**資訊三甲 10427101 馬若芸**

1. **開發平台**

Windows 10。

1. **使用開發環境**

Dev-C++。

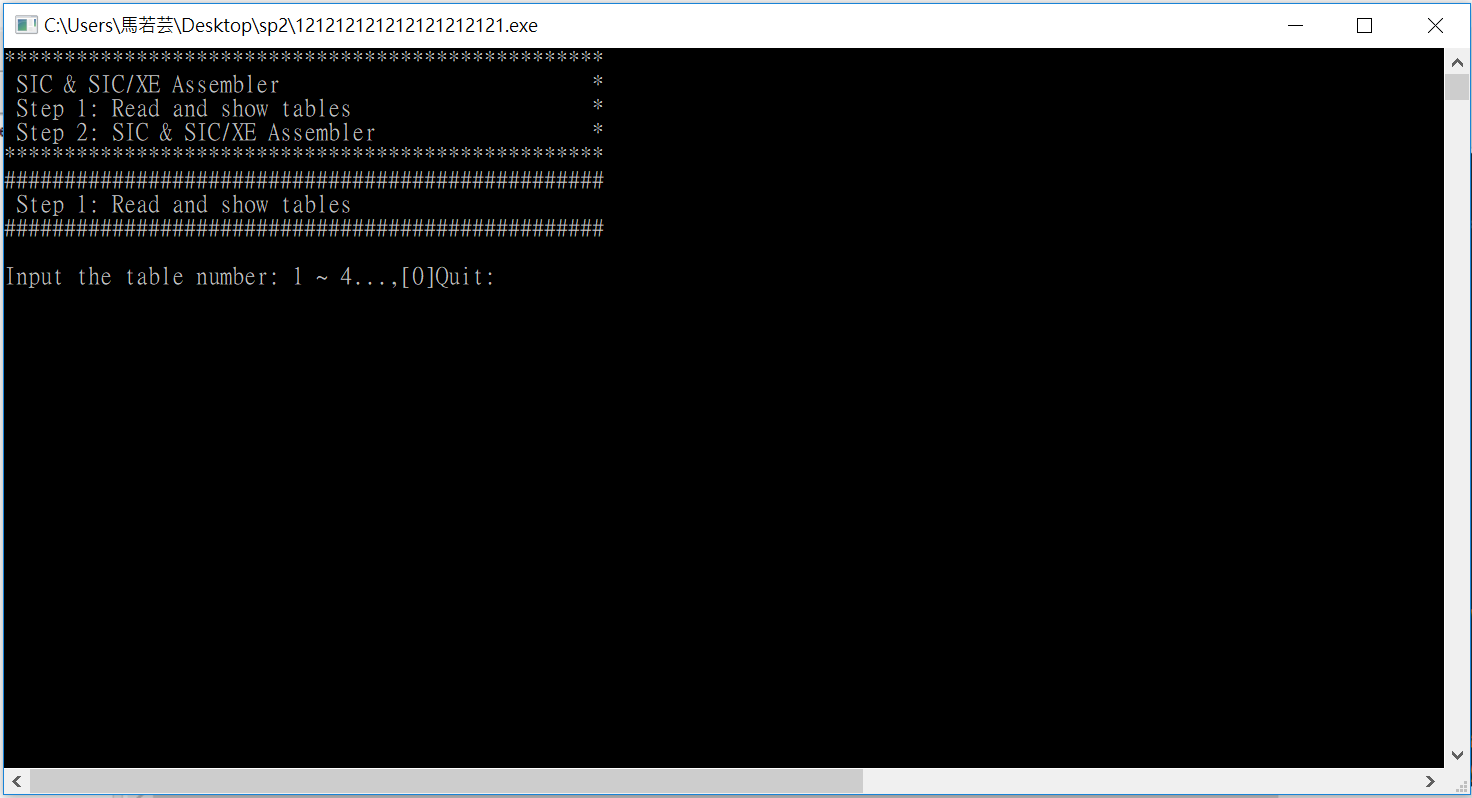
1. **使用的程式語言**

C、C++。

1. **所選擇的組合語言**

SIC、SIC/XE。

