실습문제

1

- 교재 371페이지, 실습5, 실습문제 7-11, 7-12의 혼합
  - 프로젝트명 : 학번_Chap_7_11
  - main 파일명 : 학번_Chap_7_11.cpp
  - 이번 프로젝트는 실습5에서 구현한 코드를 그대로 활용할 것이다. 먼저 실습 5에서 작성한 학번_Chap_5_8.cpp의 내용을 그대로 학번_Chap_7_11.cpp에 복사하라.
  - 기존의 MyIntStack에 푸시(push)용으로 << 연산자를, 팝(pop)을 위해 >> 연산자를, 비어 있는 스택인지 알기 위해 !연산자를 작성하라. (멤버 연산자 함수로 구현)
  - 기존 스택에 다른 스택의 내용을 대입하는 = 연산자를 작성하라. (멤버 연산자 함수로 구현)
  - 두 스택의 내용을 합쳐 새로운 스택을 만들어 리턴하는 + 연산자 함수를 작성하라. 단, 이 연산자 함수를 외부 함수로 구현하고 MyIntStack 내부에 friend 함수로 선언하라.

프로그램 전체 구성

헤드 파일 include

class MyIntStack { } 선언

MyIntStack의 각 멤버 함수 구현 코드

외부 + 연산자 함수 구현 코드

main() 함수

class MyIntStack


```
class MyIntStack {
    int *p;          // 스택 메모리로 사용할 포인터
    int size;        // 스택의 최대 크기
    int tos;         // 스택의 탑을 가리키는 인덱스
public:
    MyIntStack();
    MyIntStack(int size);
    MyIntStack(const MyIntStack& s); // 복사 생성자
    ~MyIntStack();
    bool push(int n); // 정수 n을 스택에 푸시한다.
    bool pop(int &n); // 스택의 탑에 있는 값을 n에 팝한다.

    // 새로 추가해야 할 연산자 함수
    MyIntStack& operator << (int n);  // push: 객체 자신의 참조자를 리턴
    MyIntStack& operator >> (int& n); // pop: 객체 자신의 참조자를 리턴
    bool operator ! ();               // 스택이 비었으면 true 리턴
    MyIntStack& operator = (const MyIntStack& s);
                                      // 현재 스택에 s 스택의 내용을 대입
    외부 + 연산자 함수를 프렌드로 선언; // 예제 7-12 참고
};
```

main() 함수 및 실행결과


```
int main() {
    . . .
    // 기존 main() 함수의 마지막에 다음을 추가한다.
    // 아래에서 a와 n은 기존에 이미 선언된 변수임

    a << 3 << 5 << 7;    // 3, 5, 10을 순서대로 푸시
    MyIntStack c;
    c = a + b;           // a, b 두 스택 스택을 합친 후 결과를 c에 대입
    while (true) {
        if (!c) break;   // 스택이 비었으면 빠져 나감
        c >> n;
        cout << n << ' ';
    }
    cout << endl;
}
```

- 실행결과

```
스택 a에서 팝한 값 20
스택 b에서 팝한 값 30
20 10 7 5 3 10
```

MyIntStack의 <<, >>, ! 연산자 함수


```
MyIntStack& MyIntStack::operator << (int n) { // push
    멤버 함수인 push()를 호출하여 n을 스택에 푸시함
    // 스택이 가득 찬 경우 아무 것도 하지 않음; push() 리턴 값도 무시
    이 객체의 참조 리턴 // 예제 7-14 참조
}

MyIntStack& MyIntStack::operator >> (int &n) { // pop
    멤버 함수인 pop()를 호출함(스택에서 팝한 값을 n에 저장)
    // 스택이 빈 경우 아무 것도 하지 않음; pop() 리턴 값도 무시
    이 객체의 참조 리턴 // 예제 7-14 참조
}

bool MyIntStack::operator ! () {
    스택이 비어 있으면 true 리턴 // 스택이 빈 상태 체크는 pop() 함수 참조
    그렇지 않으면 false 리턴
}
```

MyIntStack의 = 연산자 함수

- a = b; 연산에서 = 연산자는 객체 a의 기존 스택용 배열 메모리 p[]를 모두 delete시키고 객체 b의 스택 최대 크기만큼 다시 메모리를 할당 받은 후 객체 b의 배열 p[] 모든 내용을 복사하도록 작성되어야 한다. 또한 클래스의 나머지 멤버 데이터도 1:1 복사해야 한다.
- = 연산자와 복사생성자와의 차이점을 잘 구분하기 바람
  - 복사생성자: 새로 생성되는 비어 있는 객체에 기존의 객체를 복사하여 초기화함
  - a=b 연산자: 기존에 이미 값이 존재하는 a 객체 내의 모든 동적 할당 메모리를 모두 delete하여 반납(동적으로 할당 받은 멤버를 먼저 반납)하고 난 후에 기존 객체 b를 복사하여 a에 저장; 만약 동적으로 할당 받은 멤버가 없다면 b를 a에 그냥 복사해도 됨

```
MyIntStack& MyIntStack::operator = (const MyIntStack& s) {
    포인터 p가 nullptr이 아니면, // ~MyIntStack() 참조
        p가 가리키는 메모리 반환 // 동적으로 할당 받은 멤버를 먼저 반납
    복사 생성자 함수의 내용을 그대로 복사해서 넣을 것
    // 복사 생성자 MyIntStack(const MyIntStack& s) 참조
    이 객체의 참조 리턴 // 예제 7-14 참조
}
```

외부 + 연산자 함수

- a + b; 연산에서 + 연산자는 객체 a의 스택 크기와 b의 스택 크기를 합친 크기의 새로운 스택 tmp를 만든다. 그런 후 a의 스택용 배열 메모리 p[]의 내용을 tmp의 p[]에 복사하고, 그 뒤에 객체 b의 스택 메모리 p[]의 내용을 tmp의 p[]에 복사(a를 복사한 뒤쪽에)하도록 작성되어야 한다(주의: 스택 크기 만큼이 아닌 스택 탑 만큼만 복사해야 함). 또한 tmp의 tos 값도 적절히 설정해야 함.

```
MyIntStack operator + (const MyIntStack& s1, const MyIntStack& s2) {
    MyIntStack 타입의 지역변수 tmp(합친_크기) 선언; // 예제 7-12 참고
    // 이때 합친_크기는 s1 스택 크기와 s2 스택 크기를 합친 크기여야 함
    s1의 tos와 s2의 tos 멤버를 합친 값을 tmp의 tos 멤버에 저장
    for문을 이용하여          // for문은 예제 4-6 참조
        s1의 p[i]를 tmp 객체의 p[i]에 저장
        // 이때 모든 원소가 아니라 s1 스택 탑의 개수 만큼만 복사
    for문을 이용하여 // 모든 원소가 아니라 s2 스택 탑의 개수 만큼만 복사
        s2 객체의 p[i]를 tmp 객체의 p[i+?]에 저장
        // 이때 s1의 p[i]를 모두 복사하고 난 뒤쪽에,
        // 즉 tmp의 p[i+?]에 s2의 p[i]를 저장해야 함
    tmp를 리턴함 // 예제 7-12 참고
}
```