CSED 232 Object-Oriented Programing (Spring 2021)

Programming Assignment # 2

- Classes & objects–

Due date : 4월 9일

담당 조교 : 연주은 (jeyeon@postech.ac.kr)

주의사항

- 모든 문제에 대한 구현 사항을 만족하여야 하며, string, cmath를 제외한 추가적인 library의 사용은 허용하지 않습니다.
- 각 클래스마다 header file과 cpp file을 나누어 구현하여야 합니다. 추가적으로 필요한 멤버 함수(method)나 멤버 변수(member variable)가 있을 경우, 추가해도 무방합니다.
- 문제에 명시되어 있지 않더라도 각 클래스마다 생성자(Constructor), 소멸자(Destructor)는 필수입니다. 기본 생성자의 경우, 별다른 언급이 없다면 멤버 변수에 int, double은 0, string은 "", pointer는 NULL로 초기화해 주십시오.
- 클래스마다 클래스명과 파일명을 통일하여 주십시오.
- 모든 클래스의 멤버 변수는 private으로 선언하여야 합니다.

감점

- 제출 기한이 지나면 얻은 총점의 20% 감점
- 추가로 하루(24시간) 늦을 때마다 20%씩 감점
  1일 이내 지연: 20% 감점, 2일 이내 지연: 40% 감점, 5일 이상 지연: 0점
- 컴파일이 정상적으로 이루어지지 않을 경우 0점

제출방식

채점은 Windows Visual Studio 2019 환경에서 이루어집니다. 파일을 업로드하실 때, 작업하신 환경이 있는 **프로젝트 폴더를 디버그 폴더를 삭제한 후, 그대로 압축**해서 올려 주시기 바랍니다. 폴더명은 문제#_학번으로 만들어 주십시오. 또한 문제 폴더 안에 각 문제에 해당하는 Report도 같이 넣어서 zip파일로 만든 후 제출해 주시기 바랍니다. 문제마다 따로 프로젝트를 생성하고, 따로 압축하여 제출해 주시기 바랍니다. 즉, **총 3개의 파일을 제출**하셔야 합니다. 반드시 PLMS를 통해 제출해주시기 바랍니다. 이메일 제출은 인정되지 않습니다. 5일이 지날 경우 0점이므로 과제 제출 마감일로부터 5일후인 4월 14일 23시 59분 59초 이후는 PLMS를 통해 제출하실 수 없습니다.

예)prob1_20209999.zip, prob2_20209999.zip , prob3_20209999.zip

채점 기준

1. 프로그램 기능
    - 프로그램이 요구 사항을 모두 만족하면서 올바로 실행되는가?

2. 프로그램 설계 및 구현
    - 각 클래스가 header file과 cpp file로 나누어 졌는가?
    - 요구 사항을 만족하기 위한 변수 및 알고리즘 설계가 잘 되었는가?
    - 설계된 내용이 요구된 언어를 이용하여 적절히 구현되었는가?

3. 프로그램 가독성
    - 프로그램이 읽기 쉽고 이해하기 쉽게 작성되었는가?
    - 변수 명이 무엇을 의미하는지 이해하기 쉬운가?
    - 프로그램의 소스 코드를 이해하기 쉽도록 주석을 잘 붙였는가?

4. 보고서 구성 및 내용, 양식
    - 보고서는 적절한 내용으로 이해하기 쉽고 보기 좋게 잘 작성되었는가?
    - 보고서의 양식을 잘 따랐는가?

다른 사람의 프로그램이나 인터넷에 있는 프로그램을 복사(copy)하거나 간단히 수정해서 제출하면 학점은 무조건 'F'가 됩니다. 이러한 부정행위가 발견되면 학과에서 정한 기준에 따라 추가의 불이익이 있을 수 있습니다.

문제 1번 (배점 : 20점)

분수를 표현하는 Fraction class를 구현한다. Fraction class는 다음과 같은 member를 가진다.

