과제 명세

프로그램 소스코드 파일(.h, .cpp)과 문서 파일(프로그램 소스코드 설명 및 실행화면 캡쳐 설명)을 학번_이름.zip으로 압축하여 제출

1. 다음과 같은 Term 클래스를 설계하고 구현하시오. (15점)

   (1) 멤버변수로 계수(coefficient)와 지수(exponent)를 가지고, 다항식의 한 항을 표현한다.

   (2) 멤버함수로 생성자, Get/Set 함수를 가진다.

   (3) 연산자함수로 단항연산자 ++, --, -를 가진다. ++, --는 전위연산자와 후위연산자를 모두 포함해야 하 고, 계수 값을 1씩 증가하거나 감소한다.

   (4) 그 외 사항은 적절히 처리한다.

예시

Term t1, t2(2, 4);	↦ t2 : $2x^4$
t1.SetCoefficient(5);	
t1.SetExponent(3);	↦ $5x^3$
++t1	↦ $6x^3$
t2--	↦ $1x^4$
t1 = -t1;	↦ $-6x^3$

2. 다음과 같은 Polynomial 클래스를 설계하고 구현하시오. (30점)

   (1) 멤버변수로 Term 클래스의 객체 배열(최대 크기는 100)과 항의 개수를 가진다.

   (2) 멤버함수로 생성자, Get/Set 함수, 그리고 다항식을 출력할 수 이는 멤버 함수를 가진다.

   (3) 연산자함수로 +, -를 가진다. +, -는 연산은 일반적인 다항식 연산을 수행한다. (발생 가능한 경우를 고려해야 함)

예시

Polynomial A, B, C, D;	
A.SetTerm(2,4);	
A.SetTerm(3,2);	
A.SetTerm(4,0);	↦ $2x^4+3x^2+4$
B.SetTerm(5,3);	
B.SetTerm(-2,2);	↦ $5x^3-2x^2$
C = A + B;	↦ $2x^4+5x^3+x^2+4$
D = A - B;	↦ $2x^4-5x^3+5x^2+4$

3. Polynomial 클래스를 테스트하는 main 함수를 구현하시오. (20점)

   (1) Polynomial 클래스의 객체 여러개를 사용하여 각 멤버함수 및 연산자 함수를 충분히 적용한다.

   (2) 각 결과를 모두 출력해야 한다.

4. 문서 설명 내용 (35점)

과제 풀이

term.h

Term class

- 다양한 operator 을 정의해 보았다.

```
class Term
{
    private:

    int m_nCoef;
    int m_nExp;

    // methods
    void _Debug() const;


    public:

    // constructor
    Term(int coef = 0, int exp = 0):m_nCoef(coef), m_nExp(exp) {};

    // getter
    int Get_nCoef() const { return m_nCoef; };
    int Get_nExp()  const { return m_nExp; };

    // setter
    void Set_nCoef(int coef)    { m_nCoef = coef; };
    void Set_nExp(int exp)      { m_nExp = exp; };

    // methods
    void Show(bool for_debug = false, bool in_line = false) const;

    // operators
    void operator=(const Term&);

    Term &operator++();
    Term operator++(int);
    Term &operator--();
    Term operator--(int);
```

```
    Term operator-()            const;
    Term operator+(const Term&) const;
    Term operator-(const Term&) const;
};
```

Polynomial Class

- Term class 의 내용을 최대한 활용했다.
- 다양한 Setter function 을 만들어 보았다.
- 배열의 n 번째 원소가 n 차항을 나타내도록 하고, 고정크기를 사용했다.
- 상황에 따라 자유로운 배치를 가지게 하는 것은, 지나치게 클래스가 복잡해질 것이라고 생각하였다.
- 0차부터 n-1 차 항까지만 존재한다고 가정하고 클래스를 설계하였다
- 모든 항을 생성자에서 coef = 0 으로 초기화해줌으로써 다항식의 의미를 살리고자 하였다.

```
class Polynomial
{
    private:

    Term m_arrTerm[100];
    bool m_arrTermValidity[100];
    int m_nTermCnt;


    public:

    // constructor
    Polynomial();

    // getter
    Term* Get_tTerm() { return m_arrTerm; }
    bool* Get_arrTermValidigy() { return m_arrTermValidity; }
    int Get_nTermCnt() const { return m_nTermCnt; };

    // setter
    void Set_arrTerm(int coef, int exp);
    void Set_arrTermValidity(int index, bool setting);
    void Set_nTermCnt(int number);
    void Set_nTermCnt(bool plus, int number);

    // methods
    void Show();

    // operations
    void operator=(const Polynomial&);

    Polynomial operator-() const;
    Polynomial operator+(const Polynomial&) const;
    Polynomial operator-(const Polynomial&) const;
};
```

