106-1 Data Structure Program Assignment 1 Report

104403512 吳宛儒

使用語言 C++

解題模式

第一題 (檔案:104403512_1.cpp)

Step1. 先定義堆疊資料結構，包含 top 值與 items 陣列

```cpp
#include<iostream>
#include<fstream>
using namespace std;
#define MAXLEN 100

// C++
// 定義堆疊資料結構
struct stack{
    int top=0;
    char items[MAXLEN];
};
```

Step2. 宣告 line 陣列，以一行為單位存取每一行的資訊(式子)

```cpp
// 存取檔案每一行的資訊
char line[MAXLEN];
```

Step3. 定義堆疊所要用到的函數，包含 IsFull, IsEmpty, Pop, Push

```cpp
// If stack is full
int IsFull(stack *s){
    if(s->top>=MAXLEN){
        return 1;
    }else{
        return 0;
    }
}

// If stack is empty
int IsEmpty(stack *s){
    if(s->top==0){
        return 1;
    }else{
        return 0;
    }
}
```

```cpp
// Pop the stack
char Pop(stack *s){
    if(IsEmpty(s)){
//      printf("Stack is empty!\n");
//      printf("Lack of ( ! \n");
        return ' ';
    }else{
        char data = s->items[s->top-1];
        s->top--;
        return data;
    }
}

// Push the stack
void Push(char item,stack *s){
    if(IsFull(s)){
//      printf("Stack is full!\n");
//      printf("Lack of ) ! \n");
    }
    else{
        (s->top)++;
        s->items[s->top-1]=item;

    }
}
```

注 : printf 這裡的提示不需要，故註解

Step4. 撰寫主程式，先開啟兩個檔案 file1&file2，前為 input1.txt，後為 output1.txt; `ios::trunc` 的意思是覆蓋已有的內容

```cpp
// Read the file
int main(){
    fstream file1,file2;

// Open the file to be read
    file1.open("input1.txt",ios::in);

// Open the file to write
    file2.open("output1.txt", ios::out | ios::trunc);

```

Step5. While，一次讀一行 file1；每次讀的時候宣告一個堆疊

```cpp
// Read one line each time
    while(file1.getline(line,sizeof(line),'\n')){

        // Declare a stack
        stack s;

```

Step6. For 迴圈，每一次讀一個字元；如果是左括號就 push 進堆疊，如果是又括號就 pop 堆疊，如果是等於最後一個字元，視為一行的結束；其他遇到數字等情況，跳過、不做任何處理

```
76          // Read each character one time
77          for(int i=0;i<sizeof(line);i++){
78
79              if(line[i]=='('){
80                  // If meet lparen, push it in the stack
81                  Push('(',&s);
82              }else if(line[i]==')'){
83                  // If meet rparen, pop the stack
84                  Pop(&s);
85              }else if(line[i]==line[99]){
86                  // If we meet the end
87                  break;
88              }
89              // 其他可能會遇到數字等，為else情況，這裡不做任何處理
90
91          }
92          printf("\n");
```

Step7. 判斷堆疊最後跑完一行以後是否有剩下東西，也就是這運算式是否 valid；若還有東西，代表左右括號數量不等，為 invalid，反之，則 valid

```
94  //      To determine it's valid or not
95          if(IsEmpty(&s)){
96              //1 as valid
97              file2<<"1\n";
98          }else{
99              //0 as invalid
100             file2<<"0\n";
101         }
102
103     }
104
```

Step8. 關閉 file1, file2 兩檔案，結束程式

```
105 //  Close two files
106     file1.close();
107     file2.close();
108
109     system("pause");
110     return 0;
111 }
```

第二題 (檔案:104403512_2.cpp)