1. **程式設計說明**



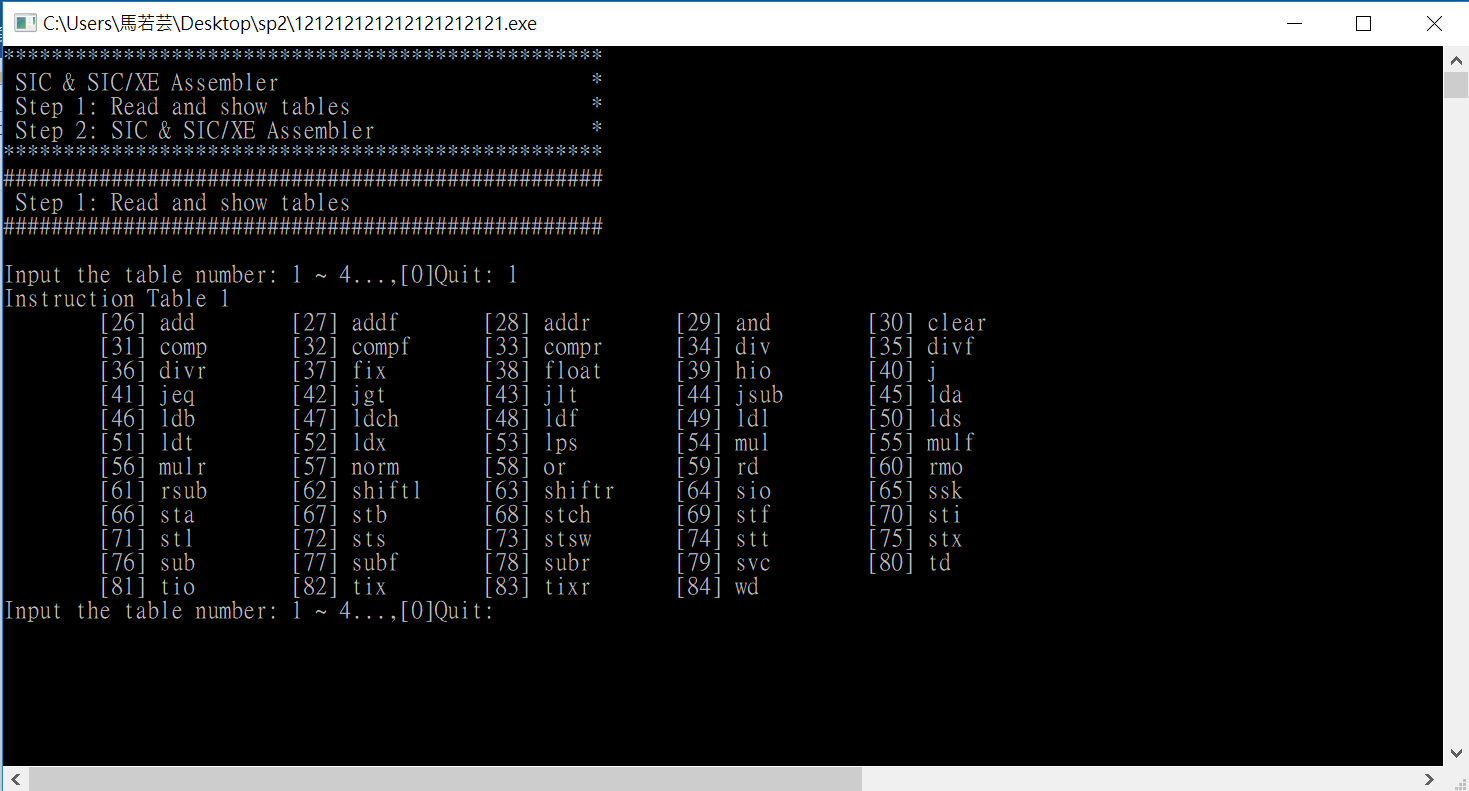
**(初始畫面)**

* 功能

程式分成兩步驟進行，先輸入完需要的instruction、pesudo and extra、register、delimiter table後，即可進行SIC & SIC/XE Assembler。

