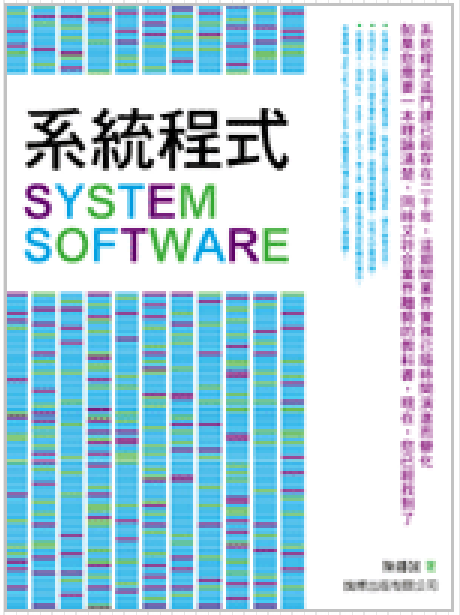
第 7 章、高階語言

作者：陳鍾誠

旗標出版社



第 7 章、高階語言

- 7.1　簡介
- 7.2　語法理論
- 7.3　語意理論
- 7.4　執行環境
- 7.5　實務案例：C 語言

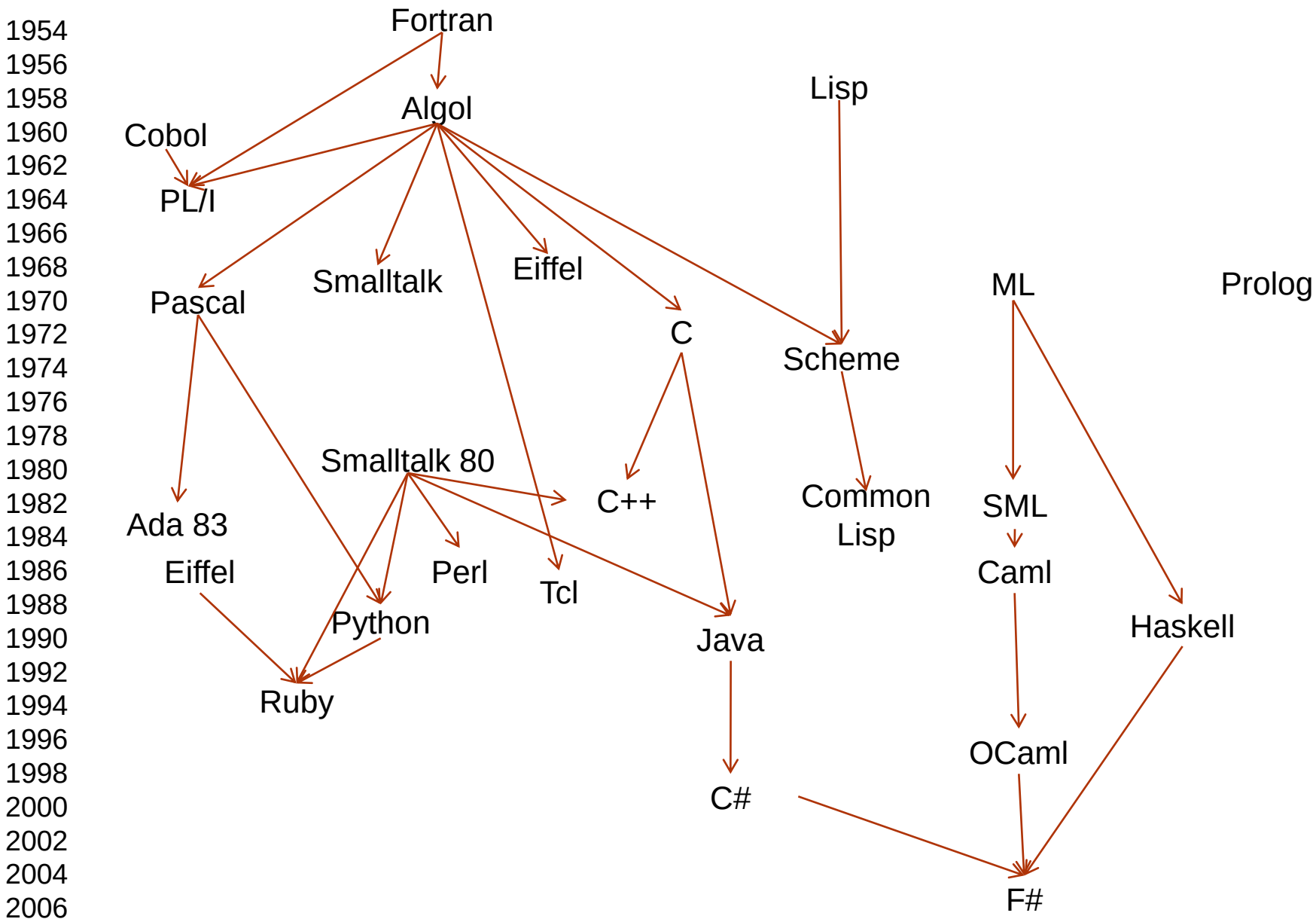
7.1 簡介

- 高階語言的核心是「語法理論」
  - 利用生成規則 (例如：BNF, EBNF 等) 描述程式的語法。
  - 根據生成規則撰寫剖析程式, 轉換成語法樹。
  - 對語法樹進行『解譯』或『編譯』的動作。

