# Object Oriented Programming

## Programming Assignment 5

Due: 2017/05/20

# ★Announcement★

1. 제출 기한은 5월 20일 자정까지 입니다. LMS 시스템에서는 제출자가 몰리면 업로드가 안 될 수 있으니, 유의해주시기 바랍니다.
2. 과제는 늦게 제출하면 이유를 불문하고 0점입니다.
3. 채점은 Visual Studio 2015 환경에서 이루어집니다. 파일을 업로드하실 때, 작업하신 환경에 있는 프로젝트 폴더를 그대로 압축해서 올려주십시오. 그리고 폴더 이름은 ProblemN_학번의 형태로 만들어주십시오. 프로젝트 폴더를 압축하지 않고 cpp 파일만 업로드하거나, Visual Studio 2015 환경에서 컴파일 되지 않을 경우 **큰 감점**이 있습니다.
   예시) `Problem1_20162939`, `Problem2-1_20162939`, `Problem2-2_20162923`
4. C++에서 배운 내용을 토대로 작성하셔야 합니다. C++로 구현 가능한 내용을 C로 구현했을 경우에는 감점으로 처리합니다.
   예시) `scanf`, `printf`, `fprintf`, `fscanf`, `malloc`, `free`
5. 문서를 꼼꼼하게 읽고 문서에 나온 `method`들은 `input argument`와 `output result`를 문서에 맡게 작성해주세요.

# 1. Set Operation (80 pts)

이전 과제의 Set class를 확장하여, 다음의 요구조건을 충족시키는 집합을 구현하여라.

클래스 Set은 다음과 같은 구현을 충족시키도록 한다.

```
template <class T>
class Set {
private:
	T* elements;
	int size;
public:
	Set();
	Set(int size, T* elements);
	~Set();
	Set operator+(const Set& set);
	Set operator-(const Set& set);
	Set operator*(const Set& set);
	Set& operator=(const Set& set);
	bool operator==(const Set& set);
	friend ostream& operator<<(ostream&, const Set&);
	Set powerSet();
	bool isSubsetOf(const Set& set);
	Set intersection(const Set& set);
};
```

- ✓ **`Set operator+(const Set& set);`**
  이 집합과 set의 합집합을 구하여 새로운 Set을 반환한다.
- ✓ **`Set operator-(const Set& set);`**
  이 집합과 set의 차집합을 구하여 새로운 Set을 반환한다.
- ✓ **`Set operator*(const Set& set);`**
  이 집합과 set의 Cartesian product를 구하여 새로운 Set을 반환한다.
  * Cartesian product는 다음 링크의 Cartesian product 항목을 참고한다.
  * https://en.wikipedia.org/wiki/Set_(mathematics)
- ✓ **`Set& operator=(const Set& set);`**
  set을 이 집합에 복사한다.
- ✓ **`bool operator==(const Set& set);`**
  두 집합이 같은지 비교한다.
- ✓ **`friend ostream& operator<<(ostream&, const Set&);`**
  "{e1, e2, e3}"의 꼴로 집합을 출력한다.
  * Pair Set을 출력할 경우 "{(e11, e12), (e21, e22), (e31, e32)}"의 꼴로 집합을 출력한다.
- ✓ **`Set power();`**
  이 집합의 모든 부분집합을 구한다.
- ✓ **`bool isSubsetOf(const Set& set);`**
  이 집합이 set의 부분집합인지 검사한다.
- ✓ **`Set intersection(const Set& set);`**
  이 집합과 set의 교집합을 구하여 새로운 Set을 반환한다.

Cartesian product를 구현하기 위한 클래스 Pair를 구현한다. Set 클래스의 operator overloading을 고려하여 Pair class를 구현하도록 한다.

```
template <class T1, class T2>
class Pair {
private:
	T1 e1;
	T2 e2;
public:
	Pair(T1 e1, T2 e2);
	//…
};
```

이 외에, 별도의 method나 member variable이 필요하다면, 추가하여도 무방하다. 이 과제는 STL을 이용할 수 없다.

이 과제는 main함수를 구현할 필요가 없다.

## 채점 기준

1. Accuracy - 50 pts
2. Program Structure - 10 pts
3. Code Readability - 10 pts
4. Comment - 10 pts

# 2. Calculator with STL (120 pts)

**중요:** Problem 2-1과 Problem 2-2로 프로젝트를 따로 작성하여 제출하여 주시기 바랍니다. Announcement 3번을 참조해주시기 바랍니다.

## 1) (80점) 사칙연산 계산기

우리가 평소에 사용하는 일반적인 사칙연산 식은 연산자가 숫자 가운데 있는 중위표현법 (Infix Notation)을 활용한다. 그러나, 컴퓨터는 사칙연산을 순차적으로 처리하기 때문에 중위표현법을 이해하는데 어려움이 있다. 그래서 연산자를 숫자의 뒤로 배치하는 후위표현법 (Postfix Notation)을 활용한다. 예를 들어 보자.

