

## Homework #2

2018. 4. 16 (월) ~ 4. 30 (월) 23:59

\* 마지막 장의 주의사항을 참고할 것.

P1 (25 점). Linked List-based Stack Class 를 구현하고자 한다. Stack 의 각 데이터는 Linked List 형태로 저장되며, 주어진 HW2\_Stack.h 를 참고하여 다음 조건을 만족하도록 HW2\_Stack.h 를 작성하시오.

제출 파일 : HW2\_Stack.h

(※ Stack Class 내부는 수정 불가하며, main() 함수는 작성하지 마시오)

(※ T 의 자료형으로는 c++의 기본 자료형(int, double, char 등)을 가정)

(※ Template 함수의 경우 헤더와 cpp 파일을 분리하여 작성하면 오류가 발생할 확률이 높으므로 cpp 파일을 따로 만들지 않고 헤더 파일 안에서 모두 구현할 것!)

### Stack Class

- **Definition**
  - Stack : LIFO (Last-In-First-Out) 구조를 갖는 자료구조
- **Node Class** (**Private**로 정의됨)
  - 실습3에서의 Node Class 형태와 동일
- **Member Variable** (**Private**로 정의됨)
  - **Node\* top** : Stack의 top을 가리키는 포인터
  - **int size** : 현재 Stack에 들어있는 데이터의 개수
- **Constructor**
  - **Stack()** : Stack의 Constructor; Top을 null pointer로, size를 0으로 초기화
- **Destructor**
  - **~Stack()** : Stack의 destructor; Stack 내에 존재하는 모든 node 삭제
- **Member Function**
  - **bool isEmpty() const** : Stack에 저장된 데이터가 없으면 true/있으면 false 반환
  - **T getTop() const** : Stack의 top 위치에 저장되어 있는 데이터 반환  
Stack이 비어있는 경우 T() 반환
  - **int getSize() const** : Stack에 들어있는 데이터의 개수 반환
  - **void push(const T& x)**
    - ◆ Stack에 데이터로 x를 가지는 새로운 node 삽입

