실습문제

- 교재 420-421페이지, 실습문제 8-5, 8-6을 응용함
  - 프로젝트명 : 학번_Chap_8_5_6
  - main 파일명 : 학번_Chap_8_5_6.cpp
  - 이번 프로젝트는 실습문제 8-5와 8-6를 참고하되 다음 페이지들에서 설명한 것처럼 이를 응용한 하나의 프로그램으로 완성하라.

  - 프로그램 실행결과

```
큐에 삽입할 5개의 정수를 입력하라>> 1 3 5 7 9
큐의 크기:5
큐의 원소를 순서대로 제거하여 출력한다>> 1 3 5 7 9
큐의 현재 크기 : 0
스택에 삽입할 5개의 정수를 입력하라>> 1 3 5 7 9
스택 크기:5
스택의 모든 원소를 팝하여 출력한다>> 9 7 5 3 1
스택의 현재 크기 : 0
```

기존 실습문제 8-5, 8-6 실습문제와의 차이점

- 스택과 큐를 구현하는 방식은 크게 배열을 이용하는 방식과 linked list로 구현하는 두가지 방식이 있다.
- 기존 실습문제 8-5와 8-6은 BaseArray를 상속받아 MyQueue와 MyStack을 구현하였다.
- BaseArray는 동적으로 배열 메모리를 할당 받은 후 적절한 배열원소에 값을 넣거나(put()) 얻는(get()) 클래스다.
- 이번 과제에서는 배열의 개념을 이용하는 BaseArray 대신 double linked list인 BaseList 클래스를 구현하고, 이 BaseList를 상속하여 MyQueue와 MyStack을 구현하도록 한다.
- 즉, MyQueue와 MyStack를 배열을 이용하는 것이 아니라 double linked list인 BaseList를 이용하여 구현해 보는 것이다.

프로그램 전체 구성 및 main() 함수


```
헤드 파일 include
class Node { } 선언

class BaseList { } 선언
BaseList의 세 개 멤버 함수 구현 코드

class MyQueue { } 선언
MyQueue의 test() 함수 구현 코드

class MyStack { } 선언
MyStack의 test() 함수 구현 코드

int main()
{
    MyQueue::test();
    MyStack::test();
    // 실행 결과는 첫 페이지 참조
}
```

리스트의 한 노드인 Node 클래스 선언


```
// 정수값 value를 저장하고 있는 double linked 리스트의 한 노드
class Node {
private:
    // 모든 멤버를 private으로 선언하여 외부에서 접근 못하게 하고,
    // BaseList에서만 Node의 모든 멤버에 접근할 수 있게 클래스 전체를 friend로 선언
    friend class BaseList;

    Node *prev;    // 리스트의    앞 노드를 포인트함
    Node *next;    // 리스트의 다음 노드를 포인트함
    int value;     // 노드에 저장할 정수값, 생성자에서 저장됨

    Node(int value) { prev = next = nullptr;   this->value = value; } // 초기화

    int getValue() { return value; } // 노드에 저장된 정수값 리턴

    // 현재 노드 다음에 n 노드를 추가한다.
    void add(Node *n) { next = n;   n->prev = this; }

    // 현재 노드를 삭제하고 이 노드의 next 노드를 리턴한다.
    Node *remove() { Node *n = next;    delete this;    return n; }
};
```

double linked 리스트인 BaseList 선언


```
// 여러 개의 노드들을 가지고 있는 double linked 리스트
class BaseList {
private:
    Node *head; // double linked 리스트의 맨 처음   노드를 가리키는 포인터
    Node *tail;   // double linked 리스트의 맨 마지막 노드를 가리키는 포인터
protected:
    int size;       // 현재 리스트 내에 있는 노드의 개수

    BaseList() { head = tail = nullptr;    size = 0; } // 생성자: 멤버 초기화

    // 소멸자: 모든 노드를 삭제함
    ~BaseList() { for (Node *n = head; n != nullptr; n = n->remove()); }

    void add_rear(int value);  // 리스트의 맨 마지막에 노드 추가 후, value저장
    int remove_rear();   // 리스트의 맨 마지막에 노드 삭제 후, value 값 리턴
    int remove_front();  // 리스트의 첫 번째 노드 삭제 후, value 값 리턴
};
```

add_rear(), remove_rear(), remove_front() 구현: 뒤쪽 페이지들 참조

MyQueue 클래스 선언

- BaseList를 상속받아 큐처럼 작동하는 MyQueue 클래스를 작성하라.

```
// 큐는 FIFO방식으로 삽입/삭제되어야 함
class MyQueue ... {   // BaseList를 상속받게 선언해야 함
public:
    // 큐의 맨 끝에 노드(value 저장된) 추가
    void enqueue(int value) {
        Base List의 적절한 멤버함수를 호출함
    }
    // 큐의 맨 앞의 노드 삭제하고 저장된 value리턴
    int dequeue() {
        return  Base List의 적절한 멤버함수를 호출함;
    }
    // 큐의 길이를 리턴
    int length() { return size; }

    static void test();
};                          // First In First Out
```

MyQueue 활용 사례 및 출력 결과

- MyQueue의 정적 함수 test(): MyQueue를 테스트하는 코드

```
MyQueue의 정적 함수 test() { // 교재 420, 실습문제 8-5 코드와 거의 동일
    MyQueue mQ; // 수정됨
    int n;
    cout << "큐에 삽입할 5개의 정수를 입력하라>> ";
    for (int i = 0; i<5; i++) {
        cin >> n;
        mQ.enqueue(n); // 큐에 삽입
    }
    cout << "큐의 크기:" << mQ.length() << endl;  // 수정됨
    cout << "큐의 원소를 순서대로 제거하여 출력한다>> ";
    while (mQ.length() != 0)
        cout << mQ.dequeue() << ' '; // 큐에서 제거하여 출력
    cout << endl << "큐의 현재 크기 : " << mQ.length() << endl;
}
```

