資料結構
Data Structure

DS_Assignment01_Group#4

Assignment #02

Name & Student ID

劉千榮 110310138
朱丞鈺 110310213
蔡宇軒 110310218

Q1. Evaluating an Arithmetic Expression

In this problem, you have to implement a calculator like what has been taught in the classes. The calculator should support the following operator on double-precision floating points with correct precedence.

Input

The input contains *T* lines (multiple lines), each representing an arithmetic expression.

Output

For each test case print a double-precisionfloating point number in one line, which indicatesthe answer of the expression. Your solution will be considered correct if the absolute or relativeerror between the answer(a)and your output(b)is less than 10^{-8},i.e., $\frac{|a-b|}{\max(1,|b|)} \leq 10^{-8}$.

Constraints

$0<$the length of each line $L<10^6$

$0<a_i<10^8$ for each number a_i in the expression

$L \cdot T \leq 10^6$

Every number in the input will be an integer containing only of digits (no decimal points). We expect the final output to be a floating-point number, though.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define dot ','

// 定義建立堆疊與串列所需的結構 (Infix 轉 Postfix 用)
typedef struct StackNode
{
    // 儲存讀入字元
    char value;
    // 判斷為數字或運算元
    int chardetect;
    // 儲存下一節點的指標
    struct StackNode *nextPtr;

} stacknode;

// 重新定義結構之一階指標的名稱
typedef struct StackNode *stackPtr;

// 插入串列尾端
void insertlist(stackPtr *nodePtr, char info, int mode);
// 輸出串列第一的元素
char poplist(stackPtr *topPtr);
// 輸入資料進入堆疊
void push(stackPtr *topPtr, char info);
// 輸出堆疊中最上層的元素
char pop(stackPtr *topPtr);
// 判斷堆疊是否為空
int isEmpty(stackPtr topPtr);
// 顯示串列的所有元素
```

```
32. void printlist(stackPtr currentPtr, int mode);
33. // 優先權判斷
34. int priority(char op);
35. 
36. // 定義建立堆疊所需的結構 (計算 Postfix 用)
37. typedef struct CalculateNode
38. {
39.     // 儲存待運算數值
40.     double value;
41.     // 儲存下一節點的指標
42.     struct CalculateNode *nextPtr;
43. 
44. } calculateNode;
45. 
46. // 重新定義結構之一階指標的名稱
47. typedef struct CalculateNode *calculatePtr;
48. 
49. // 輸入資料進入堆疊
50. void pushDouble(calculatePtr *topPtr, double info);
51. // 輸出堆疊中最上層的元素
52. double popDouble(calculatePtr *topPtr);
53. // 對堆疊中數字進行計算
54. void calculate(calculatePtr *topPtr, char sign);
55. 
56. int main()
57. {
58.     // 建立堆疊(儲存運算符號)
59.     stackPtr stackNodePtr = NULL;
60.     // 建立串列(儲存 infix 後的數字與符號)
61.     stackPtr infixHeadPtr = NULL;
62.     // 建立串列(儲存傳換成 postfix 後的數字與符號)
63.     stackPtr postfixHeadPtr = NULL;
64.     // 逐字讀取的容器
65.     char value;
66.     // 設定步數旗標
67.     int step = 1;
68.     char trush;
69. 
70.     // 系統輸出 "Please enter the formula : " 在 Console 上
71.     printf("Please enter the formula : \n");
72.     // 在讀取到換行符'\n'之前逐字讀取字元
73.     while ((value = getchar()) != '\n')
74.     {
75.         insertlist(&infixHeadPtr, value, 1);
76.     }
77. 
78.     // 將串列的首位指標設為當前指標位置(後指標)
79.     stackPtr currentPtr = infixHeadPtr->nextPtr;
80.     // 定義前個指標為空(前指標)
81.     stackPtr previousPtr = infixHeadPtr;
82. 
83.     // 當串列的後指標到達最後一位之前
84.     while (currentPtr != NULL)
85.     {
86.         // 如果兩指標的字元性質相異
87.         if (currentPtr->chardetect != previousPtr->chardetect)
88.         {
89.             // 若前指標讀取到數字
90.             if (previousPtr->chardetect == 0)
91.             {
92.                 // 在移動到後指標之前
93.                 while (currentPtr != previousPtr)
94.                 {
```

```
95.                         // 將讀取到的數字直接插入 postfix 串列
96.                         insertlist(&postfixHeadPtr, previousPtr->value, 0);
97.                         // 前指標移向下個字元
98.                         previousPtr = previousPtr->nextPtr;
99.                     }
100.                    // 以逗號區隔下個符號
101.                    insertlist(&postfixHeadPtr, dot, 0);
102.                    // 將後指標向下移動至下個字元
103.                    currentPtr = currentPtr->nextPtr;
104.                    step = 1;
105.                }
106.                // 若前指標讀取到運算符
107.                // 接下來每個 case 做的事情都一樣
108.                else
109.                {
110.                    switch (previousPtr->value)
111.                    {
112.                    // 若讀入左括號
113.                    case '(':
114.                        // 直接推入堆疊中
115.                        push(&stackNodePtr, previousPtr->value);
116.                        // 將前指標移動至後指標位置
117.                        previousPtr = currentPtr;
118.                        // 將後指標向下移動至下個字元
119.                        currentPtr = currentPtr->nextPtr;
120.                        break;
121.
122.                    // 若讀入右括號
123.                    case ')':
124.                        // 在讀取到左括號前，依序 pop 出堆疊中的運算符
125.                        while (stackNodePtr->value != '(')
126.                        {
127.                            // 將運算符推入存 postfix 的串列中
128.                            insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);
129.                            // 以逗號區隔下個符號
130.                            insertlist(&postfixHeadPtr, dot, 0);
131.                        }
132.
133.                        // 建立零時容器吸收不須輸出的左括號
134.                        trush = pop(&stackNodePtr);
135.                        // 將前指標移動至後指標位置
136.                        previousPtr = currentPtr;
137.                        // 將後指標向下移動至下個字元
138.                        currentPtr = currentPtr->nextPtr;
139.                        break;
140.
141.                    case '+':
142.                    case '-':
143.                    case '/':
144.                    case '%':
145.                    case '^':
146.                        // 若堆疊內不是空的
147.                        if (!isEmpty(stackNodePtr))
148.                        {
149.                            // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括號
150.                            if (priority(previousPtr->value) < priority(stackNodePtr
    ->value) || stackNodePtr->value == '(')
151.                            {
152.                                // 直接推入堆疊中
153.                                push(&stackNodePtr, previousPtr->value);
154.                                // 將前指標移動至後指標位置
155.                                previousPtr = currentPtr;
156.                                // 將後指標向下移動至下個字元
```

```
157.                                 currentPtr = currentPtr->nextPtr;
158.                                 // 重製步數旗標
159.                                 step = 1;
160.                                 break;
161.                             }
162.                             else
163.                             {
164.                                 // 先將運算符推入存 postfix 的串列中
165.                                 insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);

166.                                 // 以逗號區隔下個符號
167.                                 insertlist(&postfixHeadPtr, dot, 0);
168.                                 // 再將讀入的運算符直接推入堆疊中
169.                                 push(&stackNodePtr, previousPtr->value);
170.                                 // 將前指標移動至後指標位置
171.                                 previousPtr = currentPtr;
172.                                 // 將後指標向下移動至下個字元
173.                                 currentPtr = currentPtr->nextPtr;
174.                                 // 重製步數旗標
175.                                 step = 1;
176.                                 break;
177.                             }
178.                         }
179.                         else
180.                         {
181.                             // 直接推入堆疊中
182.                             push(&stackNodePtr, previousPtr->value);
183.                             // 將前指標移動至後指標位置
184.                             previousPtr = currentPtr;
185.                             // 將後指標向下移動至下個字元
186.                             currentPtr = currentPtr->nextPtr;
187.                             // 重製步數旗標
188.                             step = 1;
189.                             break;
190.                         }
191.                     case '*':
192.                         // 若讀取到的運算符為兩個字元
193.                         if (step == 2)
194.                         {
195.                             // 若堆疊內不是空的
196.                             if (!isEmpty(stackNodePtr))
197.                             {
198.                                 // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括
    號
199.                                 if (priority('p') < priority(stackNodePtr-
    >value) || stackNodePtr->value == '(')
200.                                     {
201.                                     // 直接推入堆疊中
202.                                     push(&stackNodePtr, 'p');
203.                                     // 將前指標移動至後指標位置
204.                                     previousPtr = currentPtr;
205.                                     // 將後指標向下移動至下個字元
206.                                     currentPtr = currentPtr->nextPtr;
207.                                     // 重製步數旗標
208.                                     step = 1;
209.                                     break;
210.                                 }
211.                                 else
212.                                 {
213.                                     // 先將運算符推入存 postfix 的串列中
214.                                     insertlist(&postfixHeadPtr, pop(&stackNodePtr),
     0);
215.                                     // 以逗號區隔下個符號
```

