실습 5 – 수식 계산

- 실습 목표
  - 스택을 이용하여 postfix 수식을 계산할 수 있다.
  - infix 수식을 postfix 수식으로 변환할 수 있다.
  - 최종 목표: infix 수식을 입력받아 계산 결과를 출력한다.

postfix 수식 계산

- **(5.1.)** 프로그램 3.13과 3.14 구현 (eval.cpp)
  - 정수 데이터를 저장하는 stack 구현 포함
  - 수정 사항:
    - eval() 함수에서는 수식이 전역변수인 char expr[]에 저장되었다고 가정
    - eval() 함수를 호출할 때 expr[] 배열을 인자로 넘겨주는 것으로 수정
  - 함수원형:
    int eval(char* expr)
    precedence getToken(char* expr, char* symbol, int* n)

- main 함수에서 expr[] 배열에 postfix를 저장한 후, eval() 함수를 호출하고, 실행 결과(스택의 내용)를 관찰

infix를 postfix로 변경

- **(5.2.)** 프로그램 3.15를 구현 (postfix.cpp)
  - 역시 infix를 저장하는 expr[] 배열을 인자로 받도록 수정
  - postfix를 출력하지 말고, new_expr[] 배열에 저장하도록 수정. new_expr[] 배열도 역시 인자로 받도록 수정
  - 문자 데이터를 저장하는 stack 구현 포함
  - 함수원형
    void postfix(char* expr, char* new_expr)
    char printToken(precedence token)

main 함수(infix → postfix → eval)

- main 함수에서 infix를 입력받아 expr에 저장하고,
  - postfix(expr, new_expr);
  - eval(new_expr);

  을 차례대로 호출하여, 실행 결과를 확인 할 것!

  - (1) main 함수에서 expr[] 배열에 infix를 저장한 후 postfix(expr, new_expr) 함수를 호출하고, 실행을 마치면 new_expr을 출력하여 실행 결과를 관찰
    - Stack 내용이 변경되는 과정도 관찰할 것

  - (2) postfix 함수의 수행 결과인 new_expr을 이용하여 eval(new_expr) 함수를 호출하고, 실행을 마치면 연산 결과를 출력하여 결과를 관찰
    - Stack 내용이 변경되는 과정도 관찰할 것

응용

- 연산자의 확장
  - ^ (power 연산자: 2 ^ 5 = 2^5 = 32) ← right associative
  - ! (not 연산자: !2 = 0) ← 단항 연산자

- 피연산자의 확장
  - 여러 자리 숫자 인식
  - 음수 인식

- 입력 수식에서 공백 문자 처리