資工三甲10627122陳俐欣

* ***開發平台***

Ans: MacOS

* ***使用開發環境***

Ans: xcode

* ***說明你的程式設計(功能、流程、使用的data structure)***
* 功能：
  1. 讀Input檔：手動輸入檔名(.txt不用)，並讀取完整組合語言程式碼的

檔案作為整個等待被切token的input檔，並將該input檔以行為單位放置vector<Statement> input中。

* 1. 讀Table檔：自動讀入兩個需要的table（delimiter、rsword），並將每個

table的資料整理(將字體都轉成大寫，清除空白和換行符號)後放進各自的buffer中，之後在找token是屬於哪一個table中時，直接從buffer中找。

* 1. 切token：處理從組語程式碼的input檔讀進來的buffer，分析每一

行程式碼，將一整行字串切成一個一個的token，先切出有幾個token(先不判斷是在哪個table，除了像C‘string’這種會先賦予“string”在table7)，先將token初步分開後，再將每一行程式碼的每個token一個一個去table尋找，並將token附上table編號和在該table第幾個欄位。

* 1. 文法判斷：
     1. Program：

在我的程式中，PROGRAM的文法判斷並非照著原本的文法走，因此相較於其他的指令，PROGRAM的文法寫的較死，可能會出錯，我是當判斷到有PROGRAM保留字時，下一個預期要是program的名稱，當判斷到program的名稱時，就將此program的subroutine用stack的方式存在我的共用變數區，這樣之後再判斷變數在哪一個subroutine時可以用stack的top來看，然後當遇到ENDP或ENDS時就將stack的top pop出去。

* + 1. VARIABLE：

我是照著講義中的文法一個一個寫成function，當遇到需要判斷的項目時就呼叫對應的function，用遞迴加上queue的方式進行，當此token確定符合文法後就從queue中pop掉，不再理會他，繼續往下呼叫。當文法正確時，在此指令中宣告的變數尋找當下的subroutine並一起放入table5（identifier）中，邊檢查文法邊丟資料進table。

* + 1. DIMENSION：

在DIMENSION中為了能宣告超過2dimension的陣列，我也是照著講義的文法建function，撰寫過程和VARIABLE差不多，只是DIMENSION還需要處理放資料進table7（information）的部分。

* + 1. LABEL：

和撰寫VARIABLE的過程大致相同，根據講義文法進行。

* + 1. GTO：

和撰寫VARIABLE的過程大致相同，根據講義文法進行。

* + 1. SUBROUTINE：

和撰寫PROGRAM的過程大致相同，並非完全照講義文法，再判斷到有一個新的subroutine時將此subroutine push進存放subroutine的stack中。

* + 1. CALL：

和撰寫VARIABLE的過程大致相同，根據講義文法進行。

* + 1. ASSIGNMENT：

因為assignment的情況有太多，為了避免我有疏漏，我都是完全照著講義文法寫function，並也是使用遞迴加上queue的方式做文法判斷。

* + 1. IF：

和撰寫assignment的概念差不多。

* 1. 產生中間碼：
     1. PROGRAM：

因為通過文法檢查後，如果文法正確，我們就可以預期我們看到的PROGRAM保留字位置和program名稱的位置，所以可以直接依序將token找到其對應的table和entry，丟入table6（quadruple）。

* + 1. VARIABLE：

因為文法是正確的所以可以預知我們要的token的位置，所以直接將格式轉換為table6要的後直接丟入。

* + 1. DIMENSION：

和建立VARIABLE的中間碼過程差不多，只是要確定的是，陣列的資料有正確的背書入到table7中。

* + 1. LABEL：

因為有可能會有forward reference所以我採用的是，在處理每一行statement時都要判斷是否有LABEL，若有的話就在table5中尋找此名稱的LABEL的欄位，並將pointer的位址修改成此statement的位置。

* + 1. GTO：

因為有可能GO TO的位置是個forward reference所以我先將可能要跳去的label名稱和此statement在table6的位置先記錄下來（在buffer中），等所有的statement都處理完後，理應所有label都應該被賦予位址，此時在從buffer中尋找有哪些包含forward reference的quadruple，再將label的地址指標補上。

