Home work #1

1. Delete Strictly Increasing Sequences

用於判斷是否為數字字元('0'~'9')

```
#define isdigit(x) (((unsigned int)x - '0') < 10u)
```

用於輸出數字，因 printf 較慢，故以此替代

```
void print_num(int n){//只能丟正數進來
    if (n > 9){
        int a = n / 10;
        n -= 10 * a;
        print_num(a);
    }
    putchar('0' + n);
}
```

讀取題目的條件 N 與 T(M)，使用 getchar()讀，因 scanf 過慢，讀到非數字就停止

```
    while (c = getchar(),isdigit(c)) N = N * 10 + c - '0';
    while (c = getchar(),isdigit(c)) M = M * 10 + c - '0';
```

配置記憶體與主要運作的 for 迴圈

```
    int* arr = (int*)malloc(N * sizeof(int));
    for (i = 0; i < N; ++i) {
          …
     }
```

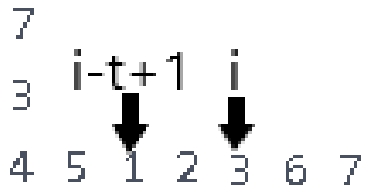
讀取輸入數列中的數字。

```
        while (c = getchar(), c == ' ');//跳過所有空白
        if (c != '-') {
            arr[i] = c - '0';//先c裡的數字取出，順便覆蓋arr[i]，因malloc不會初始化
            while (c = getchar(), isdigit(c)) arr[i] = arr[i] * 10 + c - '0';
            //讀到不是數字為止，邊讀邊加
        }
        else {//讀取到負號
            arr[i] = getchar() - '0';//再拿新的字元出來，順便覆蓋arr[i]
            while (c = getchar(), isdigit(c)) arr[i] = arr[i] * 10 + c - '0';
            //讀到不是數字為止，邊讀邊加
            arr[i] = ~arr[i] + 1;//將arr[i]乘以-1，用反相後+1，速度更快
        }
```

核心程式碼

```
        if (i < M - 1) continue;
        for (j = 1; j < M && arr[i - j + 1] > arr[i - j];++j);
        if (j == M) {
            N -= M;
            i -= M;
        }
```

`if (i < M - 1)`代表前面的數字不夠下面的 for 迴圈去檢查，所以直接`continue;`進行下一個數字的回合，數字足夠，就會跑`for (j = 1; j < M && arr[i - j + 1] > arr[i - j];++j);`若在中途被`arr[i - j + 1] > arr[i - j]`給 break 掉，就代表從 arr 的 index 為 i 到 i-t+1 之間並非遞增數列，反之，如下圖則為遞增數列



則會跑完整個 for 迴圈，導致`j == M`，接著就要刪除此遞增數列，但不須實際刪除記憶體空間，只需把 i(index)指到
i-M 處，下一次讀入數字就被覆蓋掉了，因為刪除了數列，所以數列總數 N 也要減去 M

```
        if (j == M) {
            N -= M;
            i -= M;
        }
```

輸出結果

```
    for (i = 0; i < N;++i) {//輸出結果
        if (arr[i] < 0) {//若為負數
            putchar('-');//印出負號
            arr[i] = ~arr[i] + 1;//乘上負1，轉為正數，因為print_num，只能使用正數
        }
        print_num(arr[i]);//呼叫print_num，印出數字
        putchar(' ');//每個數字間的空白
    }
    free(arr);//釋放記憶體
```

2. Calculator

一些實用的 macro，讓程式碼更簡潔，用法如同之後程式碼。

```
#define MAX_SIZE 71//stack最大容量，依題目所訂
#define PUSH (size++)//push新東西進stack，並將size+1
#define POP  (--size)// 先將size-1，就會回傳最上面的東西，相當於pop
#define TOP  (size-1)//回傳stack最上面的東西
#define isdigit(x) ((x)>='0' && (x)<='9')//判斷是否為數字字元
```

儲存輸入字串與轉後序使用的 stack

```
char input[MAX_SIZE];
char stack[MAX_SIZE];
```

預先處理動作

```
out output;
int size = 0, len = my_strlen(input);
if (!check_brackets(input, len) || !check_operator(input, len)) {
     cout << "invalid" <<endl;
     continue;
}
input[len] = ')'; input[len + 1] = '\0';++len;
stack[PUSH] = '(';
```

output 將轉後序的結果存入 output 內的 stack，因為之後要算出結果，所以不能直接印出，`my_strlen(input);`回傳 string 長度，內部實作:

```
int my_strlen(char* input) {
     int index = 0;
     while (index != MAX_SIZE-1 && input[index] != '\0') index++;
     return index;
}
```