C-Term.cpp

Term Class

```
// Term class
void Term::Show(bool for_debug, bool in_line) const
{
    if (for_debug == true)
    {
        std::cout << "variable : " << "x"      << std::endl;
        std::cout << "coef : " << Get_nCoef()  << std::endl;
        std::cout << "exp : "  << Get_nExp()   << std::endl;
    }
    else
    {
        if (m_nCoef > 0)
            std::cout << '+';
        std::cout << m_nCoef;

        if (m_nExp < 0)
            std::cout << "x^" << "(" << m_nExp << ")";
        else if (m_nExp == 0)
            std::cout;
        else
            std::cout << "x^" << m_nExp;

        if (in_line == false) std::cout << std::endl;
    }
}

void Term::operator=(const Term& righthand)
{
    this->m_nCoef = righthand.m_nCoef;
    this->m_nExp = righthand.m_nExp;
}
Term &Term::operator++()
{
    ++(this->m_nCoef);
    return *this;
}
Term Term::operator++(int)
{
    Term temp(0,0);
    temp = *this;
    ++(this->m_nCoef);
    return temp;
}
Term &Term::operator--()
{
    --(this->m_nCoef);
    return *this;
}
Term Term::operator--(int)
{
    Term temp(0,0);
    temp = *this;
    --(this->m_nCoef);
```

```
    return temp;
}
Term Term::operator-() const
{
    Term answer(0,0);
    answer = *this;
    answer.m_nCoef *= (-1);
    return answer;
}
Term Term::operator+(const Term& righthand) const
{
    Term answer(0,0);
    answer = *this;
    answer.m_nCoef = this->m_nCoef + righthand.m_nCoef;
    return answer;
}
Term Term::operator-(const Term& righthand) const
{
    Term answer(0, 0);
    answer = *this;
    answer.m_nCoef = this->m_nCoef - righthand.m_nCoef;
    return answer;
}
```

Polynomial Class

- 항이 유효한지 검사하는 것에 초점을 맞추었다.
- 배열의 n 번째 원소가 n 차 항이라는 것을 생각하여, 해당 차수에 값이 존재하는지 확인하는 절차가 있다.
- 특히, 두 함수식에 대해서 연산을 수행하는 경우, 두 식 중 하나의 식이라도 특정 항을 포함하고 있으면 연산하는 것에 초점을 맞추었다.

```
// Polynomial class
Polynomial::Polynomial()
{
    for(int i = 0; i < 100; i++)
    {
        m_arrTermValidity[i] = 0;
        m_arrTerm[i].Set_nExp(i);
        m_arrTerm[i].Set_nCoef(0);
    }
    m_nTermCnt = 0;
}

void Polynomial::Set_arrTerm(int coef, int exp)
{
    if ((exp > 100) || (exp < 0))
    {
        std::cout << "invalid exp value" << std::endl;
        std::cout << "should be 0 <= coef <= 100" << std::endl;
    }
    else
    {
        Term term(coef, exp);
        m_arrTerm[exp] = term;
```

```
        m_arrTermValidity[exp] = true;
    }

    m_nTermCnt++;
}
void Polynomial::Set_arrTermValidity(int index, bool setting)
{
    m_arrTermValidity[index] = setting;
}
void Polynomial::Set_nTermCnt(int number)
{
    m_nTermCnt = number;
}
void Polynomial::Set_nTermCnt(bool plus, int number)
{
    if (plus)
    {
        m_nTermCnt += number;
    }
    else
    {
        m_nTermCnt -= number;
    }
}


void Polynomial::Show()
{
    for (int i = 100-1; i >= 0; i--)
    {
        if (m_arrTermValidity[i] && (m_arrTerm[i].Get_nCoef() != 0))
        {
            m_arrTerm[i].Show(false, true);
        }
    }
    std::cout << std::endl;
}

void Polynomial::operator=(const Polynomial& righthand)
{
    // copy m_arrTerm
    for (int i = 0; i <= 100; i++)
        this->m_arrTerm[i] = righthand.m_arrTerm[i];

    // copy m_arrTermValidity
    for (int i = 0; i <= 100; i++)
        this->m_arrTermValidity[i] = righthand.m_arrTermValidity[i];

    // copy m_nTermCnt
    this->m_nTermCnt = this->m_nTermCnt;
}
Polynomial Polynomial::operator-() const
{
    Polynomial answer;
    answer = *this;
    for (int i = 0; i < 100; i++)
    {
        if (m_arrTermValidity[i])
```

```
        {
            answer.m_arrTerm[i] = -(m_arrTerm[i]);
        }
    }
    return answer;
}
Polynomial Polynomial::operator+(const Polynomial& righthand) const
{
    Polynomial answer;
    answer = *this;
    for (int i = 0; i < 100; i++)
    {
        if (this->m_arrTermValidity[i] || righthand.m_arrTermValidity[i])
        {
            answer.m_arrTerm[i] = this->m_arrTerm[i] + righthand.m_arrTerm[i];
            answer.m_arrTermValidity[i] = true;
        }
    }
    return answer;
}
Polynomial Polynomial::operator-(const Polynomial& righthand) const
{
    Polynomial answer;
    answer = *this;
    for (int i = 0; i < 100; i++)
    {
        if (this->m_arrTermValidity[i] || righthand.m_arrTermValidity[i])
        {
            answer.m_arrTerm[i] = this->m_arrTerm[i] - righthand.m_arrTerm[i];
            answer.m_arrTermValidity[i] = true;
        }
    }
    return answer;
}
```

