자료구조 HW5

B711222 박조은
Hongik University
mrnglory@mail.hongik.ac.kr

November 1, 2019

I. List of Source Files

- hw5
  - hw5.cpp
  - post.cpp
  - post.h

II. hw5

i. hw5.cpp

```
#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
            ↪ Expression 을 만듦

        try {
            PostFix(e);
        } catch (char const *msg) {
            cerr << "Exception: " << msg << endl;
        }
    }
}
```

ii. post.cpp

```
#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
```

```
52              throw "must be ID or NUM";
53      }
54
55      bool Token::IsOperand()
56      {
57          return type == ID || type == NUM;
58      }
59
60      ostream& operator << (ostream& os, Token t)
61      {
62          if (t.type == UMINUS)
63              os << "-u";
64
65          else if (t.type == NUM)
66              os << t.ival;
67
68          else
69              for (int i = 0; i < t.len; i++)
70                  os << t.str[i];
71
72          os << " ";
73
74          return ps;
75      }
76
77      bool GetID(Expression& e, Token& tok)
78      {
79          char arr[MAXLEN];
80          int idlen = 0;
81          char c = e.str[e.pos];
82
83          if (!(c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z'))
84              return false;
85
86          arr[idlen++] = c;
87          e.pos++;
88
89          while ((c = e.str[e.pos]) >= 'a' && c <= 'z'
90              || c >= 'A' && c <= 'Z'
91              || c >= '0' && c <= '9')
92          {
93              arr[idlen++] = c;
94              e.pos++;
95          }
96
97          arr[idlen] = '\0';
98          tok = Token(arr, idlen, ID); // return an ID
99
100         return true;
101     }
102
103     bool GetInt (Expression& e, Token& tok)
104     {
105         char arr[MAXLEN];
106         int len = 0;
107         char c = e.str[e.pos];
108
109         if (!(c >= '0' && c <= '9'))
110             return false;
111
112         arr[len++] = c;
113         e.pos++;
114
115         while ((c = e.str[e.pos]) >= '0' && c <= '9')
116         {
117             arr[len++] = c;
118             e.pos++;
119         }
120
121         arr[len] = '\0';
122         tok = Token(arr, len);
123
124         return true;
125     }
126
127     void SkipBlanks(Expression& e)
128     {
129         char c;
130
131         while (e.pos < e.len && ((c = e.str[e.pos]) == ' ' || c
                ↪ == '\t'))
132             e.pos++;
133     }
134
135     bool TwoCharOp(Expression& e, Token& tok)
136     {
137         // 7가지 두 글자 토큰들 // <= >= == != && || -u
                ↪ 을 처리
138         char c = e.str[e.pos];
139         char c2 = e.str[e.pos + 1];
140         int op; // LE GE EQ NE AND OR UMINUS
141
142         if (c == '<' && c2 == '=')
143             op = LE;
144
145         else if
146
147         else
148             return false; // 맞는 두 글자 토큰이 아니면 false
                    ↪ 를 return
149
150         tok = Token (c, c2, op);
151         e.pos += 2;
152
153         return true;
154     }
155
156     bool OneCharOp(Expression& e, Token& tok)
157     {
158         char c = e.str[e.pos];
159
160         if (c == '-' || c == '!' || c == '*' || c == '/' || c ==
```

```
            ↪ '%' ||
161         c == '+' || c == '<' || c == '>' || c == '(' || c
                ↪ == ')' || c == '=')
162     {
163         tok = Token(c);
164         e.pos++;
165
166         return true;
167     }
168
169     return false;
170 }
171
172 Token NextToken(Expression& e, bool INFIX = true)
173 {
174     static bool oprrFound = true; // 종전에 연산자
            ↪ 발견되었다고 가정
175     Token tok;
176     SkipBlanks(e); // skip blanks if any
177
178     if (e.pos == e.len) // No more token left in this
            ↪ expression
179     {
180         if (INFIX)
181             oprrFound = true;
182
183         return Token('#');
184     }
185
186     if (GetID(e, tok) || GetINT(e, tok))
187     {
188         if (INFIX)
189             oprrFound = false;
190
191         return tok;
192     }
193
194     if (TwoCharOp(e, tok) || OneCharOp(e, tok))
195     {
196         if (tok.type == '-' && INFIX && oprrFound) //
                ↪ operator 후 - 발견
197             tok = Token('-', 'u', UMINUS); // unary
                    ↪ minus (-u)로 바꾸시오
198
199         if (INFIX)
200             oprrFound = true;
201
202         return tok;
203     }
204
205     throw "Illegal Character Found";
206 }
207
208 int icp(Token& t)
209 {
210     // in-coming priority
211
212     int ty = t.type;
213
214     if (t.type == '(')
215         return 0;
216
217     else if (t.type == UMINUS || t.type == '!')
218         return 1;
219
220     else if (t.type == '*' || t.type == '/' || t.type == '%')
221         return 2;
222
223     else if (t.type == '+' || t.type == '-')
224         return 3;
225
226     else if (t.type == '<' || t.type == '>' || t.type == LE
            ↪ || t.type == GE)
227         return 4;
228
229     else if (t.type == EQ || t.type == NE)
230         return 5;
231
232     else if (t.type == AND)
233         return 6;
234
235     else if (t.type === OR)
236         return 7;
237
238     else if (t.type == '=')
239         return 8;
240
241     else if (t.type == '#')
242         return 9;
243 }
244
245 int isp(Token& t)
246 {
247     // in-stack priority
248
249     if (t.type == '(')
250         return 10;
251
252     else
253         return icp(t);
254 }
255
256 void PostFix(Expression e)
257 {
258     // HINT: STL stack 이용하고, 교재의 마지막 for
            ↪ 문으로
259     // while (stack.top() != '#') { cout << stack.top();
            ↪ stack.pop();}
260     // stack.pop();
261
262     stack<Token> stack;
263     stack.push('#');
```

```
264
265     for (Token x = NextToken(e); x != '#'; x = NextToken(
            ↪ e))
266     {
267         if (x.IsOperand())
268             cout << x;
269
270         else if (x == ')')
271         {
272             for (; stack.top() != '('; stack.pop())
273                 cout << stack.top();
274             stack.pop();
275         }
276
277         else
278         {
279             for (; isp(stack.top()) <= icp(x); stack.pop())
280                 cout << stack.top();
281             stack.push(x);
282         }
283
284         while (stack.top() != '#')
285         {
286             cout << stack.top();
287             stack.pop();
288         }
289
290         cout << endl;
291     }
292 }
```