- int numerator : 분자
- int denominator : 분모

Fraction class는 다음과 같은 method를 가진다. 분수를 나타내는 클래스에서 분모는 0이 될 수 없으므로 클래스의 생성자 및 저장 method에서 분모가 0일 경우 에러를 출력 후 프로그램을 종료하라. 음의 분수의 경우, 분자를 음수로 표현하여 음의 분수를 표현한다.

- Fraction()

  : 기본 생성자로, 분자는 0, 분모는 1인 분수를 만든다.

- Fraction(int numer)

  : 분모가 1인 분수를 만든다.

- Fraction(int numer, int denom)

  : 분자가 numer, 분모가 denom인 분수를 만든다. numer에는 양수, denom에는 음수가 들어왔을 경우, 분자를 음수로, 분모를 양수로 만들어 음의 분수를 표현한다.

- Fraction(const Fraction& copyFrom) : 복사 생성자
- ~Fraction() : 소멸자. 소멸자에서 수행하는 일은 없으므로, 공백으로 두면 됩니다.
- void store(int numer, int denom)

  : 멤버 변수 numerator, denominator를 설정하는 함수이다. (setter 함수와 같은 역할)

- inline void print() const

  : 분수를 화면에 출력한다.

  명시적 인라인 함수로 구현하여라.

Fraction class의 member들에 대한 getter, setter 함수를 구현하여라.

Fraction class에 다음 연산자들을 오버로딩하여 구현하여라.

- 전위 증가 연산자 ++fr1

  : 증가 연산자의 경우, 분자에 분모의 값을 더하는 형태로 연산을 수행합니다.

  예를 들면, 3/4 + 1 => 7/4 가 됩니다.

- 후위 증가 연산자 fr1++

  : 분자에 분모의 값을 더하는 형태로 연산을 수행합니다.

- 대입 연산자 fr1=fr2
- 이항 증가 연산자 fr1+=fr2

- 덧셈 연산자 fr3=fr1+fr2
- 뺄셈 연산자 fr3=fr1-fr2
- 곱셈 연산자 fr3=fr1*fr2
- 나눗셈 연산자 fr3=fr1/fr2

main 함수에 요구되는 구현 조건은 없다. 즉 main 함수를 공백으로 두어도 되고, 다음 페이지에 나온 것과 같이 코드를 구성하여 구현한 class가 제대로 작동하는지 테스트해볼 수도 있다. 제출 시 다른 클래스의 코드와 main.cpp코드를 포함한 프로젝트를 디버그 폴더 삭제 후, 압축하여 제출한다.

>> main.cpp

```
#include <iostream>
#include "Fraction.h"
using namespace std;

int main() {
	Fraction fr1;
	fr1.store(2, 3);

	Fraction fr2(7, 10);
	Fraction fr3;

	fr3 = fr1 + fr2;
	cout << "value of fr3 after fr3 = fr1 + fr2 operation : ";
	fr3.print();

	fr3 = fr1 - fr2;
	cout << "value of fr3 after fr3 = fr1 - fr2 operation : ";
	fr3.print();

	fr3 = fr1 * fr2;
	cout << "value of fr3 after fr3 = fr1 * fr2 operation : ";
	fr3.print();

	fr3 = fr1 / fr2;
	cout << "value of fr3 after fr3 = fr1 / fr2 operation : ";
	fr3.print();

	fr2 = fr1;
	cout << "value of fr2 after fr2 = fr1 operation : ";
	fr2.print();

	fr2 += fr1;
	cout << "value of fr2 after fr2 += fr1 operation : ";
	fr2.print();

	fr2++;
	cout << "value of fr2 after fr2++ operation : ";
	fr2.print();

	++fr2;
	cout << "value of fr2 after ++fr2 operation : ";
	fr2.print();

	return 0;
}
```

세부 채점 기준

1. 프로그램 기능 – 20%
2. 프로그램 설계 및 구현 – 60%
    - 생성자 및 소멸자 구현 (10%)
    - store() 구현 (5%)
    - print() 구현 (5%)
    - 연산자 구현 (각 5%)
3. 프로그램 가독성 – 10%
4. 보고서 구성 및 내용, 양식 – 10%

문제 2번 (배점 : 30점)

아래와 같이 기본적인 벡터 연산을 하는 Vector 클래스를 구현하여, 클래스 Line, Plane을 정의하고 투영점을 구할 수 있는 라이브러리를 구현하여라.