```
                                    insertlist(&postfixHeadPtr, dot, 0);
                                    // 再將讀入的運算符直接推入堆疊中
                                    push(&stackNodePtr, 'p');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                            }
                            else
                            {
                                // 直接推入堆疊中
                                push(&stackNodePtr, 'p');
                                // 將前指標移動至後指標位置
                                previousPtr = currentPtr;
                                // 將後指標向下移動至下個字元
                                currentPtr = currentPtr->nextPtr;
                                // 重製步數旗標
                                step = 1;
                                break;
                            }
                        }
                        else
                        {
                            // 若堆疊內不是空的
                            if (!isEmpty(stackNodePtr))
                            {
                                // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括號
                                if (priority(previousPtr->value) < priority(stackNodePtr->value) || stackNodePtr->value == '(')
                                {
                                    // 直接推入堆疊中
                                    push(&stackNodePtr, previousPtr->value);
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                                else
                                {
                                    // 先將運算符推入存 postfix 的串列中
                                    insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);
                                    // 以逗號區隔下個符號
                                    insertlist(&postfixHeadPtr, dot, 0);
                                    // 再將讀入的運算符直接推入堆疊中
                                    push(&stackNodePtr, previousPtr->value);
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                            }
                            else
```

```
                            {
                                // 直接推入堆疊中
                                push(&stackNodePtr, previousPtr->value);
                                // 將前指標移動至後指標位置
                                previousPtr = currentPtr;
                                // 將後指標向下移動至下個字元
                                currentPtr = currentPtr->nextPtr;
                                // 重製步數旗標
                                step = 1;
                                break;
                            }
                        }
                    case '>':
                        // 若讀取到的運算符為兩個字元
                        if (step == 2)
                        {
                            // 若堆疊內不是空的
                            if (!isEmpty(stackNodePtr))
                            {
                                // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括號
                                if (priority('r') < priority(stackNodePtr->value) || stackNodePtr->value == '(')
                                {
                                    // 直接推入堆疊中
                                    push(&stackNodePtr, 'r');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                                else
                                {
                                    // 先將運算符推入存 postfix 的串列中
                                    insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);
                                    // 以逗號區隔下個符號
                                    insertlist(&postfixHeadPtr, dot, 0);
                                    // 再將讀入的運算符直接推入堆疊中
                                    push(&stackNodePtr, 'r');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                            }
                            else
                            {
                                // 直接推入堆疊中
                                push(&stackNodePtr, 'r');
                                // 將前指標移動至後指標位置
                                previousPtr = currentPtr;
                                // 將後指標向下移動至下個字元
                                currentPtr = currentPtr->nextPtr;
                                // 重製步數旗標
                                step = 1;
                                break;
```

```
337.                             }
338.                         }
339.                         else
340.                         {
341.                             // 若堆疊內不是空的
342.                             if (!isEmpty(stackNodePtr))
343.                             {
344.                                 // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括
    號
345.                                 if (priority(previousPtr-
    >value) < priority(stackNodePtr->value) || stackNodePtr->value == '('
)
346.                                 {
347.                                     // 直接推入堆疊中
348.                                     push(&stackNodePtr, previousPtr->value);
349.                                     // 將前指標移動至後指標位置
350.                                     previousPtr = currentPtr;
351.                                     // 將後指標向下移動至下個字元
352.                                     currentPtr = currentPtr->nextPtr;
353.                                     // 重製步數旗標
354.                                     step = 1;
355.                                     break;
356.                                 }
357.                                 else
358.                                 {
359.                                     // 先將運算符推入存 postfix 的串列中
360.                                     insertlist(&postfixHeadPtr, pop(&stackNodePtr),
     0);
361.                                     // 以逗號區隔下個符號
362.                                     insertlist(&postfixHeadPtr, dot, 0);
363.                                     // 再將讀入的運算符直接推入堆疊中
364.                                     push(&stackNodePtr, previousPtr->value);
365.                                     // 將前指標移動至後指標位置
366.                                     previousPtr = currentPtr;
367.                                     // 將後指標向下移動至下個字元
368.                                     currentPtr = currentPtr->nextPtr;
369.                                     // 重製步數旗標
370.                                     step = 1;
371.                                     break;
372.                                 }
373.                             }
374.                             else
375.                             {
376.                                 // 直接推入堆疊中
377.                                 push(&stackNodePtr, previousPtr->value);
378.                                 // 將前指標移動至後指標位置
379.                                 previousPtr = currentPtr;
380.                                 // 將後指標向下移動至下個字元
381.                                 currentPtr = currentPtr->nextPtr;
382.                                 // 重製步數旗標
383.                                 step = 1;
384.                                 break;
385.                             }
386.                         }
387.                     case '<':
388.                         // 若讀取到的運算符為兩個字元
389.                         if (step == 2)
390.                         {
391.                             // 若堆疊內不是空的
392.                             if (!isEmpty(stackNodePtr))
393.                             {
394.                                 // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括
    號
395.                                 if (priority('l') < priority(stackNodePtr-
    >value) || stackNodePtr->value == '('
)
```

```
                                {
                                    // 直接推入堆疊中
                                    push(&stackNodePtr, 'l');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                                else
                                {
                                    // 先將運算符推入存 postfix 的串列中
                                    insertlist(&postfixHeadPtr, pop(&stackNodePtr),
    0);
                                    // 以逗號區隔下個符號
                                    insertlist(&postfixHeadPtr, dot, 0);
                                    // 再將讀入的運算符直接推入堆疊中
                                    push(&stackNodePtr, 'l');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                            }
                            else
                            {
                                // 直接推入堆疊中
                                push(&stackNodePtr, 'l');
                                // 將前指標移動至後指標位置
                                previousPtr = currentPtr;
                                // 將後指標向下移動至下個字元
                                currentPtr = currentPtr->nextPtr;
                                // 重製步數旗標
                                step = 1;
                                break;
                            }
                        }
                        else
                        {
                            // 若堆疊內不是空的
                            if (!isEmpty(stackNodePtr))
                            {
                                // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括
    號
                                if (priority(previousPtr-
    >value) < priority(stackNodePtr->value) || stackNodePtr->value == '(')
                                {
                                    // 直接推入堆疊中
                                    push(&stackNodePtr, previousPtr->value);
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                                else
                                {
```

```
                                    // 先將運算符推入存 postfix 的串列中
                                    insertlist(&postfixHeadPtr, pop(&stackNodePtr),
    0);
                                    // 以逗號區隔下個符號
                                    insertlist(&postfixHeadPtr, dot, 0);
                                    // 再將讀入的運算符直接推入堆疊中
                                    push(&stackNodePtr, previousPtr->value);
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                            }
                            else
                            {
                                // 直接推入堆疊中
                                push(&stackNodePtr, previousPtr->value);
                                // 將前指標移動至後指標位置
                                previousPtr = currentPtr;
                                // 將後指標向下移動至下個字元
                                currentPtr = currentPtr->nextPtr;
                                // 重製步數旗標
                                step = 1;
                                break;
                            }
                        }
                    case '&':
                        // 若讀取到的運算符為兩個字元
                        if (step == 2)
                        {
                            // 若堆疊內不是空的
                            if (!isEmpty(stackNodePtr))
                            {
                                // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括
    號
                                if (priority('a') < priority(stackNodePtr-
    >value) || stackNodePtr->value == '(')
                                {
                                    // 直接推入堆疊中
                                    push(&stackNodePtr, 'a');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                                else
                                {
                                    // 先將運算符推入存 postfix 的串列中
                                    insertlist(&postfixHeadPtr, pop(&stackNodePtr),
    0);
                                    // 以逗號區隔下個符號
                                    insertlist(&postfixHeadPtr, dot, 0);
                                    // 再將讀入的運算符直接推入堆疊中
                                    push(&stackNodePtr, 'a');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
```

