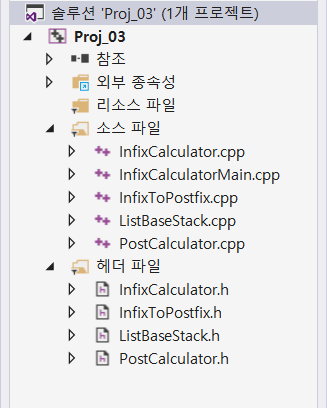
Project #3:: 프로그램 보고서:: 각 프로그램 기능에 대한 보고서

***!!!! 유의사항: 한 자리 숫자의 피연산자로만 계산 가능. 두 자리 이상의 피연산자는 받지 못하는 프로그램. 이외 문제에서 제시하는 모든 요건들은 충족시킴.***

**(0) 3번 프로그램 구성**



Proj\_03 프로젝트의 구성은 다음과 같다. c파일 5개, 헤더파일 4개 이렇게 총 9개의 파일들로 구성된다.

위 4개의 헤더 파일들은 모두 따로 정의한 사용자 정의 헤더 파일이다. 사용자 정의 헤더파일이기 때문에 c파일에서 include시킬 때에는 “ ” 쌍따옴표를 이용하여 가져오면 된다.

**(1) 사용한 각 함수에 대한 설명 (Input, Output, 알고리즘 설명)**

* **3번 계산기 프로그램의 구현 방식 설명**

1. **사용자로부터 중위 표기법으로 표현된 문자열 수식을 입력 받는다.**
2. **중위 표기법으로 받은 수식을 후위 표기법으로 변환한다.**
3. **이제 후위 표기법으로 표현된 수식을 계산한다. (여기서 스택 이용)**
4. **최종 계산 결과물을 출력하여 준다.**

**1. 사용자로부터 중위 표기법으로 표현된 문자열 수식을 입력 받는다.**

일반적인 사람들에게 익숙한 것은 중위 표기법이다. 중위 표기법(Infix)은 수식이 1 + 2 / 3 이런 식으로 표현된 기법이다. 제일 먼저 사용자가 계산되기 원하는 수식을, 사용자에게 가장 익숙한 표현인 중위 표기법으로 입력 받도록 한다.

*(EX) 9 - 2 \* 3 << 이런 식으로 사용자로부터 입력받게 될 것.*

**2. 중위 표기법으로 받은 수식을 후위 표기법으로 변환한다.**

하지만 중위 표기법으로 표현된 수식에는 연산순서에 대한 정보가 담겨있지 않다.( 컴퓨터가 중위 표기법으로 표현된 수식을 보고 우선 순위에 따라 스스로 알아서 계산하여 줄 수는 없다. 기계 학습을 따로 시키지 않은 이상…. ) 따라서 중위 표기법으로 입력 받은 수식을 후위 표기법의 수식으로 변환하여 줄 것이다. 후위 표기법 수식은 연산자의 배치순서에 따라서 연산순서가 결정되기 때문에, 연산자의 우선순위를 알 필요가 없다. 따라서 이 과정을 꼭 거치는 것이 좋다.

*(EX) 3 + 2 \* 4 🡪 3 2 4 \* + << 이런 식으로 변환하여 줄 것임.*

**3. 이제 후위 표기법으로 표현된 수식을 계산한다. (스택 이용)**

후위 표기법의 수식으로 변환되었으면, 이제 계산을 진행할 것이다. 여기 스텝에서는 ‘스택’ 개념을 이용한다. 이 부분에서의 자세한 알고리즘은 이후에 기술될 것이다.

**4. 최종 계산 결과물을 출력하여 준다.**

여태 후위 표기법으로 변환한 수식을 계산하여 그 결과를 얻어냈다. 사용자에게 그 결과 내용을 최종적으로 출력시키도록 한다.

* **각 파일마다의 역할**

**ListBaseStack.cpp ListBaseStack.h :** 스택을 활용하기 위한 코드

**InfixToPostfix.cpp InfixToPostfix.h :** 중위 표기법의 수식을 후위 표기법으로 변환

**PostCalculator.cpp PostCalculator.h :** 후위 표기법으로 된 수식을 계산

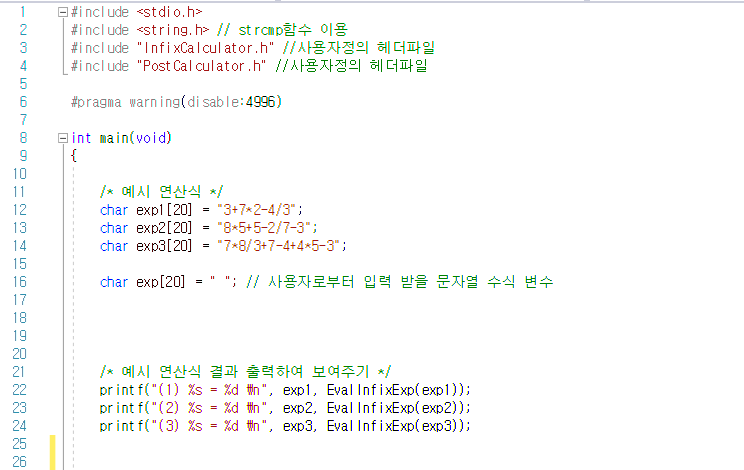
**InfixCalculator.cpp InfixCalculator.h :** 중위 표기법의 수식 최종 계산

