110705067 洪薇欣

Hw4. Infix/Postfix/Prefix Transformation

1. Algorithm for infix to postfix:

```
Input: Infix e
Output: Postfix of e

n=parsing(e,token);   //把字串分割，字串總長度為 n
push("#");   //讓 operator 皆可進
postfix ="";
for(i=0;i<n;i++)
{
        s=token[i];
        if (s is operand)
              postfix += s;
        else if (s== ")" )     //遇到 ")" ，將 "(" 前的東西全部輸出
        {
              while( (x=pop()) != "(" )
                    prefix += x;
        }
        else
        {
              while( p(s) <= q(Stack[top])   //比 operator 的優先順序大小
              {
                    x = pop();     //堆疊內元素比外的元素>=皆要 pop 出
                    prefix += x;
              }
              push(s);
        }
}
while (Stack[top] != "#")   //stack 內還有東西就將所有東西 pop 出來
{
        x = pop();
        postfix += x;
}
x = pop();
```

2. Algorithm for infix to prefix:

```
Input: Infix e
Output: Prefix of e

n=parsing(e,token);
push(“#”);
for(i=0;i<n;i++)
{
        s=token[i];
        if (s is operand)
              push_opn(s);    //push 至 operand 的 stack
        else if (s== ”)” )
        {
              while( (x=pop()) != “(” )
                    push_opn(get_prefix());
        }
        else
        {
              while( p(s) <= q(Stack[top])
              {
                    x = pop();
                    push_opn(get_prefix());
                    /*
                          string get_prefix(x)
                           {
                                string a = pop_opn();    //從 operand 的 stack pop 出
                                return x+pop_opn()+a;
                           }
                    */
              }
              push(s);
        }
 }
 while (x = pop_opn()    != “#”)
        push_opn(get_prefix());
 return pop_opn();
```

3. Algorithm for postfix to prefix:

Input: Postfix e
Output: Prefix of e

```
n = parsing(e, token);
for (i=0; i<n; i++)
{
        s = token[i];
        if ( s is operand)
                push_opn(s);    //push 至 operand 的 stack
        else
                push_opn( get_fix (s,1) );
        /*
                string get_fix (string x, int flag)    //用 flag 表示要用前序或後序
                {
                        string a = pop_opn();
                        string b = pop_opn();
                        a = (flag == 1 ) ? x+b+a : b+a+x;
                        // flag=1 做前序（x+b+a），不是就做後序（b+a+x）
                        return a;
                }
        */
}
return pop_opn();    //從 operand 的 stack pop 出
```

4. Algorithm for prefix to postfix:

Input: Prefix e
Output: Postfix of e

```
n = parsing(e, token);
for (i=0; i<n; i++)
{
        s = token[i];
        if ( s is operand)
        {
```

```
            while( Stack[top] is operand)
            {
                y = pop();   //pop 出前一個 operand
                x = pop();   //pop 出前一個 operator
                s = y+s+x;
            }  //如果結合完後 Stack[top]仍是 operand，還要再結合
            push(s);   //將已結合的 operand 丟回 Stack
        }
        else
            push(s);   //將 operator push 進 stack
}
return pop();
```

5. Algorithm for postfix to infix:

```
Input: Postfix e
Output: Infix of e

n = parsing(e, token);
for (i=0; i<n; i++)
{
        s = token[i];
        if ( s is operand)
        {
            push(2, s);
            push(1, "@");   //每個未結合的 operand  給一對應 operator @
                            // ( @有最大的優先順序，不用加括號的)
        }
        else
        {
            s1 = pop(1);
            //從 stack 1 (放 operand 的 stack)將第一個 operand pop 出
            s2 = pop(1);
            //從 stack 1 (放 operand 的 stack)將第二個 operand pop 出
            //（運算需要兩個 operand）
            x = pop(2);
            //從 stack 2 (放對應的 operator 的 stack)將 s1 的 operator pop 出
```

```
            y = pop(2);
            //從 stack 2 (放對應的 operator 的 stack)將 s2 的 operator pop 出
            if ( p(s) > p(s1) )
                x = "(" + x + ")" ;
                //如果要結合的 operator 比已結合的 operator 優先順序大，
                  以結合的 operand 要加()。
                //Ex. operand:A+B, operator: *. * > + , A+B 要加括號，變(A+B)
            if ( p(s) > p(s2) )
                y = "(" + y+ ")" ;
            push (2, y+s+x );   //結合後再放入 operand 的 stack
            push (1, s);
            //將結合後的 operand 所對應的 operator 放入 operator 的 stack
        }
}
while ( top != -1)
        x = pop(1);   //將 operator 的 stack 清空
return pop(2);
```