編譯器 v.s. 直譯器

- 直譯器
  - 利用程式解讀該語法樹, 並根據節點類型執行對應的動作, 這樣的程式就被稱為『直譯器』。
- 編譯器
  - 撰寫程式將語法樹轉換為組合語言 (或目的碼), 那麼, 這樣的程式就被稱為編譯器。

圖 7.1高階語言的歷史年表



7.2 語法理論

- 高階語言所使用的語法, 大致上分為兩個層次
- 詞彙層次
  - 使用 Regular Expression (簡稱 RE)
- 語句層次
  - 使用 Context-Free Grammar (簡稱 CFG)
- RE 與 CFG 都可以使用『生成規則』描述

生成規則

- 生成規則
  - 由近代語言學之父的喬姆斯基 (Chomsky) 所提出
  - 是生成語法 (Generative Grammar) 理論的基礎

- 用途
  - 用來描述「自然語言」的語法，像是中文、英文等。

BNF (Backus–Naur Form) 規則

- BNF
  - 由 John Backus 與 Peter Naur 所提出的規則寫法
  - 很適合用來描述程式語言的語法

- BNF 與生成規則的關係
  - BNF 是為了描述「程式語言」而發展出來的。
  - 生成語法是為了描述「自然語言」所發展出來的。
  - 兩者幾乎是同一件事，都可以描述 RE 與 CFG，因此後來多不進行區分。

BNF 語法的範例

- 簡易的範例

▶圖 7.2 簡單的生成語法範例

(a) BNF 語法

```
S = A B
A = 'a' | 'b'
B = 'c' | 'd'
```

(b) 生成的語言

```
L = {ac, ad, bc, bd}
```

- 英語語法的範例

▶圖 7.3 一個簡單的英語語法範例

(a) BNF 語法

```
S = N ' ' V
A = 'John' | 'Mary'
B = 'eats' | 'talks'
```

(b) 生成的語言

```
L = {John eats, John talks,
     Mary eats, Mary talks}
```

數學運算式的語法

▶圖 7.4 簡單的數學運算式語法

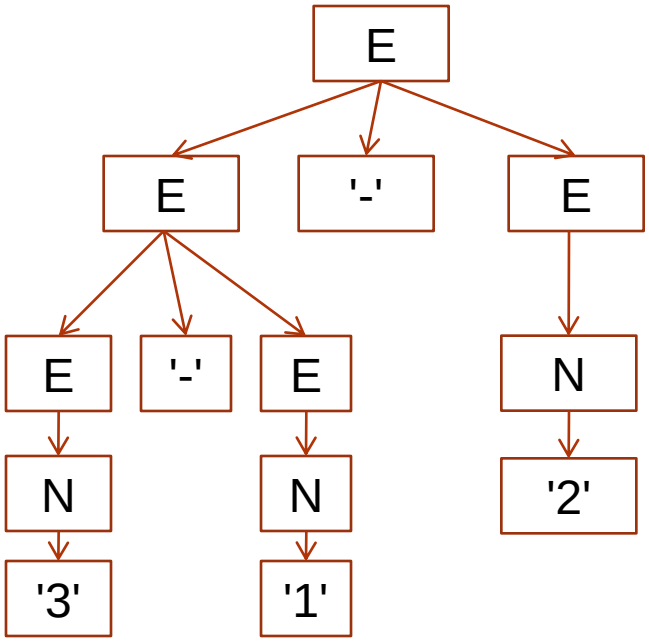
(a) BNF 語法

```
E = N | E [+-*/] E
N = [0-9]+
```

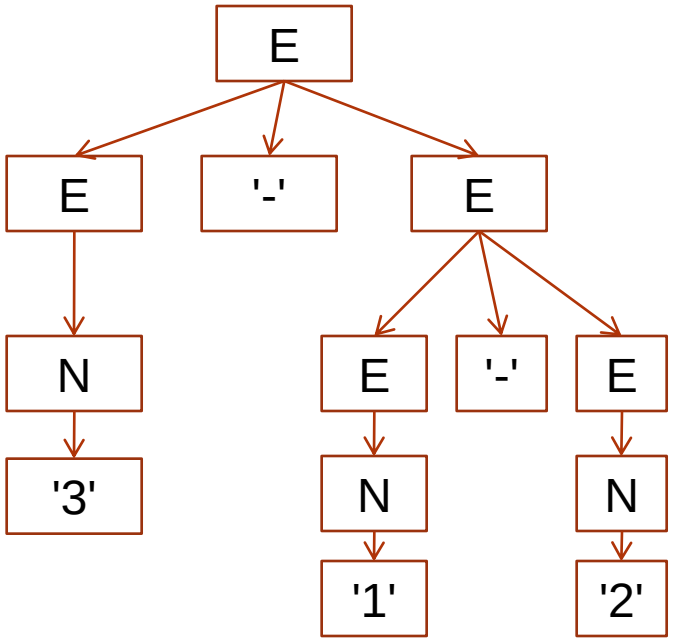
(b) 語言的實際範例

```
3
3 + 5
3 + 5 * 8 - 4 / 6
```

具有歧義的語法範例



(a). 語意：(3 - 1) - 2 = 0



(b). 語意： 3 - (1 - 2) = 4

圖 7.5 運算式 3-1-2 可能產生兩顆語意不同的語法樹

歧義性的困擾

- 程式語言的語法是不能有歧義性的
- 否則,根據圖 7.5，程式編譯後有時 3-1-2 會計算出 0, 有時卻會算出 4
- 這樣將導致程式設計師無法確定程式的執行結果, 而陷入混亂崩潰的狀況。