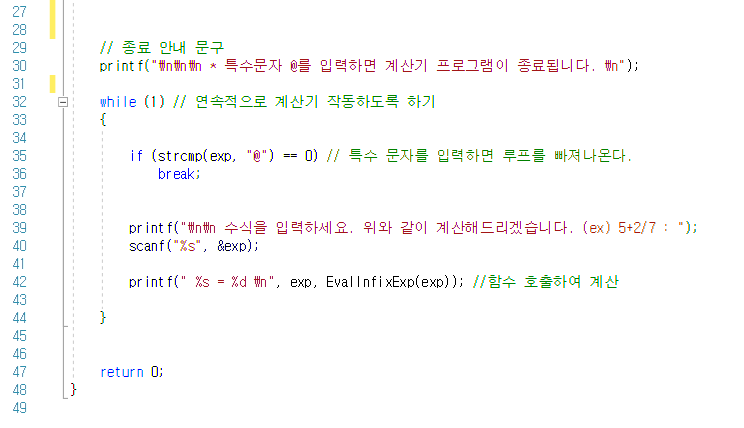
**InfixCalculatorMain.cpp :** 메인 함수

* **각 파일들에 대한 자세한 설명(파일 안의 각 함수 인풋, 아웃풋, 알고리즘 설명)**

**# 이외 자세한 설명은 소스 캡쳐의 주석을 참고하길 바람! 여기서는 각 함수 인풋 아웃풋 알고리즘만 설명!**

**< InfixCalculatorMain.cpp >**





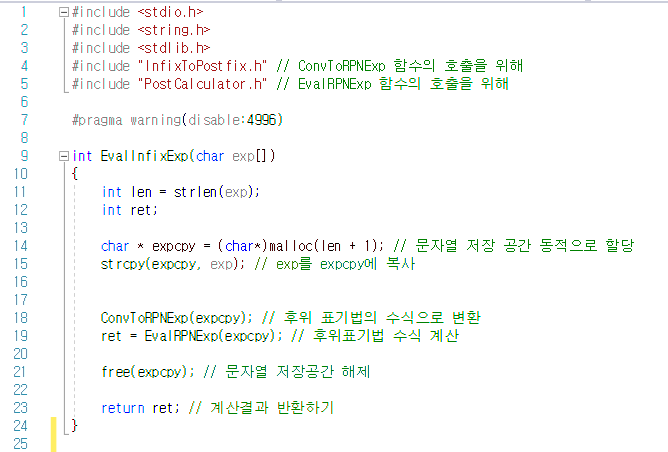
InfixCalculatorMain.cpp파일에는 프로그램 시작 시 제일 먼저 가동되는 main함수가 있다.

처음에는 사용자에게 계산기 프로그램의 예시를 보여주기 위해 exp1, exp2, exp3에 미리 문자열 수식을 넣고 그의 계산 결과를 출력시켜 보여주었다. 예시를 보여주고 난 뒤에는, 사용자에게 문자열 수식을 입력 받도록 한다. While 무한 루프를 돌려서 계산기가 연속적으로 작동하도록 만들었다. 특수문자 “@”를 입력하면, 루프를 빠져나가면서 메인 함수가 종료된다. 사용자가 프로그램을 성공적으로 돌릴 수 있도록 안내 문구도 틈틈히 프린트 시켜주었다.

특히, 진정한 계산 처리는 함수를 호출하여 진행된다. 메인 함수에서는 EvalInfixExp함수를 호출하는데, 그 함수 내부에서는 또 ‘후위 표기법으로 변환’, ‘후위표기법으로 변환된 수식을 계산’ 하는 처리를 담당한다. 그 처리들을 위해 또 각각의 필요한 함수를 호출한다. 이런 식으로 모든 파일이 유기적으로 연결되어 있다. 정리하면 메인 함수에서 함수를 호출 시키는데, 그 함수를 실행시키면 그 안의 또 다른 함수 호출을 만나고 만나고 해서 처리를 담당하고 그 함수를 종료하고 최종적으로 다시 메인함수 원래로 돌아와서 메인함수가 종료되면서 프로그램이 끝나게 된다.

