HW8: infix evaluation, postfix to infix conversion

2012003567

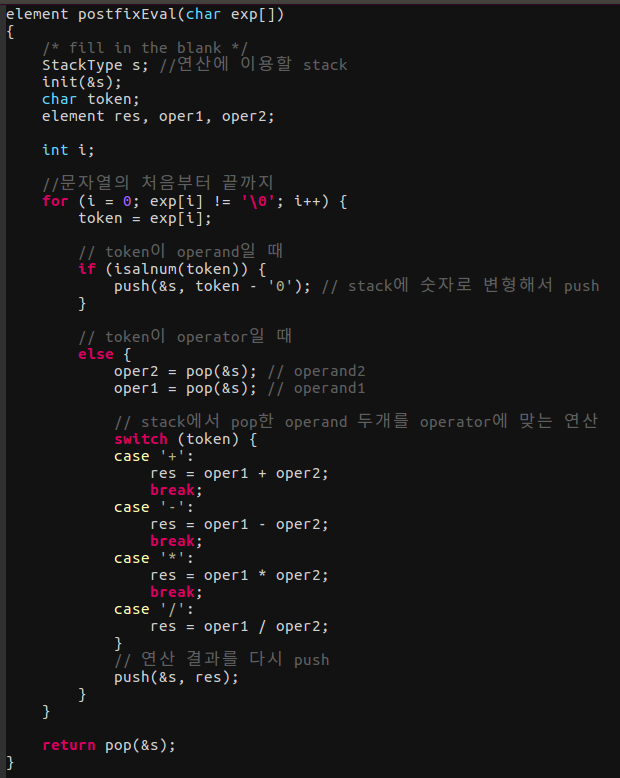
이상화

**1. 문제 해결의 주요 코드**



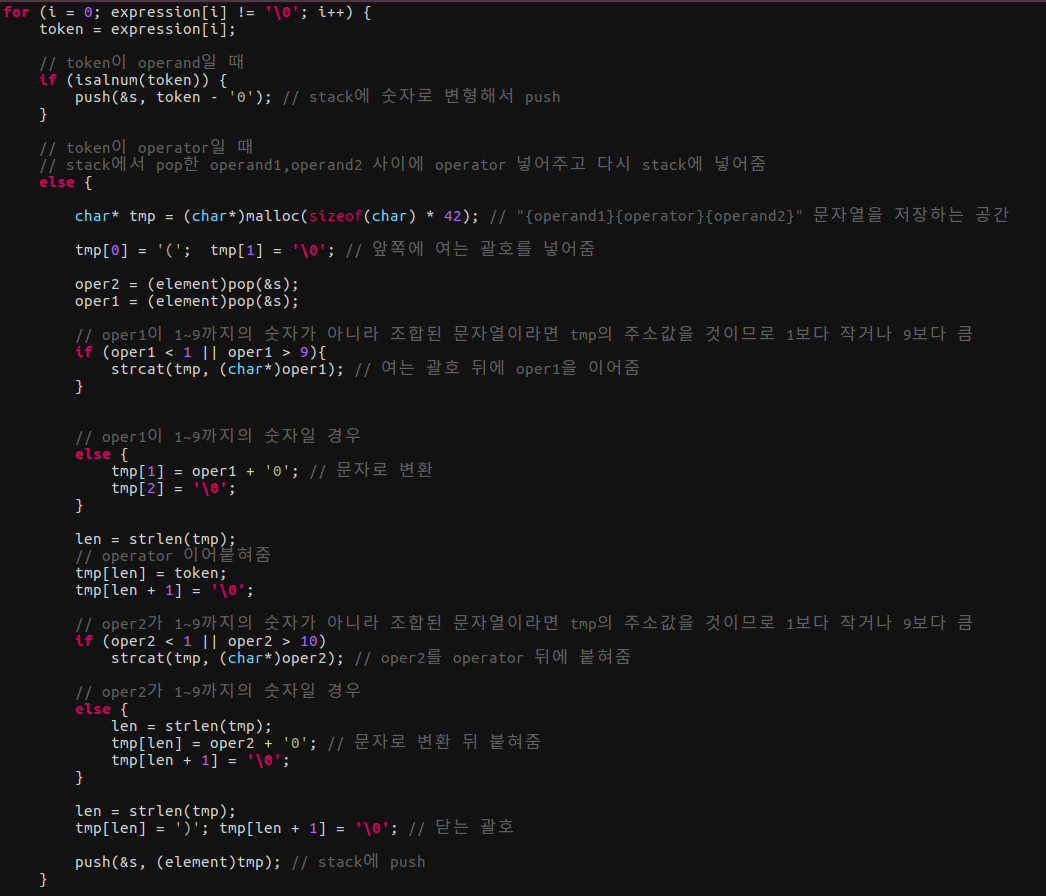
기존의 stack에 element 타입은 int였지만 long long으로 변경해주었다. 왜냐하면 postfix\_to\_infix 부분에서 char\*형 변수를 stack에 push해야하는 부분이 있는데 char\*는 64bit 머신에서 8바이트이기 때문에 stack의 element 타입을 int로 할 경우 type casting시에 오류가 발생한다. 그러므로 포인터와 크기가 같은 long형 으로 element 타입을 변경해주었다.