Infix Notation: 2 * 3 - 6 + 8 / 4

Postfix Notation: 2 3 * 6 - 8 4 / +

Postfix Notation에서 컴퓨터는 2와 3을 본 후에 *을 보았을 때 2와 3이 operand라는 것을 파악할 수 있다. 그 후 6을 다시 넣어주면, 6과 6을 본 후에 -를 보게 되어 6과 6이 operand라는 것을 파악할 수 있다. 다시 0이 들어가는데, 0 8 4 / 를 보았을 때, /는 8과 4가 operand라는 것을 파악하고 2를 다시 넣어준다. 0과 2 +를 보고 마지막으로 2를 계산해낸다. 여기서 중요한 것은 연산자 우선순위를 파악해야 한다는 것이다.

본 과제에서는 Standard Template Library를 이용해 Infix Notation을 Postfix Notation으로 바꾸고, 이를 연산하는 것을 목표로 한다.

Postfix Notation은 vector를 이용해 저장한다. 이를 계산할 때는 stack을 활용한다. 식은 다음과 같은 조건을 가진다.

1. 수는 10 미만의 한 자리 수만 나타난다.
2. 덧셈, 뺄셈, 곱셈, 나눗셈만을 활용한다.
3. 곱셈과 나눗셈을 먼저 연산하고, 덧셈과 뺄셈을 나중에 연산하며, 우선순위가 같은 연산자끼리는 왼쪽부터 연산한다.
4. 각 연산자 및 숫자의 사이에는 빈칸이 전혀 없이 붙어서 입력된다.

### Input & Output

Input, Output 은 File I/O 로 진행되며 input.txt 를 통해 입력 받고, output.txt 를 통해 출력한다. 입출력 조건은 다음 표와 같다.

| Input (input.txt) | Output (output.txt) |
|---|---|
| input.txt을 통해 입력을 받는다. input.txt의 첫 줄에는 Infix Notation의 개수가 입력된다. 그 이후의 각 줄에는 Infix Notation이 입력된다. | 각 줄에 Postfix Notation이 출력된 후, 연산결과가 다음 줄에 출력된다. 편의를 위해 각 글자의 사이에 space를 하나씩 출력한다. |
| 2<br>2*3-6+8/4<br>2*3+6/2-4 | 2 3 * 6 - 8 4 / +<br>2<br>2 3 * 6 2 / + 4 -<br>5 |

## 2. (40점) 괄호가 포함된 사칙연산 계산기

괄호가 포함된 Infix Notation을 Postfix Notation으로 변경하고, 그 결과를 계산한다. 먼저 괄호의 개수가 legal한지를 확인한다. 그 이후에는 원래 하던 것대로 Postfix Notation을 만든다. 예를 들어 보자.

Infix Notation: 4 / 2 - ( 2 + ( 3 * 6 ) )

Postfix Notation: 4 2 / 2 3 6 * + -

그 이후에는 앞에서 설명한대로 연산한다. 조건 또한 원래와 같다. 왼쪽 괄호와 오른쪽 괄호의 개수가 같지 않은 경우에는 ParanIllegalError를 출력한다.

### Input & Output

Input, Output 은 File I/O 로 진행되며 input.txt 를 통해 입력 받고, output.txt 를 통해 출력한다. 입출력 조건은 다음 표와 같다.

| Input (input.txt) | Output (output.txt) |
|---|---|
| input.txt을 통해 입력을 받는다. input.txt의 첫 줄에는 Infix Notation의 개수가 입력된다. 그 이후의 각 줄에는 Infix Notation이 입력된다. | 각 줄에 Postfix Notation이 출력된 후, 연산결과가 다음 줄에 출력된다. 편의를 위해 각 글자의 사이에 space를 하나씩 출력한다. 왼쪽 괄호와 오른쪽 괄호의 개수가 같지 않은 경우에는 ParanIllegalError만을 출력한다. |
| 2<br>4/2-(2+(3*6))<br>2/(1+1*3-4 | 4 2 / 2 3 6 * + -<br>-18<br>ParanIllegalError |

## 평가기준

### 2-1. 사칙연산 계산기

1. Accuracy - 40 pts
2. Implementation - 30 pts
    A. Infix to Postfix expression - 15 pts
    B. Evaluate using Stack - 15 pts (STL을 활용하지 않을 경우 감점)
3. Comment - 10 pts

### 2-2. 괄호가 포함된 사칙연산 계산기

1. Accuracy - 20 pts
2. Implementation - 20 pts
    A. Infix to Postfix expression - 15 pts
    B. Evaluate using Stack - 5 pts (STL을 활용하지 않을 경우 감점)