메인함수에서 EvalInfixExp함수를 이용하기 위해(수식을 계산하기 위해), 그 함수의 원형이 들어있는 InfixCalculator.h파일을 “ ”를 이용하여 include시켰다. EvalInfixExp함수 호출 시, 또 EvalInfixExp함수 내부 안에서는 또다른 함수EvalRPNExp를 호출시킨다. EvalRPNExp함수는 또 PostCalculator.h파일에 원형이 정의되어 있는데, 결국은 메인 함수에서 두 함수 모두 호출하게 되는 것이므로 InfixCalculator.h 와 PostCalculator.h 파일 둘 다 include시켜야 한다. 이렇게 필요한 헤더파일들을 include 시켜줄 때에도 주의가 필요하다.

**< InfixCalculator.cpp >**



앞서 메인 함수에서 EvalInfixExp 함수를 호출한다. EvalInfixExp함수는 바로 이 InfixCalculator.cpp파일에 정의되어 있다.

EvalInfixExp 함수는 ‘통합’하는 기능을 한다. 이전에 우선순위 계산을 진행하기 위해서는 중위표기법의 수식을 (1)후위 표기법으로 변환하고, (2)변환된 그 후위표기법의 수식을 계산해야 한다고 말했었다. (1), (2) 기능은 또 각자의 기능을 하는 함수를 호출하여 진행시키는데, 이 EvalInfixExp 함수 안에서 (1), (2) 함수를 각각 호출시켜 통합하는 기능을 한다. 다시 말해, 전체 프로그램을 깔끔하게 정리해주는 기능을 한다고 볼 수 있다.

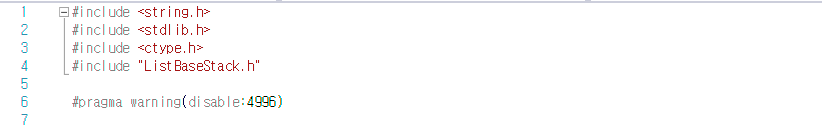
EvalInfixExp 함수의 input은 ‘문자열 수식’이다. 메인 함수에서 인자로 ‘사용자로부터 입력 받은 문자열 수식’을 넘겨줄 것이다. 그 인자로 넘어온 그 문자열 수식을 가지고 EvalInfixExp함수 내부에서의 처리가 진행된다. EvalInfixExp함수의 output은 결국 ‘최종 계산 결과 값(문자열이 아닌 정수형 숫자)’일 것이다.

ConvToRPNExp(expcpy); // 후위 표기법의 수식으로 변환

ret = EvalRPNExp(expcpy); // 후위표기법 수식 계산

이 두 함수의 호출 과정을 거치고 나면, 수식 계산 결과 값이 산출될 것이다. 그 값을 ret변수에 넣어 그를 리턴 값으로 넘겨줄 것이다.

**< InfixToPostfix.cpp >**



InfixToPostfix.cpp파일의 맨 처음 부분이다. 이 파일에는 중위 표기법의 수식을 후위 표기법으로 변환시키기 위한 코드들이 기술되어 있다.

InfixToPostfix.cpp파일에는 다음과 같은 3개의 함수들이 정의되어 있다. 함수의 원형은 다음과 같다.

int GetOpPrec(char op);

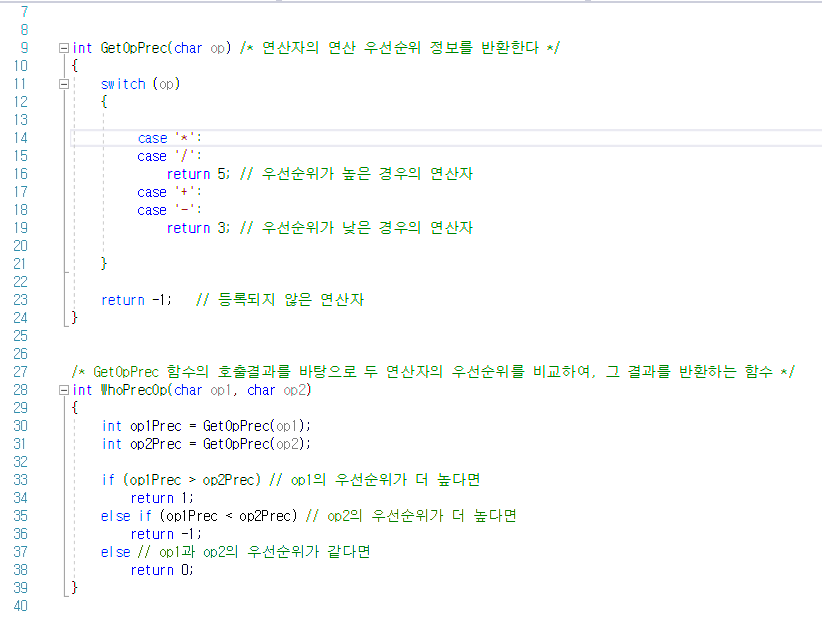
int WhoPrecOp(char op1, char op2);

void ConvToRPNExp(char exp[]);

