**[EE305] Introduction to Electronics Design Lab**

Post-lab Report Lab. 6 ‘Calculator Software’

20150603 이주현

20150651 장강욱

1. **Purpose**

① 8-Bit Microcontroller Atmega128a의 사용법을 익힌다.

② Stack의 개념을 이해한다.

③ C언어를 이용하여 Stack을 기반으로 하는 계산기를 설계하고 구현한다.

**2. Methods**

2.1 Material

1) Hardware : AVRISP mkII, USB cable, ISP cable, Calculator board

2) Software:　AVR studio, Visual Studio

2.2 Process

1) ATmega128a에서 계산기가 동작할 수 있도록 cal.c를 작성한다. 계산기는 사칙연산과 거듭제곱을 올바르게 할 수 있어야 하며, 괄호처리도 포함한다. Stack을 이용하여 수식을 계산하고, 그 결과는 LCD display를 통해 출력한다. 대신 수식에 오류가 있다면 error message를 출력한다.

2) 지시문을 따라서 컴퓨터에 AVR Studio를 설치한다.

3) AVR ISP mkII와 Calculator board를 ISP cable로 연결한 뒤에, 컴퓨터와 USB cable을 이용하여 연결한다. 기타 Setting도 지시문을 참고하여 끝마친다.

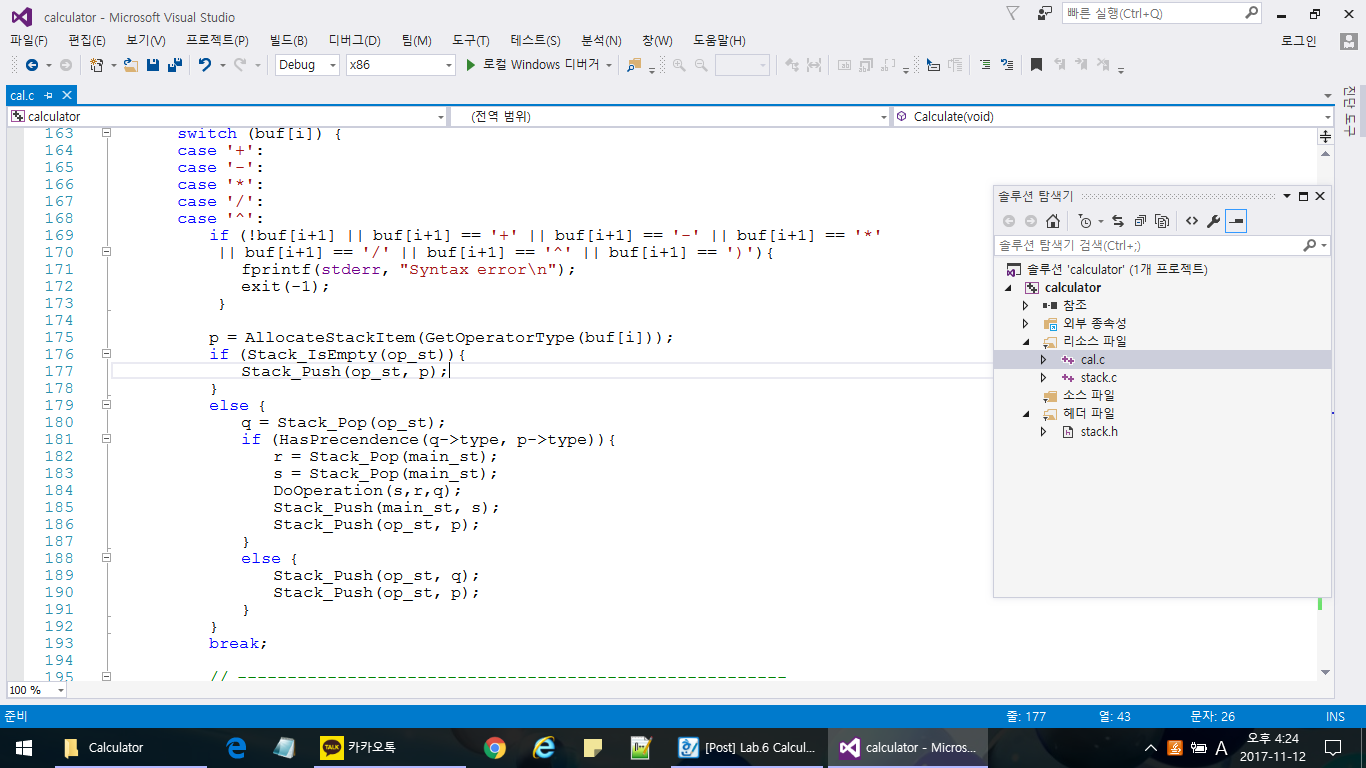
4) 작성한 cal.c를 바탕으로 구현된 계산기에 제시된 수식들을 실행해본다.

5) 오류가 생기거나 계산 결과가 맞지 않을 경우에는 코드를 수정하고 계산기를 실행하는 과정을 반복한다.

