2020년 자료구조 Postfix 실습

마감 일자

1분반: 2020년 11월 9일 저녁(밤) 12시

2분반: 2020년 11월 13일 저녁(밤) 12시

제출 방법

1분반 : submit pem_ta hw6a

2분반 : submit pem_ta hw6b

제출 파일 : hw6.cpp , post.h , post.cpp , hw6.tex , hw6.pdf

다음과 같이 infix notation 을 postfix notation으로 변환하는 프로그램을 만들고자 한다.

```
[B611047@localhost ta]$ hw6 <post.in
A B * C *
A -u B + C - D +
A B -u * C +
A B + D * E F A D * + / + C +
[B611047@localhost ta]$ hw6 < post.in2
A B && C || E F > ! ||
A B C < C D > || ! && ! C E < ||
```

1. makefile 다음과 같이 작성

```
[B611047@localhost hw6]$ cat makefile
hw6: hw6.o post.o
  g++ -o hw6 hw6.o post.o
hw6.o post.o: post.h
```

2. infix notation으로 된 즉 우리가 평소에 사용하는 표현식으로 이루어진 4가지 테스트 케이스를 만든다

```
[B611047@localhost hw6]$ cat post.in
A * B * C
-A + B - C + D
A * -B + C
(A + B) * D + E / (F + A * D) + C
[B611047@localhost hw6]$ cat post.in2
A && B||C||!(E>F)
!(A&&!((B<C)||(C>D)))||(C<E)
[B611047@localhost hw6]$ cat post.in3
34 * 56 + 11 / 2
1 + 2 * 3 + - 4 * 2
[B611047@localhost hw6]$ cat post.in4
33 + 55 * 2
an77 = 2 + 7 * 5
b = 2
an77 + b * 2
a + 5
```

3. main 파일 (hw6.cpp) 는 이렇게 이루어져 있다.

```
#include <iostream>
#include "post.h"
using namespace std;
void Postfix(Expression);
int main() {
  char line[MAXLEN];
  while (cin.getline(line, MAXLEN)) {
    Expression e(line); // line 버퍼를 이용하여 Expression을 읽음
    try {
      Postfix(e);
    }
    catch (char const *msg) {
      cerr << "Exception: " << msg << endl;
    }
  }
}
```

4. post.h 파일

```
#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);
Token NextToken(Expression&, bool); // 2nd arg=true for infix expression
#endif
```

(UMINUS → unary minus 음수표현 여기선 -u 로 바꿔 표현함)

5. 작성해야할 post.cpp 파일

icp , isp , postfix , getint

```
#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)
{
  // 이부분을 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 // .각 두글자 토큰에 대해 알맞은 type값을 op에 저장
  else return false; // 맞는 두글자 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) {
  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
  {
    return Token('#');
  }
  if (GetID(e, tok) || GetInt(e, tok))
  {
    return tok;
  }
  if (TwoCharOp(e, tok) || OneCharOp(e, tok)) {
    if (tok.type == '-' && e.str[e.pos-2]=='-') //operator후 -발견
      tok = Token('-', 'u', UMINUS); // unary minus(-u)로바꾸시오
    return tok;
  }
  throw "Illegal Character Found";
}
int icp(Token& t) { // in-coming priority
  int ty = t.type;
  /* 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
}
void Postfix(Expression e)
{
  // infix expression e를 postfix form으로 바꾸어 출력
  // e에 토큰이 없으면 NextToken은 '#' 토큰을 반환한다.
  // 스택의 밑에도 '#'를 넣고 시작한다.
```

```
}
```

함수를 추가해서는 안되지만 주어진 함수들은 변형하고 코드는 자유롭게 짜도 됩니다.

post.h 나 post.cpp의 주어진 코드들을 보며 이해하면서 코딩하면 조금 더 수월할 수 있습니다.

혹시 오타가 있을 수도 있습니다.

제출 시 실행파일 (ex hw6.o, post.o) 같은 파일들은 지우고 제출합니다.

latex 에서는 infix notation을 어떻게 postfix로 stack을 이용해서 변환하였는지 간단한 그림이나 글로 설명 해주시고 다른 부분은 자유입니다.