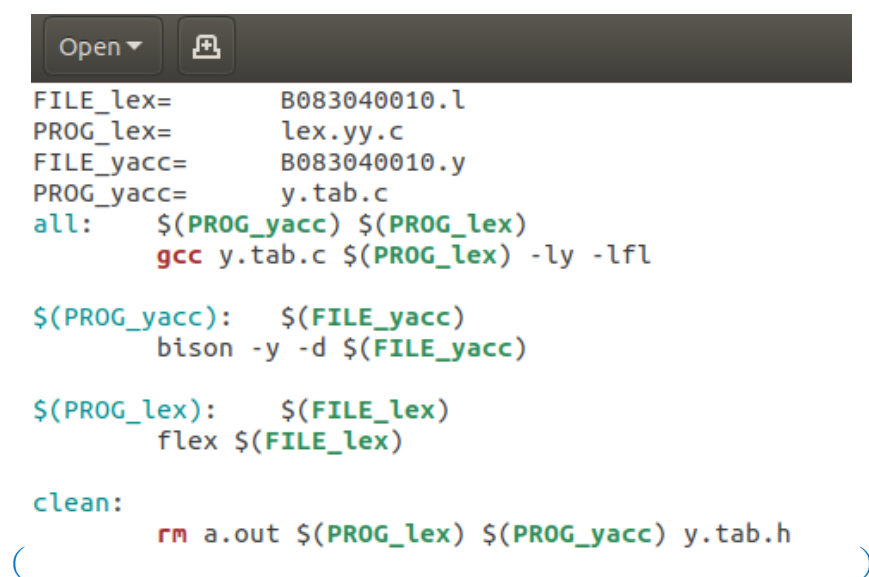


【README】

B083040010 李其儒

1. Lex, 版本：[flex 2.6.4](#) / [bison 3.0.4](#)
2. 作業平台：[Ubuntu 18.04 \(Linux\)](#)
3. 執行方式：製作 makefile，如圖：



```
FILE_lex=      B083040010.l
PROG_lex=      lex.yy.c
FILE_yacc=     B083040010.y
PROG_yacc=     y.tab.c
all:           $(PROG_yacc) $(PROG_lex)
               gcc y.tab.c $(PROG_lex) -ly -lfl

$(PROG_yacc):  $(FILE_yacc)
               bison -y -d $(FILE_yacc)

$(PROG_lex):   $(FILE_lex)
               flex $(FILE_lex)

clean:
               rm a.out $(PROG_lex) $(PROG_yacc) y.tab.h
```

執行步驟：

- I. 進入相關資料夾，並“make”。
- II. “./a.out < (要測試的檔案)”

4. 你/妳如何處理這份規格書上的問題：

1. lex：

首先修改 regular expression 的格式，並刪掉 invalid token 的偵測。用了 condition state 來蓋掉相關訊息(四則運算中)，好比說 {integer} - {integer}，要加上 - 號，所以設 BEGIN。

此外，用 yylval.name=strdup(“string”)，把 token 轉存為字串，這樣可以給丟給 Yacc 使用。Lex 主要處理輸出、切 token(scanner)、count(position 跟 line)。

2. yacc :

設定 lex 傳過來的 token(都以大寫表示)；設定 error message，規格書內有給 simplified grammar，我有針對一些內容去多處理情況，好比 | error 的運用；關於 id_list 的拆分(read/write)……

運用 pre_p 及 pre_l(extern)來記錄先前 token 的位置等資訊，方便在發生錯誤時，得知其是在哪一行；我是以多讀一個 token 的策略，去判斷有無 error。在 rule 中加上 %error-verbose 來得知完整錯誤訊息。

建了 idtable[]，因為 assign 或 exp 可能會有 id 不存在或 type 錯誤的問題，用 table 紀錄其，若 id 已存在則輸出 duplicate id；在 stmt_list 中，若有 id 未宣告，則輸出 id not found.

5. 你/妳寫這個作業所遇到的問題

問題可多了，起初完全看不懂怎麼開始，毫無頭緒，到處跟人討論。

起步一點後，發現對語法超級不熟，好比 char*(後來用 strdup 解決)等等；且語法是深度優先，要很注意判斷 error 或 statement 的寫法。我在加上別於 simplified grammar 的 statement 後，都要改好久 bug 才能正常…

Error recovery 的部分，我只單純把錯誤的行印出來，沒特別做其他處理，基本上要找到錯在哪，就要花好一番功夫了。

6. 所有測試檔執行出來的結果，存成圖片或文字檔

I. correct.pas :

```

taurus880503@ubuntu:~/Desktop/compiler/HW2$ ./a.out < correct.pas
Line 1: program test;
Line 2: var
Line 4:   i, j: integer;
Line 5:   ans: array[0 .. 81] of integer;
Line 6: begin
Line 7:   i := -1+3;
Line 8:   j := +7*8;
Line 9:   ans[0] := 7;
Line 14:   for i:=1 to 9 do
Line 15:   begin
Line 16:     for j:=1 to i do
Line 17:       ans[i*9+j] := i*j;
Line 18:   end;
Line 20:   for i:=1 to 9 do
Line 21:   begin
Line 22:     for j:=1 to i do
Line 23:       if ( ans[i*9+j] mod 2 = 0) then
Line 24:         write(i, ' ', j, '=', ans[i*9+j], ' ');
Line 25:       writeln;
Line 26:   end;
Line 27: end.

```

II. error1.pas :

```

taurus880503@ubuntu:~/Desktop/compiler/HW2$ ./a.out < error1.pas
Line 1: program test;
Line 2: var
Line 3:   i: integer;
Line 4: begin
Line 5, syntax error, unexpected '=', expecting ':='
Line 6, identifier not found 'j'
Line 7, identifier not found 'j'
Line 8, at char 5, syntax error, unexpected WRITE, expecting THEN
Line 9: end.

```

III. error2.pas :

```

taurus880503@ubuntu:~/Desktop/compiler/HW2$ ./a.out < error2.pas
Line 1: program test;
Line 2: var
Line 3:   i, j : integer;
Line 4: begin
Line 5:   i := 5*2;
Line 6:   j := 9;
Line 7:   if (i > j)
Line 8, at char 5, syntax error, unexpected WRITE, expecting THEN
Line 9: end.

```

IV. error3.pas :

```
taurus880503@ubuntu:~/Desktop/compiler/HW2$ ./a.out < error3.pas
Line 1: program test;
Line 2: var
Line 3, at char 8, syntax error, unexpected ASSIGNMENT, expecting COLON or COMMA
Line 4: begin
Line 5, identifier not found 'i'
Line 6: end
Line 6, at char 4, syntax error, unexpected $end, expecting DOT
```

V. error4.pas :

```
taurus880503@ubuntu:~/Desktop/compiler/HW2$ ./a.out < error4.pas
Line 1: program test;
Line 2: var
Line 3:   i, j : integer;
Line 4:   c : string;
Line 5: begin
Line 6:   i := 5;
Line 7:   c := 'aa';
Line 8, syntax error, unexpected '=', expecting ':='
Line 9: end.
```