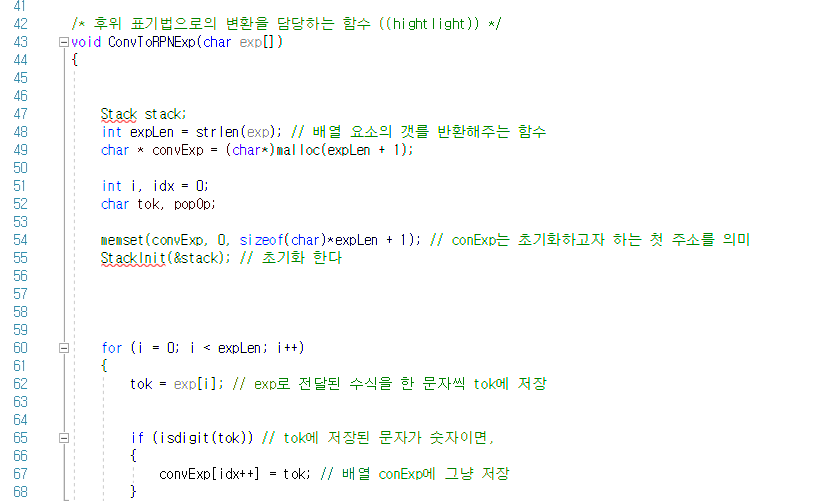
**.**

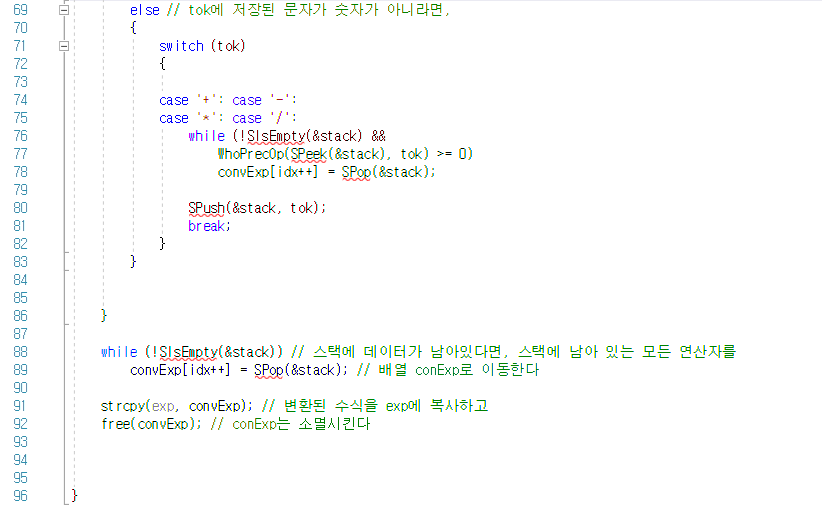
**.**

**(다음 장에 이어서 계속 설명)**

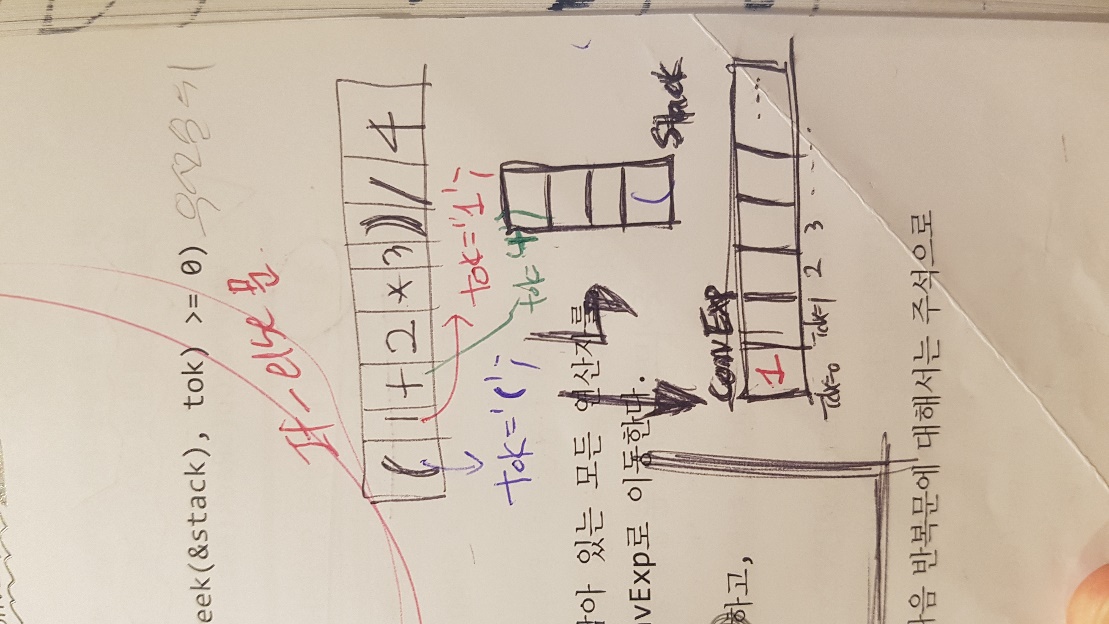


위의 두 함수, **GetOpPrec** 와 **WhoPrecOp 함수**는 ConvToRPNExp함수 구현에 앞서 이를 돕는 함수들이다. 두 개의 함수 각각의 기능은 주석으로 달아놓았다. 둘 다 input, output이 int형이다.