* + 1. SUBROUTINE：

和處理PROGRAM的手法相同。

* + 1. CALL：

尋找被呼叫的變數在table5中的位址，並得到此function在table7中放的資料（有幾個參數等），再把呼叫時放入的參數丟到table7中。

* + 1. ASSIGNMENT：

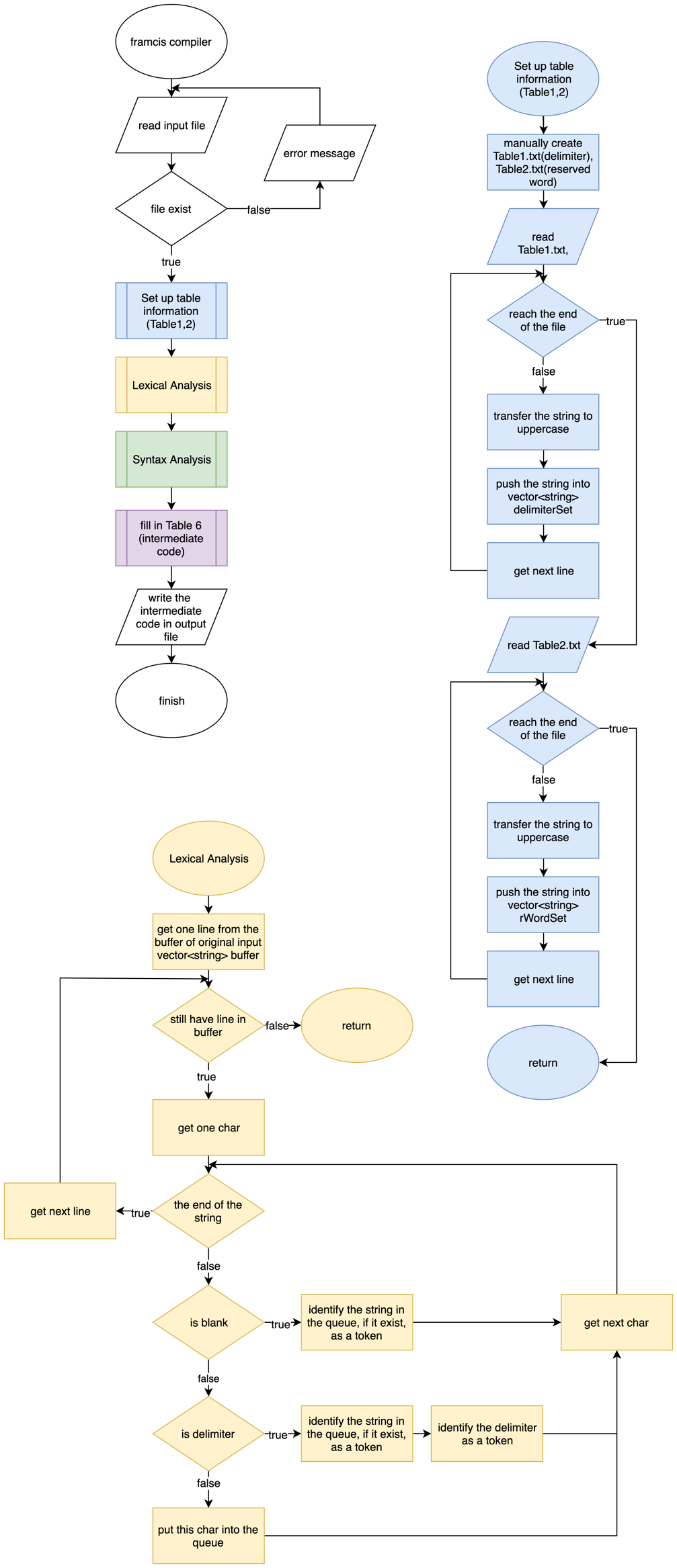
在處理各種計算式時，我花了蠻多時間在處理array的index部分，而我的寫法是可以允許array的維度超過2，應該是多少都可以，因為當assignment中出現array時不管是出現在等號左邊或是右邊，array都是最先處理，我將array獨立出去處理（將statement中array的index都存起來，也在table7中尋找該array每個index的大小，進行位址的換算）完後（array的中間碼都生成），在繼續處理其他的運算。而其他運算是照著Reverse Polish Notation的方式進行，用兩個stack（id、op）處理，每個op都被賦值，方便比大小。

* + 1. IF：

我將IF分成三部分處理，一個是IF的條件式部分，一個是THEN（true）的部分，一個是ELSE（false）的部分。IF的部分我的處理和處理assignment相同只是運算子換成GT、LT等，並可以允許條件式中有括號或是多個條件式在其中，因為也是用雙stack進行，所以條件式個數沒有限制，並在IF部分差不多處理完後儲存此IF將要放入table6的位址，因為要放入true時跳到的位址和false時跳到的位址，在此時我們並補曉得所以我們先保留，等到THEN和ELSE處理完後再將位址補上，這樣就能在一個pass完成。在THEN和ELSE的部分，我將這兩部分包含的token全部都製作成（視為）一個新的statement（舉例：X=X+2），再直接呼叫處理statement的function，就可以直接判斷他是哪一個指令並生成對應的中間碼，我就不用再重寫這段。

* 1. 寫檔：

因為在生成中間碼時，我都是丟在一個buffer中，所以寫檔時只要依序輸出就沒什麼太大的問題，而我在進行文法分析時會將發生error的地方，error message也都放在一個buffer中，所以當quadruple的buffer都輸出完後，再判斷error buffer中有沒有東西，若有則也輸出，並輸出”Syntax error”。