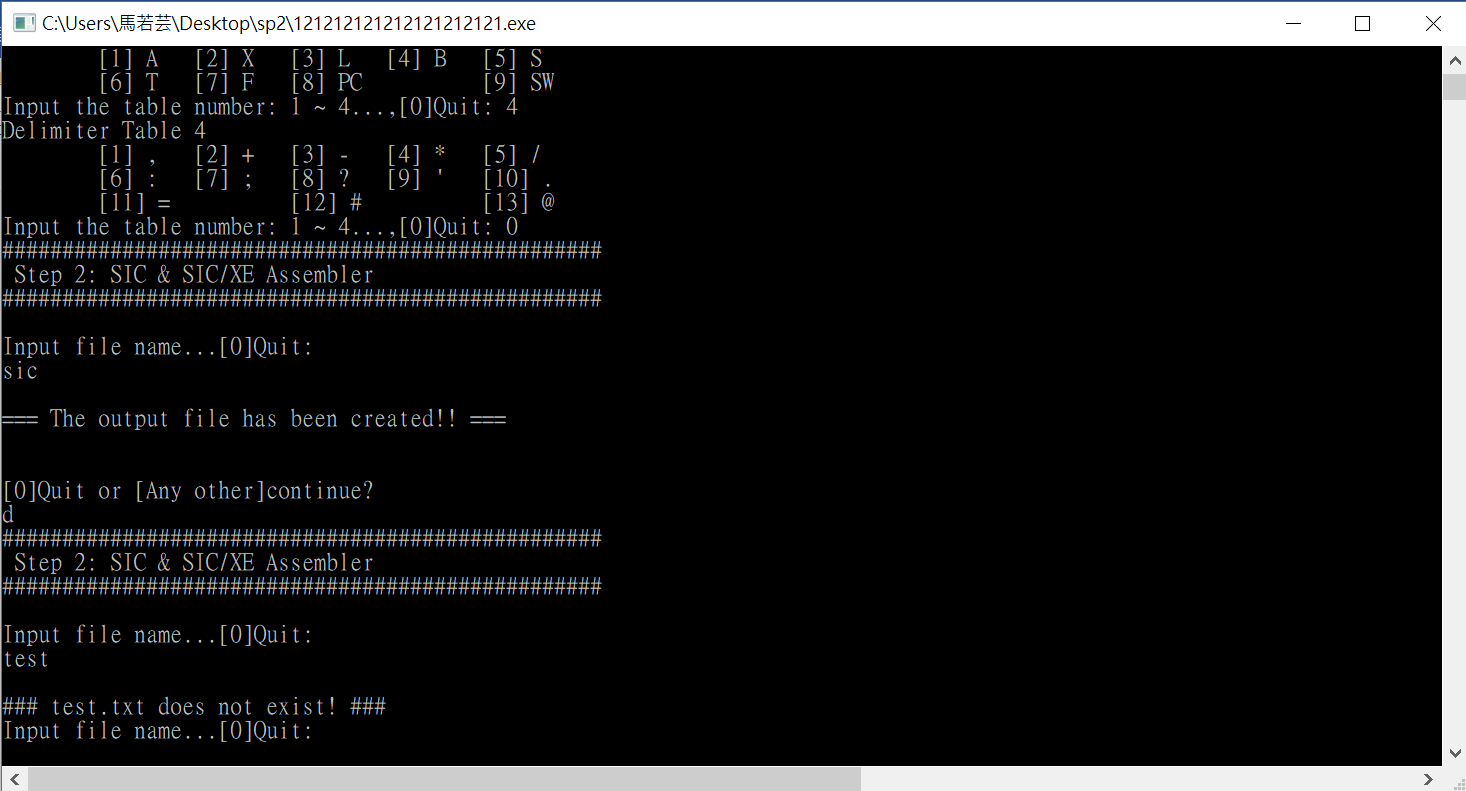
* 使用流程

1. 輸入並讀入提供的table1~4，每輸入table的編號後會輸出table的內容(圖二)。輸入需要的instruction、pesudo and extra、register、delimiter table後，輸入”0”進行下一步驟。