判斷是否 invalid，我使用 2 個 function 去 check，分別為檢查括號匹配的 `bool check_brackets(char* input,int length)`與檢查運算子左右兩邊是否有運算元的 `bool check_operator(char* input,int length)`以下是內部實作:

```
bool check_brackets(char* input,int length) {
	int count = 0;//遇到左括號+1，右括號-1
	for (int i = 0;i < length;i++) {
		switch (input[i]) {
			case'(': {
				count++;
			}break;
			case')': {
				if (!count) return false;
//若遇到右括號時count為0，代表沒有對應的左括號，或前面的左括號，都已被前面的右括號匹配走了
				count--;
			}break;
			default:break;
		}
	}
	if (count) return false;//括號若匹配正確count應是0，若不是 return false
	return true;
}
bool check_operator(char* input,int length) {
	if (input[0] == '+' || input[0] == '-' ||
		input[0] == '*' || input[0] == '/' ||
		input[length - 1] == '+' || input[length - 1] == '-' ||
		input[length - 1] == '*' || input[length - 1] == '/')return false;
//因為下面的for迴圈會用到index-1與index+1，所以先把開頭與結尾的字元過濾掉( 第一個字元與最後一個不可為任何運算子 )，for迴圈就能從1開始跑到倒數第二個了，
	for (int i = 1;i < (length-1);i++) {
		switch (input[i])
		{
			case'+':case'-':case'*':case'/': {//用switch case 看起來比較乾淨
				if ((!isdigit(input[i - 1]) && input[i - 1] != ')') ||
					(!isdigit(input[i + 1]) && input[i + 1] != '(')) return false;
//判斷運算子左邊與右邊是否為對應括號或數字，相當於判斷左右邊是否有運算元
			}break;

			default:
				break;
		}
	}
	return true;//若正確就會完整跑完到這裡
}
```

以上 2 個函式都回傳 true 後，就開始演算法的步驟，先放入'('到 stack 內，並在 input 最後加上')'，因為加入新字元，故++len，'\0'推到下一格

```
input[len] = ')'; input[len + 1] = '\0';++len;
stack[PUSH] = '(';
```

主要 for 迴圈，跑完整個 input 即可

```
for (int i = 0;i < len;i++) {
    …
}
```

數字處理

```
int index = i;//用index跑數字之後才能判斷是否有讀入數字
while (isdigit(input[index])) {//若為數字輸出到output，此處已重載<<符號
    output << input[index];
    index++;
}
if (index != i) {
//代表有讀入數字，再輸出一個空白進去，以便之後計算區隔所有數字
    output << ' ';
    i = index;
}
```

output 重載<<，這裡先將 class 內的 output 陣列當作 stack 用，與之後計算數字用的 stack 不同。

```
class out {
public:
    char output[150];//不能只設70，因為很多空白要存
    int stack[MAX_SIZE];
    int size = 0;
    out& operator<<(char a) {
        output[PUSH] = a;
        return *this;
    }
```

回到主迴圈，接下來是演算法的核心

```
switch (input[i])
{
    case'(': {
        stack[PUSH] = '(';//遇到左括號就push進stack
    }break;
    case')': {//遇到右括號就pop並輸出直到遇到左括號
        char tmp;
        while (tmp = stack[POP], tmp != '(') output << tmp << ' ';
    }break;
    case'+':case'-':case'*':case'/':{
//運算子透過priority比較stack最上層的優先權，若大於則pop掉，直到遇到比自己大的就停止
    while (priority(input[i]) >= priority(stack[TOP])) output << stack[POP] << ' ';
        //最後將自己push進stack
        stack[PUSH] = input[i];
    }break;
    default:
        break;
}
```

priority function 的內部實作:

```
int priority(char op) {
    switch (op) {
    case '*': case '/': return 0;
    case '+': case '-': return 1;
    default:            return 2;
    }
}
```

主迴圈結束後，已成功完成後序轉換，呼叫 `output.print_result();`即可。

內部實作:

```
    void print_result() {
        output[TOP] = '\0';
//There is no space at the end of line.把最後一格空白覆蓋掉
        cout << output << endl;//輸出後序表示式
        int postfix_len = size;//因為size還要被stack運用，所以存到另一個變數裡
        size = 0;//之前的size被拿來計算index用，所以要歸零
        for (int i = 0;i < postfix_len;i++) {//後序轉數字演算法
            int index = i;
            int num = 0;
            int A, B;
            while (isdigit(output[index])) {//遇到數字就把它算出來
                num = num * 10 + output[index] - '0';
                index++;
            }
            if (index != i) {//代表有讀到數字
                stack[PUSH] = num;//push到stack
                i = index;//把i指到數字結束的地方
            }
            //若遇到運算子，拿出stack上面2個A(最上) B(第二上)，並進行B <運算子> A的
運算
            switch (output[i])
            {
            case'+': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B + A;
            }break;
            case'-': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B - A;
            }break;
            case'*': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B * A;
            }break;
            case'/': {
```

```
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B / A;
            }break;
            case'%': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B % A;
            }break;
            default:
                break;
            }
        }
        //輸出數字主迴圈結束後stack最上面就是答案
        cout << stack[TOP] <<endl;
    }
```

