

8

리스트 ADT의 이중 연결 리스트 구현

목적

이번 연구에서 여러분은

- 이중 연결 리스트를 사용하여 리스트 ADT를 구현한다.
- 퍼즐 프로그램을 만든다.
- 연결 리스트를 역행한다.
- 여러분의 리스트 ADT의 이중 연결 리스트 구현의 효율성을 분석한다.

개요

여러분이 실습 7에서 만든 리스트 ADT의 단일 연결 리스트는 매우 효과적이지만 리스트에서 삭제와 역방향 이동시에는 효과적이지 않다. 이번 실습에서 여러분은 선형 이중 연결 리스트를 사용하여 리스트 ADT의 구현을 만든다. 이 구현은 대개의 리스트 ADT 기능들을 고정된 시간내에 수행한다.

List ADT

원소

리스트의 원소는 일반적인 LE 형태이다.

구조

원소들은 선형구조의 형태를 한다. 원소들의 순서는 각 원소가 삽입되는 때와 위치에 따라 결정되고, 리스트에 있는 데이터의 함수(function)가 아니다. 어느 시점에서 도 비어 있지 않은 리스트의 원소는 커서를 사용하여 표시한다. 여러분은 커서의 위치를 변경하는 기능을 사용하여 리스트에서 움직인다.

기능

List (int ignored = 0)

요구사항:

없음.

결과:

객체 생성자. 빈 리스트를 생성하라. 매개변수는 배열 구현으로 양립호출(call compatibility)에 대한 매개변수가 제공되고 무시된다.

~List ()

요구사항:

없음.

결과:

객체 생성자. 리스트 저장에 사용된 메모리 할당 해제한다.

void insert (const LE &newElement)

요구사항:

리스트는 가득 차 있지 않다.

결과:

리스트에 newElement를 삽입한다. 리스트가 비어 있지 않다면 커서 다음에 newElement를 삽입한다. 그렇지 않으면 리스트의 첫번째 원소로 newElement를 삽입한다. 커서를 newElement로 이동시킨다.

Void remove ()

요구사항:

없음.

결과:

리스트에서 커서가 표시하는 원소를 제거한다. 리스트가 비어 있으면 커서를 삭제된 원소 뒤에 있는 원소로 이동시킨다. 삭제된 원소가 리스트의 끝에 있으면 커서를 리스트의 처음으로 이동시킨다.

void replace (const LE & newElement)**요구사항:**

리스트는 비어 있지 않다.

결과:

커서가 표시하는 원소를 newElement로 교환한다. 커서는 newElement에 위치한다.

void clear ()**요구사항:**

없음.

결과:

리스트에서 모든 원소를 지운다.

int empty () const**요구사항:**

없음.

결과:

리스트가 비어 있으면 1을 아니면 0을 반환한다.

int full () const**요구사항:**

없음.

결과:

리스트가 가득 차 있으면 1을 아니면 0을 반환한다.

int gotoBeginning ()

요구사항:

없음.

결과:

리스트가 비어 있지 않으면 커서를 리스트로 끝으로 이동시키고 1을 반환하고 그렇지 않으면 0을 반환한다.

int gotoNext ()

요구사항:

리스트는 비어 있다.

결과:

커서가 리스트의 끝에 있지 않으면 커서를 리스트의 다음 원소로 커서를 이동시키고 1을 반환하고 아닌 경우는 0을 반환한다.

int gotoprior ()

요구사항:

리스트는 비어 있다.

결과:

커서가 리스트의 처음에 있지 않으면 커서를 리스트의 이전 원소로 이동시키고 1을 반환하고 그렇지 않으면 0을 반환한다.

LE getCursor () const

요구사항:

리스트는 비어 있다.

결과:

커서가 표시하는 원소의 복사본을 반환한다.

void showStructure () const

요구사항:

없음.

결과:

리스트의 원소를 출력한다. 리스트가 비어 있으면 "Empty list"를 출력하라. 이 연산자는 단지 시험과 디버깅을 목적으로 한다. C++에서 미리 지정한 타입(int, char 등)의 리스트 원소들만 지원한다.

실습 8 : 표지

날짜_____ 구분_____

성명_____

강사가 여러분에게 할당한 다음에 연습문제들에 할당된 열에 체크표시를 하고 이 겹장을 앞으로의 실습에서 여러분이 제출할 과제물 앞에 붙이시요

	할 일	완 료
실습 전 연습	✓	
연 결 연습	✓	
실습 중 연습 1		
실습 중 연습 2		
실습 중 연습 3		
실습 후 연습 1		
실습 후 연습 2		
	총 점	