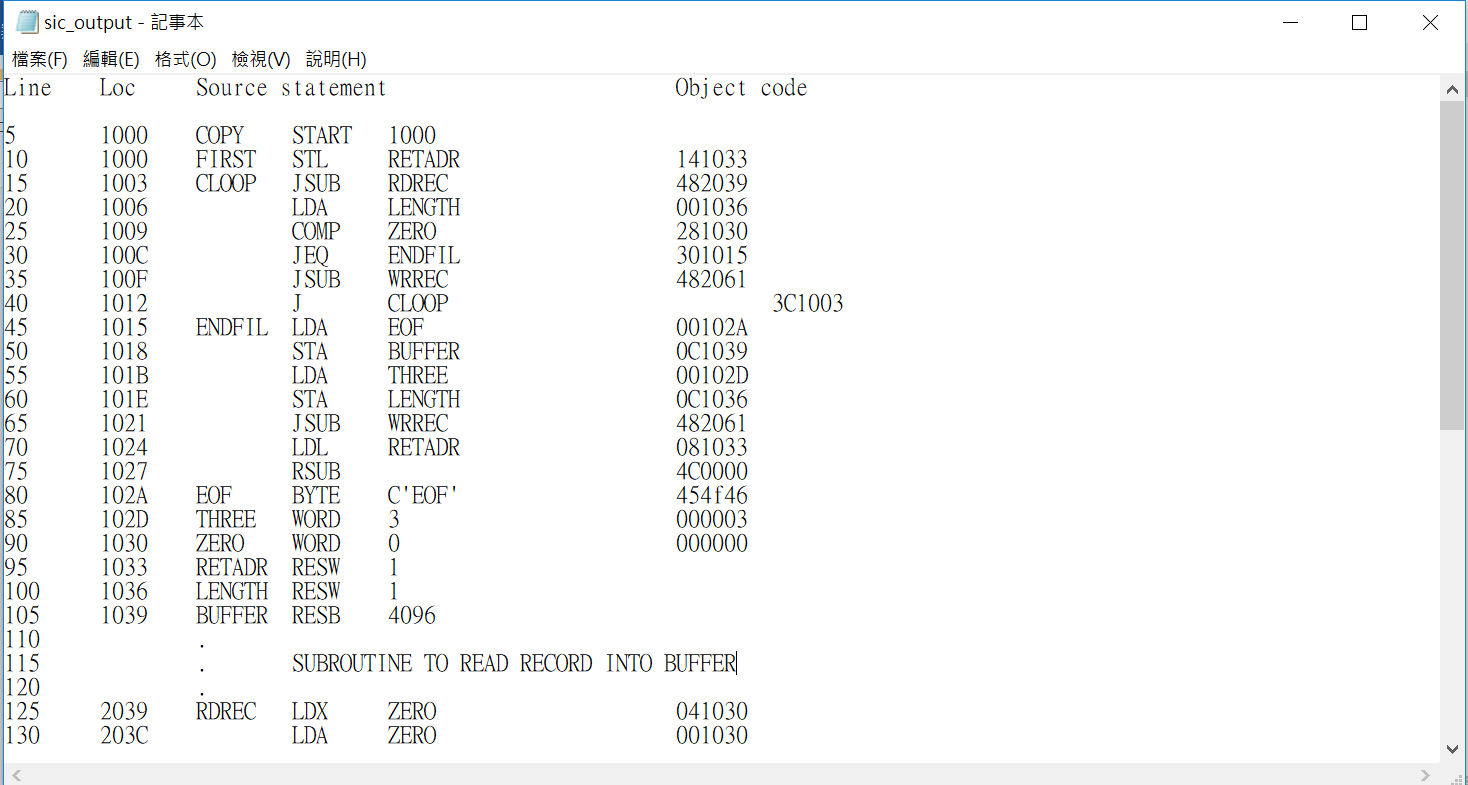
(圖二)

1. 完成步驟一接著開始SIC & SIC/XE Assembler。輸入檔案名稱，若找到該檔名並執行成功則出現”=== The output file has been created!! ===”字樣；若無此檔案則出現”### 名稱.txt does not exist! ###”(圖三)。



(圖三)

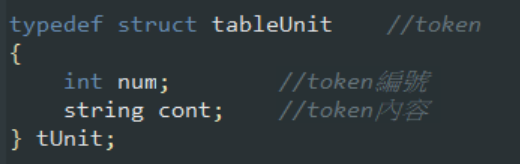
1. 完成SIC & SIC/XE Assembler後若還想再執行下一份資料時，輸入任意見繼續執行，輸入”0”結束離開。
2. SIC & SIC/XE Assembler完成產生的檔案，檔名:輸入檔\_output.txt，如在步驟二輸入”SIC\_input”，則輸出檔為”SIC\_input\_output”。輸出檔內容格式為每行包含欄位行數(Line)、在主記憶體的位置(Loc)、指令敘述(Source statement)、目的碼(Object code)(圖四)。



(圖四)

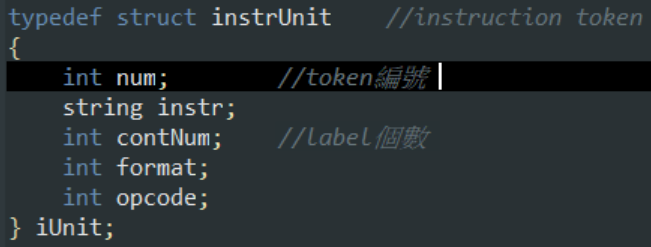
* 使用的data structure

Table的struct為tableUnit，tableUnit內有紀錄token號碼的num和此token內容的cont。(如下圖)