* 流程：

(右邊流程圖)

* data structure：
  1. struct:
     1. struct Token {

string str ; // token的字串名稱

int table ; // 此token所在的table

int no ; // 此token在table中的位址

} // Token

* + 1. struct Unit {

int tableNo ; // 放入quadruple table中的(table, entry)的table

int index ; // 放入quadruple table中的(table, entry)的entry

} // Unit

* + 1. struct Statement {

string originStr ; // 原始的指令字串

Token instr ; // 此statement的instruction

string label ; // 此statement的label若無則是空的

vector<Token> tokens ; // 從放此statement的所有token

bool gramCorrect ; // 紀錄此statement文法是否正確

} // Statement

* + 1. struct ErrorInfo {

int indexOfStmt ; // 在第幾行程式發生error

string message ; // 此error是什麼

} // ErrorInfo

* + 1. struct Table5Info {

string identifier ; // identifier的名稱

int subroutine ; // 此identifier所在的subroutine

int type ; // 此identifier的型別

Unit pointer ; // 此identifier指向哪裡

} // Table5Info

* + 1. struct quaForm {

Unit units[ 4 ] ; // 存放要輸出的quadruple格式的4個單元

string stmt ; // 要輸出的quadruple的原始指令

} // quaForm

* + 1. struct fix {

int quaIndex ; // 哪一行quadruple需要修改

int unitNo ; // 要修改哪一個單元

string labelName ; // 要修改的單元的label名稱

} // fix

* 1. class：
     1. class G {

private :

fstream file ; // 讀檔用

string fileName ; // 檔案名稱

int recentTmpSequenceNum ; // 記錄目前的暫時變數序號

public :

stack<string> subroutine ; // 紀錄subroutine

vector<string> delSet ; // 保存delimiter

vector<string> rsWord ; // 保存reserved Word

vector<Statement> input ; // 保存input

vector<Token> table0 ; // 放暫存變數

string table3[ HASHMOD ] ; // table3 ( integer )

string table4[ HASHMOD ] ; // table4 ( real number )

Table5Info table[ HASHMOD ] ; // table5 ( identifier )

vector<quaform> table6 ; // table6 ( quadruple )

vector<int> table7 ; // table7 ( information )

vector<ErrorInfo> errorTable ; // 存放error資訊

vector<fix> fixSet ; // 存放要修改的forward reference

void createOutputFile() ; // 寫檔

int getRececntTmpNum() ; // 取目前的tmp num

int getNewTmpNum() ; // 拿一個新的tmp num

……

(和其他工具用的函式ex:尋找label index等等)

} ; // class G

* + 1. class LexicalAnalysis {

private :

vector<string> buffer ; // buffer

public:

getToken() ; // 切token

test\_LexicalAnalysis() ; // 測試整個檔案切token的結果

test\_getToken() ; // 測試切token

} ; // LexicalAnalysis

* + 1. class SyntaxAnalysis {

private :

isDataType() ; // 判斷是哪個型別

\_type\_() ; // 講義中<type>的文法

\_unsignedInteger\_() ; // 講義中<unsigned integer>的文法

……

（所有依據講義文法建立的文法function）

……

} ; // SyntaxAnalysis

* + 1. class DataProcessor {

private :

void \_\_program\_\_() ; // 建立PROGRAM的中間碼

void \_\_variable\_\_() ; // 建立VARIABLE的中間碼

void \_\_dimension\_\_() ; // 建立DIMENSION的中間碼

void \_\_labelDeclare() ; // 建立LABEL的中間碼

void \_\_gto\_\_() ; // 建立GTO的中間碼

void \_\_subroutine\_\_() ; // 建立SUBROUTINE的中間碼

void \_\_call\_\_() ; // 建立CALL的中間碼

void \_\_assignment\_\_() ; // 建立ASSIGNMENT的中間碼

void \_\_if\_\_() ; // 建立IF的中間碼

void \_\_end\_\_() ; // 建立ENDP ENDS的中間碼

int getPrecedence() ; // 取運算子的大小

bool hasHigherPrecedence() ; // 比較當前和stack中的op

int classifyType() ; // 辨識型別

int getDelIndex() ; // 取得此delimiter的index

void arrayMiddleCode() ; // 處理計算array index部分

void assignLabelAddress() ; // 將label指標賦值

void processForwardReference() ; // 補上label地址

public :

void process() ; // 處理一個statement

void processAll() ; // 處理整個input檔的statement

} ; // DataProcessor

* ***未完成的功能***
  1. 在判斷整個program是否符合文法時，PROGRAM和SUBROUTINE是相

對寫的較死，可能會出錯，如果有時間我想改掉

* 1. comment處理
  2. 在計算陣列位址的時候，我的程式會需要多一個臨時變數儲存
  3. 更嚴謹的防呆措施
  4. 讀取"↑"會有問題