**3. Results**

3.1 작성한 코드

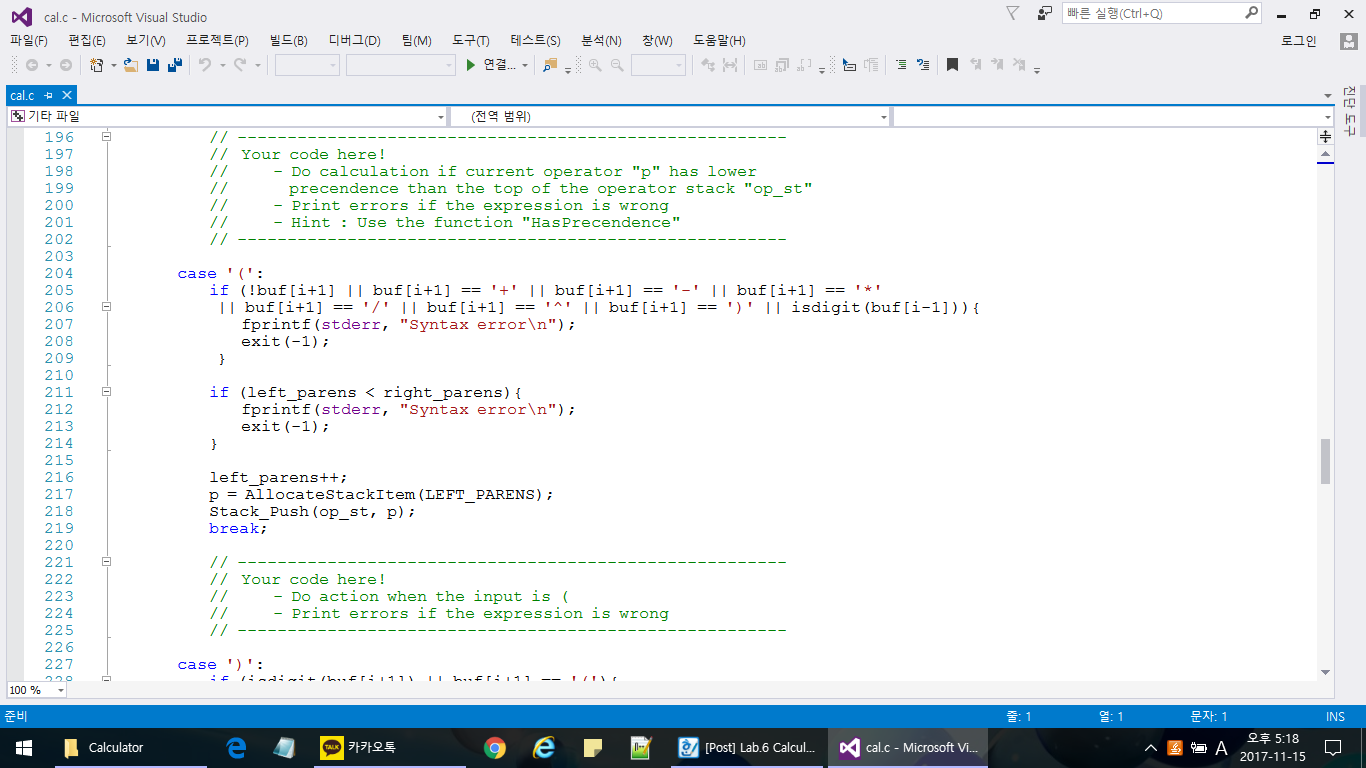
이미 skeleton code로 많은 부분이 작성되어 있었기 때문에 일부분에 대해서만 완성했다. ‘+,-,\*,/,^’ 등의 operator가 입력되었을 경우, ‘(’나 ‘)’과 같은 괄호가 입력되었을 경우, 그리고 모든 입력이 끝난 후에 어떻게 계산기가 작동해야하는지에 대하여 코드를 작성해야했다. 그리고 각각의 경우에 생길 수 있는 error에 대해서도 생각했고, 그 경우 프로그램이 멈추고 error를 띄우도록 하였다.



먼저, operator가 입력되었을 때에 관한 code이다. 가장 윗부분에서는 error를 먼저 걸러주었다. 이 때 생길 수 있는 error로는 operator 뒤에 NULL일 때(아무 입력이 없을 때), operator가 연속으로 입력되었을 때, 그리고 그 뒤에 ‘)’가 입력되었을 때이다.

이제 p에 memory를 할당하여 입력한 operator를 넣어준다. 만약 operator stack(op\_st)이 비어있을 경우, 즉 남아있는 operator가 없는 경우에는 바로 Stack\_Push를 이용해 operator stack에 넣어준다.

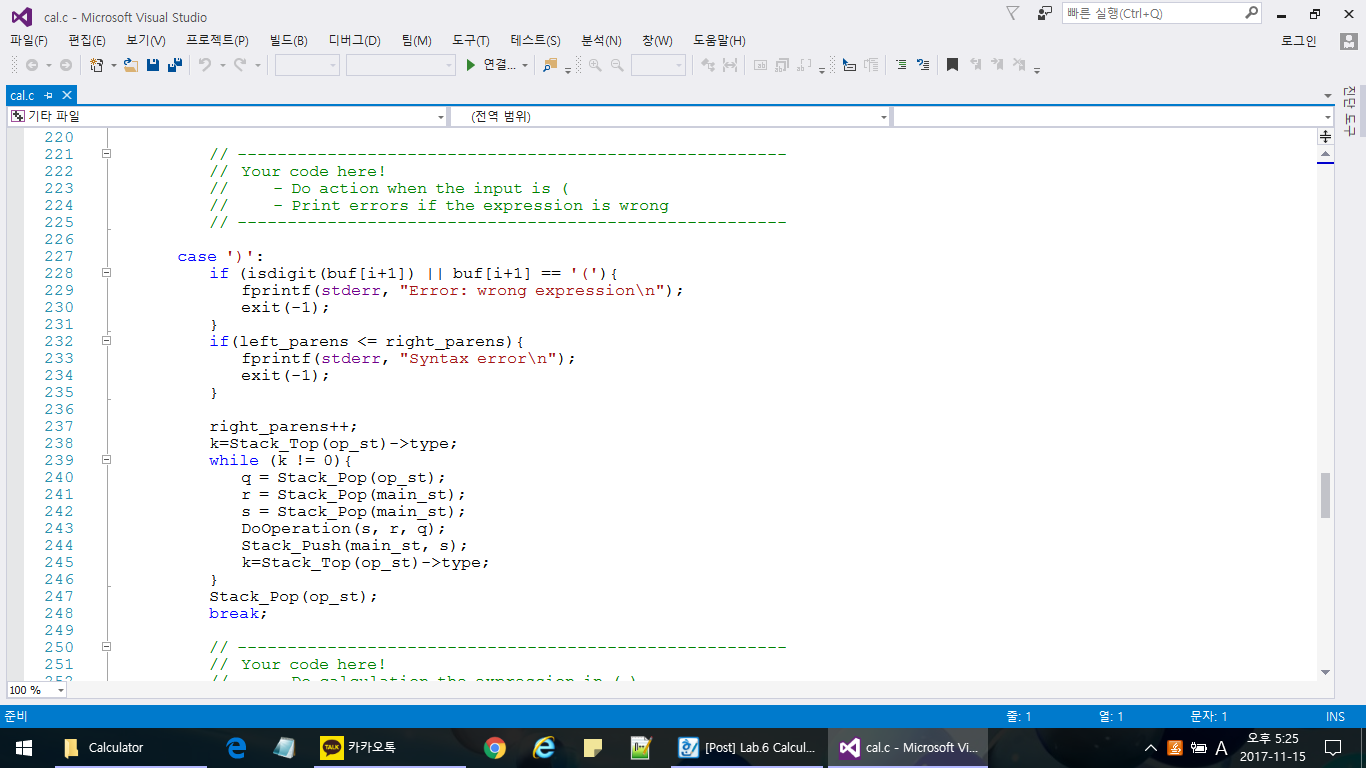
만약 op\_st에 이미 다른 operator가 있는 경우에는 HasPrecendence 함수로 우선순위를 비교해준다. 이미 operator stack에 있는 operator(q)의 우선순위가 p 이상이면, 그것의 연산을 먼저 해준다. 그래서 main stack(main\_st)에서 두 수를 pop하여 DoOperation 함수를 통해 연산을 하고, 연산 결과 얻은 수(s)와 연산자 p를 각각의 main stack에 넣어준다. p의 우선순위가 더 높을 경우에는 원래의 op\_st에 p만 push해주면 된다.



