

SIC Assembler 實作(完成)

座號：86

班級：資訊二丁

學號：D0976935

姓名：楊凱捷

Assembler : SIC (完成)

1. 程式開發過程：自行開發

2. 開發語言、開發平台、程式使用方式、使用範例與輸出結果範例：

開發語言：C 語言

平台是：DEV C++

程式的使用方式：

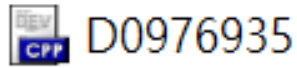
一開始先確認.c 檔、SICP.txt、OpTable.txt 是否有放在同個地方，然後打開 D0976935.c 檔，按下 F11 或點選上面的編譯並執行，之後會出現一個 CMD 視窗並會開始執行程式碼。之後，會看到 CMD 視窗上出現：

已產生 Symbol Table
已產生 LocCtr
已產生 Program Length
Object File 寫入完成

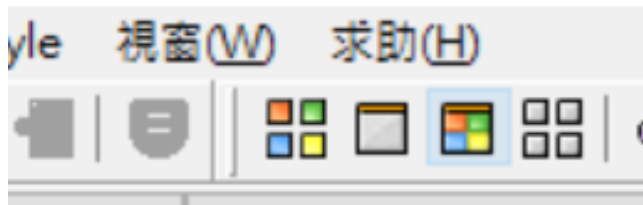
即可關掉 CMD 視窗，然後資料夾就會多 4 個 txt 檔了。

使用範例：

1.點選資料夾內的 **CPP** 檔：



2.點選有圖片上有藍框的按鈕，或者是按 **F11**：

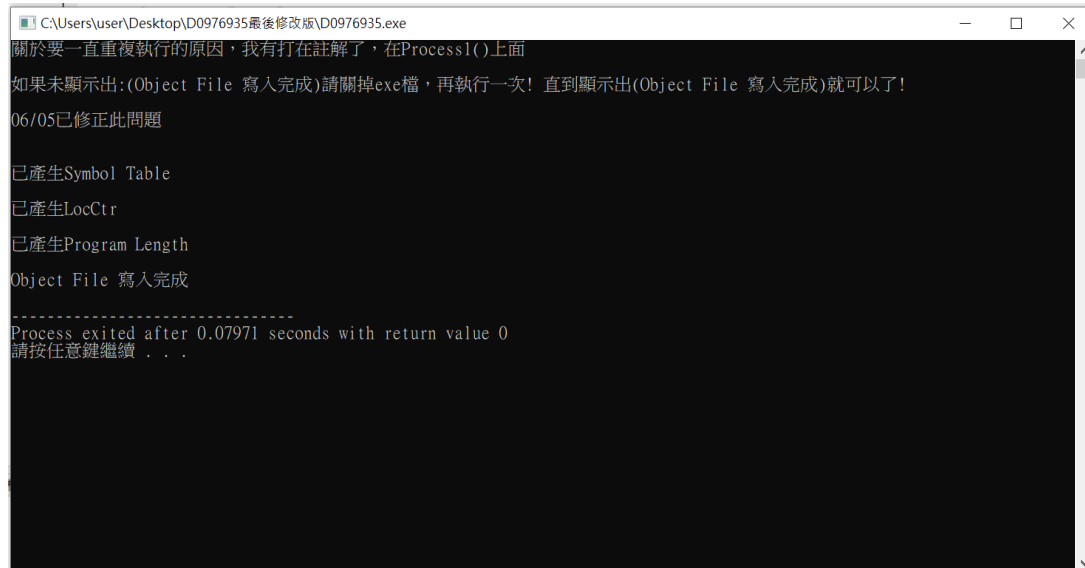


3.接下來會出現一個 **CMD** 視窗，等他就好了：



結果範例：

1.執行成功的樣子：



```
C:\Users\user\Desktop\D0976935最後修改版\D0976935.exe
關於要一直重複執行的原因，我有打在註解了，在Process1()上面
如果未顯示出:(Object File 寫入完成)請關掉exe檔，再執行一次! 直到顯示出(Object File 寫入完成)就可以了!
06/05已修正此問題
已產生Symbol Table
已產生LocCtr
已產生Program Length
Object File 寫入完成
-----
Process exited after 0.07971 seconds with return value 0
請按任意鍵繼續 . . .
```