無歧義的數學運算式語法

▶圖 7.6 無歧義的數學運算式語法

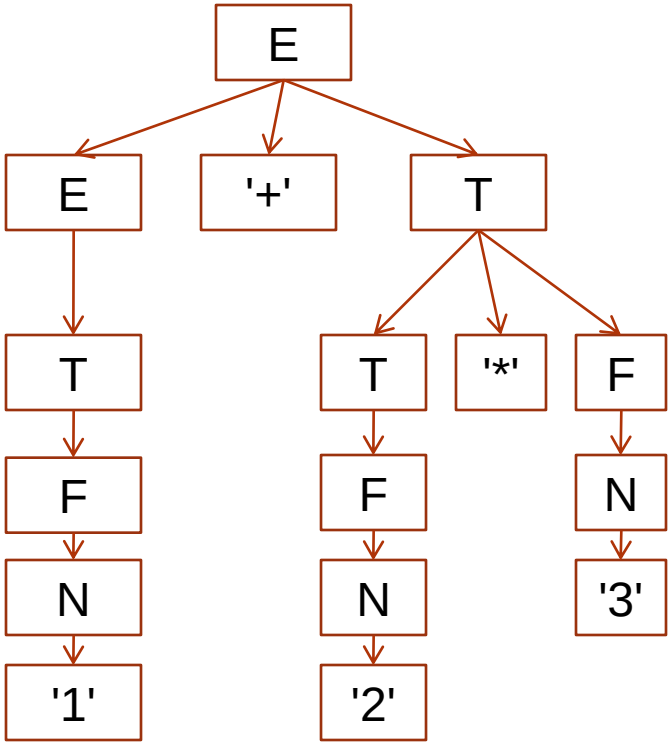
(a) BNF 語法

```
E = T | E [+-] T
T = F | T [*/] F
F = N | ' (' E ') '
N = [0-9]+
```

(b) 語言的實際範例

```
3
3 + 5
3 + 5 * 8 - 4 / 6
(3 + 5) * 8 - 6
```

圖 7.7 數學運算式 (1+2*3) 的語法樹



左遞迴的問題

- 問題描述
  - 舉例而言，規則 E = T | E [+-] T 中就有左遞迴
  - 使用遞迴下降法進行剖析時，E 會不斷被展開，因而導致無窮遞迴而當機

- 解決辦法
  - 將左遞迴的 BNF 語法改為 EBNF 語法，以消除左遞迴

▶圖 7.6 無歧義的數學運算式語法

(a) BNF 語法

```
E = T | E [+-] T
T = F | T [*/] F
F = N | ' (' E ') '
N = [0-9]+
```

(b) 語言的實際範例

```
3
3 + 5
3 + 5 * 8 - 4 / 6
(3 + 5) * 8 - 6
```

EBNF

- 說明
  - Pascal 語言的發明人 Nicklaus Wirth 延伸 BNF 成為 EBNF (Extended Backus–Naur Form)
- 方法
  - 加入『迴圈語法』用以代表重覆出現的意思

將 BNF 改為 EBNF

- 以便消除左遞迴問題，讓遞迴下降法可以剖析 EBNF 語法。

▶圖 7.8 將數學運算式的 BNF 改寫為 EBNF 語法

(a) BNF 語法

```
E = T | E [+-] T
T = F | T [*/] F
F = N | ' (' E ') '
N = [0-9]+
```

(b) EBNF 語法

```
E = T ([+-] T)*
T = F ( [*/] F)*
F = N | ' (' E ') '
N = [0-9]+
```

本書所使用的語法符號 (1)

- 中括號
  - 代表一群字的集合
  - 範例： [a-zA-Z] 代表 所有的英文字母

- 大括號
  - 單純用來括住一個範圍，並用星號、加號、問號代表重複次數。
  - 範例：E = T ([+-] T)*
    - 代表 ([+-] T) 部分可重複數次 (零次以上)。

本書所使用的語法符號 (2)

- 星號 (...)*
  - 代表 (...) 部分重複比對,
  - 其中的星號 * 代表出現「零次以上」
- 加號 (...)+
  - 代表 (...) 部分重複比對,
  - 其中的星號 * 代表出現「一次以上」
- 問號 (...)?
  - 代表 (...) 部分重複比對,
  - 其中的星號 ? 代表出現「零次或一次」。

7.3 語意理論

- 說明
  - 語意理論所探討的是語法所代表的意義,
  - 也就是某個語法應該如何被執行,
  - 或者如何轉換成組合語言的問題。

- 高階語言
  - 語法理論 → 語意理論 → 執行平台

結構化程式語言

- 說明
  - 目前的程式語言大多屬於結構化程式語言
  - 使用 if、for、while 作為控制邏輯

- 範例
  - C、C++、C#、 Java、Obj C、Python、Perl

結構化的語意

- 指定：a = 5;
- 運算：b+3*d;
- 循序： s1; s2; s3; s4; ….
- 分支：if (…) … else (…)
- 迴圈：for (…) { …} , while (…) { … }
- 函數：f() {…}, f();

結構化程式的構造方式

表格 7.1 結構化程式的構造方式

結構類型	語法	範例
指定結構	ASSIGN = ID '=' EXP	x = 3*y+5
運算結構	EXP = T ([+-] <T>)* T = F ([*/] <F>)*	3*y+5
循序結構	BASE_LIST = (BASE)*	t=a; a=b; b=t;
分支結構	IF = 'if' ' (' COND ') ' BASE ('elseif' '(' COND ')' BASE)* ('else' BASE)	if (a>b) c=a; else c=b;
迴圈結構	WHILE = 'while' '(' COND ')' BASE [2]	while (i<=10) { sum = sum+i; i++;}
函數結構	FDEF = ID '(' ARGS ')' BLOCK FCALL = ID '(' PARAMS ')' '; '	定義：max(a, b) {if (a>b) return a; else return b;} 呼叫：c = max(3, 5);