다음은 ‘(’가 입력된 경우이다. 첫 번째로 error case를 잡아낸다. ‘(’ 뒤에 입력된 것이 아무것도 없을 때, 다른 operator가 입력되었을 때, 닫는 괄호가 바로 뒤에 나올 때, 그리고 여는 괄호 이전에 숫자가 나올 때의 경우가 error이다.

left\_parens는 ‘(’의 개수를, right\_parens는 ‘)’의 개수를 의미한다. 닫는 괄호의 개수가 여는 괄호의 개수보다 더 클 수는 없으니, 그 경우도 error이다.

error인 경우를 모두 따졌으니 p가 ‘(’를 받아들이고, op\_st에 push한다. left\_parens도 1만큼 늘려준다.

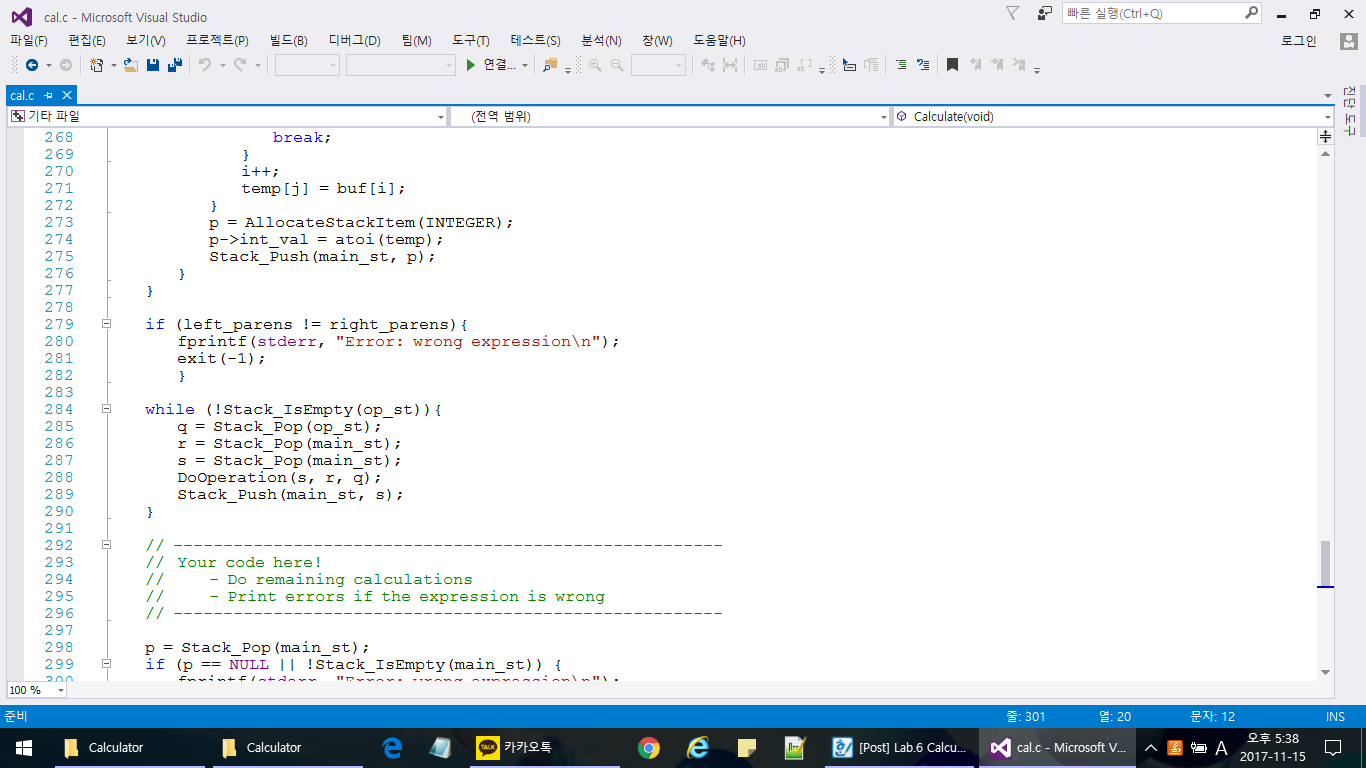


같은 괄호가 입력되어도 ‘(’의 경우와 ‘)’의 경우는 많이 다르다. ‘(’의 경우와 마찬가지로, right\_parens가 left\_parens보다 크거나 같으면 안 된다. 왼쪽과 오른쪽 괄호가 같은 상태에서 오른쪽 괄호를 한 개 더 받게 되면 오류이기 때문에 같은 경우도 배제해준다. 괄호 바로 뒤에 숫자나 열린 괄호가 와서도 안 되지만, 뒤에 아무것도 없는 경우와 operator가 오는 것은 가능하다. 바로 이전에 ‘(’나 operator가 오는 경우는 위에서 잡아주었기 때문에 생략했다.

‘)’가 입력되었을 때는 괄호를 stack에 넣지 않은 채로 다른 연산을 해주게 하였기 때문에 memory를 할당하지 않았다. 그러나 오른쪽 괄호를 읽어들었다는 것을 표시하기 위해, right\_parens는 1만큼 늘렸다.

k는 op\_st의 top에 있는 operator의 type을 말한다. ‘(’가 나올 때까지, 즉 괄호 안의 연산을 끝마칠 때까지 while문을 유지시켜야 하므로, while의 조건은 ‘k!=0’이다. 0인 까닭은 cal.c의 enum 부분에서 LEFT\_PARENS가 첫 번째 원소에 해당되기 때문이다.

While문 안에서는 stack 위에서부터 차례차례 계산을 하면 되었다. main stack에서는 두 수를, operator stack에서는 1개의 operator를 pop해서 연산하고, 그 결과를 main stack에 push해준 뒤, op stack의 top에 있는 operator가 무엇인지 판별해주는 것이다. 만약 operator가 ‘(’이면, while문을 빠져나갈 것이다.