실습 8 : 실습 전 연습

날짜_____ 구분_____

성명_____

이중 연결 리스트의 각 노드는 한 쌍의 포인터를 갖는다. 하나의 포인터는 리스트의 이전 노드(prior)를 가리키고, 또 다른 포인터는 다음 노드(next)를 가리킨다. ListNode 클래스는 여러분이 실습 7에서 사용한 것과 비슷하다.

```

Template <class LE>
class ListNode    // facilitator class for the List class
{
    private:
        // Construct
        ListNode (const LE &elem, ListNode * nextPtr);
        LE element;           // List element
        ListNode *prior,      // Pointer to the previous element
                *next;        // Pointer to the next element

        friend class List <LE>;
};

```

환형 이중 연결 리스트에서 처음과 마지막 노드들은 서로 연결되어 있다. 리스트의 마지막에 있는 노드의 next 포인터는 처음 노드를 가리키고 처음에 있는 노드의 prior 포인터는 마지막에 있는 노드를 가리킨다.

1단계 : 환형 이중 링크들 리스트를 이용하여 List ADT로 기능을 구현하라. 파일

 listdbl.h의 클래스 선언들을 기본으로 구현하라. showStructure 기능 구현은 파일

 show8.cpp에 있다.

2단계 : 여러분의 List ADT 구현을 파일 listdbl.cpp로 저장하라. 여러분의 코드를 문서화하라.

실습 8 : 실습 전 연습

날짜 _____ 구분 _____
 성명 _____

여러분의 강사와 함께 여러분의 실습 전이나 실습 중에 이 연습문제를 완료할 수 있는지 점검하라.

파일 `test8.cpp`에 있는 시험 프로그램에서 여러분은 아래의 명령어들을 사용하는 리스트 ADT 구현을 상호적으로 점검할 수 있다.

명령어	동작
+x	커서 다음에 x를 삽입
-	커서가 표시하는 원소를 삭제
=x	커서가 표시하는 원소를 원소 x로 교체
@	커서가 표시하는 원소를 화면에 출력
N	다음 원소로 이동
P	이전 원소로 이동
<	리스트의 시작으로 이동
>	리스트의 끝으로 이동
E	리스트가 비었는지 조사
F	리스트가 가득 찼는지 조사
C	리스트를 clear
Q	테스트 프로그램 종료

1단계 : 리스트 ADT 구현에 대한 시험 계획을 준비하라. 시험 계획은 처음, 중간, 마지막에 있는 원소에 대한 각 기능의 응용을 포함해야(cover)한다. 시험 항목은 아래와 같다.

2단계 : 여러분의 시험 계획을 실행하고 리스트 ADT 구현에 잘못이 있으면 수정하고 다시 실행하라.

실습 8 : 실습 중 연습 1

날짜_____ 구분_____

성명_____

리스트는 다른 클래스에서 데이터 멤버로 사용되어질 수 있다. 이 연습에서 퍼즐의 해결과 현 퍼즐의 형상을 저장하는 문자 리스트를 이용하여 아래에 서술된 철자 바꾸기 퍼즐 ADT 구현을 만든다.

ANAGRAM PUZZLE ADT

원소

알파벳 문자

구조

문자들은 환형으로 정리된다. 적절하게 정리되면 지정된 영어 단어의 철자를 쓴다.

기능

AnagramPuzzle (char answ [], char init [])

요구사항:

문자열 answ와 init은 비어 있지 않고 같은 문자들을 갖고 있다(순서는 다름).

결과:

객체 생성자. 낱말 바꾸기 퍼즐을 만든다. 문자열 answ는 퍼즐에 대한 해답이고 문자열 init는 초기의 문자 순서이다.

void shiftLeft ()

요구사항:

없음.

결과:

퍼즐에서의 문자들을 왼쪽으로 한 칸씩 이동시킨다. 가장 좌측 문자는 퍼즐의 오른쪽 끝으로 이동시킨다.

void swapEnds ()

요구사항:

없음.

결과:

퍼즐의 왼쪽과 오른쪽 끝의 문자를 바꾼다.

void display ()

요구사항:

없음.

결과:

낱말 바꾸기 퍼즐을 화면 출력한다.

int solved ()

요구사항:

없음.


결과:

퍼즐을 풀었으면 1을 반환 아니면 0을 반환한다.

아래의 코드는 "yes"를 "yse"로 섞는 퍼즐을 선언한다. 퍼즐이 "yes"의 형태로 바뀌는 방법을 보여 준다.

```
AnagramPuzzle enigma("yes", "yse"); // word is "yes",
                                     start w/ "yse"
enigma.shitLeft ( );                // Changes puzzle to "sey"
enigma.swapEnds ( );                // Changes puzzle to "yes"
```

퍼즐 프로그램은 인코딩된 퍼즐을 해결하는 것이 아니라 키보드로 명령어를 입력하여 퍼즐을 해결한다.

