HW5: Infix 2 Postfix

학교서버: 203.249.75.14 / linux2.ce.hongik.ac.kr

submit pem_ta hw5a // 1분반

submit pem_ta hw5b // 2분반

submit pem_ta hw5c // 3분반

제출 확인: submit pem_ta hwNx -l // 본인 분반 submit 명령어에 마이너스 엘 추가

제출기한:

과제 부여 후 2주 후 24시 까지

목89(2분반) => 10월 20일 목요일 24시 (10월 21일 금요일 0시) 까지

금56, 금89(1, 3분반) => 10월 21일 금요일 24시 (10월 22일 토요일 0시) 까지

필수 제출 파일 (9개 +알파):

hw5.cpp, post.h, post.cpp, hw5.tex, hw5.pdf + 보고서에 첨부한 post.in/2.in/3.in/4.in의 실행결과 캡쳐 이미지 + 그 외 보고서에 첨부한 이미지 파일들

실행결과에 본인 학번이 같이 출력될 수 있도록 코드를 임의로 수정

latex 내용:

1. stack에 관한 설명, 작성한 코드에 관한 설명
2. 무슨 작업을 한 건지 알아볼 수 있게 latex에 작성부분 코드 첨부(₩verbatim 패키지 추천)
3. 최대한 자세하게 작성.

실습 시작 전 만들어야 하는 파일들:

makefile, post.in/2.in/3.in/4.in, hw5.cpp, post.h, **post.cpp**

1(a) 다음과 같은 **makefile** 을 작성하라.

명령어 : cat makefile / vi makefile

```
hw5:    hw5.o post.o
        g++ -o hw5 hw5.o post.o
```

(b) infix notation으로 된 표현식들로 이루어진 각 파일별로 4가지 테스트 케이스들을 만든다

(※ 참고 : 입력된 파일 확인 명령어 : cat, 파일 입력 명령어 : vim)

명령어: 파일 생성 및 편집 명령어: vi post.in 파일 입력 확인(출력) 명령어: cat post.in

post.in

A * B * C

-A + B – C + D

A * -B + C

(A + B) * D + E / (F + A * D) + C

post2.in

A && B || C || !(E>F)

!(A&&!((B<C)||(C>D)))||(C<E)

post3.in

34*56+11/2

1+2*3+-4*2

post4.in

33+55*2

an77=2+7*5

b=2

an77+b*2

a+5

2. 다음 같이 동작하는 프로그램을 작성하려 한다.

make hw5

hw5 < post.in

A B * C *

A -u B + C - D +

A B -u * C +

A B + D * E F A D * + / + C +

(a) main 프로그램(hw5.cpp)은 다음과 같다. 아래와 같이 작성 후 빈칸을 채우시오.

※ main 프로그램 안의 구성 변경 가능

※ 주어진 함수는 적절하게 변형 가능

```
#include <iostream>
#include "post.h"
using namespace std;
```

```
void Postfix(Expression);
int main() {
	char line[MAXLEN];
	while (/* 어떤 함수를 써야할지 고민 */(line, MAXLEN)) {
		Expression e(line); // line 버퍼를 이용하여 Expression을 읽음
		try {
			Postfix(e);
		}
		catch (char const *msg) {
			cerr << "Exception: " << msg << endl;
		}
	}
}
```

(b) 다음 프로그램을 post.h로 저장하라.

```
#ifndef POSTFIX_H
#define POSTFIX_H
// token types for non one-char tokens
#define ID 257
#define NUM 258
#define EQ 259
#define NE 260
#define GE 261
#define LE 262
#define AND 263
#define OR 264
```

```
#define UMINUS 265
#define MAXLEN 80
struct Expression {
	Expression(char* s) : str(s), pos(0)
		{ for (len = 0; str[len] != '₩0'; len++); }
	char* str;
	int pos;
	int len;
};
struct Token {
	bool operator==(char);
	bool operator!=(char);
	Token();
	Token(char);		// 1-char token: type equals the token itself
	Token(char, char, int);		// 2-char token(e.g. <=) & its type(e.g.LE)
	Token(char*, int, int);		//operand with its length & type(defaulted to ID)
	bool IsOperand();		// true if the token type is ID or NUM
	int type;		// ascii code for 1-char op; predefined for other tokens
	char* str;		// token value
	int len;		// length of str
	int ival;		// used to store an integer for type NUM; init to 0 for ID
};
using namespace std;
ostream& operator<<(ostream&, Token t);
Token NextToken(Expression&, bool); // 2nd arg = true for infix expression
#endif
```

