과제 명세

1. 내용

- 기존 Complex 클래스에서
  (1) + 연산자, - 연산자(이항), -연산자(단항) 함수를 구현
  (2) ++, -- 증감연산자 함수 구현(각각 prefix, postfix)
  (3) = 대입연산자함수 구현
  (4) 주석으로 코드설명 (자세히)

2. 구현 조건

- 연산자함수는 총 8개
- 멤버함수로 구현
- 모든 연산자함수의 결과 제시(TEST)

3. 제출

- 클래스 헤더파일 및 소스파일
- 실행결과 캡쳐화면
- 학번_이름.zip 으로 압축하여 업로드

4. 기한

- 2020.04.27.월까지

5. 기타

- 문의 : yohans@sejong.ac.kr

과제 풀이

```
    /* operators   */
    Complex operator+(const Complex&) const;
    Complex operator-(const Complex&) const;
    void operator=(const Complex&);
    Complex operator-();
    Complex& operator++();       // pre
    Complex operator++(int);    // post
    Complex& operator--();      // pre
    Complex operator--(int);   // post
```

```
/* operators   */
Complex Complex::operator+(const Complex &complex) const
{
    // this + complex
    Complex answer(0,0);
    answer.m_nImag = this->m_nImag + complex.m_nImag;
    answer.m_nReal = this->m_nReal + complex.m_nReal;
    return answer;
}

Complex Complex::operator-(const Complex &complex) const
{
    // this - complex
    Complex answer(0,0);
    answer.m_nImag = this->m_nImag - complex.m_nImag;
    answer.m_nReal = this->m_nReal - complex.m_nReal;
    return answer;
}

void Complex::operator=(const Complex &complex)
{
    // this = complex
    this->m_nImag = complex.m_nImag;
    this->m_nReal = complex.m_nReal;
}

Complex Complex::operator-()
{
    // -this
    Complex answer(0, 0);
    answer.m_nImag = -(this->m_nImag);
    answer.m_nReal = -(this->m_nReal);
    return answer;
}

Complex& Complex::operator++()
{
    // ++this
    ++(this->m_nImag);
    ++(this->m_nReal);
    return *this;
}

Complex Complex::operator++(int)
```

```
{
    // this++
    Complex answer(0,0);
    answer = *this;
    ++(this->m_nImag);
    ++(this->m_nReal);
    return answer;
}

Complex& Complex::operator--()
{
    // --this
    --(this->m_nImag);
    --(this->m_nReal);
    return *this;
}


Complex Complex::operator--(int)
{
    // this--
    Complex answer(0,0);
    answer = *this;
    --(this->m_nImag);
    --(this->m_nReal);
    return answer;
}
```

```
C:\Users\user\Desktop\programming_PROJECTS\git-SJU-Subject\SJU-Subject\
ms-vscode.cpptools-0.27.0\debugAdapters\bin\WindowsDebugLauncher.exe --
stderr=Microsoft-MIEngine-Error-juci0nvy.uc1 --pid=Microsoft-MIEngine-P
raev0\mingw32\bin\gdb.exe" --interpreter=mi "
-1-3i
1+3i
3+5i
3+5i
1+3i

-2-5i
2+5i
4+7i
4+7i
2+5i

-1-2i
1+2i
3+8i
2+5i
2+5i
```

<과제2>

알게 된 점

- 포인터로 객체를 선언하는 것은 new 와 함께하고, 정리하는 것은 delete 로 함.
- 객체 소멸자를 호출하면 어느정도 정리해 줌.
- const 와 static 은 함께 사용할 수 없음.
- const 멤버변수를 초기화할 때에는 콜론 초기화를 이용해야 함.
- 기본 파라미터를 설정할 때에는, '선언' 에 설정하고 '정의' 에는 적어넣으면 안 됨.
- static 멤버 변수를 클래스 밖에서 초기화해야, 전역변수 영역에 메모리가 잡힘.

<과제3>

알게 된 점

- 파라미터로 객체를 넣으면, 객체가 복사되어 넘어감.
- 생각보다 Call by Reference 는 유용한 놈임.

유의사항

이름 규칙

참조

- NHN/C++ 코딩 규칙
- 안용학 교수님이 수업에서 언급했던 규칙
- 과제 명세
- Google C++ Style Guide

파일명

- 지금 하고 있는 프로젝트의 컨벤션, 안용학교수님의 컨벤션에 따른다.
- 클래스를 설계하는 경우, 파일 앞에 대문자 C 를 붙인이고, 대쉬 (-) 를 붙이고, 대표 class 이름으로 사용하며, C 를 제외하고는 소문자와 언더바를 사용한다. ex : **C-rect_base**
- 테스트용 (실행 파일) 의 경우, Test- 를 가장 처음에 포함한다. ex : **Test-rect_base**
- 파일 이름에 대쉬(-) 를 두 개 이상 사용하지 않는다.

함수명

- 일반적인 함수는 대문자로 시작하며, 각 새로운 단어마다 대문자를 사용한다. 언더라인은 사용하지 않는다. ex : **MyExcitingFunction()**
- 접근자와 수정자(get, set)는 변수 이름과 일치시킨다. ex : **Set_strMyExcitingMemberVariable()**

- True/False 값을 return 하는 경우, 함수 이름은 is 혹은 has 로 시작한다. ex : **IsHungry()**
- private 함수 이름은 언더바(_) 로 시작한다. ex : **_DontTouchMe()**

타입명

- 타입명은 대문자로 시작하며, 각 새로운 단어마다 대문자를 갖으며 언더라인을 사용하지 않는다. ex : **MyRectangle**

변수 및 상수명

- 변수명은 소문자로 시작하며, 대문자와 소문자를 섞어서 사용한다.
- 클래스 멤버 변수는 'm_' 으로 시작하며, 간단한 자료형 [string : str, integer : n]을 그 뒤에 표기한 후 이름을 붙인다. ex : **m_strMyExcitingLocalVariable**
- static 멤버 변수의 경우 's_' 으로 시작한다. ex : **s_nMyExcitingStaticVariable**
- 이름은 가능한 설명적으로 짓는다. 공간 절약이 중요한 게 아니라, 코드를 즉시 보고 이해할 수 있어야 한다. ex : **numCompletedConnections**
- 모호한 약어나 의미를 알 수 없는 임의의 문자를 사용하지 않는다. ex : **nerr** (?)
- 구조체의 데이터 멤버는 일반적인 변수처럼 이름을 짓는다. 클래스처럼 언더라인으로 끝나지 않는다.
- 전역 변수는 특별한 요구사항이 없으며, 거의 사용을 하지 않는다. 만약 사용한다면, g_로 시작하거나 로컬 변수와 구별되는 표시를 한다.
- 상수는 k로 시작하며 대소문자를 섞어서 사용한다 : ex : **kDaysInAWeek**

기타

- 들여쓰기는 Tab 을 사용한다.
- 간단한 생성자 초기화는 콜론 초기화로 한다.
- 이항 연산자 (=, >, <, 등..) 앞과 뒤에 공백을 제공한다. ex : **a = b + c**
- 단항 연산자 앞과 뒤에 공백을 제공하나, (A++), [--BB], {--KK}와 같이 사용할 때는 공백이 없어도 좋다.
- 일부 연산자(“ , “, “ ; “)는 연산자 뒤에 공백을 제공해야 한다. ex : **for(i = 0; i < 3; i++)**
- brace({)는 분리된 라인에 작성한다.

```
class People
{
    // 내용
}

void main()
{
    // 내용
}

struct DataStructure
{

}
```

참고한 내용