OurC Project 3

OurC Project 3 的主要目的是(用你寫的 OurC interpreter 來) run OurC program。

除了 function 的呼叫之外，所有其他的功能都在 Proj.3 的測試範圍內。這包括以下的 operators:

=, +=, -=, *=, /=, %=, ? :, &&, ||, !, ==, !=, <, >, <=, >=, <<, >>, +, -, *, /, %

由於 PAL 封鎖&,|,^的使用，所以&,|,^不在以上測試範圍內。但 comma expression 是在測試範圍內。

在 error message 的部分，共有三種 error messages。其例如下：

```
Line 3 : unrecognized token with first char : '$'        // lexical error
Line 2 : unexpected token : '*'                          // syntactical error (token recognized)
Line 5 : undefined identifier : 'bcd'                    // semantic error (grammar ok)
```

同時，有鑒於 line number 的計算有時會有爭議，Project 3 的 test data 將會遵守以下兩項原則：

(a) 一旦有 error 發生，此“error 行”(the line where the error occurs)的下一行將不會是空行，而且下一個 user input 將不會有 error。

(b) 如果 token 是個 ID、且下一個 token 是'(' ，那麼此'(' 一定會與 ID 出現在同一行(the same line)。

至於型別錯誤(type error)的判斷，這事實上是個很大的問題。array index out of range 是不是 type error？Identifier is an array but is referenced without '[' and ']'是不是 type error？Identifier is NOT an array but is referenced with '[' and ']'是不是 type error？Array index is not an integer 是不是 type error？The test condition of if or while is not a boolean expression 是不是 type error？相關問題不勝枚舉。

因此，有關「型別錯誤(type error)的判斷」的決定如下：

(a) 沒有「型別錯誤(type error)」的 error message，亦即：test data 中所有的數據都是 guaranteed to be compatible

(b) 但你的程式要負責作 coercion (when needed)

以下是 type compatibility 的原則：

1. string is compatible with any (primitive) type with regard to '+' . // and the result is a string
2. boolean is not compatible with any other type (except string). So is char.
3. int is only compatible with float and string (and int itself).
4. float is only compatible with int and string (and float itself)

By the way, %, <<, >> 的使用、必須左右都是 integer。 Moreover, <<, >> 的右邊必須是非零 integer

再來是有關 cout 與 cin 這兩個“保留字”。

首先，你的程式應該把 cin 與 cout 當作是(default 就已宣告的)兩個變數。

但 test data 中不會有 cin，也不會宣告名字為'cin'的變數。

至於 cout，這是一個「無法取其值」的變數、我們也無法對此變數之值作任何變更。在 test data 中，每當 cout 出現，其後一定是<<，而<<之後一定是個 expression。 (test data 中不會宣告名字為'cin'的變數)

不過請注意：雖然不能 cout + 5 或 cout = 5，但可以有 cout << (cout << 5)。 // 印出'55'

也就是說，even though we cannot access the value of cout, “cout << expr” is nevertheless a valid expression in itself, and, as an expression, “cout << expr” should rightfully have a value; and the value of “cout << expr” is naturally the value of “expr.”

最後，是有關測試數據的設計(總共有 16 題)：

1, 2 : simple operations on primitives // 無 array，但涵括所有 operations，一個 expr 頂多包括 2 個 operations
// (不含 assignment)； e.g., x = x + y*z ; cout << true&&false||w ;

3, 4 : complex operations on primitives and arrays // 所謂 complex 就是很多不同 operations 放在一起、
// 有時有括號、有時沒括號，有 array

5, 6 : conditional expressions, comma expressions and arrays

7, 8 : single while, single if, no array // 單純一個 while，單純一個 if，包括區域變數的宣告，無 array

9, 10 : single while, single if + array // 單純一個 while，單純一個 if，包括區域變數的宣告，有 array

11, 12 : while with if + array // while 中有 if，if 中有 while，包括區域變數的宣告

13, 14 : nested while with if + array // 洋蔥式的 while 與 if，包括區域變數的宣告

15, 16 : with error // 1-12 沒有 error // 原則上只是把其他題的數據拿來“加工”

基本原則與先前一樣，單數題的隱藏數據的“結構”是與可見數據相同，雙數題的隱藏數據的“結構”是與可見數據類似。

========== 請忽略以下的敘述 ; 這是為將來而準備的 ==========

一些在技術層面上有點惱人的問題是 I/O。

Interpreted program 的執行與一般 program 的 execution 不太一樣。Interpreted program 的「被 key in」與程式的「被執行」幾乎是同時發生。因此，使用者如果只能運用單一的輸入管道來作輸入，他/她必須利用此輸入管道一方面輸入程式本身(請注意：從使用者的角度來看，每輸入一個 statement 或 definition，都是當場被執行或當場生效)，一方面又輸入「程式在執行時應該要讀入的 input」。那...怎樣做比較自然？

OurC 的解決之道：

加入一個 ad hoc 指令('ad hoc' = "體制外")，此 ad hoc 指令與 statement 或 definition 是同等級。所加入的 ad hoc 指令是：