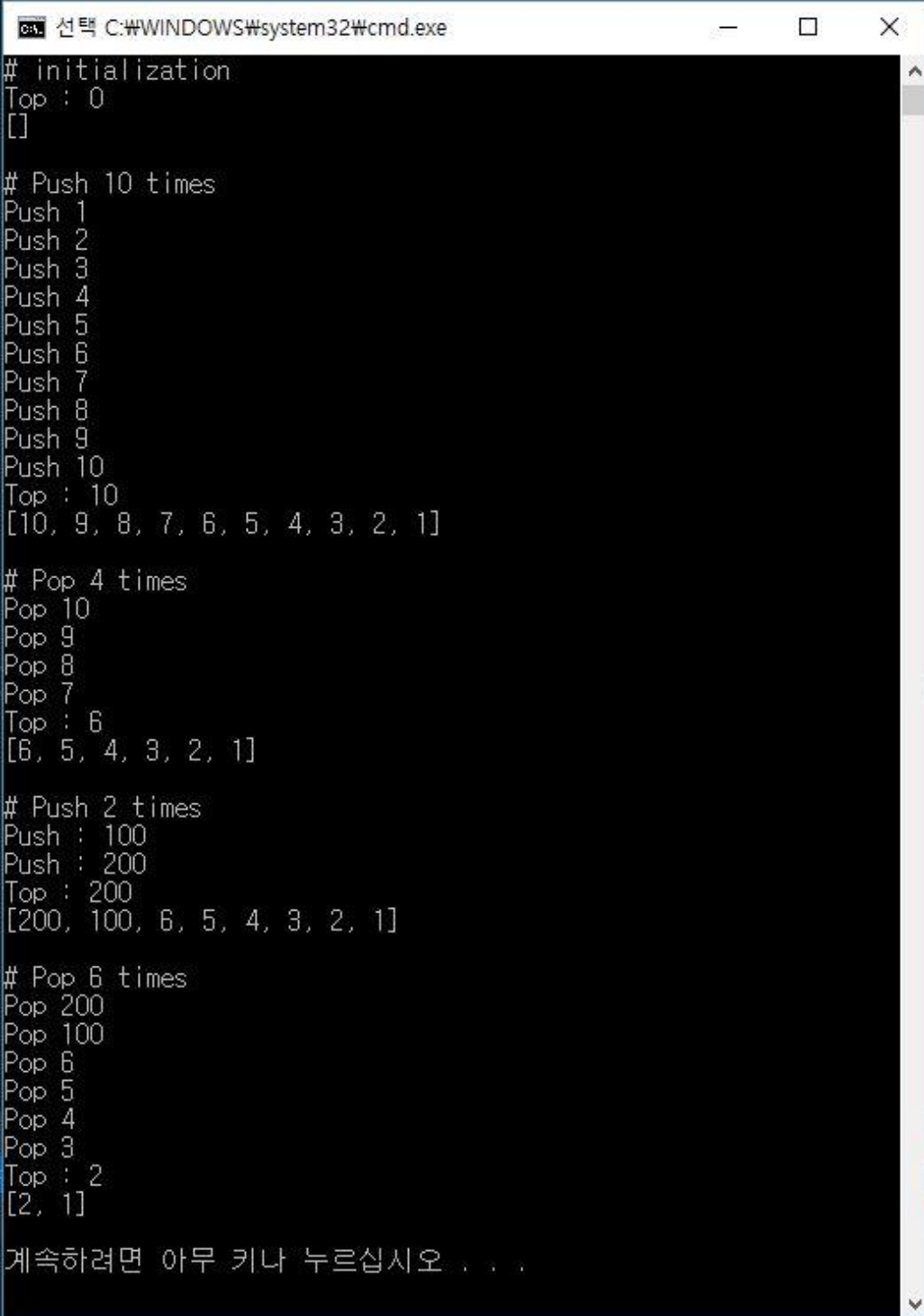
■ **T pop()**

- ◆ Stack이 비어있지 않은 경우, Top이 가리키는 위치의 data값을 반환하고 Top이 가리키는 Node 삭제 후 Top이 그 다음 Node를 가리키도록 저장
- ◆ Stack이 비어있는 경우 T() 반환

■ **void print() const**

- ◆ Stack에 저장된 모든 데이터를 다음과 같이 출력  
예) Stack이 비어있는 경우 [] 출력  
Stack에 Top에서부터 1, 2, 3순으로 들어있는 경우 [1, 2, 3] 출력

주어진 HW2\_Main\_Stack.cpp를 실행하여 간단하게 Stack Class가 잘 구현되었는지 테스트할 수 있다. (채점 시에는 더욱 복잡한 Case를 가지고 테스트할 예정)



```
선택 C:\WINDOWS\system32\cmd.exe
# initialization
Top : 0
[]

# Push 10 times
Push 1
Push 2
Push 3
Push 4
Push 5
Push 6
Push 7
Push 8
Push 9
Push 10
Top : 10
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

# Pop 4 times
Pop 10
Pop 9
Pop 8
Pop 7
Top : 6
[6, 5, 4, 3, 2, 1]

# Push 2 times
Push : 100
Push : 200
Top : 200
[200, 100, 6, 5, 4, 3, 2, 1]

# Pop 6 times
Pop 200
Pop 100
Pop 6
Pop 5
Pop 4
Pop 3
Top : 2
[2, 1]

계속하려면 아무 키나 누르십시오 . . .
```

P2 (35점). Array-based Queue Class를 Circular Queue 형태로 구현하고자 한다. Queue의 각 데이터는 Array에 저장되며, Queue의 처음과 끝은 front와 rear를 이용하여 가리킨다. 주어진 HW2\_Queue.h의 Queue Class를 참고하여 다음 조건을 만족하도록 HW2\_Queue.h를 작성하시오.

제출 파일 : HW2\_Queue.h

(※ Queue Class 내부는 수정 불가하며, main() 함수는 작성하지 마시오)

(※ T의 자료형으로는 c++의 기본 자료형(int, double, char 등)을 가정)

(※ Template 함수의 경우 헤더와 cpp 파일을 분리하여 작성하면 오류가 발생할 확률이 높으므로 cpp파일을 따로 만들지 않고 헤더 파일 안에서 모두 구현할 것!)