```
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                            }
                            else
                            {
                                // 直接推入堆疊中
                                push(&stackNodePtr, 'a');
                                // 將前指標移動至後指標位置
                                previousPtr = currentPtr;
                                // 將後指標向下移動至下個字元
                                currentPtr = currentPtr->nextPtr;
                                // 重製步數旗標
                                step = 1;
                                break;
                            }
                        }
                        else
                        {
                            // 若堆疊內不是空的
                            if (!isEmpty(stackNodePtr))
                            {
                                // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括號
                                if (priority(previousPtr->value) < priority(stackNodePtr->value) || stackNodePtr->value == '(')
                                {
                                    // 直接推入堆疊中
                                    push(&stackNodePtr, previousPtr->value);
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                                else
                                {
                                    // 先將運算符推入存 postfix 的串列中
                                    insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);
                                    // 以逗號區隔下個符號
                                    insertlist(&postfixHeadPtr, dot, 0);
                                    // 再將讀入的運算符直接推入堆疊中
                                    push(&stackNodePtr, previousPtr->value);
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                            }
                            else
                            {
                                // 直接推入堆疊中
                                push(&stackNodePtr, previousPtr->value);
                                // 將前指標移動至後指標位置
                                previousPtr = currentPtr;
                                // 將後指標向下移動至下個字元
```

```
                                currentPtr = currentPtr->nextPtr;
                                // 重製步數旗標
                                step = 1;
                                break;
                            }
                        }
                    case '|':
                        // 若讀取到的運算符為兩個字元
                        if (step == 2)
                        {
                            // 若堆疊內不是空的
                            if (!isEmpty(stackNodePtr))
                            {
                                // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括號
                                if (priority('o') < priority(stackNodePtr->value) || stackNodePtr->value == '(')
                                {
                                    // 直接推入堆疊中
                                    push(&stackNodePtr, 'o');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                                else
                                {
                                    // 先將運算符推入存 postfix 的串列中
                                    insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);
                                    // 以逗號區隔下個符號
                                    insertlist(&postfixHeadPtr, dot, 0);
                                    // 再將讀入的運算符直接推入堆疊中
                                    push(&stackNodePtr, 'o');
                                    // 將前指標移動至後指標位置
                                    previousPtr = currentPtr;
                                    // 將後指標向下移動至下個字元
                                    currentPtr = currentPtr->nextPtr;
                                    // 重製步數旗標
                                    step = 1;
                                    break;
                                }
                            }
                            else
                            {
                                // 直接推入堆疊中
                                push(&stackNodePtr, 'o');
                                // 將前指標移動至後指標位置
                                previousPtr = currentPtr;
                                // 將後指標向下移動至下個字元
                                currentPtr = currentPtr->nextPtr;
                                // 重製步數旗標
                                step = 1;
                                break;
                            }
                        }
                        else
                        {
                            // 若堆疊內不是空的
                            if (!isEmpty(stackNodePtr))
                            {
```

```
638.                                // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括
    號
639.                                if (priority(previousPtr-
    >value) < priority(stackNodePtr->value) || stackNodePtr->value == '(')
640.                                {
641.                                    // 直接推入堆疊中
642.                                    push(&stackNodePtr, previousPtr->value);
643.                                    // 將前指標移動至後指標位置
644.                                    previousPtr = currentPtr;
645.                                    // 將後指標向下移動至下個字元
646.                                    currentPtr = currentPtr->nextPtr;
647.                                    // 重製步數旗標
648.                                    step = 1;
649.                                    break;
650.                                }
651.                                else
652.                                {
653.                                    // 先將運算符推入存 postfix 的串列中
654.                                    insertlist(&postfixHeadPtr, pop(&stackNodePtr),
     0);
655.                                    // 以逗號區隔下個符號
656.                                    insertlist(&postfixHeadPtr, dot, 0);
657.                                    // 再將讀入的運算符直接推入堆疊中
658.                                    push(&stackNodePtr, previousPtr->value);
659.                                    // 將前指標移動至後指標位置
660.                                    previousPtr = currentPtr;
661.                                    // 將後指標向下移動至下個字元
662.                                    currentPtr = currentPtr->nextPtr;
663.                                    // 重製步數旗標
664.                                    step = 1;
665.                                    break;
666.                                }
667.                            }
668.                            else
669.                            {
670.                                // 直接推入堆疊中
671.                                push(&stackNodePtr, previousPtr->value);
672.                                // 將前指標移動至後指標位置
673.                                previousPtr = currentPtr;
674.                                // 將後指標向下移動至下個字元
675.                                currentPtr = currentPtr->nextPtr;
676.                                // 重製步數旗標
677.                                step = 1;
678.                                break;
679.                            }
680.                        }
681.                    }
682.                }
683.            }
684.            else
685.            {
686.                // 將後指標向後移動
687.                currentPtr = currentPtr->nextPtr;
688.                // 運算符長度+1
689.                step++;
690.            }
691.        }
692.
693.        // 若已掃描完畢還未有數字未讀出
694.        while (previousPtr != NULL)
695.        {
696.            // 將讀取到的數字直接插入 postfix 串列
697.            insertlist(&postfixHeadPtr, previousPtr->value, 0);
698.            // 移向下個指標
```

```
699.            previousPtr = previousPtr->nextPtr;
700.        }
701.        // 以逗號區隔下個符號
702.        insertlist(&postfixHeadPtr, dot, 0);
703.
704.        // 讀取結束後，將堆疊中的運算符全部依序輸出
705.        while (stackNodePtr != NULL)
706.        {
707.            // 將讀取到的運算符直接插入 postfix 串列
708.            insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);
709.            // 以逗號區隔下個符號
710.            insertlist(&postfixHeadPtr, dot, 0);
711.        }
712.
713.        // 建立堆疊 (計算 postfix 算式)
714.        calculatePtr calculateTop = NULL;
715.        // 建立暫存數字
716.        int numtemp = 0;
717.
718.        // 當讀取到的串列位址為空前，不停進行計算
719.        while (postfixHeadPtr != NULL)
720.        {
721.            // 建立暫存容器，讀取從 postfix 串列輸出的字元
722.            char temp = poplist(&postfixHeadPtr);
723.
724.            // 判斷字元為數字或字元為
725.            switch (temp)
726.            {
727.            // 若讀取到 dot 直接略過
728.            case dot:
729.                break;
730.
731.            // 字元為運算符
732.            case '+':
733.            case '-':
734.            case '*':
735.            case '/':
736.            case '%':
737.            case '>':
738.            case '<':
739.            case '&':
740.            case '|':
741.            case '^':
742.            case 'a':
743.            case 'o':
744.            case 'p':
745.            case 'r':
746.            case 'l':
747.                // 將堆疊中的數字進行相對應的運算
748.                calculate(&calculateTop, temp);
749.                break;
750.
751.            // 字元為數字
752.            default:
753.
754.                // 在讀取到 dot 前，進行連續讀取
755.                while (temp != dot)
756.                {
757.                    // 數字掃描迴圈
758.                    for (int i = 48; i <= 57; i++)
759.                    {
760.                        // 若讀入的數字符合該數字的 ascii 碼
761.                        if ((int)temp == i)
762.                        {
```

```
763.                            // 將數字算出再加上前個數字乘以 10
764.                            numtemp = numtemp * 10 + (i - 48);
765.                            break;
766.                        }
767.                    }
768.                    // 讀取下個數字
769.                    temp = poplist(&postfixHeadPtr);
770.                }
771.
772.                // 將結果強制轉型後存入堆疊
773.                pushDouble(&calculateTop, (double)numtemp);
774.                // 重置暫存數字
775.                numtemp = 0;
776.                break;
777.            }
778.        }
779.
780.        // 在 console 上輸出答案
781.        printf("\n%.13lf", popDouble(&calculateTop));
782.    }
783.
784.    void insertlist(stackPtr *headPtr, char info, int mode)
785.    {
786.        // 建立新的節點
787.        stackPtr newPtr = (stackPtr)malloc(sizeof(stacknode));
788.
789.        // 記憶體擁有足夠的空間
790.        if (newPtr != NULL)
791.        {
792.            // 若使用模式 '1'
793.            // 會幫讀入的字元區分為數字或字元
794.            if (mode)
795.            {
796.                // 若讀入為數字
797.                if ((int)info >= 48 /* 0 */ && (int)info <= 57 /* 9 */)
798.                {
799.                    // 將其字元在 chardetect 表示為'0'(數字)
800.                    newPtr->chardetect = 0;
801.                }
802.                else if (info == '(' || info == ')')
803.                {
804.                    // 將其字元在 chardetect 表示為'2'(括號)
805.                    newPtr->chardetect = 2;
806.                }
807.                // 若讀入為符號
808.                else
809.                {
810.                    // 將其字元在 chardetect 表示為'1'(符號)
811.                    newPtr->chardetect = 1;
812.                }
813.            }
814.
815.            // 將欲插入的字元存入節點中
816.            newPtr->value = info;
817.            // 將指向下的節點的指標指為空
818.            newPtr->nextPtr = NULL;
819.
820.            // 將串列的首位指標設為當前指標位置
821.            stackPtr currentPtr = *headPtr;
822.            // 定義前個指標為空
823.            stackPtr previousPtr = NULL;
824.
825.            // 將當前指標位置移動至串列最尾端
826.            while (currentPtr != NULL)
```