7.4 執行環境

- 直譯式執行環境 (本章解說)
  - 透過直譯器直接執行高階語言程式

- 編譯式執行環境 (下一章解說)
  - 利用編譯器將高階語言轉換為可目的檔
  - 目的檔可在下列兩種平台上執行
    - 虛擬機器：像是 Java 編譯後就在 JVM 虛擬機上執行。
    - 真實機器：像是 C 語言編譯後就變成機器碼，可直接載入記憶體後在 CPU 上執行。

透過直譯器執行

結構化程式的直譯過程

表格 7.2 結構化程式的直譯過程

結構類型	語法	直譯器動作
指定結構	ASSIGN = ID '=' EXP	計算 EXP 取得結果後, 將結果放入符號表的 ID 變數中
運算結構	EXP = T ([+-] <T>)*	將 T1 [+-] T2 …[+-] Tn 的結果, 放入 EXP 節點中。
循序結構	BASE_LIST = (BASE)*	循序執行 BASE_LIST 的子節點, BASE1 BASE2 … BASEn
分支結構	IF = 'if' ' (' COND ') ' BASE ('elseif' '(' COND ')' BASE)* ('else' BASE)	檢查條件 COND 節點的值, 如果為真, 則執行對應的 BASE, 若均為假, 則執行 else 語句中的 BASE
迴圈結構	WHILE = 'while' '(' COND ')' BASE	當 COND 節點的值為真時, 執行 BASE 節點, 直到 COND 節點的值為假時, 才跳到下一個語句中。
函數結構	FDEF = ID '(' ARGS ')' BLOCK FCALL = ID '(' PARAMS ')' '; '	當呼叫函數 FCALL 時, 將 ARGS 參數取代為 PARAMS, 然後執行 BLOCK 區塊

直譯器的演算法 (1)

直譯器的演算法

```
Algorithm run(node)
  switch (node.tag) {
   ...
   case ASSIGN
      id = node.childs[0]
      exp = node.childs[2]
      SymbolTable[ id ] = run(exp)
   case EXP
      term1 = node.childs[0]
      run(term1)
      node.value = term1.value
      for (i=1; i<node.childCount; i+=2)
        op = node.childs[i].tag
        term2 = node.childs[i+1]
        run(term2)
        if (op="+")
          node.value += term2.value
        else if (op="-")
          node.value - term2.value
        end if
      end for
```

說明

```
解譯 node 節點 (以遞迴方式)
判斷節點類型

ASSIGN = ID '=' EXP
  取出變數
  取出算式
  將算式的結果指定給變數
EXP = T ([+-] T)*
  取得第一個項目
  解譯第一個項目
  設定父節點的值 (運算結果)

    取得下一個運算符號
    取得下一個運算元
    解譯下一個運算元
    如果是加號
      運算結果 += 運算元
    如果是減號
      運算結果 -= 運算元
```

直譯器的演算法 (2)

```
case BASE_LIST
  for (i=0; i<node.childCount; i++)
    run(node.childs[i])
  end for
case IF
  for (i=0; i<node.chiidCount; i++)
    if (i==0 &&
      node.childs[i].token = "if")
    or (i>0 &&
      node.childs[i].token = "elseif")
      cond = node.childs[i+2]
      run(cond)
      if (cond.value = true)
        base = node.childs[i+4]
        run(base)
        break
      end if
      i += 4
    else if (node.childs[i].token =
      "else")
      base = node.childs[i+1]
      run(base)
    end if
  end for
```

```
BASE_LIST= ( BASE )*
  循序的執行每個子節點


IF = 'if' '(' COND ')' BASE 'elseif' …
  查看每個子節點
    如果是第一個 if 關鍵字

    或者是 elseif 關鍵字

      取得條件節點 [3]
      計算條件節點
      如果條件為真
        取得 BASE 節點
        執行 BASE 節點


      跳過 'if' '('EXP ')' BASE
    如果是 else 關鍵字

      取得 BASE 節點
      執行 BASE 節點
```

直譯器的演算法 (3)

```
   case WHILE
      cond = node.childs[2]
      base = node.childs[4]
      while (run(cond)==true)
        run(base)
      end
   case FCALL
      id = node.childs[0]
      params = node.childs[2]
      fdef = functionTable[id]
      call(fdef, params)
   end switch
End Algorithm

Algorithm call(fdef, params)
  body = fdef.body.replace(fdef.args, params)

  bodyNode = parse(body)
  run(bodyNode)
End Algorithm
```

```
WHILE = 'while' '(' COND ')' BASE
  取得 COND 節點
  取得 BASE 節點
  當條件 COND 為真時
     執行 BASE 節點

FCALL = ID '(' PARAMS ')' '; '
  取得函數名稱
  取得參數
  取得函數內容
  呼叫該函數

FDEF = ID '(' ARGS ')' BLOCK
  將程式內容取出，並將參數
     ARGS 取代為 PARAMS
  剖析 body 程式
  執行 body 程式
```

7.5 實務案例：C 語言

- C 語言的語法及語意
  - 基本單元、指定結構、運算結構、循序結構、分支結構、迴圈結構、函數結構
- C 語言的執行環境
  - 程式段、資料段、BSS段
  - 使用框架存取參數與區域變數