(cmath 라이브러리 사용이 가능합니다.)

(Vector 클래스는 Point를 표현하기 위해서도 사용될 수 있다.)

1. Vector: (x,y,z)

Vector class는 다음과 같은 member를 가진다.

- double x, y, z

Vector class는 다음과 같은 method를 가진다.

- Vector()
- Vector(double x, double y, double z)
- ~Vector() : 소멸자에서 수행하는 일은 없으므로, 공백으로 두면 됩니다.
- double innerProduct(Vector v1, Vector v2)

  : 두 벡터를 내적한 결과를 반환한다.
- Vector outerProduct(Vector v1, Vector v2)

  : 두 벡터를 외적한 결과를 반환한다.
- Vector plus(Vector v1, Vector v2)

  : 두 벡터의 합을 반환한다.
- Vector minus(Vector v1, Vector v2)

  : 두 벡터의 차를 반환한다.
- Vector multiply(double s, Vector v)

  : 벡터에 상수를 곱하여 반환한다.
- double magnitude()

  : 벡터의 크기를 반환한다.
- Vector normalize()

  : 벡터의 단위 벡터를 반환한다.
- string toString()

  : (x, y, z) 꼴의 string을 반환한다.

2. Line: $(\vec{P}_1 - \vec{P}_0)t + \vec{P}_0 = \vec{l}t + \vec{l}_0$

Line class는 다음과 같은 member를 가진다.

- Vector l, l0

Line class는 다음과 같은 method를 가진다.

- Line()
- Line(const Point& po1, const Point& po2)

  : 두 점을 지나는 직선의 식을 구하여 l과 l0에 알맞은 값을 대입한다. 이 때, l0의 값으로 po1을 대입한다.
- ~Line() : 소멸자에서 수행하는 일은 없으므로, 공백으로 두면 됩니다.
- Point projectTo(const Point& po)

  : 점 po를 이 직선에 투영한 점을 계산하여 반환한다. (projection 계산을 하여 반환)
- string toString()

  : Line: (l.x, l.y, l.z)t + (l0.x, l0.y, l0.z) 꼴의 string을 반환한다.

3. Plane: $\vec{n}(\vec{r} - \vec{r}_0) = \vec{0}$

Plane class는 다음과 같은 member를 가진다.

- Vector n, r0;

Plane class는 다음과 같은 method를 가진다.

- Plane()
- Plane(const Point& po1, const Point& po2, const Point& po3)

  : 세 점을 지나는 평면의 식을 구하여 n과 r0에 알맞은 값을 대입한다. 평면이 결정되지 않는 경우는 고려하지 않는다. 이 때, r0의 값으로 po1을, n의 값으로 평면의 단위 법선 벡터를 대입한다.
- ~Plane() : 소멸자에서 수행하는 일은 없으므로, 공백으로 두면 됩니다.
- Point projectTo(const Point& po)

  : 점 po를 이 면에 투영한 점을 계산하여 반환한다. (projection 계산을 하여 반환)
- string toString()

  : Plane: (n.x, n.y, n.z)(r - (r0.x, r0.y, r0.z))=0 꼴의 string을 반환한다.

main 함수에 요구되는 구현 조건은 없다. 즉 main 함수를 공백으로 두어도 되고, 다음과 같이 코드를 구성하여 구현한 class가 제대로 작동하는지 테스트해볼 수도 있다. 제출 시 다른 클래스의 코드와 main.cpp코드를 포함한 프로젝트를 디버그 폴더 삭제 후, 압축하여 제출한다.