(c) 다음 post.cpp을 완성하라.

```
#include <iostream>
#include <stack>
#include "post.h"
using namespace std;

bool Token::operator==(char b) { return len == 1 && str[0] == b; }
bool Token::operator!=(char b) { return len != 1 || str[0] != b; }
Token::Token() {}
Token::Token(char c) : len(1), type(c)
        { str = new char[1]; str[0] = c; }                  // default type = c itself
Token::Token(char c, char c2, int ty) : len(2), type(ty)
        { str = new char[2]; str[0] = c; str[1] = c2; }
Token::Token(char* arr, int l, int ty = ID) : len(l), type(ty){
        str = new char[len + 1];
        for (int i = 0; i < len; i++) str[i] = arr[i];
        str[len] = '₩0';
        if (type == NUM) {
                ival = arr[0] - '0';
                for (int i = 1; i < len; i++) ival = ival * 10 + arr[i] - '0';
        } else if (type == ID) ival = 0;
        else throw "must be ID or NUM";
}
bool Token::IsOperand() {  return type == ID || type == NUM; }
```

```
ostream& operator<<(ostream& os, Token t) {
        if (t.type == UMINUS) os << "-u";
        else if (t.type == NUM) os << t.ival;
        else for (int i = 0; i < t.len; i++) os << t.str[i];
        os << " ";
        return os;
}
bool GetID(Expression& e, Token& tok) {
        char arr[MAXLEN]; int idlen = 0;
        char c = e.str[e.pos];
        if (!(c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')) return false;
        arr[idlen++] = c;
        e.pos++;
        while ( (c = e.str[e.pos]) >= 'a' && c <= 'z'
                || c >= 'A' && c <= 'Z'
                || c >= '0' && c <= '9') { arr[idlen++] = c; e.pos++; }
        arr[idlen] = '₩0';
        tok = Token(arr, idlen, ID);            // return an ID
        return true;
}
bool GetInt(Expression& e, Token& tok)
{
        // 이 부분을 작성하세요
}
```

```
void SkipBlanks(Expression& e) {
        char c;
        while (e.pos < e.len && ((c = e.str[e.pos]) == ' ' || c == '₩t')){
                e.pos++;
        }
}
bool TwoCharOp(Expression& e, Token& tok) {
        // 7가지 두글자 토큰들 <= >= == != && || -u을 처리
        char c = e.str[e.pos]; char c2 = e.str[e.pos + 1];
        int op;                         // LE GE EQ NE AND OR UMINUS
        if (c == '<' && c2 == '=')  op = LE;
        else if    // 이 부분 작성: 각 두 글자 토큰에 대해 알맞은 type 값 op에 저장
        // 코드 작성
        else { return false; }          // 맞는 두 글자 토큰이 아니면 false를 return
        tok = Token(c, c2, op); e.pos += 2;
        return true;
}
bool OneCharOp(Expression& e, Token& tok) {
        char c = e.str[e.pos];
        if (c == '-' || c == '!' || c == '*' || c == '/' || c == '%' ||
                c == '+' || c == '<' || c == '>' || c == '(' || c == ')' || c == '=') {
                tok = Token(c); e.pos++; return true;
        }
        return false;
}
```

```
Token NextToken(Expression& e, bool INFIX = true) {
        static bool oprrFound = true;                   // 종전에 연산자 발견되었다고 가정.
        Token tok;
        SkipBlanks(e);      // skip blanks if any
        if (e.pos == e.len) { // No more token left in this expression
                if (INFIX) oprrFound = true; return Token('#'); }
        if (GetID(e, tok) || GetInt(e, tok)) {
                if (INFIX) oprrFound = false; return tok; }
        If (TwoCharOp(e, tok) || OneCharOp(e, tok)) {
                if (tok.type == '-' && INFIX && oprrFound)          // operator 후 -발견
                tok = Token('-', 'u', UMINUS);                  // unary minus(-u)로 바꾸시오
                if (INFIX) oprrFound = true; return tok;
        }
    throw "Illegal Character Found";
}
int icp(Token& t) {          // in-coming priority
        int ty = t.type;
        // 이 부분 작성
                // ty가 '('면 0,
                //UMINUS나 '!'면 1,
                //'*'나 '/'나 '%'면 2,
                //'+'나 '-'면 3,
                //'<'나 '>'나 LE나 GE면 4,
                //EQ나 NE면 5,
                //AND면 6,
                //OR이면 7,
```

```
                    //'='이면 8,
                    //'#'면 9 를 return한다.
}
int isp(Token& t)              // in-stack priority
{
        int ty = t.type;    //stack 에서의 우선순위 결정
        // 이 부분 작성
}

void Postfix(Expression e)
{
        // infix expression e를 postfix form으로 바꾸어 출력
        // e에 토큰이 없으면 NextToken은 '#' 토큰을 반환한다.
        // 스택의 밑에도 '#'를 넣고 시작한다.

        // HINT : STL stack이용하고, 교재의 마지막 for문을 아래와 같이 바꾼다
        // while (stack.top()!='#') { cout << stack.top(); stack.pop(); }
        // stack.pop()
}
```