```
827.             {
828.                 // 將前個指標的位置設為當前指標
829.                 previousPtr = currentPtr;
830.                 // 將當前指標位置向下個節點移動
831.                 currentPtr = currentPtr->nextPtr;
832.             }
833.
834.             // 若串列中沒有半個元素
835.             if (previousPtr == NULL)
836.             {
837.                 // 將新節點所指向的下個者標位置指向讀入串列的頭(NULL)
838.                 newPtr->nextPtr = *headPtr;
839.                 // 將元串列的位置指項新節點
840.                 *headPtr = newPtr;
841.             }
842.             else
843.             {
844.                 // 將前一個指標的所指向的下個位置指向新增的節點
845.                 previousPtr->nextPtr = newPtr;
846.                 // 將新節點的所指向的下個指標指向當前節點
847.                 newPtr->nextPtr = currentPtr;
848.             }
849.         }
850.         // 記憶體空間用盡
851.         else
852.         {
853.             printf("%c not inserted. No memory available.\n", info);
854.         }
855.     }
856.
857.     char poplist(stackPtr *topPtr)
858.     {
859.         // 建立暫存指標指向串列的頭
860.         stackPtr tempPtr = *topPtr;
861.
862.         // 讀取串列頭所儲存的元素
863.         char output = tempPtr->value;
864.         // 將串列頭的指標指向串列的下一個元素
865.         *topPtr = (*topPtr)->nextPtr;
866.         // 釋放串列頭的記憶體空間
867.         free(tempPtr);
868.
869.         // 輸出串列頭所儲存的元素
870.         return output;
871.     }
872.
873.     void push(stackPtr *topPtr, char info)
874.     {
875.         // 建立新的節點
876.         stackPtr newPtr = (stackPtr)malloc(sizeof(stacknode));
877.
878.         // 若還有記憶體空間
879.         if (newPtr != NULL)
880.         {
881.             // 將欲插入的字元存入新節點中
882.             newPtr->value = info;
883.             // 將新節點所指向的下個位置指向當前堆疊的頂端
884.             newPtr->nextPtr = *topPtr;
885.             // 將堆疊的頂端位置指向剛建立的節點
886.             *topPtr = newPtr;
887.         }
888.         // 記憶體空間不足
889.         else
890.         {
```

```
891.             printf("%c not inserted. No memory available.\n", info);
892.         }
893.     }
894.
895.     // pushDouble 函式只是推入堆疊的資料改成 double 型態，其運作內容相同於 push 函式
896.     void pushDouble(calculatePtr *topPtr, double info)
897.     {
898.         calculatePtr newPtr = (calculatePtr)malloc(sizeof(calculateNode));
899.
900.         if (newPtr != NULL)
901.         {
902.             newPtr->value = info;
903.             newPtr->nextPtr = *topPtr;
904.             *topPtr = newPtr;
905.         }
906.         else
907.         {
908.             printf("%lf not inserted. No memory available.\n", info);
909.         }
910.     }
911.
912.     char pop(stackPtr *topPtr)
913.     {
914.         // 建立暫存指標指向堆疊的頂端
915.         stackPtr tempPtr = *topPtr;
916.
917.         // 讀取堆疊頂端所儲存的元素
918.         char popValue = (*topPtr)->value;
919.         // 將堆疊頂端的指標指向堆疊內的下一個元素
920.         *topPtr = (*topPtr)->nextPtr;
921.         // 釋放堆疊頂端節點的記憶體空間
922.         free(tempPtr);
923.
924.         // 輸出堆疊頂端所儲存的元素
925.         return popValue;
926.     }
927.
928.     // popDouble 函式只是推入堆疊的資料改成 double 型態，其運作內容相同於 pop 函式
929.     double popDouble(calculatePtr *topPtr)
930.     {
931.         calculatePtr tempPtr = *topPtr;
932.
933.         double popValue = (*topPtr)->value;
934.         *topPtr = (*topPtr)->nextPtr;
935.         free(tempPtr);
936.
937.         return popValue;
938.     }
939.
940.     int isEmpty(stackPtr topPtr)
941.     {
942.         // 檢查串列頂端是否為空
943.         return topPtr == NULL;
944.     }
945.
946.     void printlist(stackPtr currentPtr, int mode)
947.     {
948.         // 檢查當前串列是否為空
949.         if (currentPtr == NULL)
950.         {
951.             puts("The stack is empty.\n");
952.         }
953.         else
954.         {
955.             // 將當前串列位置往後移直到到達尾端
```

```
956.            while (currentPtr != NULL)
957.            {
958.                switch (mode)
959.                {
960.                case 0:
961.                    // 輸出該串列元素
962.                    printf("%c", currentPtr->value);
963.                    // 將當前串列位置往後移
964.                    currentPtr = currentPtr->nextPtr;
965.                    break;
966.
967.                case 1:
968.                    // 輸出該串列元素
969.                    printf("%d", currentPtr->chardetect);
970.                    // 將當前串列位置往後移
971.                    currentPtr = currentPtr->nextPtr;
972.                    break;
973.                }
974.            }
975.        }
976.    }
977.
978.    int priority(char op)
979.    {
980.        // 檢查該運符之優先權
981.        switch (op)
982.        {
983.        case 'p': // 次方 power
984.            return 3;
985.
986.        case '*': // 乘法
987.        case '/': // 除法
988.        case '%': // 取 mod
989.            return 5;
990.
991.        case '+': // 加法
992.        case '-': // 減法
993.            return 6;
994.
995.        case 'r': // 位元右移
996.        case 'l': // 位元左移
997.            return 7;
998.
999.        case '>': // 大於
1000.       case '<': // 小於
1001.           return 8;
1002.
1003.       case '&': // 位元 and
1004.           return 10;
1005.
1006.       case '^': // 位元 xor
1007.           return 11;
1008.
1009.       case '|': // 位元 or
1010.           return 12;
1011.
1012.       case 'a': // 邏輯 and
1013.           return 13;
1014.
1015.       case 'o': // 邏輯 or
1016.           return 14;
1017.
1018.       default:
1019.           return 0;
```

```
1020.     }
1021. }
1022.
1023. void calculate(calculatePtr *topPtr, char sign)
1024. {
1025.     // 分別讀出堆疊中的頂端兩組數字
1026.     double temp2 = popDouble(topPtr);
1027.     double temp1 = popDouble(topPtr);
1028.
1029.     // 對不同運算符對兩數字進行相對應的運算
1030.     switch (sign)
1031.     {
1032.     // 加法
1033.     case '+':
1034.         pushDouble(topPtr, temp1 + temp2);
1035.         break;
1036.
1037.     // 減法
1038.     case '-':
1039.         pushDouble(topPtr, temp1 - temp2);
1040.         break;
1041.
1042.     // 乘法
1043.     case '*':
1044.         pushDouble(topPtr, temp1 * temp2);
1045.         break;
1046.
1047.     // 除法
1048.     case '/':
1049.         pushDouble(topPtr, temp1 / temp2);
1050.         break;
1051.
1052.     // 取 mod
1053.     case '%':
1054.         pushDouble(topPtr, (double)((int)temp1 % (int)temp2));
1055.         break;
1056.
1057.     // 大於
1058.     case '>':
1059.         pushDouble(topPtr, temp1 > temp2);
1060.         break;
1061.
1062.     // 小於
1063.     case '<':
1064.         pushDouble(topPtr, temp1 < temp2);
1065.         break;
1066.
1067.     // 位元 and
1068.     case '&':
1069.         pushDouble(topPtr, (double)((int)temp1 & (int)temp2));
1070.         break;
1071.
1072.     // 位元 xor
1073.     case '^':
1074.         pushDouble(topPtr, (double)((int)temp1 ^ (int)temp2));
1075.         break;
1076.
1077.     // 位元 or
1078.     case '|':
1079.         pushDouble(topPtr, (double)((int)temp1 | (int)temp2));
1080.         break;
1081.
1082.     // 次方 power
1083.     case 'p':
```

```
1084.           pushDouble(topPtr, pow(temp1, temp2));
1085.           break;
1086.
1087.       // 位元右移
1088.       case 'r':
1089.           pushDouble(topPtr, (double)((int)temp1 >> (int)temp2));
1090.           break;
1091.
1092.       // 位元左移
1093.       case 'l':
1094.           pushDouble(topPtr, (double)((int)temp1 << (int)temp2));
1095.           break;
1096.
1097.       // 邏輯 and
1098.       case 'a':
1099.           pushDouble(topPtr, (double)((int)temp1 && (int)temp2));
1100.           break;
1101.
1102.       // 邏輯 or
1103.       case 'o':
1104.           pushDouble(topPtr, (double)((int)temp1 || (int)temp2));
1105.           break;
1106.
1107.       default:
1108.           break;
1109.       }
1110. }
```

Result

#Test 1 - 1+3-2 (Regular Data)

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
1+3-2

2.0000000000000
```