>> main.cpp

```
#include <iostream>
#include "Vector.h"
#include "Line.h"
#include "Plane.h"
using namespace std;

int main() {
	Point p1(0, 0, 0);
	Point p2(1, 1, 1);
	Line l1(p1, p2);
	cout << l1.toString() << endl;

	Point p3(1, 2, 1);
	Plane pl1(p1, p2, p3);
	cout << pl1.toString() << endl;

	return 0;
}
```

세부 채점 기준

1. 프로그램 기능 – 20%
2. 프로그램 설계 및 구현 – 60%
   - Vector class 구현 (30%)
   - Line class 구현 (15%)
   - Plane class 구현 (15%)
3. 프로그램 가독성 – 10%
4. 보고서 구성 및 내용, 양식 – 10%

문제 3번 (배점 : 50점)

자료구조 중 하나인 Sorted Doubly Linked List를 설계 및 구현하고, Iteration 기능을 독립적인 클래스로 구현하라. 이를 테스트하기 위한 driver로 Application 클래스도 구현하여라.

1. Application

완성된 클래스들의 테스트를 위하여 Application class를 구현한다. Application class에서는 DoublyLinkedList의 method를 호출하는 형태로 각 command의 기능을 수행한다.

Application class는 다음과 같은 member를 가진다.

- DoublyLinkedList *dList
- int mCommand : 입력 받은 command를 저장. 입력 받은 것이 없다면 0으로 설정한다.

Application class는 다음과 같은 method를 가진다.

- 생성자와 소멸자
- void Run()

  : 사용자들에게 입력받은 command를 멤버 변수로 저장하고, 실행하는 함수이다. 올바르지 않은 command를 입력 받았을 경우, 오류 메시지를 출력한 후 함수를 종료한다.
- int GetCommand()

  : 사용자로부터 command를 입력 받는 함수이다. 아래와 같이 작성한다.

```
#include "Application.h"

int Application::GetCommand(){
        int command;

        cout << endl << endl;
        cout << "₩t---ID -- Command ----- " << endl;
        cout << "₩t    1 : Add new item" << endl;
        cout << "₩t    2 : Delete item" << endl;
        cout << "₩t    3 : Update item" << endl;
        cout << "₩t    4 : Search item by ID" << endl;
        cout << "₩t    5 : Print all on screen" << endl;
        cout << "₩t    6 : Make empty list" << endl;
        cout << "₩t    0 : Quit" << endl;

        cout << endl << "₩t Choose a Command--> ";
        cin >> command;
        cout << endl;

        return command;
}
```

- int AddItem()

  : 사용자로부터 원하는 item을 받아 리스트에 추가한다. 성공하였을 경우 1, 실패하였을 경우 0을 반환한다.

- int DeleteItem()

  : 사용자로부터 원하는 key값(ID)을 받아 해당 item을 리스트에서 삭제한다. 성공하였을 경우 1, 실패하였을 경우 0을 반환한다.

- int UpdateItem()

  : 사용자로부터 어떤 item을 어떻게 갱신할 것인지 받아 수행한다. 성공하였을 경우 1, 실패하였을 경우 0을 반환한다.

- int SearchItem()

  : 사용자로부터 받은 key 값으로 item을 찾는다. 찾았을 경우 해당하는 item을 출력하고, 그렇지 않을 경우 찾지 못했다는 메시지를 출력한 후 0을 반환한다.

- void DisplayAllItem()

  : 리스트의 모든 item을 출력한다.

- void MakeEmptyList()

  : 리스트의 모든 item을 삭제한다.

2. ItemType

사용되는 레코드를 위한 class이다.

ItemType class는 다음과 같은 member를 가진다.

- int id : key 값으로 사용되는 data. 초기값은 –1로 설정된다.
- string name

ItemType class는 다음과 같은 method를 가진다.