(※ shift() 함수는 Queue의 기본 기능은 아니며, 과제를 위해 임의로 추가되었음)

#### Queue Class

- **Definition**
  - Queue : FIFO (First-In-First-Out) 구조를 갖는 자료구조
- **Member Variable** (Private로 정의)
  - **int front** : Queue의 front 위치,  
Queue의 데이터는 front가 가리키는 위치의 다음부터 시작
  - **int rear** : Queue의 rear 위치, Queue가 비어있을 경우 front 위치와 동일  
Queue의 데이터는 rear가 가리키는 위치까지 존재
  - **int size** : 현재 Queue에 들어있는 data의 개수
  - **T\* array** : Queue의 데이터를 저장하는 array
  - **int capacity** : Queue의 데이터를 저장하는 array의 크기
- **Constructor**
  - **Queue(int c)** : Queue의 constructor; c(>0) 크기의 array를 생성하고,  
front, rear, size를 각각 0으로, capacity를 c로 초기화
- **Destructor**
  - **~Queue()** : Queue의 destructor; Queue 데이터를 저장하는 array 삭제
- **Member Function**
  - **bool isEmpty() const** : Queue에 저장된 데이터가 없으면 true/있으면 false 반환
  - **T getFront() const** : Queue의 맨 처음 데이터 반환; 비어있는 경우 T() 반환
  - **T getRear() const** : Queue의 맨 마지막 데이터 반환; 비어있는 경우 T() 반환
  - **int getSize() const** : Queue에 들어있는 데이터의 개수 반환
  - **int getCapacity() const** : Queue 데이터를 저장하는 array의 크기 반환

■ **void push(const T& x)**

- ◆ Queue에 빈 공간이 있는 경우에는 Queue에 x를 추가하고 rear 위치 조정
- ◆ Queue가 꽉 찬 경우 Queue의 array 크기를 2배로 늘리고 push 과정 수행

■ **T pop()**

- ◆ Queue가 비어있지 않은 경우, 맨 처음 데이터를 반환하고 front 위치 조정
- ◆ Queue가 비어있을 경우 T() 반환

■ **void print() const**

- ◆ Queue에 저장된 모든 데이터를 다음과 같이 출력

예) Queue가 비어있는 경우 [] 출력

Queue에 1, 2, 3이 들어있는 경우 [1, 2, 3] 출력

■ **void shift(int amount)**

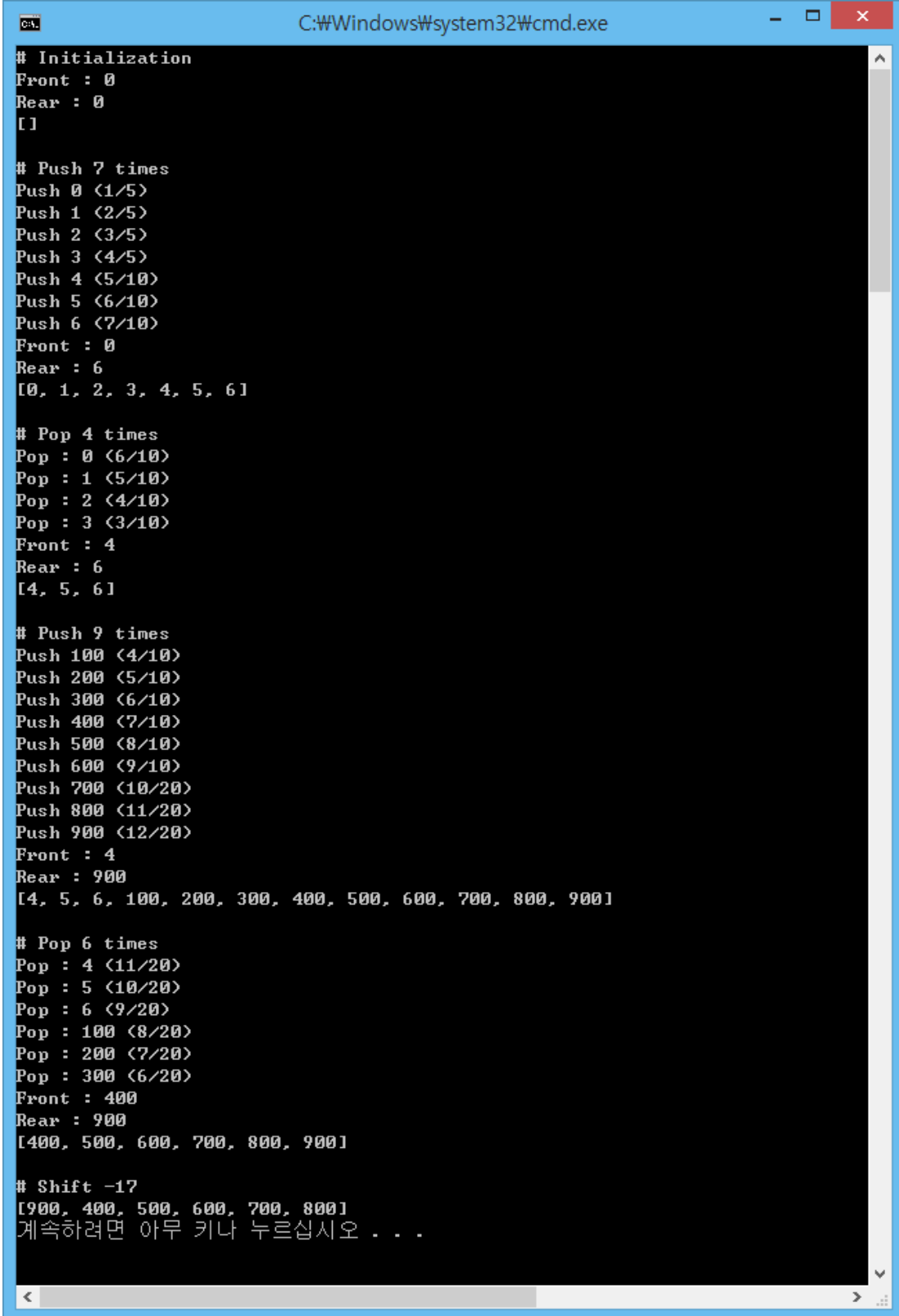
- ◆ amount가 음수인 경우 왼쪽 방향으로 amount의 절대치만큼 Shift
- ◆ amount가 양수인 경우 오른쪽 방향으로 amount의 절대치만큼 Shift

