HW4 Infix/Postfix/Prefix Transformation

A102074 顏佑霖

Design algorithms for the following transformation problems:

1. **infix ==> postfix;**
2. **infix ==> prefix;**
3. **postfix ==> prefix;**
4. **prefix ==> postfix;**
5. **postfix ==> infix;**

Algorithm 1: infix ⇒ postfix

Input: Infix e

Output: Postfix of e

```
n = parsing (e, token) ;                    //token: 把一詞/一組數字變成1 token
push("#");                                  //先將一#放進去堆疊
for (i=0; i<n; i++)
{
  s = token[i];
  if (s ∈ Operand) output(s);
  else if (s == ")")                        //右括號的目的為尋找左括號
    while ((x=pop())!="(") output(x);       //將stack中第一個(之前的運算子皆POP出並印出
  else
  {
    while ( p(s) <= q(Stack[top]))          // p() 堆疊外 q()堆疊內
    {
       x = pop();                           //較大的離開 pop出去
       output(x);
    }
    push(s);                                //因為裡面的都比它小了 所以push進去
```

```
  }

while (Stack[top] != "#")
{
  x = pop();
  output(x);
}
x = pop();                                        //最後要把#pop出去 因為是事先加的
}
```

Algorithm 2: infix ⇒ prefix

Input: Infix e

Output: Prefix of e

```
n = parsing(e, token);                          //token: 把一詞/一組數字變成1 token
push("#");                                      //先將一#放進去堆疊
for (i=0;i<n;i++)
{
  s = token(i);
  if (s ∈ Operand) postfix += s;                //直接在stack push_opn(s)
  else if (s ==")")
    while ((x=pop()) != "(")
      postfix += s;
  else
  {
    while (p(s) <= q(Stack[top]))               //String get_prefix{String a = pop_opn();
    {                                           //            return x+pop_opn() + a;}
      x = pop(s);
      postfix += s;
    }
    push(s);
  }
}
while (x = pop() != "#") postfix += s;          //push_opn(get_prefix(x))
return pop_opn();
```

Algorithm 3: postfix ⇒ prefix

Input: Postfix e

Output: Prefix of e

```
//利用flag讓get_fix知道要往prefix還是postfix
String get_fix(String x, int flag)
{
String a = pop_opn();
String b = pop_opn();
a = (flag == 1) ? x+a+b : b+a+x;  //flag = 1 --> prefix 不是1-->postfix
return a
}


n = parsing(e, token);                        //token: 把一詞/一組數字變成1 token
for (i=0;i<n;i++)
{
  s = token(i);
  if (s ∈ Operand) push_opn(s);
  else push_opn(get_fix(s, 1);
}
return pop_opn();
```

Algorithm 4: prefix ⇒ postfix

Input: Prefix e

Output: Postfix of e

```
n = parsing(e, token);                    //token: 把一詞/一組數字變成1 token
for (i=0;i<n;i++)
{
  s = token(i);
  if (s ∈ Operand) push_opn(s);
  {
    while (Stack[top] ∈ Operand)          //若Stack頂端同時是operand, s必須要和它結合
    {
      y = pop();                          //pop出前一個operand
      x = pop();                          //pop出動硬的operator
      s = y+s+x;                          //operator x 所對應之 postfix
    }
    push(s);                              //push所得到的operand
  }
  else push(s);                           //push operator s
}
return pop();
```

Algorithm 5: postfix ⇒ infix

Input: Postfix e

Output: Infix of e

```
n = parsing(e, token);                    //token: 把一詞/一組數字變成1 token
for (i=0;i<n;i++)
{
  s = token(i);
  if (s ∈ Operand) push_opn(s);           //如果operand有找到 則push進去
  else                                    //若沒找到 則把operand pop出去
  {
    s1 = pop(1);  s2 = pop(1);            //stack2:Operands
    x  = pop(2);  y  = pop(2);            //stack1:Operators(corresponding operands in Stack2)
    if (p(s)>p(s1)) x ="("+x+")";
    if (p(s)>p(s2)) y ="("+y+")";
    push(2, y+s+x);
    push(1, s);
  }
    // 最後應該只剩一元素在stack中 且該元素符合infix的排序
}
while(top!=-1) x=pop(1);
return pop(2);
```