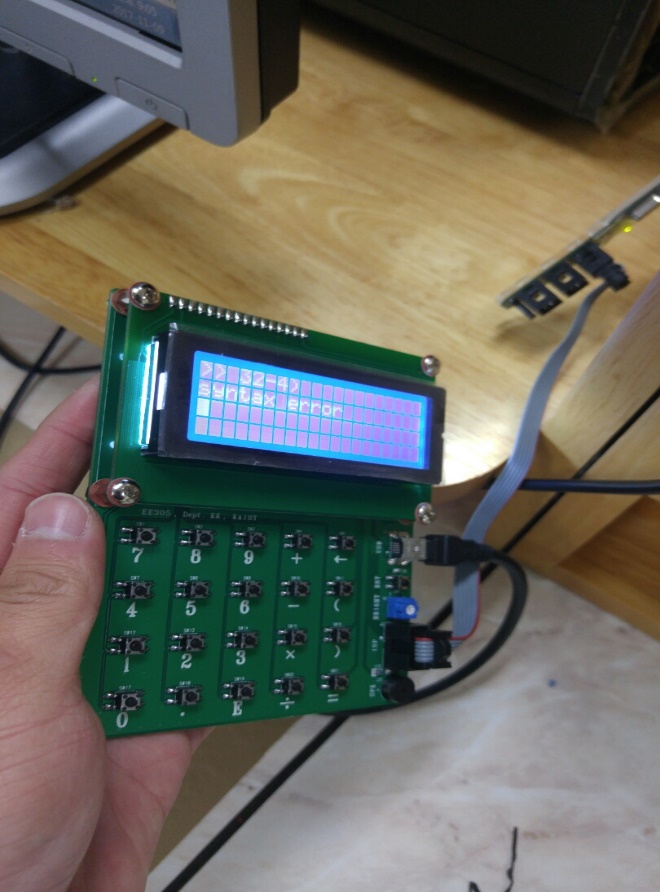


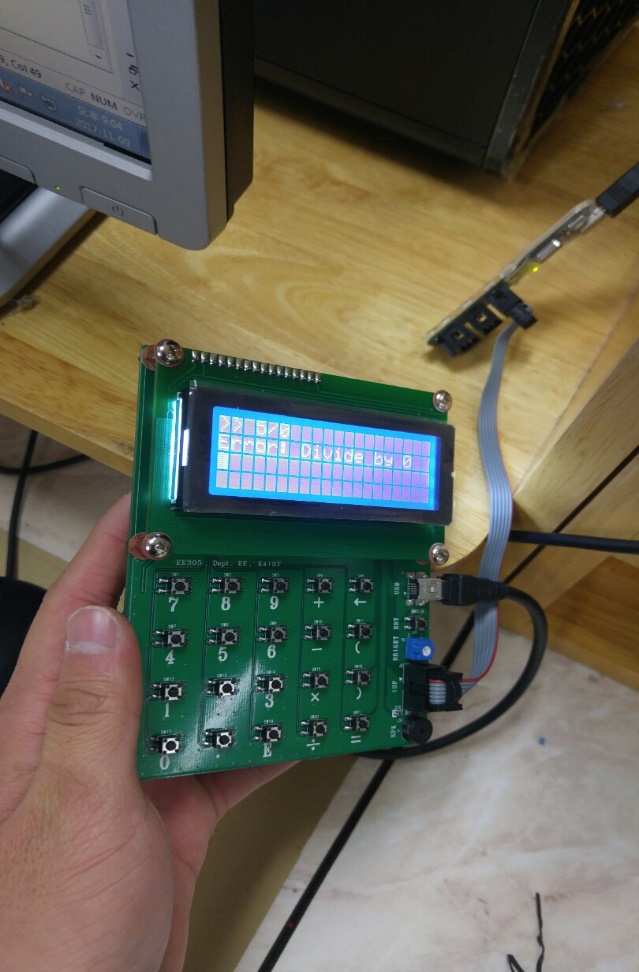
For문이 다 끝났을 때, 즉 모든 입력 값을 program이 받고 연산을 한 뒤에 stack에 남아있는 것들이 있다. 이것을 가지고 연산을 끝내는 부분이다. 먼저, 괄호의 개수가 서로 같은지 확인해준다. 위에 있던 코드에서 다 다루었지만 혹시 모를 error를 위해 입력했다. 연산이 끝난다는 것은 op\_st에 남아 있는 operator가 없다는 뜻이다. 그것이 while의 조건이 되었다. 연산은 이전에 했던 것처럼 해주면 된다.

여기까지의 코드는 실험시간 동안에 작성한 것이다. 나머지는 이미 skeleton으로 주어져 있어서 더 수정하거나 추가할 것이 없었다.

3.2 Demo 결과

주어진 calculator board와 AVR studio를 이용해서 직접 실행시켜보았다. 아래의 그림 5~8은 몇 가지 경우에 대한 연산결과이다. 결과를 뚜렷하게 나타내기 위해서 board를 기울여서 사진을 찍었다. 제대로 된 연산 결과도 입력했고, syntax error를 만드는 식과 'Divide by 0'이라는 문구를 띄우는 경우도 입력했다.