위가 바로 **ConvToRPNExp함수**에 대한 정의이다. 이 함수에서 본격적으로 후위 표기법으로의 변환 처리를 다룬다. 자세한 설명은 주석에 달아놨으며, 코드 이해를 위해 그린 그림을 바로 다음 장에 첨부해 놓겠다.

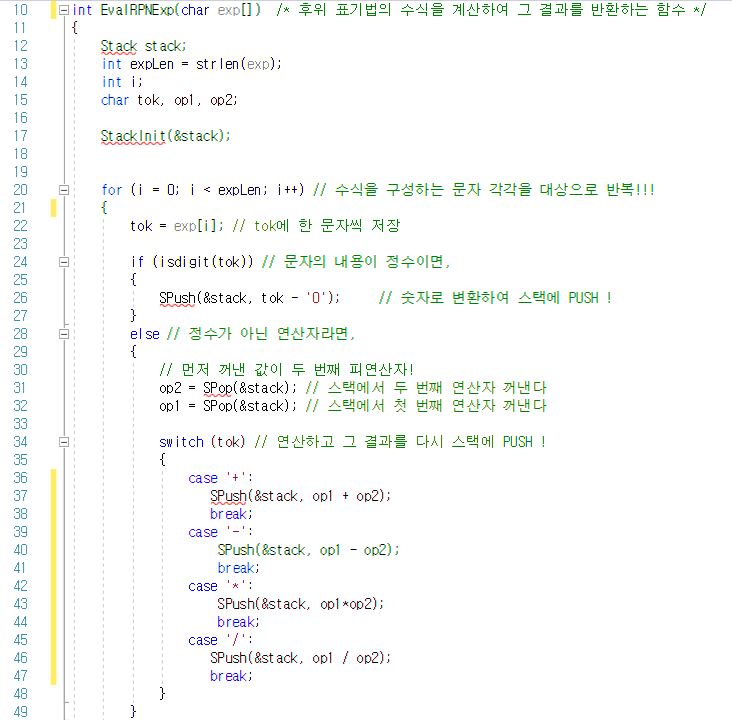


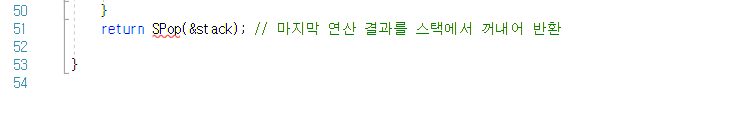
위와 같이, ConvToRPNExp함수의 알고리즘은 다음과 같다.

*Input의 인자로 exp 문자열 수식이 넘어온다. 이는 결국 한 줄 짜리 char형 배열이라고 볼 수 있는데, 이 문자열 배열에 있는 값을 하나씩 꺼내서 새로 동적으로 할당한 convExp배열에 저장할 것이다. 하나씩 꺼내서 처리를 할 때마다 for루프 한 번이 돌아가게 되는데, 그 for루프 안에서 ‘하나 꺼낸 바로 그 값이 숫자라면’, ’하나 꺼낸 바로 그 값이 연산자라면’의 두 경우로 나누어 처리를 진행할 것이다. 만약 꺼낸 값이 숫자라면 바로 convExp배열에 바로 저장하고, 연산자라면 스택에 저장한다. 그러면 스택에는 연산자들이 차례대로 넣어져서 들어가 있을 것이다. 그를 스택 꺼내는 방식으로 pop하여 convExp배열에 차례대로 다 넣는다. 그러면 이제 후위 연산자 표현으로 convExp배열이 구성되어 있을 것이다. 그 문자열 배열convExp을 exp에 복사하여, 기존의 exp값을 바꾸어준다. 이 함수의 output은 존재하지 않는다. 대신 input으로 들어온 exp값(중위표기법)이 후위표기법으로 표현된 문자열 수식으로 바뀌어 있을 것이다. ConvToRPNExp함수는 이렇게 작동된다. 참고로, convExp는 동적으로 할당시킨 새로운 배열인데, 임시 배열변수라고 생각하면 된다. Exp에 결과 값을 잘 옮겨놨으니 free를 통해 없애도 된다.*

여기까지 내가 스스로 작성한 ConvToRPNExp함수에 대한 설명이다.