Test-term.cpp

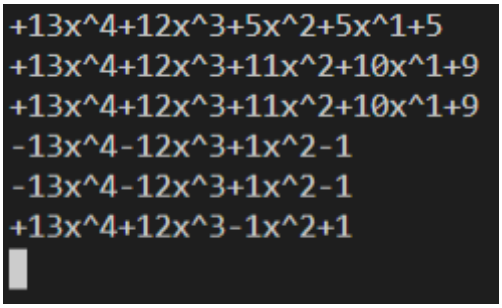
Source code1

- 멤버함수 Set 이 잘 작동하는지 확인하는 과정이 있다.
- Polynomial Class 의 각 Operator (특히, = + -) 가 정상적으로 작동하는지 확인하는 과정이 하단부에 있다.

```
int main ()
{
    Polynomial fx, gx, kx;
    fx.Set_arrTerm(6, 2);
    fx.Set_arrTerm(5, 1);
    fx.Set_arrTerm(4, 0);
    fx.Show();

    gx.Set_arrTerm(13, 4);
    gx.Set_arrTerm(12, 3);
    gx.Set_arrTerm(5, 2);
    gx.Set_arrTerm(5, 1);
```

```
    gx.Set_arrTerm(5, 0);
    gx.Show();

    (fx + gx).Show();
    kx = (fx + gx);
    kx.Show();
    (fx - gx).Show();
    kx = (fx - gx);
    kx.Show();
    (-kx).Show();
}
```