- 생성자와 소멸자 (기본 생성자 외에 복사 생성자도 필요합니다.)
- getter, setter 함수
- key 값인 id를 비교하기 위한 연산자 오버로딩 (>, <, ==, = 필수)

3. NodeType

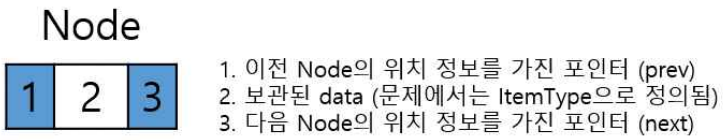
Doubly Linked List의 구현을 위해 사용되는 자료형으로 NodeType struct를 구현한다. NodeType struct는 DoublyLinkedList의 헤더 파일에 구현한다.

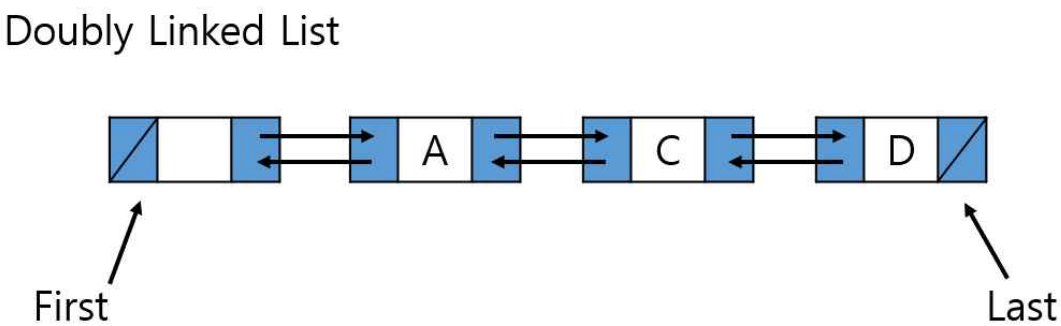
NodeType struct는 다음과 같은 member를 가진다.

- ItemType data : 노드에서 관리할 레코드
- NodeType *prev : 이전 노드를 가리키는 pointer
- NodeType *next : 다음 노드를 가리키는 pointer

4. DoublyLinkedList

Doubly Linked List는 각 node가 이후 node 뿐만 아니라 이전 node와도 서로 연결된 형태의 Linked List이다.





DoublyLinkedList class는 다음과 같은 member를 가진다.

- NodeType *pFirst : 리스트의 처음 노드를 가리키는 포인터
- NodeType *pLast : 리스트의 마지막 노드를 가리키는 포인터
- int length : 리스트에 저장된 레코드의 수

DoublyLinkedList class는 다음과 같은 method를 가진다.

- 생성자와 소멸자, getter, setter 함수
- bool IsEmpty()

  : 리스트가 비어 있는지 확인한다.
- void MakeEmpty()

  : 리스트를 빈 리스트로 만든다.
- int GetLength() const

  : 리스트가 보유하고 있는 레코드의 개수를 반환한다.
- int Add(ItemType item)

  : 새로운 레코드 item을 리스트에 추가한다. 빈 리스트였을 경우, 맨 앞에 item을 추가한다. 만약 이미 존재하는 Id를 가진 item이었을 경우, 오류 메시지를 출력한 후 0을 반환한다. 구현하는 것은 Doubly Sorted Linked List이므로, key 값인 Id 값에 따라 삽입하는 위치가 결정된다. 맨 앞 또는 맨 뒤에 item이 추가되는 경우, 멤버 변수를 알맞게 갱신한다. 성공하면 1 아니면 0을 반환한다.
- int Delete(ItemType item)

  : 기존에 존재하는 레코드(item과 같은 key 값을 가진 레코드)를 삭제한다. 성공하면 1 아니면 0을 반환한다.
- int Replace(ItemType item)

  : 기존에 존재하는 레코드(item과 같은 key 값을 가진 레코드)를 item으로 갱신한다. 성공하면 1 아니면 0을 반환한다.
- int Get(ItemType &item)