#Test 2 - 1+2*3 (Regular Data)

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
1+2*3

7.0000000000000
```

#Test 3 - 1-2*3 (Regular Data)

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
1-2*3

-5.0000000000000
```

#Test 4 - (1+2)*3 (Regular Data)

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
(1+2)*3

9.0000000000000
```

#Test 5 – 取 mod

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
7%3+1

2.0000000000000
```

#Test 6 - 大於小於

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
(1>2)+(2<1)

0.0000000000000
```

#Test 7 - 位元左移右移

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
7<<1*2

28.0000000000000
```

#Test 8 - 位元 and 及位元 or

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
(7&0)+(12|3)

15.0000000000000
```

#Test 9 - 次方 power (次方運算符為”**”)

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
2**10>2**9*2+1

0.0000000000000
```

#Test 10 - 邏輯 and 及邏輯 or

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
(7&&0)+(12||3)

1.0000000000000
```

#Test 11 - 多位數運算

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
(123-23)**2/36

277.7777777777778
```

#Test 12 - 長算式計算(整數部分超出 double 範圍)

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
11111+22222*33333/44444|55555&66666^77777%88888>99999<121212||232323&&343434<<555>>66

27778.0000000000000
```

#Test 13 - 假輸出(不合理的答案)

```
PS C:\Users\chann\Desktop\Data Strcture\Assignment 02 - Evaluating an Arithmetic Expression> ./main.exe
Please enter the formula :
2*=10

1024.0000000000000
```

Discussion

(一)、程式運作原理

我們將這 1000 多行的程式做個簡單分類，而 main 函式中可分為兩部分，分別是 infix 轉制 postfix 以及 postfix 的計算。因為本程式全程使用串列(linked list)及串列堆疊(linked stack)，故會產生整體程式龐大的錯覺。接下來我們來分析程式的運作流程。

(1)、infix 轉制 postfix

這邊採取的方法，是將讀入的所有數字及運算符以 char 的型態先存入 infixHeadPtr 的串列中(第 61 行)。其中比較特別的是我們自定義的結構

```
1.  // 定義建立堆疊與串列所需的結構 (Infix 轉 Postfix 用)
2.  typedef struct StackNode
3.  {
4.  // 儲存讀入字元
5.  char value;
6.  // 判斷為數字、運算元或括號
7.  int chardetect;
8.  // 儲存下一節點的指標
9.  struct StackNode *nextPtr;
10.
11. } stacknode;
```

除了有儲存資料的 value 以外，還有一個 chardetect 的元素儲存讀入字元的屬性，以判斷其為數字、運算元或括號，再搭配 insertlist 函式(第 786 行)使用 mode 1 進行元素插入。在完成讀入一開始的 infix 算式後，會形成下面的型態，假設輸入為(11+22)/33，我們將儲存完的 infixHeadPtr 串列以表格表示會是

value	(	1	1	+	2	2	)	/	3	3
chardetect	2	0	0	1	1	1	2	1	0	0
nextPtr	→	→	→	→	→	→	→	→	→	NULL

“→”表示指向下一節點的指標

Infix 算式串列

再來我們又定義了兩個指標分別是 previousPtr 與 currentPtr，接下來我將稱呼兩個指標為「前指標」以及「後指標」，兩個指標如下

```
1.      // 將串列的首位指標設為當前指標位置(後指標)
2.      stackPtr currentPtr = infixHeadPtr->nextPtr;
3.      // 定義前個指標為空(前指標)
4.      stackPtr previousPtr = infixHeadPtr;
```

用表格表示會變成以下這樣，我們為求方便就不看 nextPtr。前指標所指向的節點以藍色表示，而後指標以紅色表示，指標在哪個節點下方代表指向哪個節點。

value	(	1	1	+	2	2	)	/	3	3
chardetect	2	0	0	1	1	1	2	1	0	0
前指標	↑									
後指標		↑								

指標移動的結束條件為後指標最後移動到空指標(NULL)(第 84 行)，所以整個轉置成 postfix 算式的過程，是將 infix 算式又進行一次掃描。接著將判斷前後指標指向的 chardetect 格中，兩數是否相同(第 87 行)。接續上方表格可知，2 不等於 0，代表左括號是不同於下個節點的屬性。我們遇見左括號需要將它直接塞入堆疊中(第 113 行)，完成後將前指標移動到後指標的位置，再將後指標向下個節點移動。綜合以上動作，我們以表格來表示當前的指標位置及堆疊內著狀況。

value	(	1	1	+	2	2	)	/	3	3
chardetect	2	0	0	1	0	0	2	1	0	0
前指標		↑								
後指標			↑							

Infix 算式串列

(			
↑ topPtr			

Stack 堆疊內狀況

此時前後指標指向的 chardetect 格中，兩數是相同的，所以我們便把後指標再移向下一個節點，如下表格所示

value	(	1	1	+	2	2	)	/	3	3
chardetect	2	0	0	1	0	0	2	1	0	0
前指標		↑								
後指標				↑						

Infix 算式串列

發現兩指標所指向的 chardetect 不同後，先以 chardetect 的數字判斷它的屬性，發現其為數字(第 90 行)，便開始將前指標所指向的 value 直接輸出至 postfix 的串列(第 93 行)，直到後指標移動到前指標的位置才停止，即代表數字輸出完畢。最執行的程式如下

```
1.  // 若前指標讀取到數字
2.  if (previousPtr->chardetect == 0)
3.  {
4.      // 在移動到後指標之前
5.      while (currentPtr != previousPtr)
6.      {
7.          // 將讀取到的數字直接插入 postfix 串列
8.          insertlist(&postfixHeadPtr, previousPtr->value, 0);
9.          // 前指標移向下個字元
10.         previousPtr = previousPtr->nextPtr;
11.     }
12.     // 以逗號區隔下個符號
13.     insertlist(&postfixHeadPtr, dot, 0);
14.     // 將後指標向下移動至下個字元
15.     currentPtr = currentPtr->nextPtr;
16.     step = 1;  }
```

所以此時的 infix 及 postfix 串列如下

value	(	1	1	+	2	2	)	/	3	3
chardetect	2	0	0	1	0	0	2	1	0	0
前指標				↑						
後指標					↑					

Infix 算式串列

(			
↑ topPtr			

Stack 堆疊內狀況

1	1	,															

Postfix 算式串列

現在我們讀取到符號，我們要判斷要塞入堆疊中還是輸出至 postfix 算式，便需要提及運算符的優先權問題。在每個符號判斷的 case 中都是相同的輸出方法，因為不知道如何將 switch 中的程式獨立成個別函式(break 不可用於一般函式)，才導致程式攏長。所以我們拉出其中一個出來解釋

```
1.  // 若堆疊內不是空的
2.  if (!isEmpty(stackNodePtr))
3.  {
4.      // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括號
5.      if (priority(previousPtr->value) < priority(stackNodePtr-
    >value) || stackNodePtr->value == '(')
6.      {
7.          // 直接推入堆疊中
8.          push(&stackNodePtr, previousPtr->value);
9.          // 將前指標移動至後指標位置
10.         previousPtr = currentPtr;
11.         // 將後指標向下移動至下個字元
12.         currentPtr = currentPtr->nextPtr;
13.         // 重製步數旗標
14.         step = 1;
15.         break;
16.     }
17.     else
18.     {
19.         // 先將運算符推入存 postfix 的串列中
20.         insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);
21.         // 以逗號區隔下個符號
22.         insertlist(&postfixHeadPtr, dot, 0);
23.         // 再將讀入的運算符直接推入堆疊中
24.         push(&stackNodePtr, previousPtr->value);
25.         // 將前指標移動至後指標位置
26.         previousPtr = currentPtr;
27.         // 將後指標向下移動至下個字元
28.         currentPtr = currentPtr->nextPtr;
29.         // 重製步數旗標
30.         step = 1;
31.         break;
32.     }
```