**< PostCalculator.cpp >**





PostCalculator.cpp파일에는 EvalRPNExp함수에 대한 정의가 기술되어 있다. EvalRPNExp 함수는 후위 표기법의 수식을 계산하여 그 결과를 반환해주는 기능을 한다. EvalRPNExp함수의 input은 ‘후위 표기법으로 표현된 수식 문자열’이 된다. 이 함수를 호출하기 전에 이미 ConvToRPNExp함수 호출 단계를 거침으로써, 중위 표기법의 수식 문자열이 후위 표기법의 수식 문자열로 바뀌어 있을 것이다. 따라서 EvalRPNExp함수에서의 input으로는 후위 표기법으로 바뀐 수식 문자열이 인자로 들어올 것이다. / EvalRPNExp함수의 output으로 계산한 결과 값이 나갈 것이다. 즉 후위 표기법 수식을 계산한 결과 값으로 int형 정수가 반환된다.

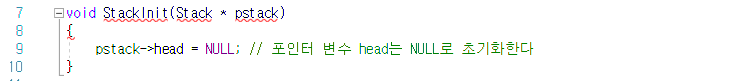
EvalRPNExp 함수의 알고리즘은 다음 규칙을 만족시켜야 한다.

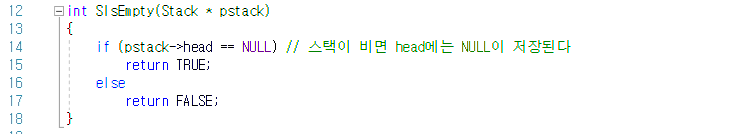
* 피연산자는 무조건 스택으로 옮긴다.
* 연산자를 만나면 스택에서 두 개의 피연산자를 꺼내서 계산을 한다.
* 계산 결과는 다시 스택에 넣는다.

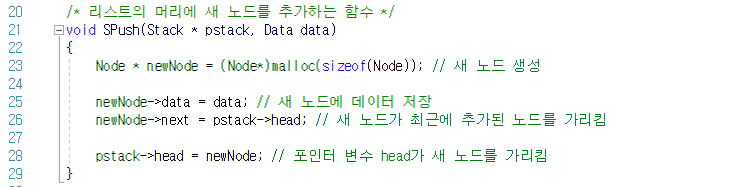
스택의 그림을 연상하며 프로그램이 어떻게 돌아가는지 손수 확인하였다.

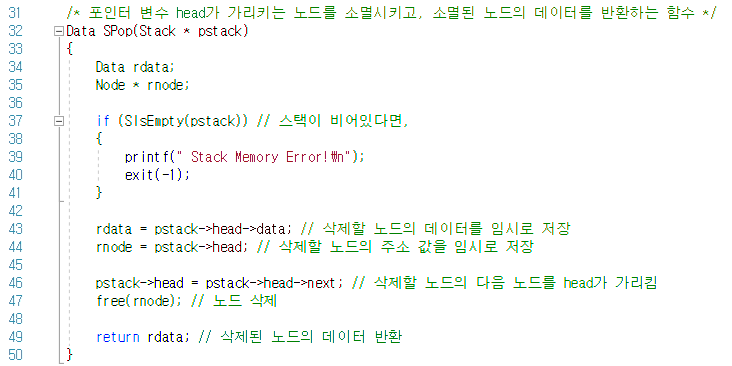
**< ListBaseStack.cpp >**

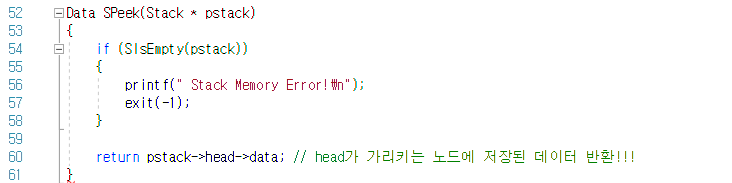
ListBaseStack.cpp파일에는 스택의 연결 리스트 기반을 구현한 코드들이 기술되어 있다. 일반적으로 ‘스택 개념’을 3번 프로그램에서 사용하기 위해서 스택 관련 함수들을 정의해놓은 것이다. 다음과 같이 5개의 스택 관련 함수들이 정의 되어있다.