Step1. 同第一題，先宣告 stack 結構以及 line 陣列、堆疊基本函數

```cpp
#include<iostream>
#include<fstream>
using namespace std;
#define MAXLEN 100

// 宣告stack結構
struct stack{
    int top=0;
    char items[MAXLEN];
};

// 宣告line陣列，儲存等下要讀的檔案中 每一行的式子
char line[MAXLEN];

// 堆疊基本函數
int IsFull(stack *s){
    if(s->top>=MAXLEN){
        return 1;
    }else{
        return 0;
    }
}

int IsEmpty(stack *s){
    if(s->top==0){
        return 1;
    }else{
        return 0;
    }
}
char Pop(stack *s){
    if(IsEmpty(s)){
        printf("Stack is empty!\n");
        return ' ';
    }else{
        char data = s->items[s->top-1];
        (s->top)--;
        return data;
    }
}

void Push(char item,stack *s){
    if(IsFull(s)){
        printf("Stack is full!\n");
    }
    else{
        (s->top)++;
        s->items[s->top-1]=item;

    }
}
```

Step2. 同第一題，打開要讀取的以及要寫入的檔案

```cpp
//read the file
int main(){
    fstream file1,file2;
//    open input2.txt to read the lines
    file1.open("input2.txt",ios::in);
//    open output2.txt to write things in
    file2.open("output2.txt", ios::out | ios::trunc);
```

Step3. 中序轉後序式：建立堆疊 s 以及將結果存入的 postfix 陣列

```cpp
//  read one line every time
    while(file1.getline(line,sizeof(line),'\n')){

        // infix to postfix 中序式轉後序式

        // 宣告s這個堆疊
        stack s;

        // 初始化postfix陣列
        char postfix[MAXLEN]={};

        // 為postfix陣列的index值，以一個字元一為單位存入後序式
        int index = 0;
```

Step4. 一次讀一行，判斷讀到的是(、)、運算子或運算元

```
76          // 一次讀一行
77          for(int n=0;n<sizeof(line);n++){
78
79              // 設字元變數token為目前所讀到的字元
80              char token = line[n];
81
82              // 如果token等於最後一個字元，視為運算式的結束
83              if(token == line[MAXLEN])
84                  break;
85
86              if(token == ')'){
87                  // when meet rparen ')'
88                  // pop the stack until we meet '('
89                  while (s.items[s.top-1] != '(' && !IsEmpty(&s)){
90                      // 將pop出來的結果存入postfix陣列
91                      postfix[index++] = Pop(&s);
92                  }
93                  // pop the '('
94                  Pop(&s);
95              }
96              else if(token == '('){
97                  // when we meet '('
98                  // push it into the stack directly
99                  Push(token,&s);
00              }
```

```
101             else if(token == '+' || token == '-'){
102                 // if we meet + or -
103                 // if the top of stack is * or / , pop it
104                 while(s.items[s.top-1]=='*'||s.items[s.top-1]=='/'||s.items[s.top-1]=='+'||s.items[s.top-1]=='-'){
105                     // pop出來並存入postfix陣列
106                     postfix[index++] = Pop(&s);
107                 }
108                 // 把現在遇到的這個+或-放進堆疊
109                 Push(token,&s);
110             }
111             else if(token == '*' || token == '/'){
112                 // if we meet * or /
113                 // if the top of stack is * or /
114                 while(s.items[s.top-1]=='*'||s.items[s.top-1]=='/'){
115                     // pop出來並存入postfix陣列
116                     postfix[index++] = Pop(&s);
117                 }
118                 // 把現在遇到的這個*或/放進堆疊
119                 Push(token,&s);
120             }
121             else{
122                 // if we meet operand
123                 postfix[index++] = token;
124             }
125         }
```

Step5. 將堆疊中剩餘的 pop 出來並存入 postfix 陣列

```
127         // 做完前面的操作，如果堆疊非空，將剩餘的pop出來存入postfix陣列
128         while(!IsEmpty(&s)){
129             postfix[index++] = Pop(&s);
130         }
131
```

Step6. 將後序式寫入 file2(也就是 output2.txt)

```
132         // 將postfix後序式寫入file2檔案並換行
133         file2 << postfix;
134         file2 << "\n";
```

Step7. 後序式求值，宣告一個堆疊 post

```
136         // postfix to value 後序式求值
137
138         //宣告一個堆疊 post
139         stack post;
140
```