```
33.      else
34.       {
35.           // 直接推入堆疊中
36.           push(&stackNodePtr, previousPtr->value);
37.           // 將前指標移動至後指標位置
38.           previousPtr = currentPtr;
39.           // 將後指標向下移動至下個字元
40.           currentPtr = currentPtr->nextPtr;
41.           // 重製步數旗標
42.           step = 1;
43.           break;
44.       }
45.}
```

首先，先判斷堆疊中是不是空的，如果是就直接推入堆疊中，否則繼續執行以下判斷。如果讀入的運算符優先權高於堆疊中的，也會直接推入堆疊中，不然就會將堆疊內的先輸出至 postfix 串列，再將讀入的運算符推入堆疊中。其中遇到右括號，就會將堆疊中的運算符輸出至 postfix 串列中，直到遇到左括號。至於步數旗標則是用在多字元的運算符如"**", '"&&", ">>"等等，就會在輸出時轉換成相對應的英文字母取代運算符，其餘的指標移動方式都和單位元運算符相等。

而當我們的後指標移動到空指標時，便代表掃描結束，此時前指標一定會停留在最後的數字上，如表格所示

value	(	1	1	+	2	2	)	/	3	3
chardetect	2	0	0	1	0	0	2	1	0	0
前指標									↑	
後指標										

此時，我們要先把數字輸出至 postfix 串列，才能將堆疊中剩餘的運算符送出來，只要將前指標指向的值輸出，將前指標再指向下個節點，直到指向空指標為止。程式如下

```
1.      // 若已掃描完畢還未有數字未讀出
2.      while (previousPtr != NULL)
3.      {
4.          // 將讀取到的數字直接插入 postfix 串列
5.          insertlist(&postfixHeadPtr, previousPtr->value, 0);
6.          // 移向下個指標
7.          previousPtr = previousPtr->nextPtr;
8.      }
9.      // 以逗號區隔下個符號
10.     insertlist(&postfixHeadPtr, dot, 0);
```

接著將堆疊中的所有運算符輸出，就完成 infix 轉置 postfix，程式如下

```
1. // 讀取結束後，將堆疊中的運算符全部依序輸出
2.      while (stackNodePtr != NULL)
3.      {
4.          // 將讀取到的運算符直接插入 postfix 串列
5.          insertlist(&postfixHeadPtr, pop(&stackNodePtr), 0);
6.          // 以逗號區隔下個符號
```

```
7.          insertlist(&postfixHeadPtr, dot, 0);  }
```

最終轉換完的 postfix 如下

1	1	,	2	2	,	+	,	3	3	,	/	,		

Postfix 算式串列

(2)、postfix 算式計算

接下來就和一般計算 postfix 的演算法相同，先將數字讀入堆疊中，再來把讀取到運算符當作堆疊上兩個數字的運算，最後剩下的數字便是答案。

讀取數字時，因為程式可以支援多位數的計算，我便在把數字或運算符插入 postfix 串列時，連帶插入逗號。因此在我讀取到逗號前，代表數字還沒完全的被讀取，其程式如下

```
// 在讀取到 dot 前，進行連續讀取
while (temp != dot)
{
                    // 數字掃描迴圈
    for (int i = 48; i <= 57; i++)
    {
        // 若讀入的數字符合該數字的 ascii 碼
        if ((int)temp == i)
        {
            // 將數字算出再加上前個數字乘以 10
            numtemp = numtemp * 10 + (i - 48);
            break;
        }
    }
    // 讀取下個數字
    temp = poplist(&postfixHeadPtr);
}

// 將結果強制轉型後存入堆疊
pushDouble(&calculateTop, (double)numtemp);
// 重置暫存數字
numtemp = 0;
break;
```

細節如下，我先建立暫存數字為 numtemp，從 postfix 串列讀入的字元為 temp，以 temp 的 ascii 碼判斷所代表的數字，在將數值存入 numtemp 中，若讀取到下個數字，只要將 numtemp 乘以 10 再加上 temp 的數字，便能計算出原本代表的數字。最後再強制轉型成 double 再 push 入堆疊就完成一個數字的完整讀取。其餘的符號，只要做相對應的運算，就能完成答案的計算。

(二)、問題與討論

(1) 多位元符號讀取問題

由於時間上的關係，在讀取多位元符號方法上，處理得相當草率。其判斷的方法是只看第一個字元及整體長度，所只適用於兩個字元都相同的運算符，如輸出結果的 #Test 13。正確的作法應該就和數字的讀取方式相同，一邊移動指標一邊判斷字元，防止無效的輸入。

(2) 一元運算子讀取問題

一元運算子是指只需要一個運算元就能進行計算的運算符號，其中包括常見的負號、不等於的「!」還有「i++」其中的「++」。這程式不支援一元運算子，若不小心讀入則會因為找不到或找錯成另一個運算元，進而導致程式崩潰。因此若要進行改進，應該在運算的 switch 迴圈中，額外的進行規範，避免掉例外狀況發生。

(3) 多行讀取問題

這程式只支援單行讀取，也就是指讀取一條算式。我們認為最好的解決方法，應該使用檔案讀取的方式實現，在檔案讀取完畢(EOF)後，再逐行的計算結果。

(4) infix 轉置 postfix 的演算法

在這程式中，我使用的演算法是自己想出來的。我有試著用網路上的演算法來進行，也有成功的計算出答案，這我會放在附錄中。不過要改成讀取超過兩字元的運算子時，在塞入堆疊時，運算子的字元會產生反轉。如讀取「and」這個運算子，分別會以「a」、「n」、「d」的順序進入堆疊，然而從堆疊取出時，卻因為堆疊先進後出(FILO)的特性，導致輸出到 postfix 串列時變成「d」、「n」、「a」。這個麻煩困擾我很久，最後從逐字讀入的讀取方式，改成一次先讀入整串後，再用掃描的方式轉置成 postfix。這兩個方法的時間複雜度還都是 O(n)，整體上不會相差太大。

(5) 多位元運算子的額外處理

為了要避免出現上個問題中，要一次讀取或存入多位元的運算子。我選擇以英文字母來取代那些運算子，這樣就可以保證運算子上的處理永遠只有一個位元。不過這個問題是治標不治本的，可能還是要更好的演算法來解決這個問題。

結論與心得

這程式全程使用串列連接，相比於用陣列寫成的，其好處當然是可以有彈性的擴充儲存算式的串列。不過在缺少安全的指標傳遞方式下，再加上 C 語言中本身缺乏「垃圾回收器」的機制，可能使指標在傳遞的過程中產生錯誤，導致程式無法正常運作的連結性問題，就是個美中不足的點。

這次的程式撰寫也都是自己完成的，花了一個星期有，是真的勞心又費神。不過還是對看這份報告的人很抱歉，本人的程式基底還是不太穩固，不能將程式收斂到乾淨再交出來，讓你們看 1000 多行的程式。整個程式唯一的優點，就是用 linked list 寫成的，其他部分還是有一堆 bug，或是沒做題目要求功能。這個程式是第 2 版，主要就是要解決多元運算子的問題。至於另外兩位同學，交出來的程式很明顯是參考過來的，不知道在改的人會看到多少的一模一樣的程式 XD。想說線性代數考差了就在這補回來，最後還是拜託老師高抬貴手，拜託不要當我啦。

Appendix

朱丞鈺 以陣列及 Queue 實作

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <time.h>
#define MAX_QUEUE 50
#define MAX 20
char new_str[MAX] = { "\0" };

    char Queue[MAX_QUEUE];
    char ac[MAX_QUEUE];
    int front = 0, rear = 0;

    int isFull() {
        return rear == MAX_QUEUE && front == 0;
    }

    int isEmpty() {
        return front == rear;
    }

    void Add(char* queue, char item) {
        if (isFull()) {
            printf("Queue is full!\n");
            return;
        }
        queue[rear++] = item;
    }

    void Delete(char* queue) {
        if (isEmpty()) {
            printf("Queue is empty!\n");
            return;
        }
        queue[front++];
    }

    void printQueue(char* queue) {
        if (isEmpty()) {
            printf("Queue is empty!\n");
            return;
        }
        printf("Queue: ");
        for (int i = front; i < rear; i++)
            printf("%c ", queue[i]);
        printf("\n");
    }


    void reverse(char str[MAX]) {
        int i, j;
        char c;
        for (i = 0, j = strlen(str) - 1; i < j; ++i, --j)
            c = str[i], str[i] = str[j], str[j] = c;
    }
    int compare(char op) {
        switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
        case '%':
```

```
            return 2;
        default:
            return 0;
        }
    }
    void Prefix(char* str, char* new_str) {
        char stack[MAX];
        int top = 0, j = 0, i;
        for (i = strlen(str) - 1; i >= 0; i--) {

            switch (str[i]) {
            case '+':
            case '-':
            case '*':
            case '/':
            case '%':
                while (compare(str[i]) < compare(stack[top])) {
                    new_str[j++] = stack[top--];
                }

                stack[++top] = str[i];

                continue;
            case ')':
                stack[++top] = str[i];
                continue;

            case '(':
                while (stack[top] != ')') {
                    new_str[j++] = stack[top--];
                }
                top--;
                continue;

            default:
                new_str[j++] = str[i];
                continue;

            }
        }
        while (top != 0) {
            new_str[j++] = stack[top--];
        }
    }

    void Infix_to_Prefix(char str[]) {

        Prefix(str, new_str);
        reverse(new_str);
    }
    int get_value(int op1, int op2, char op) {
        //printf("(%d %c %d)\n", op1, op, op2);
        switch (op) {
        case '+':
            return op1 + op2;
        case '-':
            return op1 - op2;
        case '*':
            return op1 * op2;
        case '/':
            return op1 / op2;
        case '%':
            return op1 % op2;
        default:
            return 0;
        }
```

```
    }
    void Calculation(char* str) {
        char stack[MAX];
        int top = 0, j = 0, i;
        for (i = strlen(str) - 1; i >= 0; i--) {
            // printf("\n%c\n", str[i]);
            switch (str[i]) {
            case '+':
            case '-':
            case '*':
            case '/':
            case '%':
                stack[top - 1] = get_value(stack[top], stack[top - 1], str[i]);

                top--;
                continue;

            default:
                stack[++top] = str[i] - 48;
                continue;

            }
        }
        int f = stack[top];
        float b = (float)f;
        printf("%f\n", b);
        //結果
    }
    int main() {
        char str1[MAX];
        fgets(str1, 20, stdin);
        Infix_to_Prefix(str1);
        Calculation(new_str);


        return 0;
    }
```

