編譯器研究

劉瑞姿

資工二 |110710540

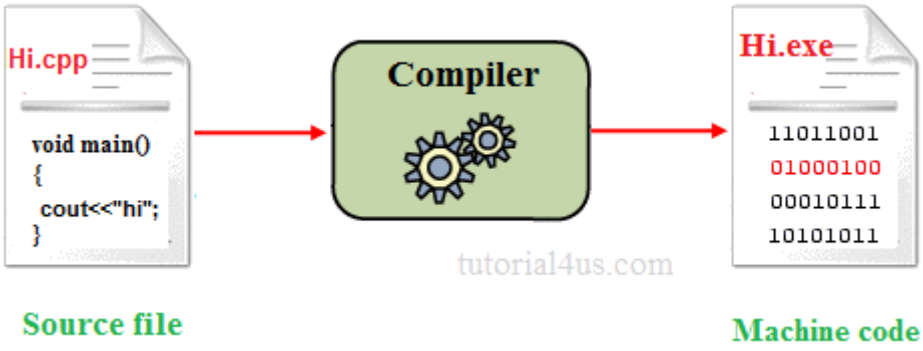
目錄

說明: 本報告的節錄內容主要來源為—編譯器(Compiler)與連結器(Linker)的運作原理、Interpreter (直譯器) 與 complier (編譯器) 的區別、輕鬆理解 C 語言相關的編譯器 gcc 和 g++、編譯器

一、 編譯器是甚麼

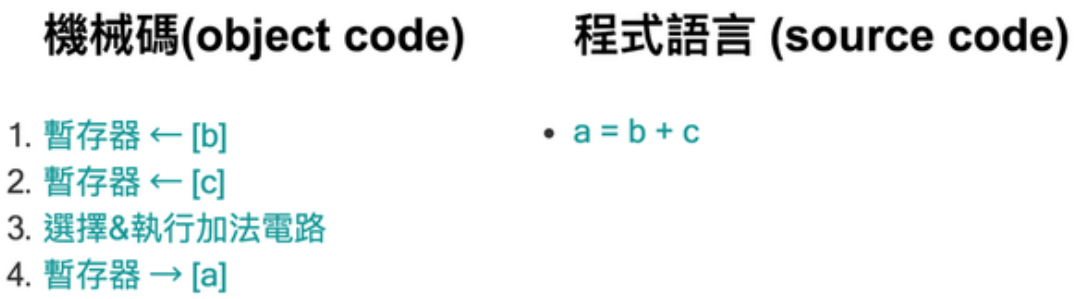
編譯器用途是把一個語言（source language）的程式碼轉換成另外一個語言（target language）的程式碼，簡單來說，就是將高階語言轉換稱組合語言或機器碼的程式。

主要目的是方便人編寫、閱讀、維護的進階電腦語言所寫作的原始碼程式，翻譯為電腦能解讀、執行的低階機器語言的程式，也就是執行檔。編譯器將原始程式作為輸入，翻譯產生使用目標語言的等價程式。原始碼一般為高階語言，如 C、C++、C#、Pascal、Java 等，而目標語言則是組合語言或目標機器的目的碼。