3. 完整程式碼連結(gist)
   Delete Strictly Increasing Sequences
   Calculator
   imgur 圖片

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#define isdigit(x) (((unsigned int)x - '0') < 10u)
void print_num(int n){//只能丟正數進來
    if (n > 9){
        int a = n / 10;
        n -= 10 * a;
        print_num(a);
    }
    putchar('0' + n);
}
int main(void) {
    int N = 0, M = 0, i, j;
    char c;
    while (c = getchar(),isdigit(c)) N = N * 10 + c - '0';
    while (c = getchar(),isdigit(c)) M = M * 10 + c - '0';
    int* arr = (int*)malloc(N * sizeof(int));
    for (i = 0; i < N; ++i) {
        while (c = getchar(), c == ' ');//跳過所有空白
        if (c != '-') {
            arr[i] = c - '0';//先c裡的數字取出，順便覆蓋arr[i]，因malloc不會初始化
            while (c = getchar(), isdigit(c)) arr[i] = arr[i] * 10 + c - '0';//讀到不是數字為止，邊讀邊加
        }
        else {//讀取到負號
            arr[i] = getchar() - '0';//再拿新的字元出來，順便覆蓋arr[i]
            while (c = getchar(), isdigit(c)) arr[i] = arr[i] * 10 + c - '0';//讀到不是數字為止，邊讀邊加
            arr[i] = ~arr[i] + 1;//將arr[i]乘以-1，用反相後+1，速度更快
        }


        /*
        if (i < M - 1)代表前面的數字不夠下面的for迴圈去檢查，所以直接continue;進行下一個數字的回合，
        數字足夠，就會跑for (j = 1; j < M && arr[i - j + 1] > arr[i - j];++j);
        若在中途被arr[i - j + 1] > arr[i - j]給break掉，就代表從arr的index為i到i-t+1之間並非遞增數列*/

        if (i < M - 1) continue;
        for (j = 1; j < M && arr[i - j + 1] > arr[i - j];++j);
        if (j == M) {
            N -= M;
            i -= M;
        }
        /*則會跑完整個for迴圈，導致j == M，接著就要刪除此遞增數列，但不須實際刪除記憶體空間，
        只需把i(index)指到i-M處，下一次讀入數字就被覆蓋掉了，因為刪除了數列，所以數列總數N也要減去M
        */
    }
    for (i = 0; i < N;++i) {//輸出結果
        if (arr[i] < 0) {//若為負數
            putchar('-');//印出負號
            arr[i] = ~arr[i] + 1;//乘上負1，轉為正數，因為print_num，只能使用正數
        }
        print_num(arr[i]);//呼叫print_num，印出數字
        putchar(' ');//每個數字間的空白
    }
    free(arr);//釋放記憶體
    return 0;
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#define MAX_SIZE 71//stack最大容量，依題目所訂
#define PUSH (size++)//push新東西進stack，並將size+1
#define POP  (--size)//從stack pop出最上面的東西，並將size-1
#define TOP  (size-1)//回傳stack最上面的東西
#define isdigit(x) ((x)≥'0' && (x)≤'9')//判斷是否為數字字元
using namespace std;
int my_strlen(char* input);
bool check_brackets(char* input, int length);
bool check_operator(char* input, int length);
int priority(char op);

class out {
public:
    char output[150];//不能只設70，因為很多空白要存
    int stack[MAX_SIZE];
    int size = 0;
    /*output重載<<，這裡先將class內的output陣列當作stack用，與之後計算數字用的stack不同。*/
    out& operator<<(char a) {
        output[PUSH] = a;
        return *this;
    }
    void print_result() {
        output[TOP] = '\0';//There is no space at the end of line.把最後一格空白覆蓋掉
        cout << output << endl;//輸出後序表示式
        int postfix_len = size;//因為size還要被stack運用，所以存到另一個變數裡
        size = 0;//之前的size被拿來計算index用，所以要歸零
        for (int i = 0;i < postfix_len;i++) {//後序轉數字演算法
            int index = i;
            int num = 0;
            int A, B;
            while (isdigit(output[index])) {//遇到數字就把它算出來
                num = num * 10 + output[index] - '0';
                index++;
            }
            if (index ≠ i) {//代表有讀到數字
                stack[PUSH] = num;//push到stack
                i = index;//把i指到數字結束的地方
            }
            //若遇到運算子，拿出stack上面2個A(最上) B(第二上)，並進行B <運算子> A的運算
            switch (output[i])
            {
            case'+': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B + A;
            }break;
            case'-': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B - A;
            }break;
            case'*': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B * A;
            }break;
            case'/': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B / A;
            }break;
            case'%': {
                A = stack[POP];
                B = stack[POP];
                stack[PUSH] = B % A;
            }break;
            default:
                break;
            }
        }
        //輸出數字主迴圈結束後stack最上面就是答案
        cout << stack[TOP] <<endl;
    }
};
```