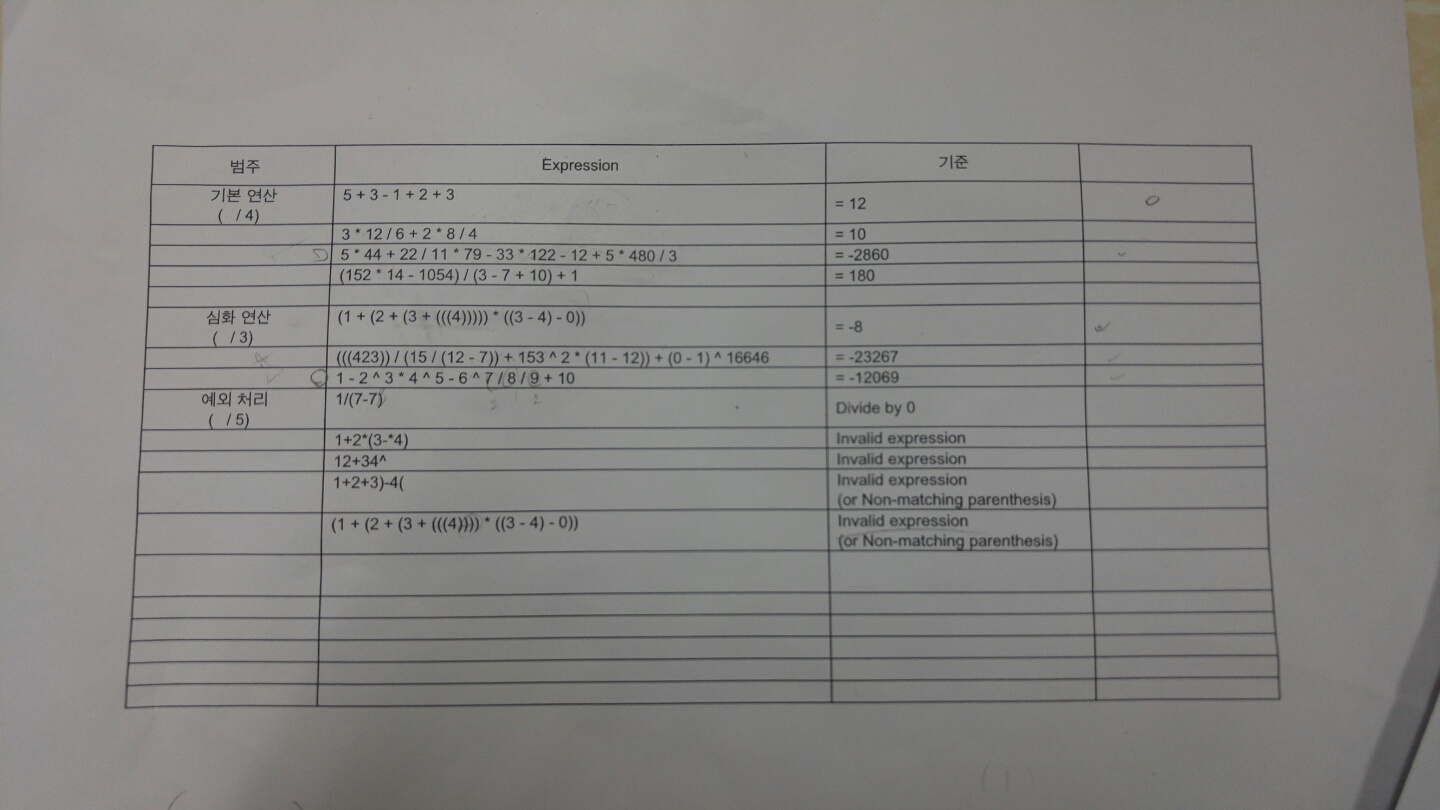
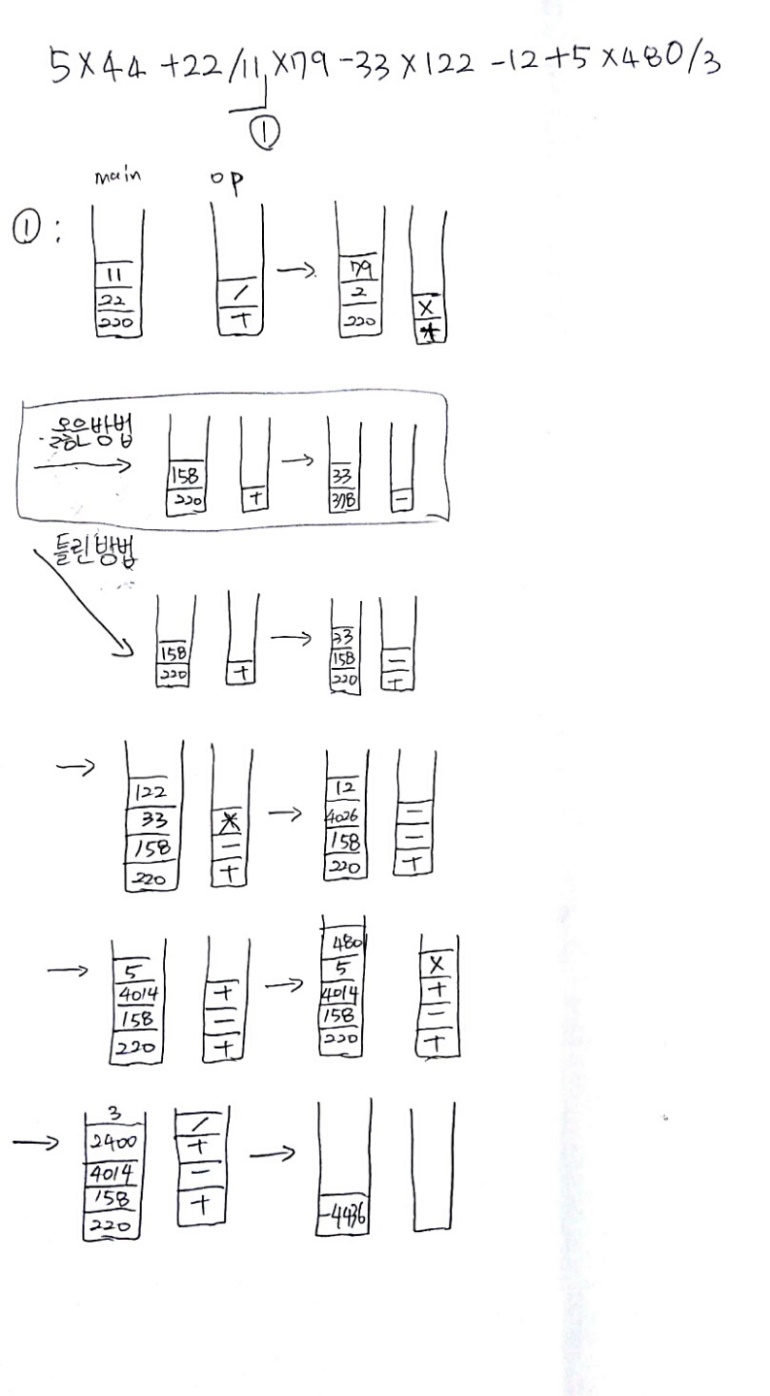


그림 9의 표를 참고하여 Demo를 받았다. 3번째, 7번째 식을 제외하고는 모두 올바른 계산 결과가 나왔다.