Capture1

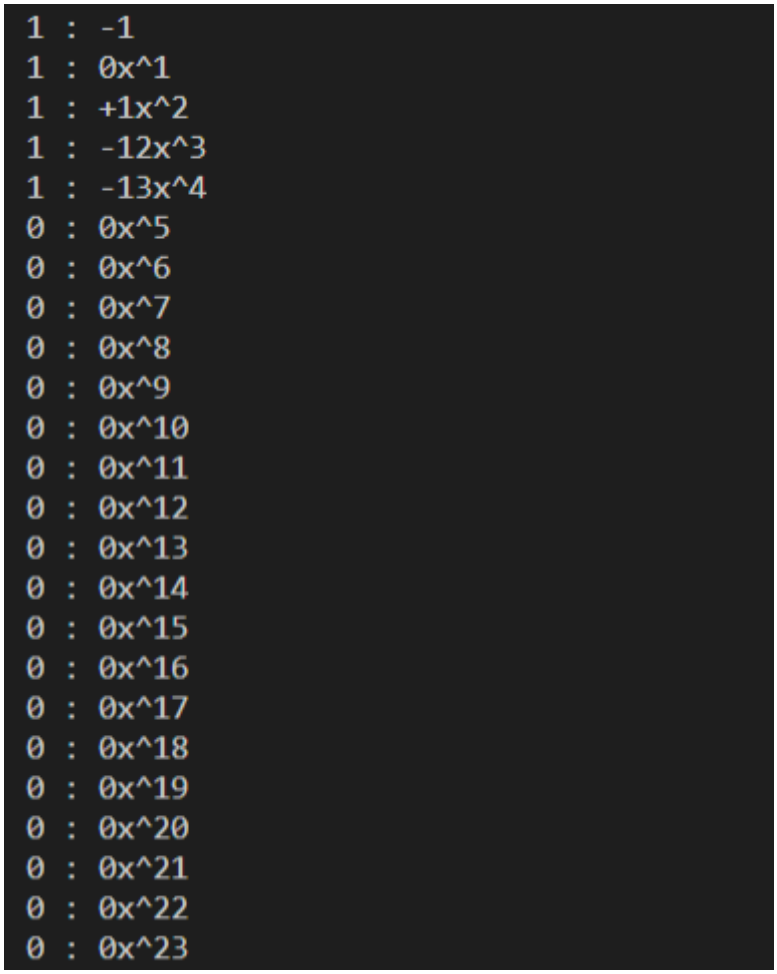


기대 출력과 동일한 출력을 보였다.

Source Code2

```
{
    /* member function test */
    // getter
    Term *term_tester;
    bool *bool_tester;

    term_tester = kx.Get_arrTerm();
    bool_tester = kx.Get_arrTermValidigy();
    for (int i = 0; i < 100; i++)
    {
        std::cout << bool_tester[i] << " : ";
        term_tester[i].Show();
    }
}
```

Capture 2

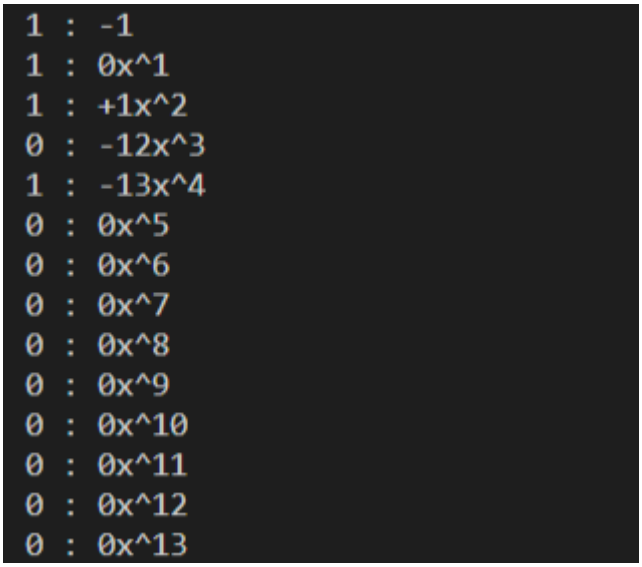


값이 들어 있는 초반부에는 1 로 true 를 보이고 있고, 모든 차수에 대해서 해당 항의 정보를 출력하고 있다.

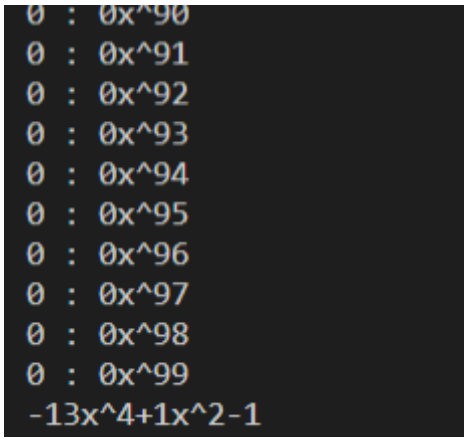
Source Code3

```
{
    // setter
    kx.Set_arrTermValidity(3, 0);
    kx.Set_nTermCnt(false, -1);
    kx.Show();
}
```

Capture3



세 번째 항이 비활성화되었다.



그리고 하나의 항이 사라졌음을 알 수 있다.