예) Queue a가 [1, 2, 3, 4, 5, 6, 7]일 때

amount = 2로 Shift했을 때 a.print() 결과는 [6, 7, 1, 2, 3, 4, 5]

amount = -3으로 Shift했을 때 a.print() 결과는 [4, 5, 6, 7, 1, 2, 3]

주어진 HW2\_Main\_Queue.cpp 를 실행하여 간단하게 Queue Class 가 잘 구현되었는지 테스트할 수 있다. (채점 시에는 더욱 복잡한 Case 를 가지고 테스트할 예정)



```
# Initialization
Front : 0
Rear : 0
[]

# Push 7 times
Push 0 <1/5>
Push 1 <2/5>
Push 2 <3/5>
Push 3 <4/5>
Push 4 <5/10>
Push 5 <6/10>
Push 6 <7/10>
Front : 0
Rear : 6
[0, 1, 2, 3, 4, 5, 6]

# Pop 4 times
Pop : 0 <6/10>
Pop : 1 <5/10>
Pop : 2 <4/10>
Pop : 3 <3/10>
Front : 4
Rear : 6
[4, 5, 6]

# Push 9 times
Push 100 <4/10>
Push 200 <5/10>
Push 300 <6/10>
Push 400 <7/10>
Push 500 <8/10>
Push 600 <9/10>
Push 700 <10/20>
Push 800 <11/20>
Push 900 <12/20>
Front : 4
Rear : 900
[4, 5, 6, 100, 200, 300, 400, 500, 600, 700, 800, 900]

# Pop 6 times
Pop : 4 <11/20>
Pop : 5 <10/20>
Pop : 6 <9/20>
Pop : 100 <8/20>
Pop : 200 <7/20>
Pop : 300 <6/20>
Front : 400
Rear : 900
[400, 500, 600, 700, 800, 900]

# Shift -17
[900, 400, 500, 600, 700, 800]
계속하려면 아무 키나 누르십시오 . . .
```

P3 (40점). 연산 표기법에는 다음과 같은 세 가지 방법이 있다.