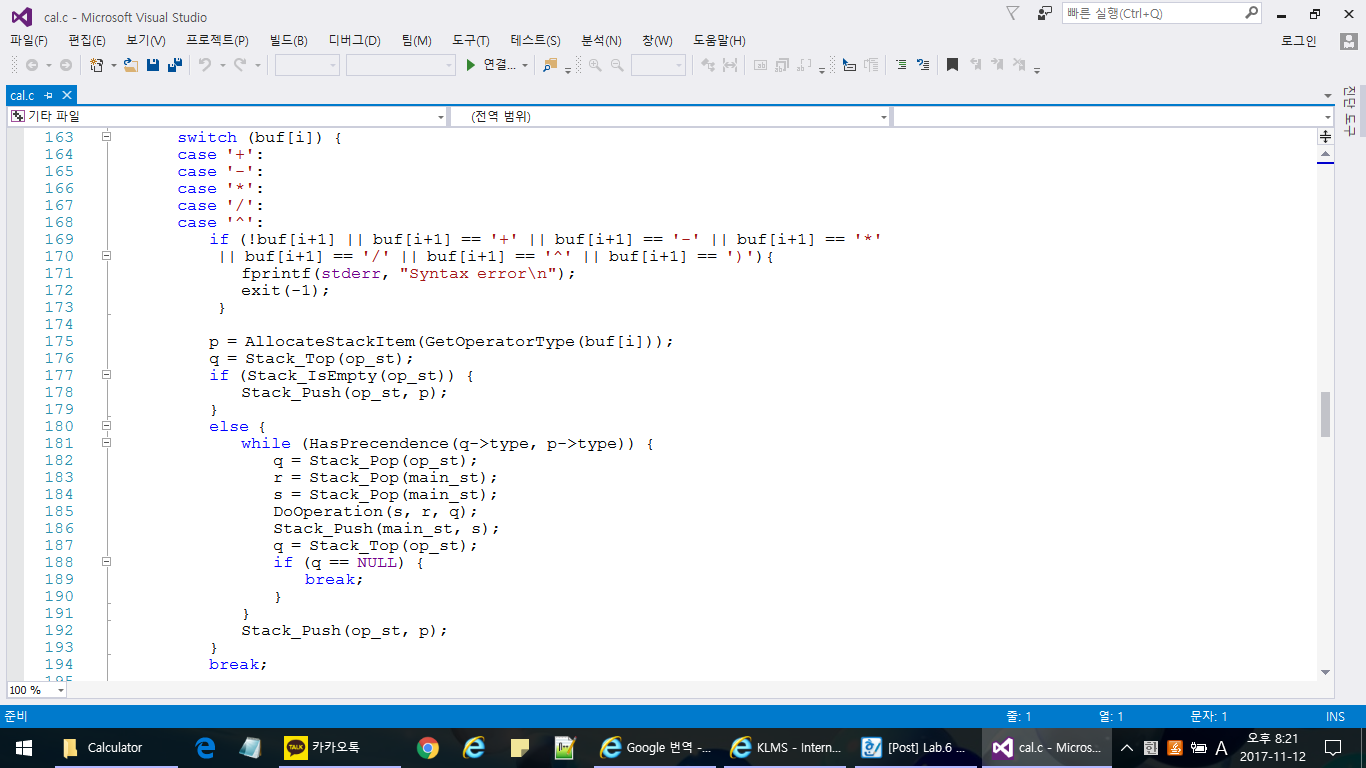
**4. Discussion**

4.1 올바른 수식에 대하여 모두 잘 동작을 하였는가? 잘 동작하지 못했다면 그 부분과 수정 방법은 무엇인가?

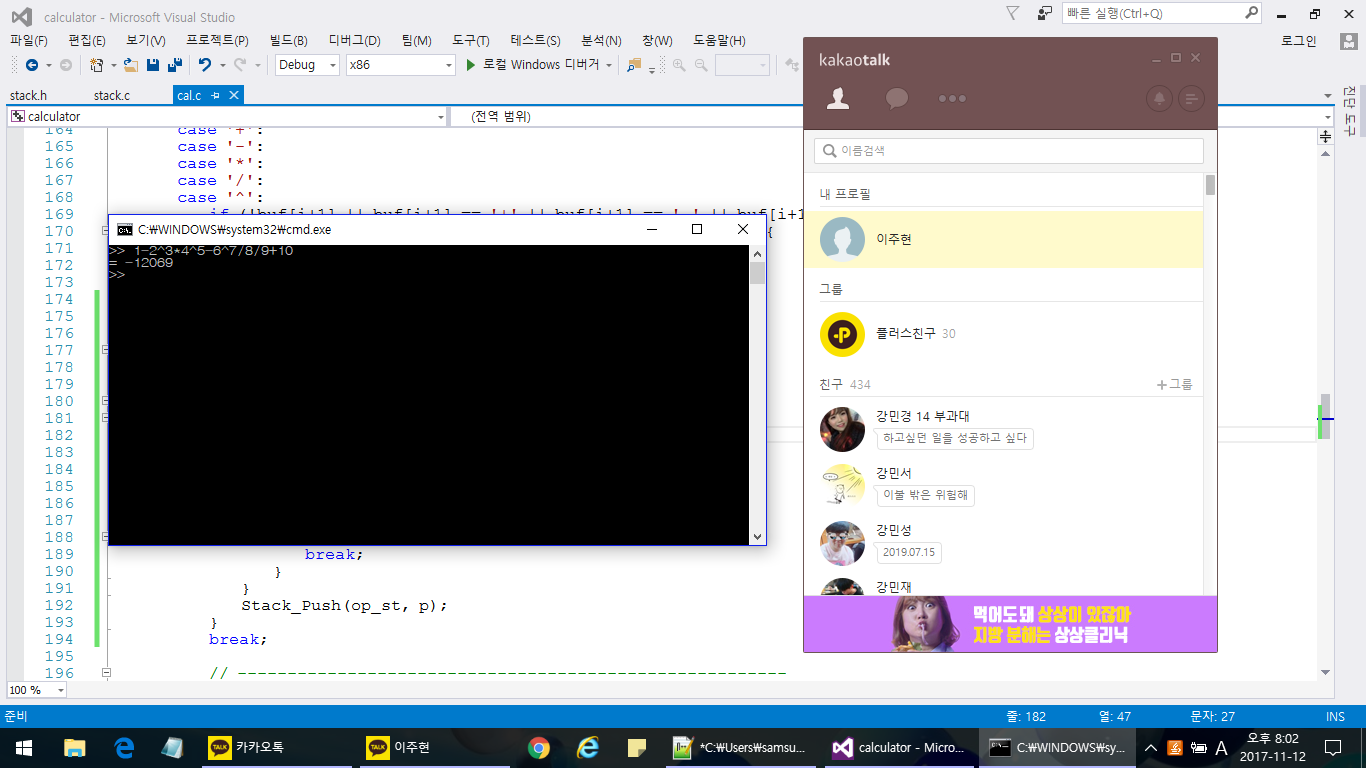
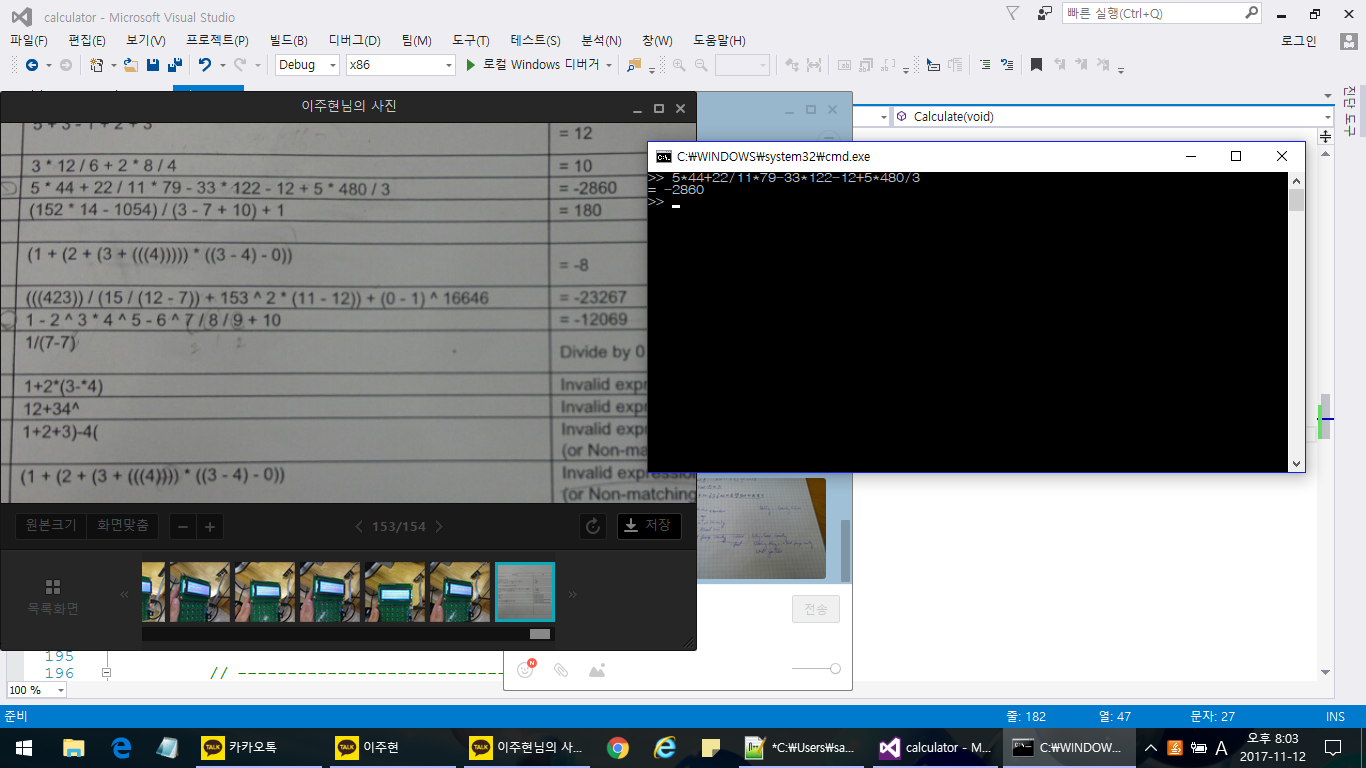
Demo를 받을 때 우리의 코드는 위 그림 1과 같았다. 저 코드에서는 연산의 오류가 생길 수밖에 없다. 우리는 operator를 stack에 넣기 전에 stack 가장 위의 operator와 우선순위를 비교했고, 그에 따라 연산을 먼저 하거나 operator를 바로 stack에 넣는 작업을 했다. 하지만 operator 사이의 우선순위를 비교하는 과정은 새로운 operator가 들어갈 때마다 단 1번뿐이었다. 이것이 문제가 되었다.