```
int main() {
    char input[MAX_SIZE];
    char stack[MAX_SIZE];
    while (cin >> input) {
        out output;//output將轉後序的結果存入output內的stack，因為之後要算出結果，所以不能直接印出
        int size = 0, len = my_strlen(input);
        /*判斷是否invalid，我使用2個function去check，
        分別為檢查括號匹配的bool check_brackets(char* input,int length)
        與檢查運算子左右兩邊是否有運算元的bool check_operator(char* input,int length)*/
        if (!check_brackets(input, len) || !check_operator(input, len)) {
            cout << "invalid" <<endl;
            continue;
        }
        /*
        以上2個函式都回傳true後，就開始演算法的步驟，先放入'('到stack內，並在input最後加上')'，因為加入新字元，故
++len，'\0'推到下一格*/
        input[len] = ')'; input[len + 1] = '\0';++len;
        stack[PUSH] = '(';
        for (int i = 0;i < len;i++) {
            int index = i;//用index跑數字之後才能判斷是否有讀入數字
            while (isdigit(input[index])) {//若為數字輸出到output，此處已重載<<符號
                output << input[index];
                index++;
            }
            if (index != i) {//代表有讀入數字，再輸出一個空白進去，以便之後計算區隔所有數字
                output << ' ';
                i = index;
            }
            switch (input[i])
            {
                case'(': {
                    stack[PUSH] = '(';//遇到左括號就push進stack
                }break;
                case')': {//遇到右括號就pop並輸出直到遇到左括號
                    char tmp;
                    while (tmp = stack[POP], tmp != '(') output << tmp << ' ';
                }break;
                case'+':case'-':case'*':case'/':{
                    //運算子透過priority比較stack最上層的優先權，若大於則pop掉，直到遇到比自己大的就停止
                    while (priority(input[i]) >= priority(stack[TOP])) output << stack[POP] << ' ';
                    //最後將自己push進stack
                    stack[PUSH] = input[i];
                }break;
                default:
                    break;
            }
        }
        output.print_result();
    }
    return 0;
}
```

```c
int my_strlen(char* input) {
    int index = 0;
    while (index != MAX_SIZE-1 && input[index] != '\0') index++;
    return index;
}

bool check_brackets(char* input,int length) {
    int count = 0;//遇到左括號+1，右括號-1
    for (int i = 0;i < length;i++) {
        switch (input[i]) {
            case'(': {
                count++;
            }break;
            case')': {
                if (!count) return false;
                //若遇到右括號時count為0，代表沒有對應的左括號，或前面的左括號，都已被前面的右括號匹配走了
                count--;
            }break;
            default:break;
        }
    }
    if (count) return false;//括號若匹配正確count應是0，若不是 return false
    return true;
}
bool check_operator(char* input,int length) {
    if (input[0] == '+' || input[0] == '-' || input[0] == '*' || input[0] == '/' ||
        input[length - 1] == '+' || input[length - 1] == '-' || input[length - 1] == '*' || input[length - 1]
== '/')return false;
    //因為下面的for迴圈會用到index-1，所以先把index==0的字元過濾掉( 第一個字元不可為任何運算子 )，for迴圈就能從1開始跑
了，
    for (int i = 1;i < (length-1);i++) {
        switch (input[i])
        {
            case'+':case'-':case'*':case'/': {//用switch case 看起來比較乾淨
                if ((!isdigit(input[i - 1]) && input[i - 1] != ')') || (!isdigit(input[i + 1]) && input[i +
1] != '(')) return false;
                //判斷運算子左邊與右邊是否為對應括號或數字，相當於判斷左右邊是否有運算元
            }break;

            default:
                break;
        }
    }
    return true;//若正確就會完整跑完到這裡
}

int priority(char op) {
    switch (op) {
    case '*': case '/': return 0;
    case '+': case '-': return 1;
    default:            return 2;
    }
}
```