(1) 'InputBufferAppend'開頭(請加入這一個保留字)
(2) 'EndInputBufferAppend'結束(請加入這一個保留字)
(3) 在此二保留字之間所 key in 的任何字元(包括 line-enter)都要加入 input buffer
(包括'EndInputBufferAppend'之前的 white-space(s)，
但 不包括 'InputBufferAppend' 之後的第一個 white space(通常是個 line-enter)!!!
)

你的程式必須在其內部 maintain 一個所謂的 input buffer (也不過就是個字串，不過要紀錄目前讀到哪裡)。每當接受到'InputBufferAppend'指令，就把其內容 append 到此 input buffer 之中。而'cin'每次讀、就是到這個 input buffer 之中去讀，如果讀不到東西或所讀到的東西不對，就會印出類似"No input!"或"Input error : x" 之類的 message，其中 x 是所偵測到的"非 expect"字元。原則上，C++的 cin 怎麼運作，你的 cin(只是個 C++的 cin 的簡化版"而已")就應該要怎麼運作。

接下來，我們談一談 cin 與 cout 的運作方式。

1. cin 只能像這樣寫(可以跨行)： // 否則是 error。
   cin >> x >> y >> x ; // 如果'x'是 boolean，所讀到的就必須是'true'或'false'，不可是 integer 或其他字串

2. cout 比照辦理(可以跨行)： // 否則是 error
   cout << "The value of " + "sum is : " << x+y*5 ;
   (注意：任何能算出值之 expression 皆可 cout；如果算出的值是個 boolean 就印出'true'或'false')

3. 以上的 cin 與 cout 的 statement 是到';'才結束，而 evaluation 是以 statement 或 definition 為"執行單位"
   (到齊了！沒問題了！才執行。)。 所以，如果 cin 與 cout 的 statement 中有任何 syntax 上的問題發生，
   將 不 會 讀入或印出任何東西。

   (是的！ 對使用者所輸入的每一 statement 或 definition， 你的程式都應該先 parse 之再 evaluate 之。
   如果第一關(syntax check)沒通過，就不會有第二關(evaluate)。
)

4. 如果到 執 行 時(才)碰到問題(e.g., cin 的"wrong input"與 cout 的"array index out of range")，

該 statement 的執行就到問題點發生之處結束。
但「在碰到問題前已 cin 進來的或已 cout 出去的」就已經"木已成舟"、無法挽回了。
(事實上，這是整個系統在 evaluation 上的通則，並不侷限於 cin 與 cout。)

5. 如果在執行'cin >> x'時碰到"無法讀入 x 值"的問題，'x'的值將不會被改變(此做法與 scanf()同)

6. 如果應該要讀一個 integer 但 input buffer 目前的字元是'a'，那麼當然會印 run-time error message。
要注意的是：下一次再去讀 input buffer 內容時，目前的字元依舊是'a'(未被上次的 read action 改變)

7. cout 指令如果沒有在最後印個 line-enter，系統的 output 看起來會略為奇怪。
例：
> cout << 5 ; // 在使用文字介面的 OS 跑此程式時，使用者要按了 ENTER
5Statement executed ... // 之後，"cout << 5 ;\n"才會被你的程式"讀到"。
>

以下是個 I/O 範例(假設是 interactive I/O)：

```
Our-C running ...
>
  int
 x
     , a[30]              ;
Definition of x entered ...
Definition of a entered ...
> cin >> x ;
Run-time error : No input when cin 'x'!
Statement executed ...
> InputBufferAppend
100  200   300
400
EndInputBufferAppend
Input-buffer appended ...
> cin >> x ;
Statement executed ...
> cout
  << x
  ;
100Statement executed ...
> cout <<
x
<< "\n"
;
100
Statement executed ...
> void AddThree(int)
Line 1 : unexpected token : ')'
```

```
> void AddThree(int& y)
{y = y + 3;} // AddThree()
Definition of AddThree() entered ...
> AddThree(x);
Statement executed ...
> cout << "Value of x is : " << x << "\n" ;
Value of x is : 103
Statement executed ...
> ListAllVariables();
a
x
Statement executed ...
> ListAllFunctions();
AddThree(int & y)
Statement executed ...
> ListVariable("x");
int x ;
Statement executed ...
> ListFunction("AddThree");
void AddThree( int & y )
{
  y = y + 3 ;
}
Statement executed ...
> int i ;
Definition of i entered ...
> i = 0 ;
Statement executed ...
> while ( i < 30 ) {
    cin >> a[ i ] ;
    i++ ;
  }
Runtime error : No input when cin 'a[ 3 ]'!
Statement executed ...
> cout << a[2] ;
400Statement executed ...
> cout << a[3] ;
0Statement executed ...     // Our-C system offers automatic initializations
> InputBufferAppend
1 2 3 4 5 6 7
EndInputBufferAppend
Input-buffer appended ...
> cout << i ;
3Statement executed ...
> while ( i < 8 ) {
    cin >> a[i] ;
    i++ ;
```

```
  }
Statement executed ...
> cout << a[7] ;
5Statement executed ...
> i = 0 ;
Statement executed ...
> while ( i < 31 ) {
    a[i] = a[i] + 1 ;
    i++ ;
  }
Runtime error : array 'a' index 'i' = 30 out of range!
Statement executed ...
> Done();
Our-C exited ...
```