**系統程式-課程筆記**

**˙**指令

|  |  |
| --- | --- |
| gcc sum.c –o sum | 用gcc編譯器去編譯sum.c程式，輸出一個執行檔sum |
| ./sum | 執行sum |

˙把codeblocks編譯器加進vs code：

1. 找到資料夾......codeblocks\MinGW\bin並複製路徑
2. 去控制台，選擇系統及安全性->系統->進階系統設定
3. 環境變數，選擇系統變數的path->新增並貼上路徑->重開vs code

˙hash雜湊：把一個字串用固定的方式轉成一種數字

ex:

usigned int h = 37

h = h\*147 + \*p; //\*p是ASCII碼

hash( )=37 hash(h) = 5543 //5543=37\*147+104(h的ASCII)

hash(he) = 814922 = 5543\*147+101 //e是101的ASCII

***04-map***

**(main.c)循序搜尋**

mapNew(&jMap, 17);

呼叫mapNew : 建立一個大小為17的表格，變數名為jmap

jMap.table = jList; : 把jList塞進去

jMap.top = 8; : 有8個元素

mapLookup(&jMap, "JLE"); : 在jMap裡尋找”JLE”結構

**(map.c)**

map->size = size; : 在main.c裡設為17，東西不能塞超過17個

mapFind : 在map陣列裡找

***生成語法***

N = cat | dog |:or的意思

S = N Y

句子 = 名詞 動詞

V = run | eat

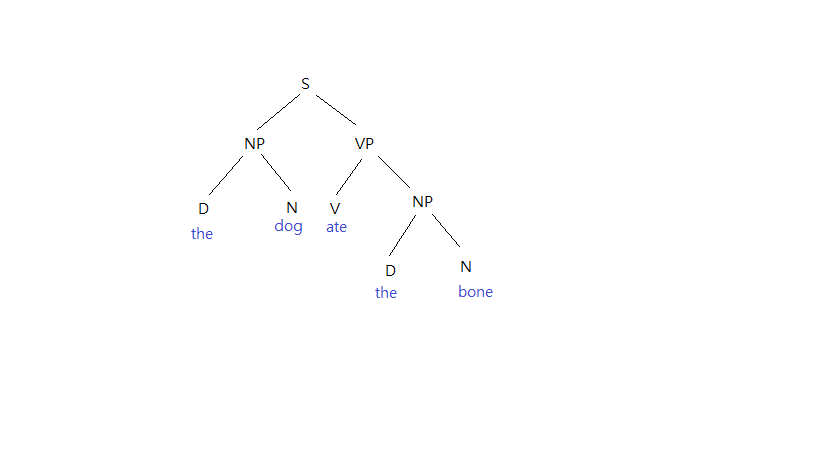
N->dog , V->run產生dog run

˙BNF語法

S = NP VP 句子 = 名詞子句 + 動詞子句

NP = D N 名詞子句 = 定詞 + 名詞

VP = V NP 動詞子句 = 動詞 + 名詞子句

語法樹：

E = F( [ + - ] )\*

E:運算式 F:Factor ( )\*:可出現0次以上

F = Number | ‘ ( ‘ E ‘ ) ‘

//E只能寫0~9 ex:’ 3 + 5 ‘

***運算式編譯器 (exp0)***

**ex:**

E = F( [ + - ] )\* //假設輸入’ 3 + 5 ‘

F = Number | ‘ ( ‘ E ‘ ) ‘

parse: 3 + 5

**結果：**

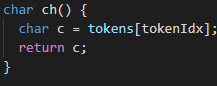
t0 = 3 //t0是一個暫存器

t1 = 5 //t1存5

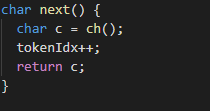
t2 = t0 + t1 //t2 = t0 + t1 = 3 + 5，最後產生的東西放在t2