- 전위 표기법 (Prefix Notation) : 연산자를 피연산자 앞에 표기
- 중위 표기법 (Infix Notation) : 연산자를 피연산자 사이에 표기
- 후위 표기법 (Postfix Notation) : 연산자를 피연산자 뒤에 표기

예를 들어, 중위 표기법으로  $(5 - 6) * 7$ 로 표기된 식은 전위 표기법으로 표기하였을 때  $*(-56)7$  또는  $*-567$ 로 표기할 수 있고, 후위 표기법으로 표기하였을 때  $56-7*$ 로 표기할 수 있다. 그러나 중위 표기법은 연산자의 우선순위에 따라 계산순서가 바뀌는 단점이 있기 때문에 컴퓨터는 중위 표기법을 바로 해석하기 어렵다. 따라서 연산자의 우선순위에 상관없이 순서대로 계산이 가능한 전위 표기법 또는 후위 표기법으로 수식을 변형하여 계산한다. 본 과제에서는 앞에서 구현한 Stack, Queue Class를 이용하여 (1) 중위 표기법으로 입력된 수식을 후위 표기법으로 변환하고 (2) 그 계산 결과를 도출하는 계산기를 만들고자 한다.

※ 본 과제에서 피연산자는 0 이상 10 미만의 한 자리 정수만 사용됨을 가정한다.

#### (1) 중위 표기법 → 후위 표기법으로 변환

중위 표기법을 후위 표기법으로 변환하기 위해서는 Stack 이 필요하다. 예를 들어, 중위 표기법으로  $1+2$ 로 표기된 식이 입력으로 들어왔을 때에, 피연산자인 1은 바로 출력하면 되지만, 연산자 +는 피연산자 2 뒤에 출력해야 한다. 이 연산자는 Stack에 보관하고, 피연산자 2를 먼저 출력하게 된다. 또한, 입력값이  $5-2*3$ 인 경우, 5와 2를 출력하고 -는 Stack에 보관한 상태에서, \*는 -보다 연산자 우선순위가 높기 때문에 \*가 나중에 출력되어야 한다. 따라서 나중에 들어온 연산자가 먼저 출력되기 위해서는 후입선출(LIFO) 구조인 Stack이 연산자 보관을 위한 자료구조로 적합하다.

그러나, 항상 나중에 들어온 연산자가 먼저 출력되는 것은 아니다.  $5*2-3$ 의 경우 위와 같은 방법을 사용하면  $2-3$ 이 먼저 계산되기 때문에, 연산자에 우선순위에 따라 달리해야 한다. 연산자 우선순위는 여는 괄호 (, 닫는 괄호 )가 가장 낮고, 덧셈과 뺄셈, 그리고 곱셈과 나눗셈 순으로 높아진다. 특히 괄호는 무조건 먼저 계산하기 때문에, 여는 괄호 (가 나올 때에는 무조건 Stack에 추가하고, 닫는 괄호 )가 나오면 Stack에서 여는 괄호 (가 나올 때까지 Pop을 수행해야 한다.

전위 표기법을 후위 표기법으로 변환하는 방법은 아래의 pseudo code 에 나와있다.

```

exp_infix ← 중위 표기법으로 표현된 수식 (수식에 공백은 입력하지 않음)
exp_postfix ← 빈 문자열
stack ← 빈 Stack

for (수식 exp_infix의 문자 ch) { // exp_infix[0]부터 exp_infix[size-1]까지 반복
    if (ch가 피연산자)
        exp_postfix에 ch 추가
    else if (stack이 비어있음)
        stack에 연산자 ch push
    else if (stack의 최상위 Node가 ch보다 연산자 우선순위가 낮음)
        stack에 연산자 ch push
    else if (stack의 최상위 Node가 ch보다 연산자 우선순위가 높음)
        stack의 최상위 Node가 ch보다 우선순위가 낮을 때까지
        stack에서 pop하고, 그 결과를 exp_postfix에 추가
        stack에 연산자 ch push
}

stack에 남은 연산자를 모두 pop하고, 그 결과를 exp_postfix에 추가

return exp_postfix

```

위의 pseudo code 는 괄호에 대해서는 배제되어 있으므로, 연산자에 괄호가 추가될 때를 고려하여 구현해야 한다.

## (2) 후위 표기법을 이용하여 계산 수행

중위 표기법을 후위 표기법으로 변환하면서, 연산자가 연산 순서대로 배열되었다. 이제 Stack 을 이용하여 모든 연산을 수행할 때까지 다음의 계산을 반복하면 된다.

- 1) 피연산자를 만나면 Stack 에 push
- 2) 연산자를 만나면 Stack 에서 두 피연산자를 pop 하여 연산 수행 후 그 결과 Stack 에 push



예시) 중위 표기법으로 표현된  $(2+5)*3*(2+1)$ 은 후위 표기법으로 변환하면  $25+3*21+*$ 로 표현된다. 이를 계산하는 과정은 다음과 같다.