(1) element postfixEval(char exp[])



기본적인 알고리즘은 postfix 문자열을 앞에서부터 한 개씩 확인하면서 피연산자가 나오면 선언해놓은 stack에 push해주고 operator가 나오면 stack에서 2개를 pop해서 연산한 뒤 나온 결과를 push해주는 과정을 반복하는 것이다. isalnum 함수를 이용해 연산자인지 피연산자인지를 확인해서 피연산자이면 stack에 그대로 push해주는데 token이 문자이므로 –‘0’을 해줘서 숫자로 바꿔줘야 한다. token이 연산자인 경우에는 먼저 꺼낸 피연산자가 뒤로 가도록 연산을 해주면 된다.

(2) char\* postfix\_to\_infix(char expression[])



postfix를 infix로 바꾸는 알고리즘은 postfix 문자열을 앞에서부터 확인하면서 피연산자가 나오면 stack에 push하고 연산자가 나오면 두개를 pop해서 가운데에 연산자를 붙인 뒤에 {operand1}{operator}{operand2} 형태의 문자열을 만들어주고 다시 push해주는 것을 반복한다. int형 element를 저장하는 stack에 어떻게 문자열을 넣어야 할지 고민했는데 그냥 element를 포인터까지 저장할 수 있는 8바이트의 long long형으로 바꿔주고 문자열을 동적으로 할당한 뒤 주소값만 stack에 저장하였다. 위의 코드에서 token이 피연산자인 경우에는 그대로 push해준다. 연산자인 경우 일단 tmp라는 문자열을 동적으로 할당해준다. 괄호가 붙을 것을 생각하여 넉넉하게 42바이트로 잡아주었다. 그 뒤 여는 괄호를 맨 앞부분에 넣어준 뒤에 oper1과 oper2를 pop한다. 이 때 oper1과 oper2는 순수한 long long형 숫자일 수도 있고 char\*형 포인터 일수도 있다. 정확한 방법인지 확실치 않지만 그 구분을 위해서 1보다 작거나 9보다 크면 char\*형 포인터라고 가정했다. char\*형인 경우 여는 괄호만 들어가 있는 tmp에 strcat함수를 이용해서 oper1을 붙여준다. oper1이 숫자인 경우에는 +’0’을 해줘서 문자로 변환시킨 뒤에 여는 괄호 뒤에 붙여준다. 그 다음 연산자를 이어서 붙여준다. oper2도 oper1과 같은 방법으로 연산자 뒤에 부쳐준 뒤 닫는 괄호로 닫아주면 {operand1}{operator}{operand2}(eg,3+5) 형태의 문자열이 완성되었고 이것을 push해준다. 이 때 type casting에 신경을 써줘야 한다. visual studio에서 할시에는 이상이 없었지만 gcc에서 테스트 할 때 type관련 에러가 많이 발생했다.

**2. 실행결과**