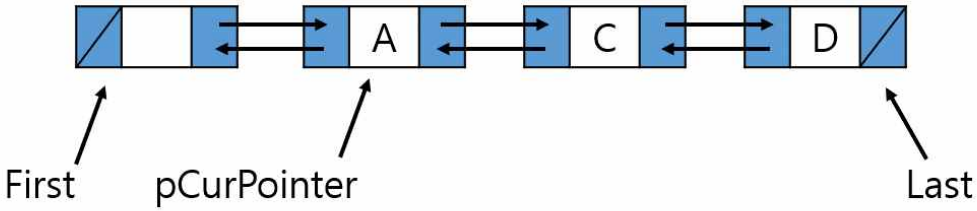
  : 데이터를 검색하고, 해당하는 데이터를 item으로 가져온다. 성공하면 1 아니면 0을 반환한다.

5. Iterator

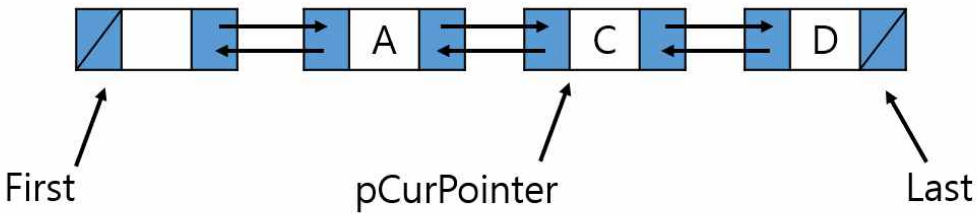
Iterator class는 구조화된 자료에서 data를 체계적(혹은 순차적)으로 접근하는 클래스이다. Iterator class를 별도로 구현하는 이유는 Iterator가 함수로 구현될 경우 여러 곳에서 동시에 iteration 기능을 사용할 수 없기 때문이다. Doubly Linked List class의 복잡한 연산을 돕기 위해 전체 list를 순차적으로 접근하는 기능을 별도로 제공한다.

이해를 돕기 위해 DoublyLinkedList에서 Add 동작이 어떻게 수행되는지를 표현하면 아래와 같다.

Add("B") B라는 data를 삽입한다고 할 때, Iterator의 pCurPointer는 리스트를 First 노드부터 원하는 노드(이 경우, data를 삽입해야 하는 위치)에 도달할 때까지 하나씩 방문합니다.

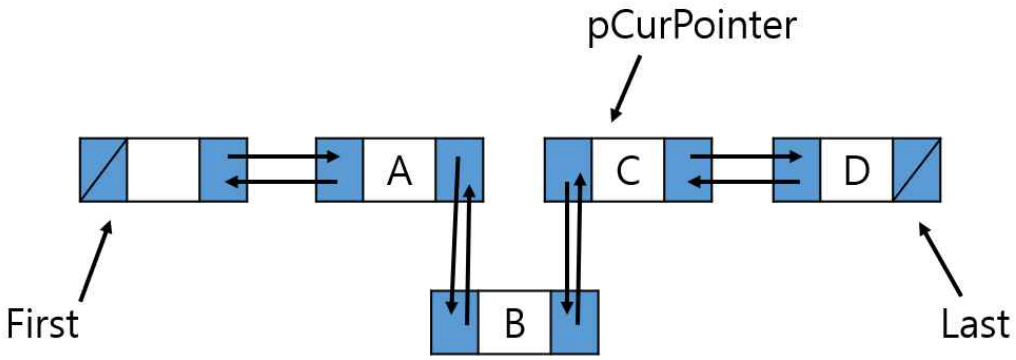


A > B true?
No
(원하는 노드가 아니므로 다음 노드로 이동합니다.)



C > B true?
Yes
(원하는 노드에 도착했으므로 Add 동작을 수행합니다.)

A data를 가진 노드의 next 포인터와, C data를 가진 노드의 prev 포인터가
B data를 가진 노드를 가리키도록 변경된 후, B data를 가진 노드의 포인터들이
각각 A, C data를 가진 노드를 가리키도록 바뀌는 것으로 Add 동작이 수행됩니다.



Iterator class는 다음과 같은 member를 가진다.