iii. post.h

```
1  #ifndef POSTFIX_H
2  #define POSTFIX_H
3  // token types for non one-char tokens
4  #define ID 257
5  #define NUM 258
6  #define EQ 259
7  #define NE 260
8  #define GE 261
9  #define LE 262
10 #define AND 263
11 #define OR 264
12 #define UMINUS 265
13 #define MAXLEN 80
14
15 struct Expression
16 {
17     Expression (char* s): str(s), pos(0)
18     {for (len = 0; str[len] != '\0'; len++);}
19
20     char * str;
21     int pos;
22     int len;
23 };
24
25 struct Token
26 {
27     bool operator == (char);
28     bool operator != (char);
29
30     Token();
31
32     Token(char);  // 1-char token: type equals the token
            ↪ itself
33     Token(char, char, int); // 2-char token (e.g. <=) &
            ↪ its type (e.g. LE)
34     Token(char*, int, int); // operand with its length &
            ↪ type (defaulted to ID)
35     bool IsOperand(); // true if the token type is ID or
            ↪ NUM
36     int type;  // ascii code for 1-char op; predefined for
            ↪ other tokens
37     char *str;  //token value
38     int len;  // length of str
39     int ival;  // used to store an integer for type NUM;
            ↪ init to 0 for ID
40 };
41
42 using namespace std;
43 ostream& operator << (ostream&, Token);
44 osteram& operator << (ostream&, Expression);
45 Token NextToken(Expression&, bool); // 2nd arg = true
       ↪ for infix expression
46 void PostFix(Expression e);
47
48 #endif
```

```
/*
 * 갓직히 왜 컴파일이 안되는지 납득이 안됨.

 */
```