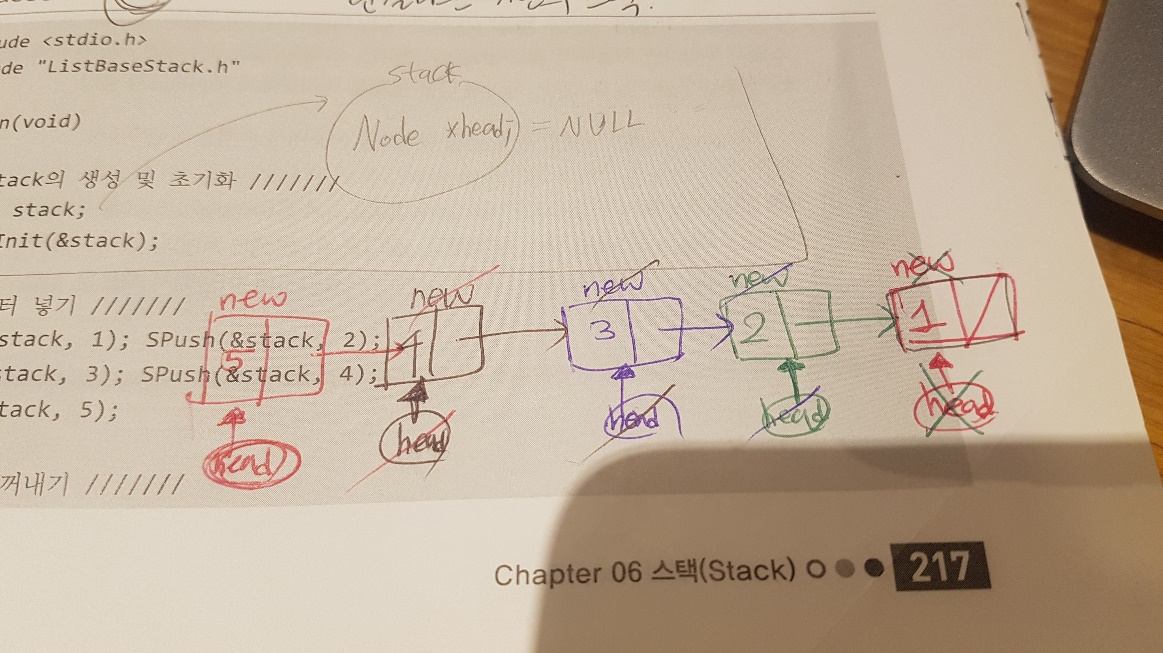
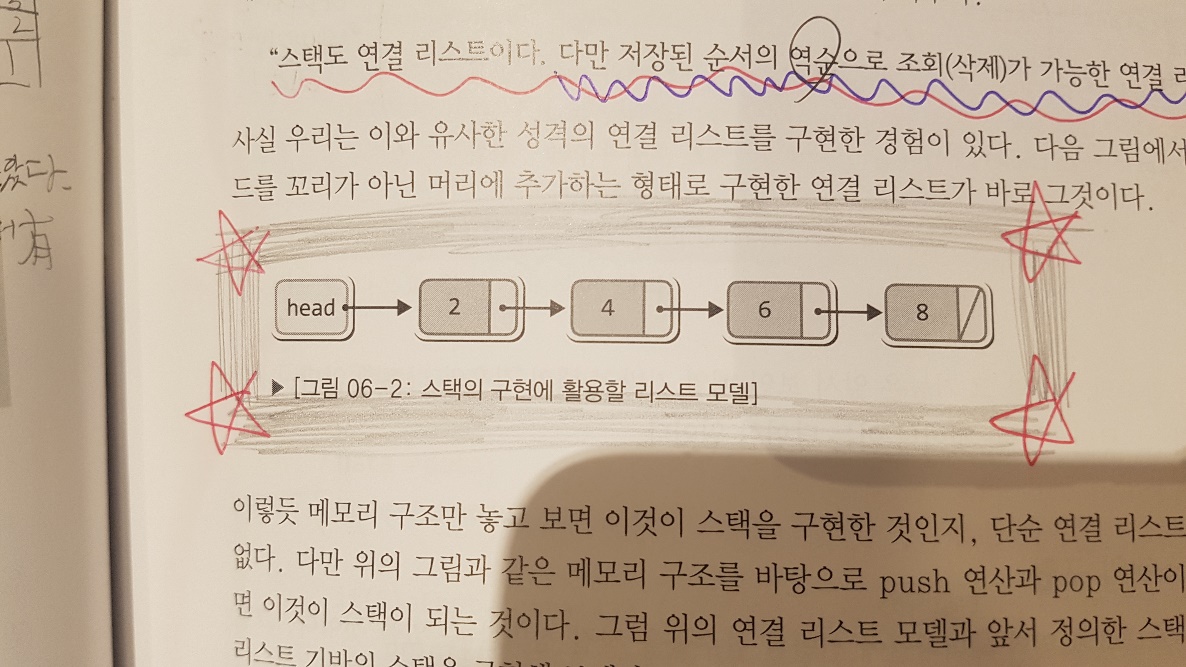




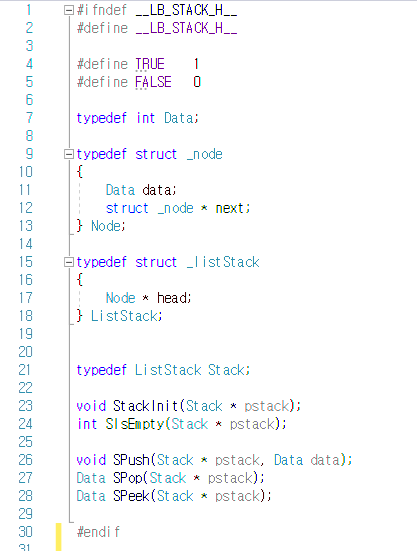


여기까지가 스택 관련 함수들에 대한 정의이다. 스택 함수들의 내용은 너무나도 일반적이므로 생략하도록 한다.