C 語言的語法及語意

- 基本單元
  - x, 35, "hello!", x[3], f(x), f(), rec.x, rec->x, x++, x--
- 指定結構
  - a=3*x
- 運算結構
  - a * 3 + b[5]
- 循序結構
  - i=1; x=f(3); t=a; a=b; b=t;
- 分支結構
  - if(a>b) x=a; else x=b;
- 迴圈結構
  - for (i=0; i<10; i++) sum += i;
- 函數結構
  - int max(x,y) { return x>y?x:y; } c = max(a,b);

語法：基本單元

- 範例
  - X
  - 35
  - "hello!“
  - x[3]
  - f(x)
  - f()
  - rec.x
  - rec->x
  - x++
  - x--

圖 7.10 C 語言基本單元的語法

C 語言的 EBNF 語法 (基本單元)	說明
postfix_exp =	後置算式 =
primary_exp	基本算式
| postfix_exp '[' exp ']'	陣列索引 x[3]
| postfix_exp '(' arg_exp_list ')'	函數呼叫 f(x)
| postfix_exp '(' ')'	函數呼叫 f()
| postfix_exp '.' id	結構欄位 rec.x
| postfix_exp '->' id	結構欄位 rec->x (指標版)
| postfix_exp '++'	x++
| postfix_exp '--'	x—
;	
primary_exp =	基本算式
id	變數
| const	常數
| string	字串
| '(' exp ')'	(運算式)
;	

語法：指定結構

- 範例
  - a=3*x
  - a = b = 3*x
  - a = (x != y)

圖 7.11 C 語言指定結構的語法

C 語言的 EBNF 語法 (指定結構)	說明
assign_exp = (var_ref assign_op)* cond_exp	指定運算

語法：運算結構

- 範例：a * 3 + b[5]
  - additive_exp (a*3+b) : additive_exp (a*3) + mult_exp (b[5])

圖 7.12 C 語言運算結構的語法

C 語言的 EBNF 語法 (基本單元)	說明
cond_exp = logic_or_exp ('?' exp : logic_or_exp)*	條件運算
logic_or_exp = logic_and_exp (logic_or_op logic_and_exp)*	邏輯運算
logic_and_exp = bit_or_exp (logic_and_op bit_or_exp)*	
bit_or_exp = bit_xor_exp (bit_or_op bit_xor_exp)*	位元運算
bit_xor_exp = bit_and_exp (bit_xor_op bit_and_exp)*	
bit_and_exp = equal_exp (bit_and_op equal_exp)*	
equal_exp = relational_exp (equal_op relational_exp)*	
relational_exp = shift_exp (relational_op shift_exp)*	關係運算
shift_exp = add_exp (shift_op add_exp)*	
add_exp = mult_exp (add_op mult_exp)*	數學運算
mult_exp = cast_exp (mult_op cast_exp)*	
cast_exp = ((type_name))* unary_exp	轉型運算
unary_exp = unary_op cast_exp	單元運算
| (prefix_op)* postfix_exp |'sizeof' '(' type ')'	
postfix_exp = primary_exp postfix_phrase	後置運算

語法：循序結構

- 範例
  - i=1; x=f(3); t=a; a=b; b=t;

圖 7.13 C 語言循序結構的語法

C 語言的 EBNF 語法 (基本單元)	說明
exp = seq_exp seq_exp = assign_exp (seq_op assign_exp)*	循序結構 a = b; b = t;

語法：分支結構

- 範例：
  - if (a>b) x=a; else x=b;
  - switch (c) {
    case ‘a’: x+=a;
    case ‘b’: x+=b;
    default: x+=c;
    }

圖 7.14 C 語言分支結構的語法

C 語言的 EBNF 語法 (基本單元)	說明
sel_stat = 'if' '(' exp ')' stat ('else' 'if' '(' exp ')' stat)* ('else' stat)? | switch (exp) stat	分支結構 if (ab) x=a; else x=b; switch (c) {…}

語法：迴圈結構

- 範例
  - while (true) { …}
  - do { … } while (! end)
  - for (i=0; i<10; i++) { sum += i; }

圖 7.15 C 語言迴圈結構的語法

C 語言的 EBNF 語法 (基本單元)	說明
iter_stat	迴圈結構
= 'while' '(' exp ')' stat	while 迴圈
| 'do' stat 'while' '(' exp ')' ';'	do while 迴圈
| 'for' '(' exp ';' exp ';' exp ')' stat	for 迴圈
…	

語法：函數結構

圖 7.16 C 語言函數結構的語法

C語言的 EBNF 語法 (基本單元)	說明
function_def =	函數本體
decl_specs declarator	static int f(n)
decl_list	int n;
compound_stat	{ … }
…	
declarator =	函數宣告
pointer d_declarator	static int f(n)
…	
d_declarator =	函數宣告 (無指標)
id	x
| '(' declarator ')'	(int (f*)(int))
| d_declarator '[' const_exp ']'	x[10]
| d_declarator '[']'	x[]
| d_declarator '(' param_types ')'	f(int x)
| d_declarator '(' id_list ')'	f(x, y)
| d_declarator '(')'	f()
…	…

函數結構的範例

- 語法
  - function_def = decl_specs declarator decl_list compound_stat

- 範例
  - static int f(n) int n; {return n*n; }
    - decl = static
    - spec = int
    - declarator = f(n)
    - decl_list = int n;
    - compound_stat = {return n*n; }