알게 된 점

<과제2>

- 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>

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

<중간고사 대체과제>

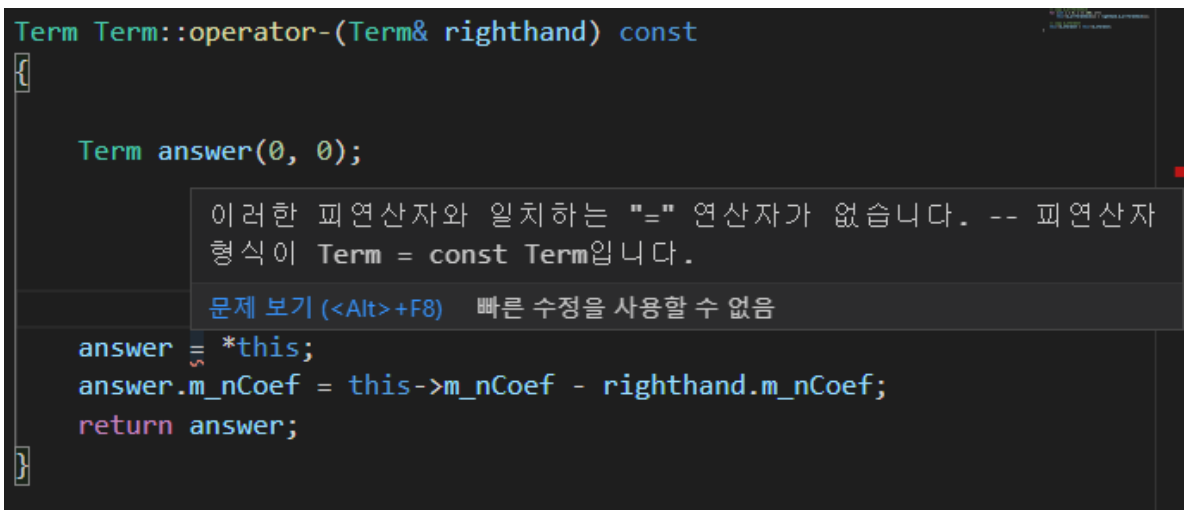
- C++ 에서 NULL 은 좋은 서술이 아님.
- 기본 파라미터는 어디에 선언하는 것인가

default parameter는 함수 선언에 표시해야 합니다. 그래야지 다른 위치에서 함수를 호출할 때 볼 수 있어요.

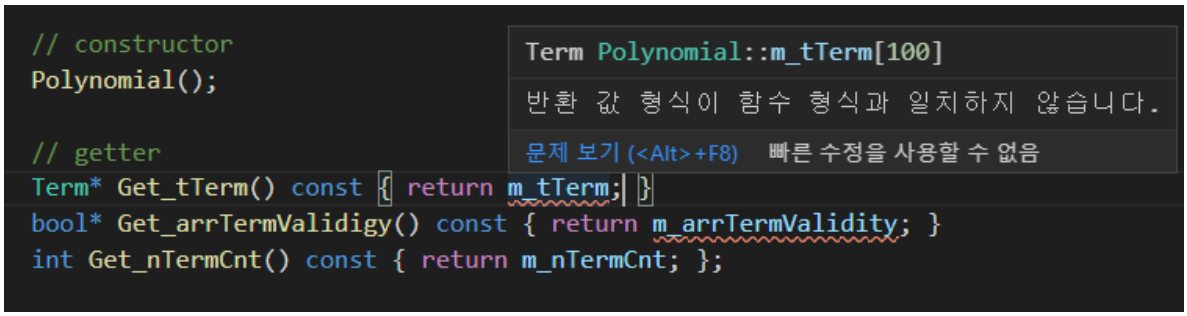
만약에 헤더 파일에 default 파라미터 없이 선언해놓고 `A.cpp`에 함수를 정의하면서 `default` 파라미터를 쓴다면 `B.cpp` 같은 다른 곳에서는 `default` 파라미터가 있는 걸 알 수 없습니다.

```
//myheader.hpp

void myfunc(int nonDefault, float Default);

/******* 파일 나눔 ********/

//A.cpp

int myfunc(int nonDefaultParam, float DefaultParam=3){
    cout << "hello" << endl;
    return 1;
}

void funcA(){
    myfunc(3); //ok
    myfunc(3, 3.0); //ok
}


/******* 파일 나눔 ********/

//B.cpp

void funcA(){
    myfunc(3); //error!!!!!!!!!!
    myfunc(3, 3.0); //ok
}
```

- 이해하기 힘들었던 이것. 분명히 this 의 값을 변경하지 않는데도 오류가 나서, 굉장히 헤맸지만 오류가 갑자기 사라졌다.



- const 에 생긴 여러 문제들. 분명히 return 만 하는 함수인데 cosnt 로 선언할 때에만 오류가 난다.



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

https://hashcode.co.kr/questions/399/default%ED%8C%8C%EB%9D%BC%EB%AF%B8%ED%84%B0%EB%8A%94-%ED%95%A8%EC%88%98-%EC%84%A0%EC%96%B8%EC%A0%95%EC%9D%98-%EC%A4%91-%EC%96%B4%EB%94%94%EC%97%90-%EC%A0%81%EC%96%B4%EC%95%BC-%ED%95%98%EB%82%98%EC%9A%94