**部分程式碼註解：**

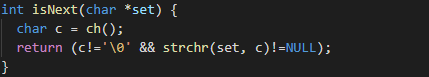
// 取得目前字元，同時進到下一格

******

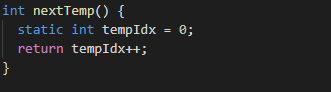
// 取得目前字元，同時進到下一格



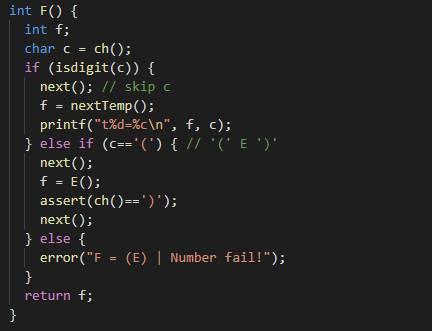
// ex: isNext("+-") 用來判斷下一個字元是不是 + 或 -



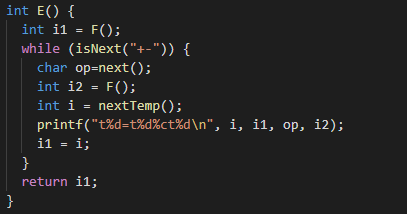
// 產生下一個臨時變數的代號， ex: 3 代表 t3。

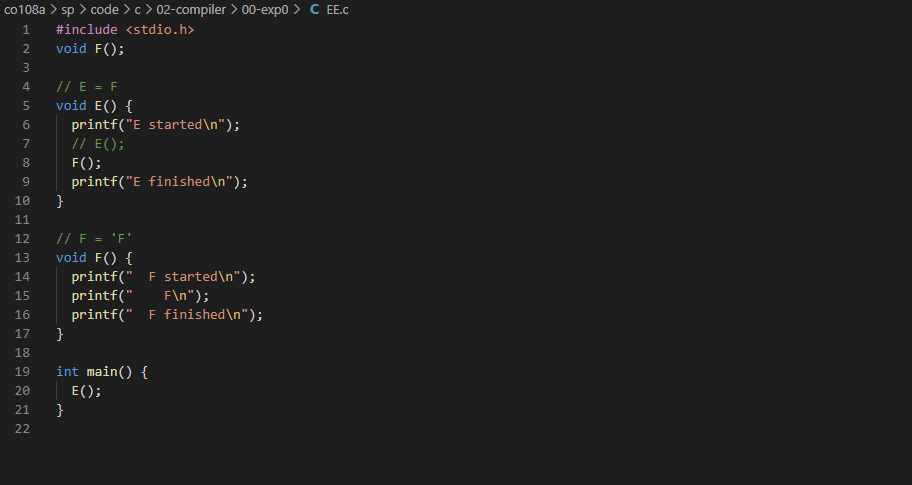


// F = Number | '(' E ')'



// E = F ([+-] F)\*



**輸入指令：gcc exp0.c –o exp0 : 編譯出一個執行檔exp0.exe**

主程式呼叫一個E

E印出E started

E呼叫了F，此F就是下面的F，但F比後面定義，但在前面就呼叫了，所以前面要加一個void F ( )

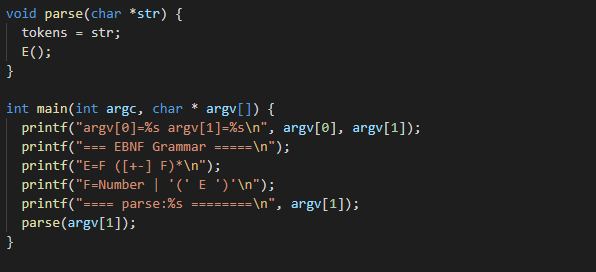
**結果：**

E started

F started

F finished

E finished

parse(argv[1]); //把argv[1]傳進parse函數

arg : 參數 argc : 參數個數 argv : 參數變數，陣列

argv[1] : 第一個參數

./lexer 第0個參數 sum.c第一個參數

˙若輸入./exp0 ‘ x + 5 – y ‘就會編譯中間碼

#t0 = x , @x (變數), D = M / @5 (數字), D =A , #t1 = 5

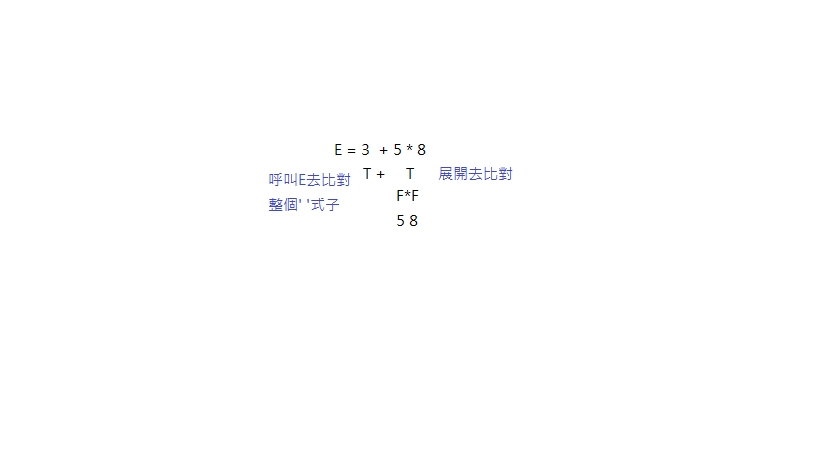
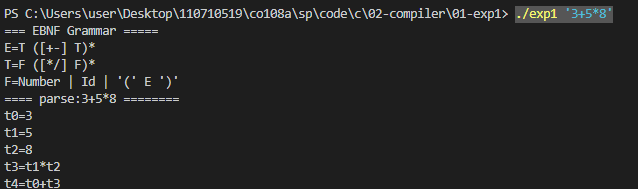
***exp1***

 printf("E=T ([+-] T)\*\n");