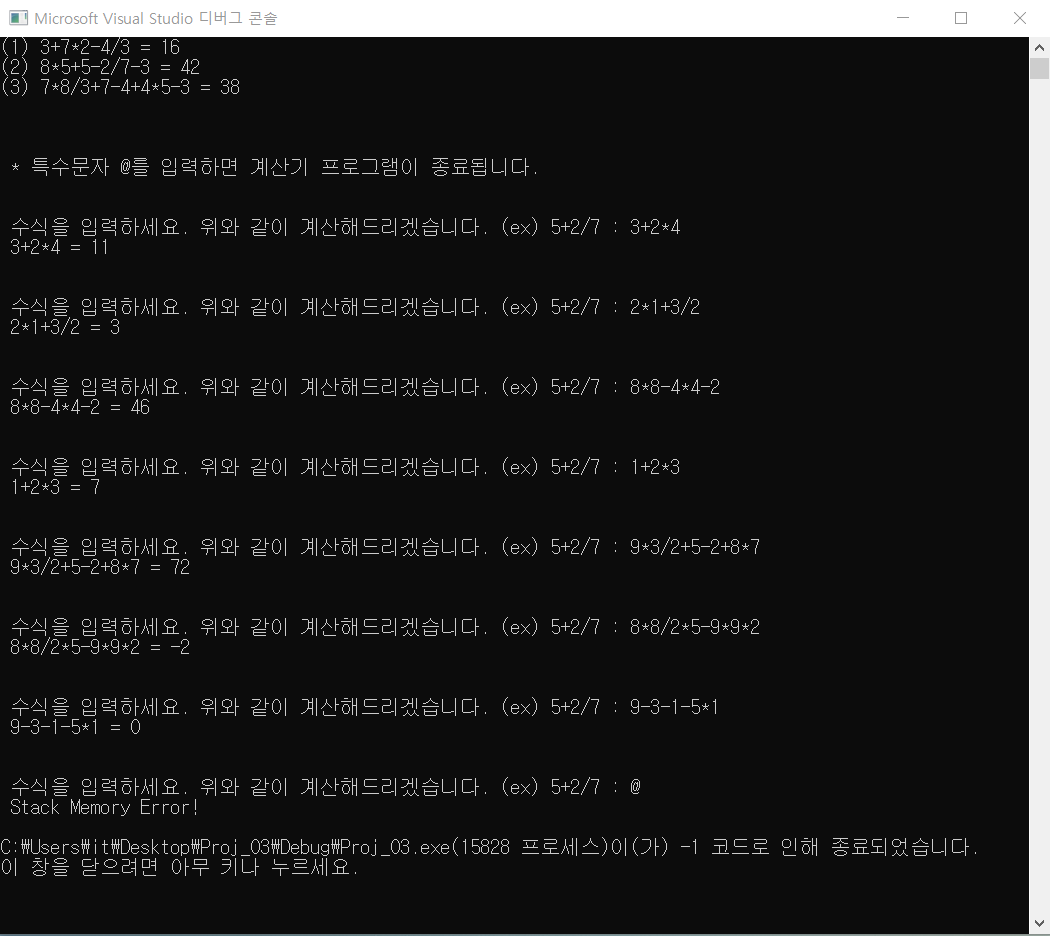
다음은 위의 스택 구현에 있어서 사용한 모델이다.



아래는 스택 헤더파일의 모습인데 노드 구조체를 하나 정의한다. 노드 하나 당, data값 하나와 다음 구조체 노드의 주소 값을 가진다. 다시 한 번 말해, data값(int) 하나와 다음 구조체 노드를 가리키는 주소 값을 가진다. 이 노드들을 포인터로 다음 노드를 가리키고 가리키고 가리키고하면서 연결시키면 위와 같은 스택 모델이 나오게 된다.



**(2) 실행 결과 화면 단계별 Capture (작은 size 화면으로 Capture)**



실행 결과 화면은 다음과 같다. 루프를 무한으로 돌림으로써 연속적인 계산기가 작동 된다. 계산기를 종료하고 싶으면 특수 기호 @를 입력하면 루프를 빠져나와 메인함수가 종료되면서 프로그램 역시 종료한다. 이 프로그램의 단점은 피연산자가 한 자리수 밖에 안된다는 점이다.

***!!!! 유의사항: 한 자리 숫자의 피연산자로만 계산 가능. 두 자리 이상의 피연산자는 받지 못하는 프로그램. 이외 문제에서 제시하는 모든 요건들은 충족시킴.***