 **1단계 :** 파일 `puzzle.cs`에 있는 낱말 바꾸기 프로그램을 낱말 바꾸기 퍼즐 ADT 구현을 만들어서 완성하라. 다음의 선언을 기본으로 구현하라.

```

class AnagramPuzzle
{
public:

    AnagramPuzzle(char answ[], char init[]); // Construct
                                           puzzle

    void shiftLeft( );           // shift letters left
    void swapEnds( );           // Swap end letters
    void display( );            // Display puzzle
    int solved( );              // puzzle solved

private:
    // Data members
    List<char> solution, // Solution to puzzle
              puzzle;   // Current puzzle configuration
};

```

퍼즐의 해결책을 저장하는 문자의 리스트와 현재의 형상을 표현하는데 List ADT의 환형 이중 연결 리스트 구현을 사용하라.

2단계 : 여러분의 낱말 바꾸기 프로그램을 아래의 시험 프로그램을 사용하여 시험하라.

퍼즐 맞추기(Anagram Puzzle Program) 시험 계획	
시험 항목	검사
"yse"로 바뀐 퍼즐 단어 "yes"	
"irthg"로 바뀐 퍼즐 단어 "right"	

실습 8 : 실습 중 연습 2

날짜 _____ 구분 _____

성명 _____

리스트는 두 가지 방법으로 역행할 수 있다. 여러분이 리스트의 노드를 새로운 순서로 다시 연결하거나 노드의 구조를 손상하지 않고 노드들 간에 원소를 변경할 수 있다. 다음의 리스트 ADT 기능을 구현하기 위해 이 두 방법 중의 하나를 사용한다.

void revers ()

요구사항:

없음.

결과:

리스트 원소의 순서를 역순으로 한다.

1단계 : 이 기능을 구현하고, 파일 *listdbl.cpp*에 추가하라. 이 기능의 프로토타입은 파일 *listdbl.h*의 List class에 선언되어 있다.

2단계 : 파일 *test8.cpp*의 시험 프로그램에서 "//R"로 시작되는 행에서 주석 경계자와 문자 R을 삭제하여 'R'명령어를 활성화하라.


3단계 : 단일 원소를 갖는 리스트를 포함해서 다양한 길이의 리스트를 다루는 reverse 기능에 대한 시험 계획을 준비하라. 시험 계획은 다음과 같이 구성한다.

4단계 : 여러분의 시험 계획을 수행하라. Reverse 기능의 구현에 잘못이 있으면 수정하고 다시 실행하라.

실습 8 : 실습 중 연습 3

날짜 _____ 구분 _____
 성명 _____


여러 리스트 응용 프로그램에서 여러분은 리스트의 원소의 갯수와 커서의 관련된 위치를 알 필요가 있다. 요청할 때마다 이 속성들을 계산하는 것이 아니라 여러분은 원소를 삽입, 삭제하고 커서를 이동할 때마다 갱신하는 두 개의 데이터 멤버로 이 정보를 저장할 수 있다.

 **1단계 :** 다음의 데이터 멤버를 파일 *listdbl.h*에 있는 List class 선언에 추가하고 결과를 파일 *listdbl2.h*로 저장하라.

size : 리스트내의 원소들의 갯수.

pos : 커서의 숫자의 위치 시작부터 끝까지 0부터 번호가 부여된 리스트 원소들의 커서의 숫자적인 위치.

2단계 : 여러분의 List ADT의 환형 이중 연결 리스트 구현에서 필요시 데이터 멤버들의 갱신이 이뤄지도록 프로그램들을 수정하라. 수정된 여러분의 구현을 파일 *listdbl2.cpp*에 저장하라.

 **3단계 :** size와 pos 데이터 멤버를 응용프로그램에서 참조한다면 이 값들을 반환하는 List ADT 기능을 가지고 있어야 한다. 다음의 연산자에 대한 프로토타입을 파일 *listdbl2.h*에 있는 List class 선언에 추가하라.

int length () const

요구사항:

없음.

결과:

리스트내의 원소의 갯수를 반환한다.

int position () const

요구사항:

리스트는 비어 있다.

결과:

처음부터 끝까지 0부터 번호가 매겨진 리스트 원소들의 커서의 위치를 반환한다.

4단계 : 이 기능들을 구현하고 파일 *listdl2.cpp*에 추가하라.

5단계 : 여러분의 변경과 유기적으로 동작하는 루틴들이 여러분이 실습 전에 만든 파일 *test8.cpp*의 시험 프로그램을 수정하라.

6단계 : "//#"로 시작하는 행에서 주석 경계자와 문자 '#'를 제거하여 '#'명령어(길이와 위치)를 활성화하라.

7단계 : 빈 리스트를 포함해서 다양한 리스트의 길이와 리스트의 시작, 중간, 마지막 원소의 숫자상의 위치를 검사하는 기능들의 시험 계획을 준비하라.

8단계 : 시험 계획을 수행하라. 기능들의 구현에서 잘못이 발견되면 수정하고 다시 실행하라.

Test Plan for the length and position Operation			
시험 항목	명령어	예상 결과	검사