- MyQueue::test() 실행 결과

```
큐에 삽입할 5개의 정수를 입력하라>> 1 3 5 7 9
큐의 크기:5
큐의 원소를 순서대로 제거하여 출력한다>> 1 3 5 7 9
큐의 현재 크기 : 0
```

MyStack 클래스 선언

- BaseList를 상속받아 스택으로 작동하는 MyStack 클래스를 작성하라.

```
// 스택은 LIFO방식으로 삽입/삭제되어야 함
class MyStack ... {   // BaseList를 상속받게 선언해야 함
public:
    // 스택의 맨 끝에 노드(value 저장) 추가
    void push(int value) {
        Base List의 적절한 멤버함수를 호출함
    }
    // 스택의 맨 끝의 노드 삭제하고 저장된 value리턴
    int pop() {
        return Base List의 적절한 멤버함수를 호출함;
    }
    // 스택의 길이를 리턴
    int length() { return size; }

    static void test();
};                          // Last In First Out
```

MyStack 활용 사례 및 출력 결과

- MyStack의 정적 함수 test(): MyStack을 테스트하는 코드

```
MyStack의 정적 함수 test() { // 교재 421, 실습문제 8-6 코드와 거의 동일
    MyStack mStack; // 수정됨
    int n;
    cout << "스택에 삽입할 5개의 정수를 입력하라>> ";
    for (int i = 0; i<5; i++) {
        cin >> n;
        mStack.push(n); // 스택에 푸시
    }
    cout << "스택 크기:" << mStack.length() << endl; // 수정됨
    cout << "스택의 모든 원소를 팝하여 출력한다>> ";
    while (mStack.length() != 0)
        cout << mStack.pop() << ' '; // 스택에서 팝
    cout << endl << "스택의 현재 크기 : " << mStack.length() << endl;
}
```

- MyStack::test() 실행 결과

```
스택에 삽입할 5개의 정수를 입력하라>> 1 3 5 7 9
스택 크기:5
스택의 모든 원소를 팝하여 출력한다>> 9 7 5 3 1
스택의 현재 크기 : 0
```

BaseList의 멤버 함수 구현


```
// 새로운 노드(value 값을 저장하고 있음)를 생성한 후 리스트의 맨 마지막에 추가
BaseList의 add_rear(int value) 멤버 함수 선언 {
    Node의 포인터 변수 n을 선언하고
    새로운 Node를 동적으로 생성(value 값 전달)하여 n에 저장;
    만약 tail이 nullptr이 아니면 // NULL 대신에 nullptr 사용
        tail의 멤버함수 add(n)호출하여 리스트의 맨 끝에 n을 추가
        리스트 끝에 새로 추가 되었으므로 tail을 n으로 설정
    아닌 경우 // 리스트에 노드가 없을 경우 n이 첫 노드가 됨
        tail과 head 값을 n으로 설정
    리스트의 노드 개수를 1 증가
}
```

BaseList의 멤버 함수 구현


```
// 리스트의 마지막 노드를 제거한 후 그 노드에 저장된 value를 리턴
BaseList의 remove_rear() 멤버 함수 선언 {
    리스트의 노드 개수가 0이면 -1 리턴 // list empty error
    정수형 변수 val을 선언하고,
    tail노드에 저장된 value 값을 얻어와 val에 저장
    // 삭제할 노드의 앞 노드를 저장할 변수 prev 선언 및 초기화
    Node * 변수 prev를 선언하고 tail의 prev 멤버 값을 변수 prev에 저장
    tail 노드를 메모리에 반납함
    리스트의 노드 개수를 1 감소
    변수 prev가 nullptr이 아니면 // 삭제할 노드의 앞 노드가 존재할 경우
        변수 prev가 포인트하는 노드의 next 멤버를 nullptr로 설정함
        tail에 변수 prev을 설정 // 이제 마지막 노드는 삭제된 노드의 앞 노드가 됨
    아닌경우 즉, 변수 prev가 nullptr이면 // 더 이상 남은 노드가 없는 경우
         tail과 head 값을 nullptr로 설정
    val 값을 리턴
}
```

BaseList의 멤버 함수 구현


```
// 리스트의 맨 처음 노드를 제거한 후 그 노드에 저장된 value를 리턴
BaseList의 remove_front() 멤버 함수 선언 {
    리스트의 노드 개수가 0이면 -1 리턴 // list empty error
    정수형 변수 val을 선언하고,
    head 노드에 저장된 value 값을 얻어와 val에 저장
    // 삭제할 노드의 뒤 노드를 저장할 변수 next 선언 및 초기화
    Node * 변수 next를 선언하고 head의 next 멤버 값을 변수 next에 저장
    head 노드를 메모리에 반납함
    리스트의 노드 개수를 1 감소
    변수 next가 nullptr이 아니면 // 삭제할 노드의 뒤 노드가 존재할 경우
        변수 next가 포인트하는 노드의 prev 멤버를 nullptr로 설정함
        head에 변수 next를 설정 // 이제 맨 첫 노드는 삭제된 노드의 뒤 노드가 됨
    아닌경우 즉, 변수 next가 nullptr이면 // 더 이상 남은 노드가 없는 경우
         tail과 head 값을 nullptr로 설정
    val 값을 리턴
}
```