- const DoublyLinkedList &dList : 사용할 리스트의 참조 변수
- NodeType *pCurPointer : Iterator 변수

Iterator class는 다음과 같은 method를 가진다. (기본 생성자는 구현하지 않습니다.)

- Iterator(const DoublyLinkedList &list)

  : 생성자. dList의 값으로 list를, pCurPointer의 값으로 list의 pFirst member를 저장한다.
- 소멸자, getter, setter 함수
- bool NotNull()

  : list의 현재 원소가 null인지 아닌지 검사한다.
- bool NextNotNull()

  : list의 다음 원소가 null인지 아닌지 검사한다.
- bool PrevNotNull()

  : list의 이전 원소가 null인지 아닌지 검사한다.
- ItemType First()

  : list의 처음 노드의 item을 반환한다.
- ItemType Next()

  : 다음 노드로 이동하고 해당 노드의 item을 반환한다.
- ItemType Prev()

  : 이전 노드로 이동하고 해당 노드의 item을 반환한다.
- NodeType GetCurrentNode()

  : 현재 노드를 반환한다.

main 함수는 아래와 같이 코드를 구성하며, 변경하지 않는다. 제출 시 다른 클래스의 코드와 main.cpp코드를 포함한 프로젝트를 디버그 폴더 삭제 후, 압축하여 제출한다.

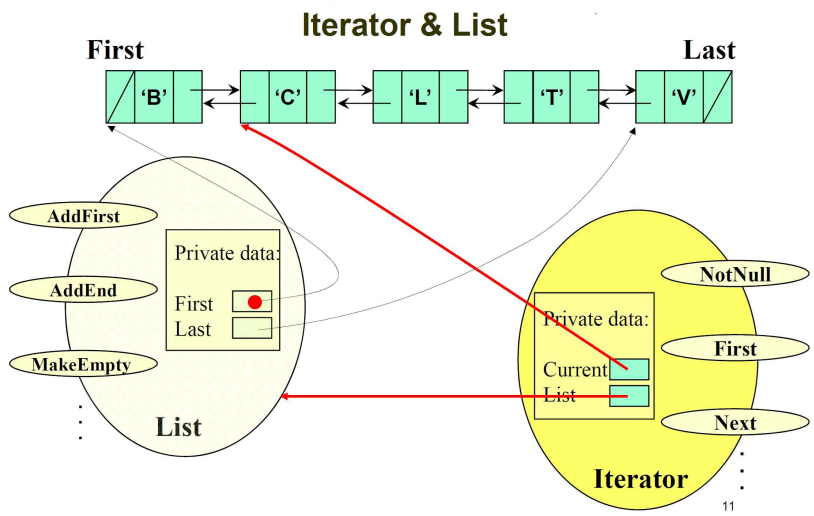
>> main.cpp

```
#include "Application.h"

int main() {
        Application app;
        app.Run();

        return 0;
}
```

세부 채점 기준

1. 프로그램 기능 – 20%
2. 프로그램 설계 및 구현 – 60%
    - Application class 구현 (8%)
    - ItemType clcass 구현 (5%)
    - NodeType struct 구현 (2%)
    - DoublyLinkedList 구현 (30%)
    - Iterator class 구현 (15%)
3. 프로그램 가독성 – 10%
4. 보고서 구성 및 내용, 양식 – 10%

<참고. Iterator와 List>



<참고. Iterator의 사용 예>

```
void DoublyLinkedList::MakeEmpty() {
        NodeType *pItem;
        Iterator itor(*this);          // this 포인터를 이용하여 Iterator 선언

        while(itor.NotNull())          // itor가 가리키는 현재 위치가 Null이 아닐 경우 반복
        {
                pItem = itor.getpCurPointer();      // itor가 가리키는 위치의 노드 pItem
                itor.Next();                        // itor가 다음 노드를 가리키도록
                delete pItem;                       // pItem 노드를 삭제
        }

        this->pFirst = NULL;
        this->pLast = NULL;
        this->length = 0;

        return;
}
```