- 1) push(2), push(5)
- 2) 2 와 5 를 pop()하고, push(2+5)
- 3) push(3)
- 4) 3 과 7 을 pop()하고, push(3\*7)
- 5) push(2), push(1)
- 6) 2 와 1 을 pop()하고, push(2+1)
- 7) 21 과 3 을 pop()하고, push(21\*3)
- 8) 최종 결과인 63 을 pop()

이제 위에서 설명한 기능을 수행하는 Calculator Class 를 주어진 HW2\_Calculator.h 에 맞게 아래와 같이 구현하시오. (Stack class 는 HW2\_Stack.h 에 정의된 Class 를 이용하며, 추가로 멤버 변수 또는 함수를 추가할 수 없다.)

제출 파일 : HW2\_Calculator.cpp

#### Calculator Class

- **Member Variable** (Private로 정의)
  - **string exp\_infix** : 중위 표기법으로 표현된 수식
- **Constructor**
  - **Calculator(string str)** : Calculator의 Constructor; exp\_infix를 str로 초기화
- **Member Function**
  - **void setInfixExp(string str)** : exp\_infix를 str로 설정
  - **int getWeight(char op) const**
    - ◆ Operator op(괄호, +, -, \*, /)의 우선순위 반환  
우선순위는 곱셈/나눗셈 연산자 (\*, /) > 덧셈/뺄셈 연산자 (+, -) > 괄호 순
  - **string getPostfixExp() const**
    - ◆ Stack과 getWeight()를 이용하여 exp\_infix를 후위 표기법으로 변환하여 출력
  - **int calcTwoOperands(int operand1, int operand2, char op) const**
    - ◆ operand1 op operand2의 연산 결과를 반환  
예) operand1 = 2, operand2 = 3, op = '\*'일 때, 2\*3의 결과인 6 반환
  - **int calculate()**
    - ◆ getPostfixExp()를 이용하여 중위 표기법으로 변환된 수식을 후위 표기법으로 변환한 후, Stack과 calcTwoOperand()를 이용하여 계산 결과를 반환

- 참고

- 과제 채점은 Microsoft Visual Studio 2017에서의 동작을 기준으로 함
- 주어진 변수, 함수 이름 절대 고치지 말것! (대소문자 구분) 이를 어길 시 감점
- 주어진 예시와 같은 양식으로 입/출력되게 할 것! 이를 어길 시 감점
- 채점은 문제에 주어진 예시보다 복잡한 Test Case로 진행
- Compile 오류 시 해당 문제 0점 처리할 예정이므로, 충분한 Test 후 제출 요망
- 표절 금지. 표절 적발 시 해당 과제 0점 처리 및 교수님께 통보
- 질문이 있는 경우, 검색을 충분히 해 본 후에도 해결되지 않는 것에 한해 질문 요망 (단순히 소스코드를 보내고 디버그 요청하는 질문에는 답변하지 않을 예정)

- 풀이 및 제출 방법

- 각 문제에 대한 코드는 문제에서 특별한 언급이 없으면 선언과 구현을 분리하여 각각의 .h / .cpp 파일에 작성 (대소문자 반드시 구분, 이를 어길 시 감점)
- 파일 이름은 각 문제에 주어진 대로 정의 (이를 어길 시 감점)
- 작성한 코드를 하나의 압축 파일로 압축
  - ◆ 압축 파일 이름은 "HW2\_(이름)\_(학번).zip"으로, zip 방식 압축 사용
  - ◆ 예) "HW2\_김태환\_2017-11111.zip"
- 만든 압축 파일을 eTL 강의 페이지에 마련된 "과제2" 제출함으로 제출

- 제출 기한

- 4월 30일 (월) 오후 11시 59분까지
- 오류 발생으로 eTL 과제함에 제출이 되지 않는 경우 이메일로 제출  
E-mail : [ds@snucad.snu.ac.kr](mailto:ds@snucad.snu.ac.kr)
- 지연 제출은 받지 않음(eTL / 이메일 모두), 지연 제출 또는 미제출 시 0점
- 파일이 첨부되어 제출되었는지 반드시 확인! 첨부되지 않은 채로 제출 시 0점