Step8. 將剛剛的結果，一次讀一個字元，將目前讀到的字元存入 token 變數，並宣告兩個整數運算元 op1、op2

```
141         // 以一次一個字元為單位 讀剛剛存入postfix陣列的值
142         for(int i=0;i<sizeof(postfix);i++){
143
144             // 宣告 token 變數來存放目前讀到的字元
145             char token = postfix[i];
146
147             // 宣告op1、op2兩個整數運算元
148             int op1, op2;
149
150             // 當目前讀到的字元和最後一個字元一樣 視為後序式的結束
151             if(token == postfix[MAXLEN])
152                 break;
153
```

Step9. 判斷讀到的字元是運算子還是運算元，並作相對應運算，再將結果 push 回堆疊

※ 首先，要將 op2、op1 以加上" -'0'"的方式，將 pop 出來的東西字串轉整數，才可以運算；push 結果進去的時候要把整數轉成字串才能放入堆疊(因為先前宣告的堆疊裡面的陣列是存字串的)

```
154             // 如果讀到運算子
155             if( token =='+' || token =='-' || token =='*' || token =='/'){
156 
157                 // pop 出來堆疊最上面的兩個運算元
158                 // 並將字串轉為整數 -'0'
159                 op2= Pop(&post)-'0';
160                 op1 = Pop(&post)-'0';
161 
162                 // 判斷目前讀到的是哪一個運算子
163                 switch(token){
164                 case '+': Push ((op1+op2)+'0',&post);
165                           break;
166                 case '-': Push ((op1-op2)+'0',&post);
167                           break;
168                 case '*': Push ((op1*op2)+'0',&post);
169                           break;
```

※ 這邊要注意，因為考量整數相除會有不整除的結果，於是有了以下四捨五入

```
170                 case '/':
171                 // 如果是除號，有可能出現小數的結果
172                 // 先將結果以float存放到result變數內
173                 float result = (float)op1/(float)op2 ;
174                 // 宣告一個num整數變數
175                 int num;
176                 // 當op1/op2不整除的時候
177                 if(op1%op2 != 0){
178                     // 將result+0.5存入num 當作四捨五入
179                     // 因為0.4+0.5不會跳到下一數，0.5以上+0.5會跳到下一數
180                     // 用num整數就會完全捨棄小數部分，剛好就是我們要的四捨五入的值
181                     num = result + 0.5;
182                 }else{
183                     // 整除的話，就直接把num設為result結果
184                     num = result;
185                 }
186                 //  將結果num整數轉字串，push進堆疊
187                 Push (num +'0',&post);
188                           break;
189                 }
190 
```

※ 如果是遇到運算元，就直接 push 進堆疊

```
191                 }else{
192                     // 如果讀到運算元
193                     // push進堆疊
194                     Push(token, &post);
195 
196                 }
```

Step10. 最後留在堆疊的就是後序式求值結果，將其字串轉成整數寫入 file2

```
200         // 最後留在堆疊的是運算結果
201         // 將結果字串轉整數 寫入file2
202         file2 << Pop(&post)-'0';
203         file2 << "\n";
204     }
```

Step11. 結束，關閉檔案

```
206 //     關閉檔案
207     file1.close();
208     file2.close();
```

所遇問題

第一題

第一題相對比較容易，主要是之前並沒有寫過 c++讀檔寫檔的功能，上網查了一下才知道。

第二題

第二題本來上課聽了覺得課本的方法很有效率、很酷，想要試試看，後來覺得還是以自己的思考模式寫寫看自己的，所以跳過優先權的設定，直接去判斷遇到的字元是甚麼。

遇到的問題大概就是

(1)因為 stack 裡面 top 的初始值設為 0，所以查看堆疊最上面的元素應該是要找 `s.items[s.top-1]`，之前一開始忘了減一。

(2)stack 裡面的陣列設定是存字串的陣列，所以必須先把裡面的字串轉成整數計算結果後才能再轉字串 push 進堆疊，之前一直沒發現這個卡了許久。

(3)四捨五入本來以為會很複雜，後來想到+0.5 的方法也是蠻直覺方便的，以後想再試試看 rounding 函數。

作業心得

之前大二上有修過資管系上的資料檔案結構，一直沒有實作機會，雖然了解邏輯但沒有實作根本不會實用；這學期修了資工的這門課，終於有機會打程式了，覺得很開心能夠有實作機會，也期待自己能好好學習並生存下去。