C 語言的執行環境

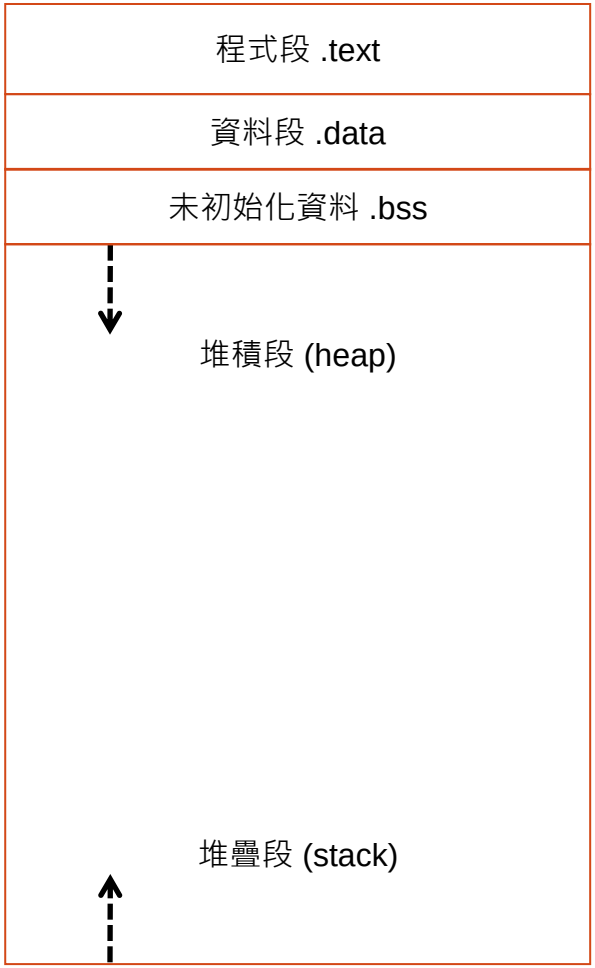
- 編譯式語言
  - C 語言通常採用編譯的方式, 先將程式編譯為機器碼(目的檔或執行檔), 然後才在目標平台上執行 C 語言。

- 目標：特定 CPU 的機器碼
  - C 語言編譯後的機器碼通常是與平台相關的, 是可以直接被 CPU 執行的 2 進位碼, 因此速度非常的快, 這也是 C 語言的優點之一。

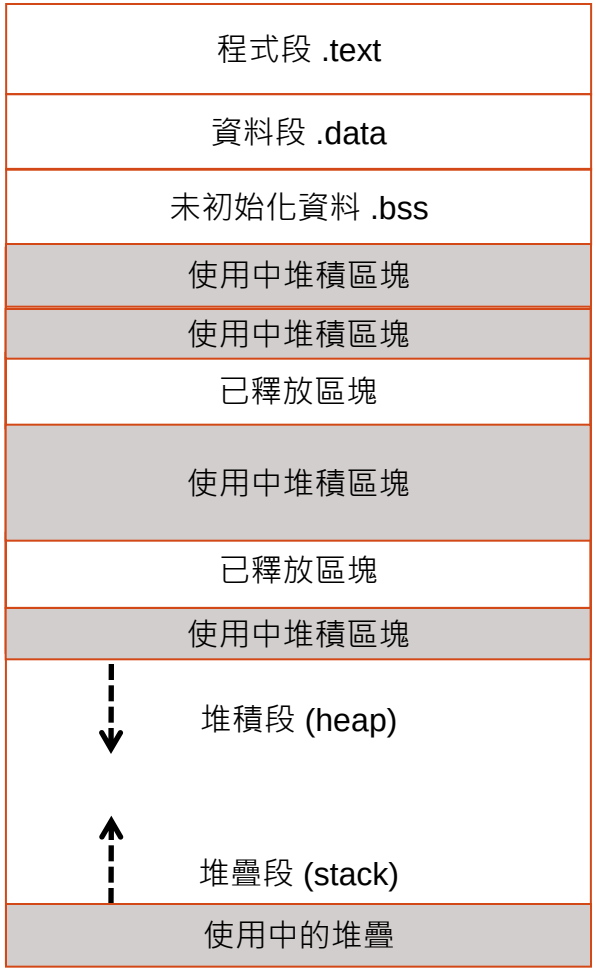
編譯與執行環境

- 編譯後
  - C 語言在執行時, 通常會編譯為目的檔或執行檔的形式, 這個些檔案包含程式段、資料段、BSS 段等區域
- 執行時
  - 在執行時還會多出堆疊 (Stack) 與堆積 (Heap) 等兩個區段。

圖 7.17 C 語言的執行時的記憶體配置圖



(a) 程式開始時的記憶體分配情況



(b) 程式執行中的記憶體分配情況

堆疊與堆積

- 堆疊段的用途：
  - 儲存「區域變數」、「參數」、「返回點」
- 堆積段的用途：
  - 提供 malloc() 的空間，並在 free() 時回收。
- 比較
  - 與堆積段的成長方向是相反的, 假如堆積由上往下成長, 堆疊段的成長方向就會是由下往上。堆疊與堆積兩段共用同一塊記憶體空間, 但是起始點與成長方向完全相反。

C 語言如何存取堆疊中的區域變數

- 問題：必須支援遞回呼叫
  - 當 C 語言的函數想要存取參數或區域變數時, 通常不能透過變數名稱存取這些變數, 否則就不能支援遞迴呼叫了。
  - 因為在遞迴呼叫的過程中, 參數名稱與區域變數的名稱雖然相同, 但是不同層次的遞迴所『看見的』變數內容是不同的。

