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

S	=	N	Y
句子	=	名詞	動詞

N = cat | dog　　|:or 的意思

V = run | eat

N->dog , V->run 產生 dog run

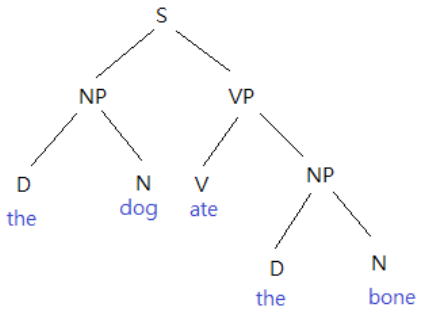
- BNF 語法

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

```
char ch() {
  char c = tokens[tokenIdx];
  return c;
}
```

// 取得目前字元，同時進到下一格

```
char next() {
  char c = ch();
  tokenIdx++;
  return c;
}
```

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

```
int isNext(char *set) {
  char c = ch();
  return (c!='\0' && strchr(set, c)!=NULL);
}
```

// 產生下一個臨時變數的代號， ex: 3 代表 t3。

```
int nextTemp() {
  static int tempIdx = 0;
  return tempIdx++;
}
```

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

```
int F() {
  int f;
  char c = ch();
  if (isdigit(c)) {
    next(); // skip c
    f = nextTemp();
    printf("t%d=%c\n", f, c);
  } else if (c=='(') { // '(' E ')'
    next();
    f = E();
    assert(ch()==')');
    next();
  } else {
    error("F = (E) | Number fail!");
  }
  return f;
}
```

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

```c
int E() {
  int i1 = F();
  while (isNext("+-")) {
    char op=next();
    int i2 = F();
    int i = nextTemp();
    printf("t%d=t%d%ct%d\n", i, i1, op, i2);
    i1 = i;
  }
  return i1;
}
```

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

co108a > sp > code > c > 02-compiler > 00-exp0 > C EE.c

```c
#include <stdio.h>
void F();

// E = F
void E() {
  printf("E started\n");
  // E();
  F();
  printf("E finished\n");
}

// F = 'F'
void F() {
  printf("  F started\n");
  printf("    F\n");
  printf("  F finished\n");
}

int main() {
  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 ( )

主程式呼叫一個 E

結果：

E started
F started
F finished
E finished

```c
void parse(char *str) {
  tokens = str;
  E();
}

int main(int argc, char * argv[]) {
  printf("argv[0]=%s argv[1]=%s\n", argv[0], argv[1]);
  printf("=== EBNF Grammar =====\n");
  printf("E=F ([+-] F)*\n");
  printf("F=Number | '(' E ')'\n");
  printf("==== parse:%s ========\n", argv[1]);
  parse(argv[1]);
}
```

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

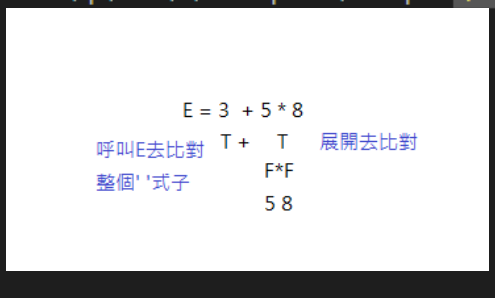
```c
printf("E=T ([+-] T)*\n");
printf("T=F ([*/] F)*\n");
```

越下層的運算，優先序越高

執行./exp1 '3+5*8'

結果：

```
PS C:\Users\user\Desktop\110710519\co108a\sp\code\c\02-compiler\01-exp1> ./exp1 '3+5*8'
=== EBNF Grammar =====
E=T ([+-] T)*
T=F ([*/] F)*
F=Number | Id | '(' E ')'
==== parse:3+5*8 ========
t0=3
t1=5
t2=8
t3=t1*t2
t4=t0+t3
```



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