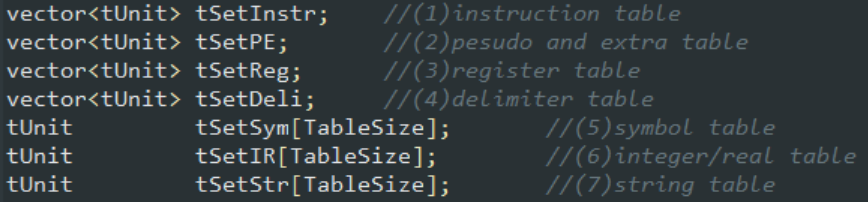
pesudo and extra、register、delimiter 皆使用vector儲存，每讀入table中一個token依類型放入相對應的vector中，三個table對應的vector分別為tSetPE、tSetReg、tSetDeli。

為了記下每個指令的opcode、format和會使用的label個數，特別把指令的struct用instrUnit儲存，instrUnit內有指令的token號碼(num)、內容(instr)、opcode(opcode)、format(format)和會使用的label個數(contNum)。(如下圖)

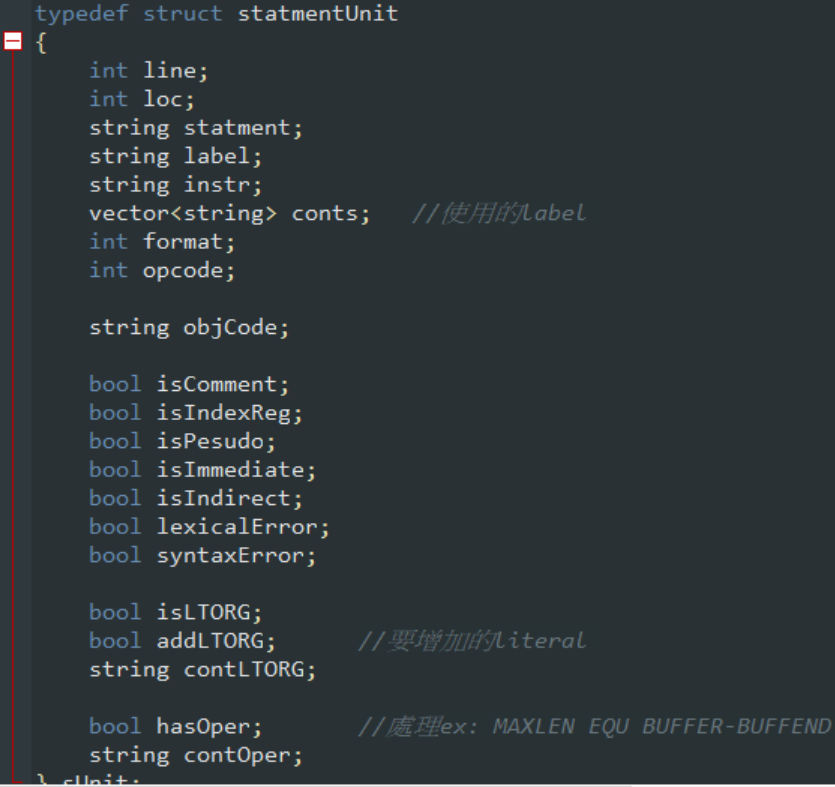


使用vector儲存，每讀入table中一個token依類型放入相對應的vector為tSetInstr。

剩下的symbol、integer/real、string table是用linear hash方式存入空間大小為100的陣列中，相對應的陣列分別為tSetSym、tSetIR、tSetStr。在進行lexical analysis時，若在對應的陣列中沒有該token，則使用mod100的hash function放入hash table中，若產生碰撞則向後遞增至空的位置。



負責儲存每一行敘述的vector為statmentUnit，內有記欄位行數(line)、在主記憶體的位置(loc)、指令敘述(statment)、該敘述宣告的label(label)、使用的指令(instr)、呼叫到的label(conts)、format(format)、opcode(opcode)、目的碼(object code)、是否是註解(isComment)、是否用到index register(isIndexReg)、是否屬於pesudo(isPesudo)、是否是立即定址法(isImmediate)、是否是間接定址法(isIndirect)、是否lexical error(lexicalError)、是否syntax error(syntaxError)、是否是有關literal(isLTORG)、是否為literal的內容(addLTORG)、literal內容(contLTORG)、是否含有運算子要運算(hasOper)、要運算的內容(contOper)。



