저장하는 포인터 변수 location을 선언한다. location의 크기를 sizeof(ItemType)씩 키워가며 info배열 안의 값과 item을 length번만큼 비교한다.

e. c, d에서의 함수 둘 다 O(N)만큼의 시간 복잡도를 가진다. / 순서가 없으므로 info배열 안의 length개(=N) 만큼의 원소들과 하나씩 비교해봐야 한다.

1-2.

a. 기능: UnsortedType타입의 리스트 안에 ItemType타입의 객체가 있는지 확인

precondition: UnsortedType타입의 객체와 ItemType타입의 객체가 초기화돼야 한다.

postcondition: item이 list안에 있을 경우 true를, 그렇지 않을 경우 false를 반환한다.

b.

bool isThere(UnsortedType &list, const ItemType &item) {

ItemType compareItem;

list.ResetList();

for (int location = 0; location < list.LengthIs(); location++) {

list.GetNextItem(compareItem);

if (item.ComparedTo(compareItem) == EQUAL)

return true;

}

return false;

}

int main() {

ItemType item1, item2, item3, item4, item5;

UnsortedType list;

item1.Initialize(5);

item2.Initialize(1);

item3.Initialize(3);

item4.Initialize(4);

item5.Initialize(2);

list.InsertItem(item1);

list.InsertItem(item2);

list.InsertItem(item3);

list.InsertItem(item4);

bool result = isThere(list, item4);

cout << result << endl;

result = isThere(list, item5);

cout << result << endl;

return 0;

}

c. 메소드일 때의 isThere 함수와 달리 client함수일 때의 isThere함수는 입력파라미터에 UnsortedType타입의 객체와 ItemType타입의 객체 모두 필요하다. 또한 UnsortedType객체의 멤버 변수(length, info)에 직접적으로 접근할 수 없으므로 isLength()나 GetNextItem()과 같은 public 메소드를 이용해 멤버 데이터에 접근해야한다.

d. O(N) / 순서가 없으므로 info배열 안의 원소 length개(=N) 만큼 GetNextItem()을 이용해 currentPos를 늘리며 하나씩 비교해봐야 한다.

1-3.

a. void UnsortedType::SplitLists(ItemType item, UnsortedType &list1, UnsortedType &list2) {

for (int idx = 0; idx < this->length; idx++) {

if (item.ComparedTo(this->info[idx]) == EQUAL || item.ComparedTo(this->info[idx]) == GREATER) //list원소가 item보다 작거나 같은 경우

list1.InsertItem(this->info[idx]);

else

list2.InsertItem(this->info[idx]);

}

}

b. (linked list를 아직 배우지 않았기 때문에 안 해도 됨)

2.

a. library ADT Specification

- Structure: library클래스는 책 제목을 저장하는 1차원 배열과 남은 권수를 저장하는 1차원 배열을 가진다. 또한 iteration을 위해 현재 위치를 저장하는 변수(iterName, iterCount)를 이용해 리스트 내의 원소들을 차례로 확인할 수 있다. 두 배열은 최대 MAX\_ITEMS 크기만큼의 원소를 가질 수 있다.

- Member Data:

string name[]

int count[]

int iterName

int iterCount

int length

- Operations:

MakeEmpty

Function: 두 개의 배열을 초기화한다. (MAX\_ITEMS 크기만큼)

Preconditions: x

Postconditions: 배열이 초기화된다.

IsFull

Function: 배열 안에 원소가 꽉 찼는지 확인한다.

Preconditions: 배열이 초기화돼야 한다.

Postconditions: 배열이 꽉 찼으면 (원소가 MAX\_ITEMS개만큼 있으면) true를, 그렇지 않으면 false를 반환한다.

LenghtIs

Function: 배열 안에 저장된 원소들의 개수 = length를 반환한다.

Preconditions: 배열이 초기화돼야 한다.

Postconditions: length가 반환된다.

IsThere

Function: name배열 안에 입력한 책이 있는지 확인한다.

Preconditions: name배열이 초기화돼야 한다.

Postconditions: 입력한 책이 name배열의 length길이 내에 있으면 true를, 그렇지 않으면 false를 반환한다.

InsertBook

Function: name배열 안에 입력한 책을 추가하고 count배열 안에 입력한 권수를 동일한 index에 추가한다.

Preconditions: 배열이 초기화돼야 하고 배열이 꽉 차지 않아야 한다. (원소의 개수가MAX\_ITEMS 미만이어야 한다. name배열 안에 입력한 책과 동일한 원소가 없어야 한다.

Postconditions: 입력한 책이 name배열에 추가되고 입력한 개수가 count배열의 같은 위치에 추가된다.

DeleteBook

Function: name배열에 입력한 책을 삭제하고 count배열의 동일한 index에 위치한 값(해당 책의 잔여 권수)을 삭제한다.

Preconditions: 배열이 초기화돼야 하고 length가 1이상이어야 한다. name배열 안에 입력한 책이 존재해야 한다.

Postconditions: 입력한 책이 name배열에서 삭제되고 count배열의 동일한 index에 위치한 값이 삭제된다.

ResetName

Function: iteration을 위해서 iterName을 제일 -1로 초기화한다.

Preconditions: 배열이 초기화돼야 한다.

Postconditions: iterName에 -1이 저장된다.

ResetCount

Function: iteration을 위해서 iterCount를 제일 -1로 초기화한다.

Preconditions: 배열이 초기화돼야 한다.

Postconditions: iterCount에 -1이 저장된다.

GetNextName

Function: name배열의 iterName+1위치의 원소를 가져온다.

Preconditions: name배열이 초기화돼야 한다. iterName이 length미만이어야 한다.

Postconditions: iterName+1위치의 원소를 가져온다.

GetNextCount

Function: count배열의 iterCount+1위치의 원소를 가져온다.

Preconditions: count배열이 초기화돼야 한다. iterCount가 length미만이어야 한다.

Postconditions: iterCount+1위치의 원소를 가져온다.

Borrow

Function: 입력한 책 index에 있는 count배열의 원소에서 1을 뺀다. count배열의 해당 위치의 값이 0인 경우 대출 불가능하다.

Preconditions: name배열에 입력한 책이 존재해야 한다.

Postconditions: count배열의 해당 위치에서 1을 뺀다. count값이 0인 경우 대출 불가 메세지를 출력한다.

Return

Function: 입력한 책 index에 있는 count배열의 원소에서 1을 더한다.

Preconditions: name배열에 입력한 책이 존재해야 한다.

Postconditions: count배열의 해당 위치에서 1을 더한다.

b.

#include <iostream>

#include <string>

using namespace std;

int MAX\_ITEMS = 50;

class Library {

private:

int length;

string name[MAX\_ITEMS];

int count[MAX\_ITEMS];

int iterName;

int iterCount;

public:

void MakeEmpty();

bool IsFull();

int LenghtIs();

bool IsThere(string book);

void InsertBook(string book, int num);

void DeleteBook(string book);

void ResetName();

void ResetCount();

void GetNextName(string &book);

void GetNextCount(int &num);

void Borrow(string book);

void Return(string book);

};