蔡宇軒 以陣列及 Stack 實作

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 80

void inToPostfix(char*, char*); // 中序轉後序
int priority(char); // 運算子優先權
double eval(char*);
double cal(char, double, double);

int main() {

    char infix[MAX] = {'\0'};

    printf("請輸入中序運算式：");
    gets(infix);
    printf("得到的結果為 : %.8f", eval(infix));

    return 0;
}

void inToPostfix(char* infix, char* postfix) {
    char stack[MAX] = {'\0'};
    int i, j, top;
    for(i = 0, j = 0, top = 0; infix[i] != '\0'; i++){
```

```
26.              switch(infix[i]) {
27.         case '(':               // 運算子堆疊
28.             stack[++top] = infix[i];
29.             break;
30.         case '+': case '-': case '*': case '/':
31.             while(priority(stack[top]) >= priority(infix[i])) {
32.                 postfix[j++] = stack[top--];
33.             }
34.             stack[++top] = infix[i]; // 存入堆疊
35.             break;
36.         case ')':
37.             while(stack[top] != '(') { // 遇 ) 輸出至 (
38.                 postfix[j++] = stack[top--];
39.             }
40.             top--;  // 不輸出 (
41.             break;
42.         default:  // 運算元直接輸出
43.             postfix[j++] = infix[i];
44.         }
45.     }
46.     while(top > 0) {
47.         postfix[j++] = stack[top--];
48.     }
49. }
50.
51. int priority(char op) {
52.     switch(op) {
53.         case '+': case '-': return 1;
54.         case '*': case '/': return 2;
55.         default:            return 0;
56.     }
57. }
58.
59. double eval(char* infix) {
60.     char postfix[MAX]= {'\0'};
61.     char opnd[2] = {'\0'};
62.     double stack[MAX] = {0.0};
63.
64.     inToPostfix(infix, postfix);
65.
66.     int top, i;
67.     for(top = 0, i = 0; postfix[i] != '\0'; i++){
68.          switch(postfix[i]) {
69.         case '+': case '-': case '*': case '/':
70.             stack[top - 1] = cal(postfix[i], stack[top - 1], stack[top]);
71.             top--;
72.             break;
73.         default:
74.             opnd[0] = postfix[i];
75.             stack[++top] = atof(opnd); //將字串中的數字轉換為 double 型態的浮點
    數
76.      }
77.         }
78.
79.     return stack[top];
80. }
81.
82. double cal(char op, double p1, double p2) {
83.     switch(op) {
84.         case '+': return p1 + p2;
85.         case '-': return p1 - p2;
86.         case '*': return p1 * p2;
87.         case '/': return p1 / p2;
88.     }
89. }
```

劉千榮 以 linked list 及 Stack 實作(僅支援單位元輸入的運算)

```c
#include <stdio.h>
#include <stdlib.h>

// 定義建立堆疊與串列所需的結構 (Infix 轉 Postfix 用)
typedef struct StackNode
{
    char value;
    struct StackNode *nextPtr;
} stacknode;

// 重新定義結構之一階指標的名稱
typedef struct StackNode *stackPtr;

// 插入串列尾端
void insertlist(stackPtr *nodePtr, char info);
// 輸出串列第一的元素
char poplist(stackPtr *topPtr);
// 輸入資料進入堆疊
void push(stackPtr *topPtr, char info);
// 輸出堆疊中最上層的元素
char pop(stackPtr *topPtr);
// 判斷堆疊是否為空
int isEmpty(stackPtr topPtr);
// 顯示串列的所有元素
void printlist(stackPtr currentPtr);
// 優先權判斷
int priority(char op);

// 定義建立堆疊所需的結構 (計算 Postfix 用)
typedef struct CalculateNode
{
    double value;
    struct CalculateNode *nextPtr;
} calculateNode;

// 重新定義結構之一階指標的名稱
typedef struct CalculateNode *calculatePtr;

// 輸入資料進入堆疊
void pushDouble(calculatePtr *topPtr, double info);
// 輸出堆疊中最上層的元素
double popDouble(calculatePtr *topPtr);
// 對堆疊中數字進行計算
void calculate(calculatePtr *topPtr, char sign);

int main()
{
    // 建立堆疊(儲存運算符號)
    stackPtr stackNodePtr = NULL;
    // 建立串列(儲存傳換成 postfix 後的數字與符號)
    stackPtr postfixHeadPtr = NULL;
    // 逐字讀取的容器
    char value;

    // 系統輸出 "Please enter the formula : " 在 Console 上
    printf("Please enter the formula : \n");
    // 在讀取到換行符'\n'之前逐字讀取字元
    while ((value = getchar()) != '\n')
    {
        // 判斷讀入的資訊為合
        switch (value)
```

```
62.         {
63.         // 若讀入左括號
64.         case '(':
65.
66.             // 直接推入堆疊中
67.             push(&stackNodePtr, value);
68.             break;
69.
70.         // 若讀入右括號
71.         case ')':
72.
73.             // 在讀取到左括號前，依序 pop 出堆疊中的運算符
74.             while (stackNodePtr->value != '(')
75.             {
76.                 insertlist(&postfixHeadPtr, pop(&stackNodePtr));
77.             }
78.
79.             // 建立零時容器吸收不須輸出的左括號
80.             char temp = pop(&stackNodePtr);
81.             break;
82.
83.         default:
84.
85.             // 若讀取到數字(字元的 ascii 碼為 48 ~ 57)
86.             if ((int)value >= 48 /* 0 */ && (int)value <= 57 /* 9 */)
87.             {
88.                 // 直接將數字推入存 postfix 的串列中
89.                 insertlist(&postfixHeadPtr, value);
90.                 break;
91.             }
92.             // 若讀取到符號
93.             else
94.             {
95.                 // 若串列不是空的
96.                 if (!isEmpty(stackNodePtr))
97.                 {
98.                     // 若串列中的符號優先權大於讀入的 或 串列中的符號為左括號
99.                     if (priority(value) < priority(stackNodePtr-
    >value) || stackNodePtr->value == '(')
100.                    {
101.                        // 直接推入堆疊中
102.                        push(&stackNodePtr, value);
103.                    }
104.                    else
105.                    {
106.                        // 先將運算符推入存 postfix 的串列中
107.                        insertlist(&postfixHeadPtr, pop(&stackNodePtr));
108.                        // 再將讀入的運算符直接推入堆疊中
109.                        push(&stackNodePtr, value);
110.                    }
111.                }
112.                else
113.                {
114.                    // 直接推入堆疊中
115.                    push(&stackNodePtr, value);
116.                }
117.            }
118.        }
119.    }
120.
121.    // 讀取結束後，將堆疊中的運算符全部依序輸出
122.    while (stackNodePtr != NULL)
123.    {
124.        insertlist(&postfixHeadPtr, pop(&stackNodePtr));
```