위 코드가 잘못되었다는 것을 이 연산을 통해 알 수 있었다. 우선, 이 연산의 답은 -2860이지만, 우리의 연산결과는 -4436이었다. 식의 ⓵이라고 되어있는 부분까지는 문제가 없다. 그 뒤에 \*를 넣고, 79를 stack에 넣었다. 이제 -를 넣을 차례이다. 먼저, -를 넣기 전에 \*와 우선순위를 비교한다. \*가 더 높으니 79\*2를 계산하고, 158을 main stack에 push한다. 올바른 계산대로라면 새로운 operator stack의 가장 위에 있는 +와 -를 다시 한 번 비교하여, +연산을 한 뒤에 -가 operator stack에 들어가야 한다. 하지만 우리의 code로는 +와 -를 비교하는 과정이 없었고, 그대로 -가 stack에 push 됐다. 따라서 위와 같은 잘못된 결과가 나온 것이다.



그래서 다음과 같이 수정했다. Operator Stack의 top에 있는 operator의 우선순위가 새로운 operator보다 작을 때까지 while문은 계속해서 돌아가게 함으로써, 올바른 연산을 할 수 있게 하였다. 만약 operator stack이 비었으면 우선순위를 비교할 수 없으므로, 그럴 경우에는 break를 통하여 while문을 빠져나갈 수 있게 하였다.



새로 수정한 코드의 실행 결과, 모든 연산이 올바르게 이루어졌다. 위 사진은 Demo 때 실패했던 두 식의 결과이다.

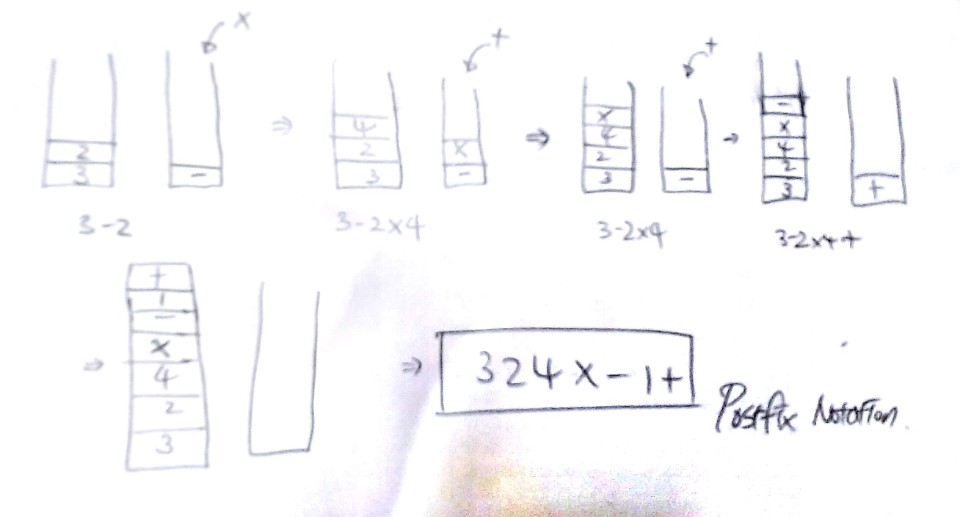
4.2 올바르지 않은 수식에 대하여 모두 error 메시지를 출력하였는가? 모두 출력하지 못했다면 출력하지 못한 올바르지 않은 수식은 무엇이고, 수정 방법은 무엇인가?

Demo에서 error로 분류해야 하는 식에 대해서는 코드가 모두 올바르게 분류했다. 또한, 각 operator의 앞, 뒤 buffer에 대해서 발생할 수 있는 error는 3.1에서 모두 설명했으므로 이 질문은 생략하겠다.

**5. Questions**

Q1.

Postfix notation(RPN, reverse Polish notation)은 operator를 operand의 뒤에 쓰는 연산 표기법이다. 우리가 흔히 쓰는 연산 표기법은 Infix notation으로 operand와 operand 사이에 operator가 들어간다. 이 표기법의 장점은 컴퓨터가 수식을 받아들일 때 별다른 변환 없이, 그저 앞에서부터 수식을 읽어나가면서 operator와 operand를 stack에 저장하면 된다는 것이다. 컴퓨터의 올바른 연산을 위해선 operator의 우선순위를 고려해줘야 하므로, Infix notation을 operator의 우선순위를 고려한 Postfix notation으로 변환한다.