框架

- 一個函數的參數與區域變數所形成的堆疊區塊， 通常稱之為框架 (Frame)

- 為了要存取這個框架, 我們可以設定一個框架暫存器 (Frame Pointer, FP), 然後使用相對定址的方式存取這些變數。

- 在 CPU0 中, 我們會習慣以 R11 作為框架暫存器, 因此我們也用 FP 稱呼 R11。

使用框架存取參數與區域變數

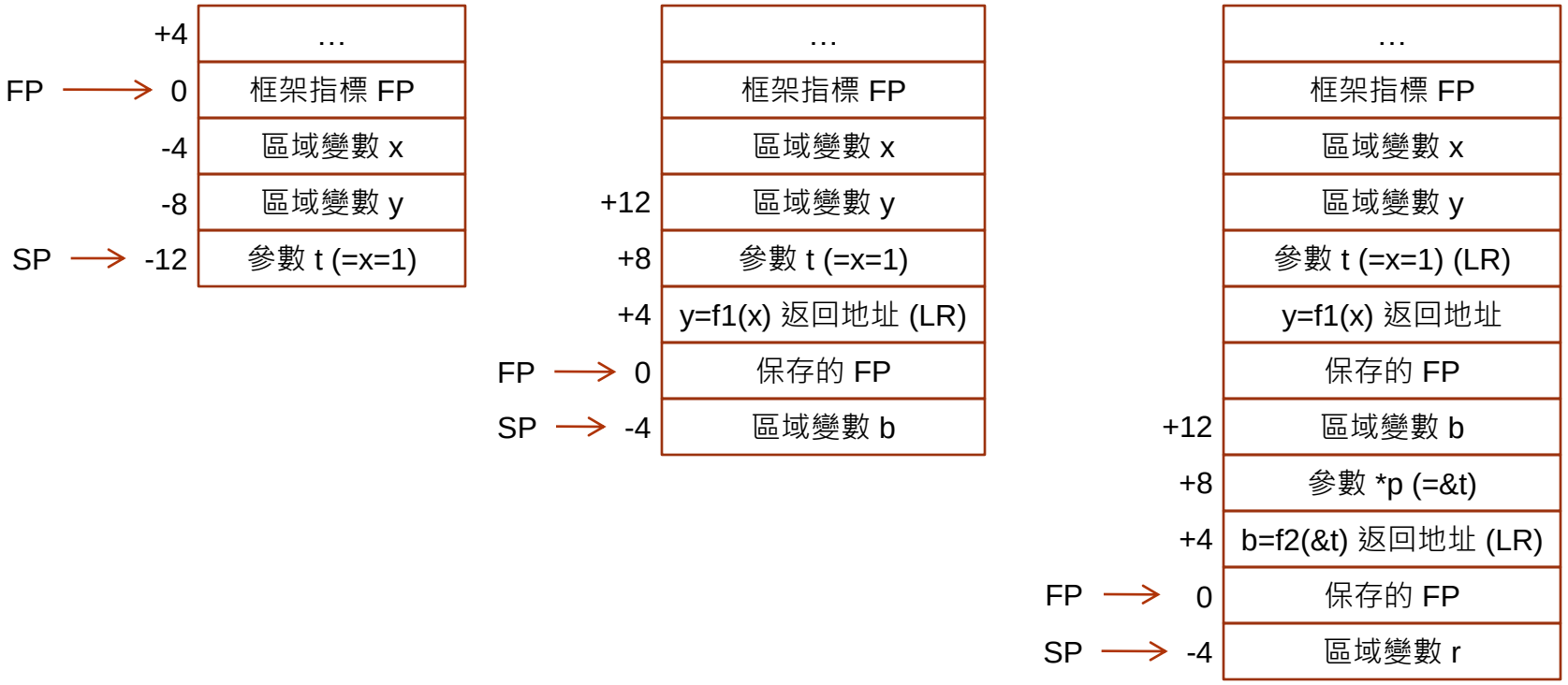
- 使用相對於框架暫存器的定址法
- 進入函數前設定好框架暫存器的內容

具有兩層函數呼叫的 C 語言程式

▶範例 7.1 具有兩層函數呼叫的 C 語言程式

```
1   int main() {
2     int x = 1;
3     int y;
4     y = f1(x);
5     return 1;
6   }
7   int f1(int t) {
8     int b = f2(&t);
9     return b+b;
10  }
11  int f2(int *p) {
12    int r= *p+5;
13    return r;
14  }
```

圖 7.18 函數呼叫時的堆疊與框架變化情形



(a) 呼叫 y=f1(x) 之前

(b) 呼叫 y=f1(x) 之後

(c) 呼叫 b=f2(&t) 之後

如何用組合語言存取堆疊中的區域變數？ (main)

▶範例 7.2 範例 7.1 程式對應的組合語言

	組合語言	說明	C 語言 (真實版)
1	_main:	void main()	void main() {
2	//****前置段********		
3	PUSH LR	將 LR 推入堆疊	
4	PUSH FP	將 FP 推入堆疊	
5	MOV FP, SP	設定新的 FP	
6	SUB SP, SP, 8	分配參數空間	
7	//****主體段********		
8	CALL _init	呼叫 _init 進行初始化	
9	MOV R3, 1	R3=1	int x = 1;
10	ST R3, [FP-4]	x = [FP-4] // = R3 = 1	int y;
11	PUSH R3	將 x 推入堆疊	
12	CALL _f1	呼叫函數 f1();	y = f1(x);
13	ADD SP, SP, 4	恢復原先堆疊指標	
14	MOV R3, R1	R3=R1 //=回傳值 f1(x)	
15	ST R3, [FP-8]	y=[FP-8] = R3	
16	//****結束段********		
17	MOV SP, FP	恢復 SP	
18	POP FP	恢復 FP	
19	RET	PC=LR, 回到呼叫點	}