2.輸出檔案：

- LocCtr
- OJ Program
- Program Length
- Symbol Table

這邊說明程式的檔案名稱各代表甚麼檔案：

SICP.txt：原始 SIC 程式

Symbol Table.txt：符號表

Program Length.txt：程式長度

OpTable.txt：指令集

LocCtr.txt：Location Counter 表

OJ Program.txt：Object code 表

3. 可以處理怎樣格式的輸入：

1. 標籤、指令都用一個 **Tab** 隔開。
2. 標籤、指令一律都大寫。
3. 註解行前面一定要加(.)。
4. **START** 跟 **END** 一定要寫。
5. 指令一定要在 **OpTable.txt** 裡的才能處理

指令、格式其實都跟課本上的 **SIC** 程式一樣。

4. 有哪些輸出：

輸出的檔案：

Symbol Table.txt(符號表)

Program Length.txt(程式長度)

LocCtr.txt(Location Counter 表)

OJ Program.txt(Object code 表)

錯誤訊息：

1. 重複定義的標籤：

Ex. NUM LDA EOF

NUM STA BUFFER

5. 可以處理的 **addressing modes** :

只要是 SIC 語法允許的都可以，像是：

direct、indexed。

不能處理的就是像 XE 的 #或@，或者是

Extended、Register-Register、PC-Relative、
Base-Relative。

6. 可以處理的 **Assembler directives** :

START、END、BYTE(X、C)、WORD、
RESB、RESW、COMP、DIV、J、ADD、
AND、JEQ、JGT、JLT、JSUB、LDA、
MUL、STL、STX、SUB、OR、RD、
RSUB、LDCH、LDL、LDX、STA、STCH、
TIX、WD、TD

7. **Machine-independent features** :

沒有，EQU、Literal、ORG、Expression、
Program-Block、Control-Section 這些都不能處理。

8. 有哪些 **Data structures** :

其實我寫這個程式用到的資料結構只有用到陣列、結構、指標而已，主要是檔案跟字串的處理比較麻煩。

一開始先開啟 **SICP.txt**，並讀入此檔案且存到陣列裡面，接下來先判斷開頭是否為 **START**，這步是為了讓 **LOCCTR** 從哪裡開始數，且並將第一行先寫入檔案裡。

接下來判斷是否有註解行(.)，不是的話就接下去判斷 **Program** 中，每列有幾個字串(也就是判斷 **Label**、指令(**EX.ONE STA TWO**)這樣)，因為不一定每列都是 3 個，有些會有 1 個(像是 **RSUB**)，或 2 個的(**LDA ONE**)。

接下來就是判斷指令，除了 **OPTABLE** 裡的跟 **RESW**、**RESB**、**WORD**、**BYTE**，其中 **BYTE** 又有分 **C** 跟 **X** 的。

這樣第一階段就完成了，產生 **Symbol Table** 跟 **Program Length** 跟 **LOCCTR**。

第二階段則是讀入 Symbol Table、Program Length、LOCCTR、OPTABLE 而產生 Object Program(H 卡、T 卡、E 卡)。分兩種，一種是直接編入的、另一種則是有 X 的，像是 BUFFER ONE,X 這樣。

一開始一樣先讀檔，然後先將第一行寫成 H 卡片並且把開始位置寫到 T 卡片的第一欄的起始位置

再來利用 `sscanf()`，把讀入的字串依照模式分成 4 個一為陣列存入，分別是 `loc,label,op1,op2`。接下來分別處理 2 個跟 3 個(也就是在上述 4 個陣列中有 2 個或 3 個值得(`EX.RSUB`、`LDA ABC`))。

接下來判斷陣列裡是否為 `RESW`、`RESB`、`END`，`END` 就跳出迴圈不繼續判斷，我先將長度寫入 T 卡片(`Ex.^1E^`)，這邊我使用了(`%02X`)，表示不足兩位數就在字串前面補 0。這邊主要是為了處理 T 卡片的起始位置(`T^001000^`)、長度，所以才需要判斷 `RESW`、`RESB`。這邊的起始位置我使用了(`%06X`)，表示不足六位數就在字串前面補 0。

處理完起始位置、長度之後，因為我是邊處理邊寫檔，所以在進行每次寫檔時，我都會使用 **fseek()**、**ftell()**來移動檔案指針，除了方便寫檔，也防止寫檔錯誤。

接下來處理 **Object code** 的寫入，首先分為下列幾種，直接編入的、有 **X** 的、**RSUB**、**BYTE(C、X)**、**WORD**。

1. 直接編入:將指令的編碼以及 **Label** 的地址直接寫入。
2. 有 **X** 的:我用 **strlen()**取得每行指令的長度，如果長度-2 的位置是(,)號，那就表示該行的 **code** 為(,X 這樣的格式(Ex. **STCH BUFFER,X**))，上述只是判斷格式的程式寫法，判斷完後就簡單了，因為有 **X** 就是表示要+8(**xbpe=1000**)，所以只要加上 **8** 並利用 **OPTABLE** 檔裡的指令邊碼，兩個合併再寫入檔案即可。