* 程式碼解說

**Function說明:**

1. bool ReadWriteFile():讀進input檔，每一次都只讀一行，讀入該行後呼叫GetToken()進行切token，直到讀完input檔結束，並開始寫入output檔。若成功讀寫檔案，回傳true讓螢幕輸出完成訊息；若失敗，回傳false。
2. void WhichTable():根據輸入的table編號，來決定table內容要放入對應的vector中，並把table的內容輸出至螢幕上。
3. void InputTable():讀進table檔，一個個放入vector中，最後再呼叫WhichTable()來決定是vector是屬於哪個table的。
4. void CheckTable():分析token，用迴圈一一比對來找該token屬於哪一table的內容，若找到則把分析結果寫入output檔中並return；若無表示此token可能因為沒有以white space間隔導致是多個token組成的ex:C’EOF’、X’123’，或是屬於需要使用hash的table但未在該table中的token，所以需要呼叫ExtraTable()來作進一步的切割、分析。
5. void ExtraTable():把在CheckTable()無法完成分析的token來依不同情況再切割，切割完成後呼叫CheckTable()進行分析。還有要新增在table的token，會呼叫insertHashLinear()來新增至hash表。
6. void GetToken():因為每一次都讀取一行，所以用white space分割一個個token，呼叫CheckTable()分析token。
7. bool isDeli():檢查token是否為符號，若是則回傳true。
8. bool isNum():檢查token是否為數字，若是則回傳true。
9. int SetHashKey():計算hash值，hash functoin是把token中每一字元的ASCII碼相加後除以100(hash table size)取餘數，最後回傳該token的hash值。
10. void insertHashLinear():先呼叫SetHashKey()算出hash值要放入哪一位置，若產生碰撞則向後遞增至空的位置，並把完成的結果寫入output檔中。
11. void resetList():初始化hash table。
12. void setObjCodeSIC():若是SIC則設定SIC程式碼各個objCode。
13. void setObjCodeSICXE():若是SIC/XE則設定SIC/XE程式碼各個objCode。
14. void setLoc():根據pesudo、、SIC、SIC/XE的指令來設定各個程式碼在主記憶體間的位置。
15. void CheckError():檢查錯誤，檢查敘述用的label數是否和該指令用的label數相同，不相同表示syntax error；相同則正確。

**main說明:**

呼叫ReadWriteFile()來進行上述一連串各function間運作，若ReadWriteFile()回傳true表已成功輸出output檔。

1. **未完成功能**

從範例來看，發現有些line的數值不是間隔為5來計算，想了很久也不知其原因、不知道要寫什麼方法來解決，所以都統一用間隔5來計算。

EQU的loc計算方式也不太了解。

程式中切token的方式，都是根據範例檔的內容或網路上所查到的SIC程式碼裡的情況來切的，雖然盡可能把預想到的情況來分析，但可能還是有遺漏的部分或是可能程式碼的排版不同ex: STCH RECORD, X和STCH RECORD,X(RECORD、X間有無white space)導致沒辦法正確的切到token。

雖然在錯誤檢查方面設想了很多面向，但我認為還是無法做到十全十美的樣子；在遇到間接定址法時也是一樣的，看了許多來自網路上的SIC/XE程式碼的範例，也盡量以較直接的方法來計算出目的碼，但可能我寫的算法錯誤導致間接定址的目的碼是錯誤的，還有處理literal的方法寫得很繁複，可能會有錯誤。

1. **心得**

相較於上一次lexical analysis的作業，這次的製作的時間只有短短的三個禮拜，雖然作業一開始時因為剛好期中考完想好好放鬆一下而變得有些懶惰，還一度想放棄寫這份作業，但是回首過去兩年來的努力以及在SP上花費的時間與精神，如果說放棄就放棄的話那實在是枉費了之前的努力和狠狠地背叛了過去的自己，為了能想完成便早早開始寫，一開始光是寫SIC就寫寫改改了三、四天，結果最後覺得自己把整支程式寫壞了，索性打掉成練完整思考清楚、從上次作業接續開始寫，最後才好不容易完成。