如何用組合語言存取堆疊中的區域變數？ (f1)

20	f1:		int f1(int t) {
21	//****前置段********		
22	PUSH LR	將 LR 推入堆疊	
23	PUSH FP	將 FP 推入堆疊	
24	MOV FP, SP	設定新的 FP	
25	SUB SP, SP, 4	分配區域變數空間 b	
26	//****主體段********		
27	ADD R3, FP, 8	R3 = FP+8 = &t	
28	PUSH R3	PUSH R3 // (&t)	
29	CALL f2	呼叫函數 f2()	int b = f2(&t);
30	ADD SP, SP, 4	恢復原先堆疊指標	
31	ST R1, [FP-4]	b = R1	
32	LD R3, [FP-4]	R3= [FP-4] // = b	
33	LD R2, [FP-4]	R2 = [FP-4] // = b	
34	ADD R3, R3, R2	R3 = R3+R2 = b + b	return b+b;
35	MOV R1, R3	傳回值 R1 = R3	
36	//****結束段********		
37	MOV SP, FP	恢復堆疊	
38	POP FP	將 FP 從堆疊取出	
39	POP LR	恢復 SP	
40	RET	恢復 FP	}

如何用組合語言存取堆疊中的區域變數？ (f2)

```
41  f2:                  PC=LR, 回到呼叫點          int f2(int *p) {
42  //****前置段********
43     PUSH LR           將 LR 推入堆疊
44     PUSH FP           將 FP 推入堆疊
45     MOV FP, SP        設定新的 FP
46     SUB SP, SP, 4     分配區域變數空間 r            int r=*p+5;
47  //****主體段********
48     LD R3, [FP+8]     R3=[FP+8] // =*p 的位址
49     LD R2, [R3]       R2 = [R3] = *p
50     ADD R3, R2, 5     R3 = R2+5 = *P+5
51     ST R3, [FP-4]     r = [FP-4] = R3
52     MOV R1, R3        傳回值 R1 = R3               return r;
53  //****結束段********  ******結束段********
54     POP FP            恢復 FP
55     POP LR            PC=LR, 回到呼叫點          }
56     RET               }
```

C 語言進入函數時的基本動作

- 1. 先保存連結暫存器 LR 的值, 以避免該函數再度呼叫子函數時, LR 的值會被覆蓋。
- 2. 接著再保存舊的框架暫存器 FP, 以便函數返回前可以恢復 FP。
- 3. 接著, 將框架暫存器更新為堆疊的頂端 (MOV FP, SP)。
- 4. 最後, 再分配好區域變數的空間之後, 前置段的工作就完成了。

前置段程式	說明
PUSH LR	將 LR 推入堆疊
PUSH FP	將 FP 推入堆疊
MOV FP, SP	設定新的 FP
SUB SP, SP, <N>	分配大小為 <N> 的參數空間

C 語言進入函數時的基本動作

- 1. 先保存連結暫存器 LR 的值, 以避免該函數再度呼叫子函數時, LR 的值會被覆蓋。
- 2. 接著再保存舊的框架暫存器 FP, 以便函數返回前可以恢復 FP。
- 3. 接著, 將框架暫存器更新為堆疊的頂端 (MOV FP, SP)。
- 4. 最後, 再分配好區域變數的空間之後, 前置段的工作就完成了。

前置段程式	說明
PUSH LR	將 LR 推入堆疊
PUSH FP	將 FP 推入堆疊
MOV FP, SP	設定新的 FP
SUB SP, SP, <N>	分配大小為 <N> 的參數空間

C 語言離開函數時的基本動作

- 1. 恢復堆疊指標 SP
- 2. 恢復框架指標 FP
- 3. 恢復連結暫存器 LR
- 4. 用 RET 返回呼叫點

```
36  //****結束段********
37     MOV SP, FP
38     POP FP
39     POP LR
40     RET
41  f2:
```

恢復堆疊

恢復 FP
恢復 LR
返回

```
}
int f2(int *p)
```

結語

- C 語言的語法及語意
  - 基本單元、指定結構、運算結構、循序結構、分支結構、迴圈結構、函數結構
- C 語言的執行平台
  - 直接編譯成目標 CPU 上的機器碼
  - 目的檔：包含程式段、資料段、BSS段
  - 執行時：增加堆疊段與堆積段

習題

- 7.1 請說明何謂 BNF 語法？何謂 EBNF 語法？並比較兩者的異同。
- 7.2 請將 BNF 語法 A = B | A '.' B 轉換為 EBNF 語法。
- 7.3 請寫出 C 語言當中 for 迴圈的 BNF 語法。
- 7.4 請說明何謂直譯器？
- 7.5 請說明何謂編譯器？
- 7.6 請比較直譯器與編譯器兩者的異同。
- 7.7 請說明何謂語法理論？
- 7.8 請說明何謂語意理論？
- 7.9 請說明何謂框架 ？
- 7.10 請舉例說明 C 語言如何利用框架暫存器存取參數與區域變數？