3. **RSUB**: 這個因為後面都不會有 **Label**，所以直接寫入 **4C+0000** 即可。
4. **BYTE(C 版本)**: 將 **C** 後面的字串轉成 **ASCII** 碼後寫入
5. **BYTE(X 版本)**: 利用 **strlen()** 函數取得 **X** 後面的字串長度(Ex. **X'05'**，=2)，並設為 **for** 迴圈的長度，接下來直接寫入檔案裡。
6. **WORD**: 利用 **strtol()** 把 **WORD** 後面的轉成長整數(**long int**)，並以 **16** 進位的方式寫入檔案內。

9.特別的 **function** :

1. **FILE** 的處理：像是讀檔或寫檔、利用 **C** 語言提供的函數來做檔案處理。

fopen(開檔)、**fclose**(關檔)

fscanf(讀檔)、**fprintf**(寫檔)

fgets(讀入一行)、**feof**(檢測檔案是否為空)

ftell(用來獲取我的文件當前的讀寫位置)

fseek(用來設定我文件的當前讀寫位置)

2. 字串處理：把檔案讀入的字串，利用 **C** 語言提供的字串處理函數來處理字串。

strcmp(字串比較)、**strcpy**(複製字串)

strlen(計算字串長度)

strtol(將字串轉成 **long int**(長整數))

3. **atoi**(將數字字串轉成整數 **int**)

rewind(用來把文件內的指針，重新指向開頭)

10.心酸血淚史：

其實我對 C 語言比其他語言來的熟悉，但是卻沒說到精通。所以我在寫 **Assembler** 時其實很痛苦，一開始讀檔基本的是沒問題，然後存入陣列加上之後的處理我也沒問題，但是檔案指標這塊我就比較頭痛了，因為我從大一到二二的資料結構幾乎不會用到檔案指標，最多都是用到讀檔跟寫檔而已，而且還要一直記說他現在指到哪個位置。

這部分在 **Pass-1** 還好，主要是 **Pass-2** 的地方要開 3 個檔案，然後還要用 3 個檔案指標去產出 **Object code**。這 3 個檔案指標幾乎是用來兩兩比對檔案資料用的，像是我在處理標籤重複定義的那裡。

之後比較麻煩的點是 **Object code** 的寫檔，空格跟排版我弄了很久。

全部都寫完後，我在執行的時候才發現我的程式會當機，我花了好幾天的時間在處理這個錯誤，原因很快就找到在哪個部份了

(在 **Pass-2**)，但是卻卡在怎麼處理。我一直反覆改了 **Pass-2**，但是卻完全沒有解決這個問題。最後，我去網路上看看卡死的原因大多是怎麼發生的，但是網路上的卡死原因都不像是我那種會機率性發生的，要嘛完全卡死、不然就是完全成功，搞得我那幾天心情真的超不好，越改火氣越上來。

問了一些朋友，跟他們討論出來的結果是檔案的部分出問題，這邊我也在修改了好幾次，依舊沒有好的結果，朋友給我的建議是重寫另一份，但是重寫其實讓我覺得我之前的那份 **Assembler** 是白費工夫，而且我的這份 **Assembler** 其實花了我很多時間跟心力，要直接重新開始實在讓我很心累，原本想說決定交了這份有缺點的 **Assembler**，結果在 6/5 號，繳交的前一天被我修改好了!!!結果只是因為陣列的初始設定不夠大，因為字元放入陣列的位置並不會照著 1,2,3 這樣放，有可能是 1,空格,2 這樣，所以有時候空間夠，有時候空間不

夠，而導致有時候成功，有時候失敗。我改個初始陣列大小就解決了這個問題，改完之後，突然有一種我到底在幹嘛的感覺....。

我突然發現，程式越寫越多，越會錯一些簡單的地方，除了這次初始陣列設定以外，像是：`if(x=1)`，正確應該是：`if(x==1)`這樣，但是我有時候就是會少打一個邏輯運算子的符號。

在寫完這份報告後，我有去開始著手寫 **XE** 版的東西，雖然目前只有寫出"`@`"跟"`Literal`"，但是就慢慢寫，看自己能寫到哪種程度，因為 **XE** 還得考慮比 **SIC** 更多的指令、模式，但是感覺少了時間的壓力，能讓我能寫出 **XE** 版的更多功能。