```
125.     }
126.
127.     // 在 console 上顯示已排列後的 postfix 算式
128.     printlist(postfixHeadPtr);
129.
130.     // 建立堆疊 (計算 postfix 算式)
131.     calculatePtr calculateTop = NULL;
132.
133.     // 當讀取到的串列位址為空前，不停進行計算
134.     while (postfixHeadPtr != NULL)
135.     {
136.         // 建立暫存容器，讀取從 postfix 串列輸出的字元
137.         char temp = poplist(&postfixHeadPtr);
138.
139.         // 判斷字元為數字或字元為
140.         switch (temp)
141.         {
142.         // 字元為運算符
143.         case '+':
144.         case '-':
145.         case '*':
146.         case '/':
147.         case '%':
148.         case '>':
149.         case '<':
150.         case '&':
151.         case '|':
152.         case '^':
153.             // 將堆疊中的數字進行相對應的運算
154.             calculate(&calculateTop, temp);
155.             break;
156.
157.         // 字元為數字
158.         default:
159.             for (int i = 48; i <= 57; i++)
160.             {
161.                 // 若讀入的數字符合該數字的 ascii 碼
162.                 if ((int)temp == i)
163.                 {
164.                     // 將數字算出後再強制轉型成 double，最後推進堆疊中
165.                     pushDouble(&calculateTop, (double)(i - 48));
166.                     break;
167.                 }
168.             }
169.             break;
170.         }
171.     }
172.
173.     // 在 console 上輸出答案
174.     printf("\n%.13lf", popDouble(&calculateTop));
175. }
176.
177. void insertlist(stackPtr *headPtr, char info)
178. {
179.     // 建立新的節點
180.     stackPtr newPtr = (stackPtr)malloc(sizeof(stacknode));
181.
182.     // 記憶體擁有足夠的空間
183.     if (newPtr != NULL)
184.     {
185.         // 將欲插入的字元存入節點中
186.         newPtr->value = info;
187.         // 將指向下的節點的指標指為空
188.         newPtr->nextPtr = NULL;
```

```
189.
190.         // 將串列的首位指標設為當前指標位置
191.         stackPtr currentPtr = *headPtr;
192.         // 定義前個指標為空
193.         stackPtr previousPtr = NULL;
194.
195.         // 將當前指標位置移動至串列最尾端
196.         while (currentPtr != NULL)
197.         {
198.             // 將前個指標的位置設為當前指標
199.             previousPtr = currentPtr;
200.             // 將當前指標位置向下個節點移動
201.             currentPtr = currentPtr->nextPtr;
202.         }
203.
204.         // 若串列中沒有半個元素
205.         if (previousPtr == NULL)
206.         {
207.             // 將新節點所指向的下個者標位置指向讀入串列的頭(NULL)
208.             newPtr->nextPtr = *headPtr;
209.             // 將元串列的位置指項新節點
210.             *headPtr = newPtr;
211.         }
212.         else
213.         {
214.             // 將前一個指標的所指向的下個位置指向新增的節點
215.             previousPtr->nextPtr = newPtr;
216.             // 將新節點的所指向的下個指標指向當前節點
217.             newPtr->nextPtr = currentPtr;
218.         }
219.     }
220.     // 記憶體空間用盡
221.     else
222.     {
223.         printf("%c not inserted. No memory available.\n", info);
224.     }
225. }
226.
227. char poplist(stackPtr *topPtr)
228. {
229.     // 建立暫存指標指向串列的頭
230.     stackPtr tempPtr = *topPtr;
231.
232.     // 讀取串列頭所儲存的元素
233.     char output = tempPtr->value;
234.     // 將串列頭的指標指向串列的下一個元素
235.     *topPtr = (*topPtr)->nextPtr;
236.     // 釋放串列頭的記憶體空間
237.     free(tempPtr);
238.
239.     // 輸出串列頭所儲存的元素
240.     return output;
241. }
242.
243. void push(stackPtr *topPtr, char info)
244. {
245.     // 建立新的節點
246.     stackPtr newPtr = (stackPtr)malloc(sizeof(stacknode));
247.
248.     // 若還有記憶體空間
249.     if (newPtr != NULL)
250.     {
251.         // 將欲插入的字元存入新節點中
252.         newPtr->value = info;
```

```
253.         // 將新節點所指向的下個位置指向當前堆疊的頂端
254.         newPtr->nextPtr = *topPtr;
255.         // 將堆疊的頂端位置指向剛建立的節點
256.         *topPtr = newPtr;
257.     }
258.     // 記憶體空間不足
259.     else
260.     {
261.         printf("%c not inserted. No memory available.\n", info);
262.     }
263.}
264.
265.// pushDouble 函式只是推入堆疊的資料改成 double 型態，其運作內容相同於 push 函式
266.void pushDouble(calculatePtr *topPtr, double info)
267.{
268.     calculatePtr newPtr = (calculatePtr)malloc(sizeof(calculateNode));
269.
270.     if (newPtr != NULL)
271.     {
272.         newPtr->value = info;
273.         newPtr->nextPtr = *topPtr;
274.         *topPtr = newPtr;
275.     }
276.     else
277.     {
278.         printf("%lf not inserted. No memory available.\n", info);
279.     }
280.}
281.
282.char pop(stackPtr *topPtr)
283.{
284.     // 建立暫存指標指向堆疊的頂端
285.     stackPtr tempPtr = *topPtr;
286.
287.     // 讀取堆疊頂端所儲存的元素
288.     char popValue = (*topPtr)->value;
289.     // 將堆疊頂端的指標指向堆疊內的下一個元素
290.     *topPtr = (*topPtr)->nextPtr;
291.     // 釋放堆疊頂端節點的記憶體空間
292.     free(tempPtr);
293.
294.     // 輸出堆疊頂端所儲存的元素
295.     return popValue;
296.}
297.
298.// popDouble 函式只是推入堆疊的資料改成 double 型態，其運作內容相同於 pop 函式
299.double popDouble(calculatePtr *topPtr)
300.{
301.     calculatePtr tempPtr = *topPtr;
302.
303.     double popValue = (*topPtr)->value;
304.     *topPtr = (*topPtr)->nextPtr;
305.     free(tempPtr);
306.
307.     return popValue;
308.}
309.
310.int isEmpty(stackPtr topPtr)
311.{
312.     // 檢查串列頂端是否為空
313.     return topPtr == NULL;
314.}
315.
316.void printlist(stackPtr currentPtr)
317.{
```

```
318.     // 檢查當前串列是否為空
319.     if (currentPtr == NULL)
320.     {
321.         puts("The stack is empty.\n");
322.     }
323.     else
324.     {
325.         // 將當前串列位置往後移直到到達尾端
326.         while (currentPtr != NULL)
327.         {
328.             // 輸出該串列元素
329.             printf("%c", currentPtr->value);
330.             // 將當前串列位置往後移
331.             currentPtr = currentPtr->nextPtr;
332.         }
333.     }
334. }
335. 
336. int priority(char op)
337. {
338.     // 檢查該運符之優先權
339.     switch (op)
340.     {
341.     case '*':
342.     case '/':
343.     case '%':
344.         return 5;
345. 
346.     case '+':
347.     case '-':
348.         return 6;
349. 
350.     case '>':
351.     case '<':
352.         return 8;
353. 
354.     case '&':
355.         return 10;
356. 
357.     case '^':
358.         return 11;
359. 
360.     case '|':
361.         return 12;
362. 
363.     default:
364.         return 0;
365.     }
366. }
367. 
368. void calculate(calculatePtr *topPtr, char sign)
369. {
370.     // 分別讀出堆疊中的頂端兩組數字
371.     double temp2 = popDouble(topPtr);
372.     double temp1 = popDouble(topPtr);
373. 
374.     // 對不同運算符對兩數字進行相對應的運算
375.     switch (sign)
376.     {
377.     case '+':
378.         pushDouble(topPtr, temp1 + temp2);
379.         break;
380. 
381.     case '-':
382.         pushDouble(topPtr, temp1 - temp2);
```

```
383.         break;
384.
385.     case '*':
386.         pushDouble(topPtr, temp1 * temp2);
387.         break;
388.
389.     case '/':
390.         pushDouble(topPtr, temp1 / temp2);
391.         break;
392.
393.     case '%':
394.         pushDouble(topPtr, (double)((int)temp1 % (int)temp2));
395.         break;
396.
397.     case '>':
398.         pushDouble(topPtr, temp1 > temp2);
399.         break;
400.
401.     case '<':
402.         pushDouble(topPtr, temp1 < temp2);
403.         break;
404.
405.     case '&':
406.         pushDouble(topPtr, (double)((int)temp1 & (int)temp2));
407.         break;
408.
409.     case '^': // bitwise xor
410.         pushDouble(topPtr, (double)((int)temp1 ^ (int)temp2));
411.         break;
412.
413.     case '|':
414.         pushDouble(topPtr, (double)((int)temp1 | (int)temp2));
415.         break;
416.
417.     default:
418.         break;
419.     }
420.        }
```