越下層的運算，優先序越高

  printf("T=F ([\*/] F)\*\n");

**執行./exp1 '3+5\*8'**

**結果：**

***Compiler.c語法***

PROG = STMTS 一個程式就是一堆陳述

BLOCK = { STMTS } 一個區塊就是有{ }中間加一堆陳述

STMTS = STMT\* STMT是一堆0次以上的STMT

STMT = WHILE / BLOCK / ASSIGN

一個陳述有可能是while迴圈/BLOCK區塊/ASSIGN指定

WHILE = while ( E ) STMT

一個while迴圈是用while開頭，再接(運算式)，最後陳述

ASSIGN = id = ‘ = ‘ E 一個指定是一個變數名稱，他等於一個運算式

E = F ( OP E )\*

F = ( E ) | Number | ID

***05-compiler-run***

執行指令：

minge32-make ：編譯原始碼的建置工具

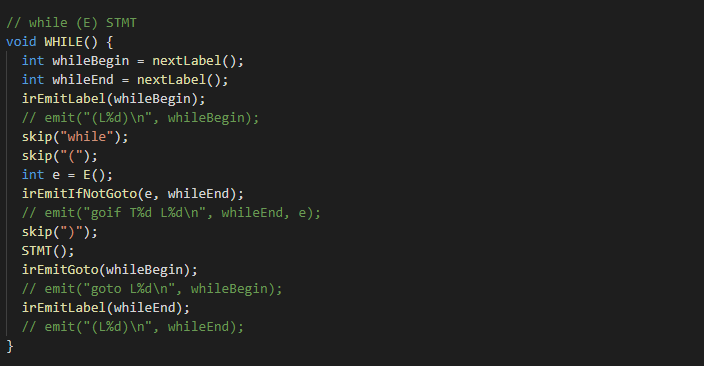
./complier test/sum.c –ir –run

只打 –ir 代表會輸出中間碼但不執行

若加上 –run 代表除了輸出中間碼，還會執行

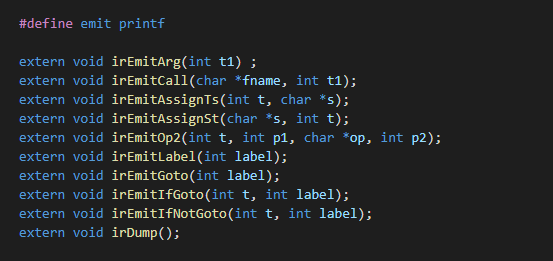
irvm.c裡的trace = printf 印出指令 = emit

\*xxxx.c檔裡面放程式碼 xxxx.h檔裡面放標頭，放define的東西



irEmitLabel：

本來是直接emit，變成直接呼叫irEmitLabel。目的是讓它格式統一，可以存在陣列裡。

typedef struct {

ir.h裡面定義的東西

  IrType type;

  int t, t1, t2, label;

  char \*s, \*op;

} IR;

t,t1,t2:臨時參數 lable:標記代號

op:做什麼動作。Ex:加法 \*s:目標參數

整個結構會儲存成一個陣列，放在ir[ ];裡面

void irEmitAssignTs(int t, char \*s) {

  irNew((IR) {.type=IrAssignTs, .op="t=s", .t=t, .s=s});

}

IrAssignTs：t = sum 之類的 t:取得代號 s:取得名字

void irDump() {

  printf("=======irDump()==========\n");

  for (int i=0; i<irTop; i++) {

    printf("%02d: ", i);

    irPrint(&ir[i]);

  }

}

irDump:把中間碼全部印出來

irvm.c:用來執行中間碼的程式

 if (eq(op, "s=t")) { //如果是s=t這樣的指令

    int \*vp = varLookup(p->s);

    \*vp = t[p->t]; //先把t取出來，再塞到新的變數

    trace("%s = t%d (%d)\n", p->s, p->t, \*vp); //印出來

  }