따라서 operator stack에 어떤 연산자가 오는지에 따라, top operator stack의 연산자를 pop하여 main stack으로 옮길지, 아니면 새로 온 연산자를 operator stack에 push할지 결정한다. 첫 번째 그림에서, -와 \*의 연산 우선순위가 \*가 높으므로 stack에 push했다. 왜 우선순위가 높으면 push해야 하는지에 대해선 이번 실험의 Pre-lab 보고서에 설명했으므로 생략하겠다. 두 번째 그림에서, +와 \*의 우선순위를 비교했을 때, \*가 높으므로 \*를 pop하여 main stack으로 push했다. 세 번째 그림에서, -와 +의 우선순위가 같으므로 –를 pop하여 main stack으로 push한 다음에 +를 operator stack에 push했다. 식을 모두 받아들였으므로, main stack의 밑에서부터 숫자와 연산자를 순서대로 쓰면 postfix notation이 완성된다. 답은 324\*-1+.

Q2.

HasPrecendence 함수는 operator 사이의 우선순위를 정해주는 함수로, 이 함수가 없을 경우 컴퓨터는 operand와 operator를 buffer로부터 받아들일 때마다 바로바로 계산해버리게 된다. 따라서 operator의 우선순위를 전혀 고려하지 않고, 그저 식의 왼쪽에서부터 순서대로 계산을 진행하면 된다.

a. 7-4+1+4-7

(준식) = 3+1+4-7 = 4+4-7 = 8-7 = 1

-> +와 –는 왼쪽에서 오른쪽으로 결합하는 연산자이고 둘의 우선순위 역시 같으므로, HasPrecendence 함수가 없어도 이 식의 결과는 변하지 않고 그대로 1일 것이다.

b. 7\*44+33/11\*44

(준식) = 308+33/11\*44 = 341/11\*44 = 31\*44 = 1364

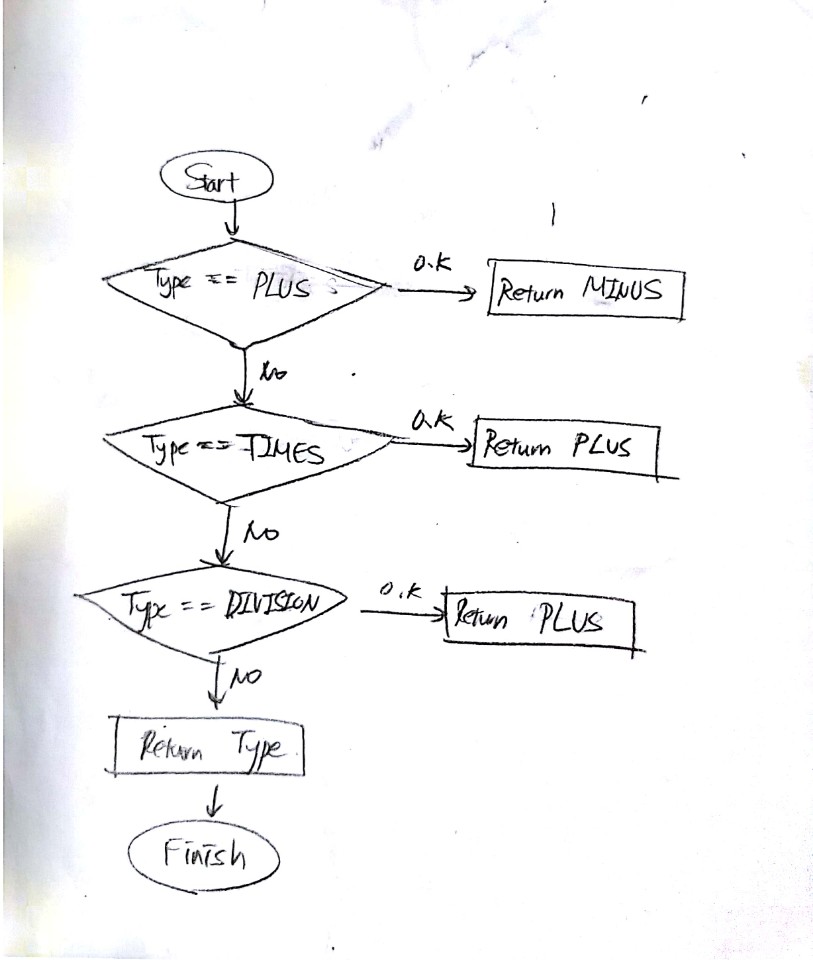
-> 사칙연산자 모두가 오른쪽에서 왼쪽으로 결합하는 연산자이지만, \*와 /는 +보다 더 먼저 연산되어야 한다. 따라서 HasPrecendence 함수가 없을 경우, 이 식의 결과는 1364로 바뀔 것이다. HasPrecendence 함수가 있으면, 이 식의 결과는 440이다.

Q3.

cal.c에서 HasPrecendence 함수를 통해, 새로운 operator를 operator stack에 push할지 아니면 top operator stack에 있는 operator를 계산할지 정할 수 있었다. HasPrecendence 함수를 보면, 새로운 operator와 top stack operator의 type을 입력으로 하여 GetOperatorPrec 함수를 쓴 것을 확인할 수 있다. type은 정수로, enum에서 이미 operator에 대한 그 숫자가 정해져 있다. 결국, 우선순위를 GetOperatorPrec 함수에서 숫자화 하고, 이것을 HasPrecendence 함수에서 비교함으로써, operator 사이의 우선순위에 따른 코드의 행동을 정할 수 있다.

우리가 원하는 것은 +,-와 \*,/의 우선순위를 바꾸는 것이다. 기존의 GetOperatorPrec 함수에 +,-,\*,/의 type을 입력했을 때의 출력은 각각 3,3,5,5이다. 이를 +,-와 \*,/의 출력은 서로 같게 해주면서, +,-의 출력을 \*,/의 것보다 크게 해주면 +,-와 \*,/의 우선순위가 뒤바뀔 것이다.

예를 들어, +,-,\*,/의 출력을 각각 3,3,2,2로 바꿔 보는 경우를 생각해 볼 수 있다. ^는 문제에서 언급되지 않았으므로 기존의 GetOperatorPrec 함수의 출력인 6을 유지한다. 이를 이용해서 순서도를 그리면 그림 15와 같으며, 이를 기반으로 GetOperatorPrec 함수의 코드를 다시 짜면 다음과 같다. 순서도는 여러 가지 경우가 나올 수 있다.



return (type == PLUS) ? MINUS :

((type == TIMES) ? PLUS : ((type == DIVISION) ? PLUS : type));

**6. References**

1. Hyung-Joun Yoo - EE305 Calculator S/W

2. EE305 전자설계 및 실험, ‘실험 6. 계산기 설계 (software)’

3. Instructions for calculator board for EE305