post_pre():

時間複雜度:O(n), n 為字串總長度

空間複雜度:O(3n) = O(n), n 為字串總長度

程式碼:

```
for(i=n-1;i>=0;i--)//scan form back to front: str
   {
     if(str_postpre[i]=='+'||str_postpre[i]=='-'||

str_postpre[i]=='*'||str_postpre[i]=='/'||str_postpre[i]=
='^')
     {
        push(str_postpre[i]);//store operator
     }
     else
     {
       c[j++]=str_postpre[i];//store numbers
       while((top!=-1)&&(stack[top]=='@'))//tag the top
       {
          a=pop(); //get rid of '@'
          c[j++]=pop();
       }
       push('@');
     }
   }
   c[j]='\0';//last entry of c

   for(int k=strlen(c)-1;k>=0;k--)
     printf("%c",c[k]);
}
```

---

1. 由後往前 scan 字串，遇到 operator 及放進堆疊 :

2. 若為數字，放進另一個字串

3. 當堆疊非空時，將堆疊內的 operator 放進字串

4. 最後反過來 print

Time complexity:
Step 1 + Step 2 = O(n)
Step 3 = O(n)
Step 4 = O(n)
Step 1+2+3+4 =O(n)

Space complexity:

輸入字串的空間+堆疊大小+存放新字串的空間= O(n)

pre_post():

程式碼:

時間複雜度: O(n), n 為字串總長度

空間複雜度: O(n), n 為字串總長度

```
for(i=0;i<n;i++)
   {
     if(str_prepost[i]=='+'||str_prepost[i]=='-'||

str_prepost[i]=='*'||str_prepost[i]=='/'||str_prepost[i]=
=='^')
     {
        push(str_prepost[i]);
     }
     else
     {
       c[j++]=str_prepost[i];
```

```
        while((top!=-1)&&(stack[top]=='@'))
        {
            a=pop();
            c[j++]=pop();
        }
        push('@');
      }
   }
   c[j]='\0';
   printf("%s",c);
}
```

---

1. 由前往後 scan 字串，遇到 operator 及放進堆疊

2. 若為數字，放進另一個字串。

3. 當堆疊非空時，將堆疊內的 operator 放進字串

4. 依序 print

Time complexity:
Step 1 + Step 2 = O(n)
Step 3 = O(n)
Step 4 = O(n)
Step 1+2+3+4 =O(n)

Space complexity:

輸入字串的空間+堆疊大小+存放新字串的空間= O(n)