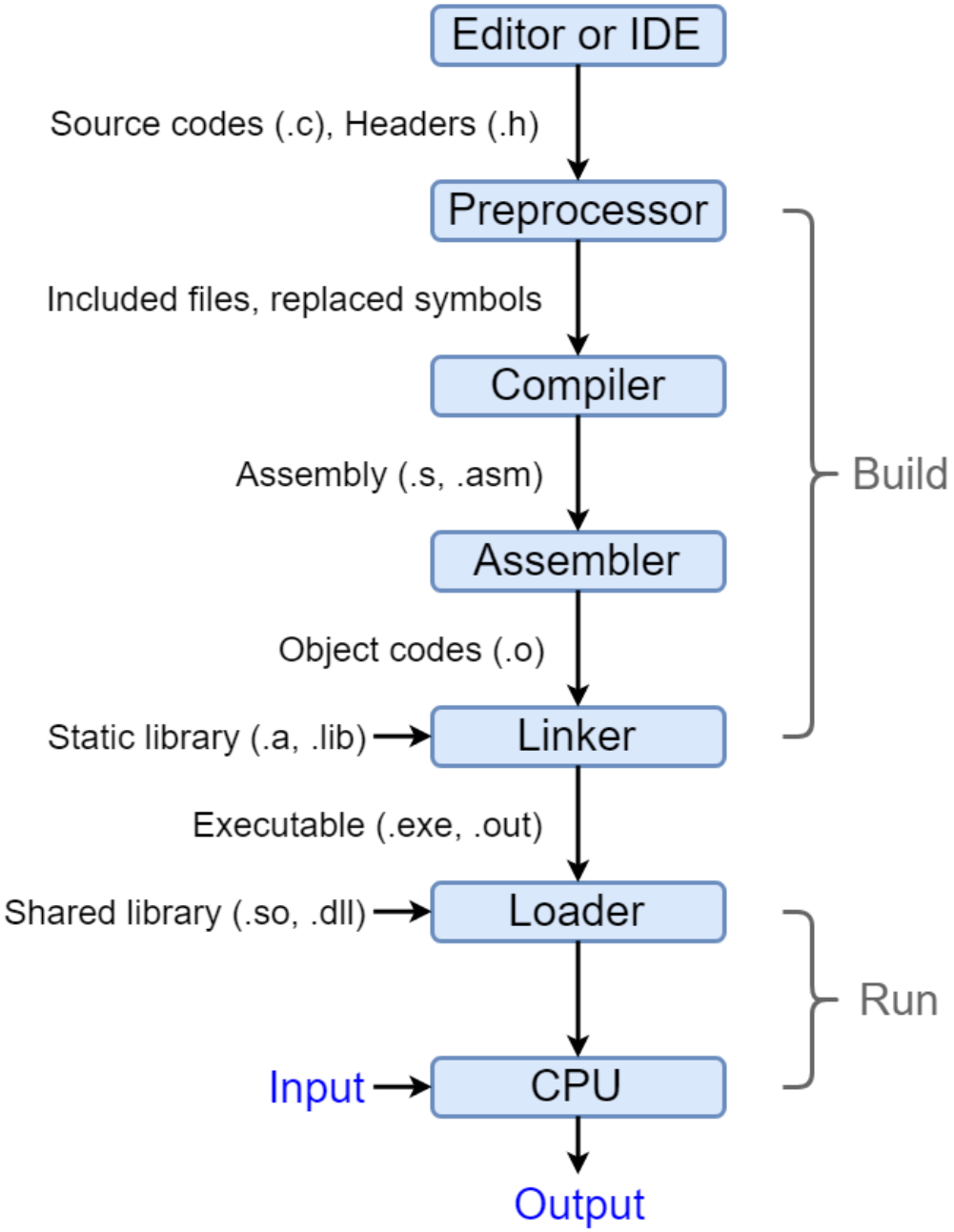
原始碼和機械碼並不是一行對一行翻譯的， 因為 CPU 並無法處理複雜的動作，它只能夠單純地載入資料、讓訊號流過電路、輸出結果，CPU 無法處理有語意的指令，所以如下圖的例子，其中的 [b] 代表變數 b 在記憶體中的位址(address)，原始碼裡面一行 a=b+c，這述句的意思是把 b 和 c 相加以後，以此結果賦予 a

變數的值，編譯器所產生的機械碼可能是把 [b] 位置的數值搬移到 CPU 的暫存器，接著把 [c] 位置的數值搬移到 CPU 的另一個暫存器，接著如圖一所示，輸出加法運算的機械碼(設定 CPU 周圍的針腳以選用加法電路)，若干 clock 以後，某個暫存器中的值變成方才兩個暫存器相加的值，接著下一行，另一個搬移的機械碼又把結果搬移到記憶體的 [a] 位置。



二、 編譯器用在何處

在了解編譯器之前，我們必須知道編譯器是用在麼地方，就是高階語言到生成個執行檔案的處理過程的其中一個階段。



◆ 程式的編譯、組譯、連結、載入之流程

編輯器/ 整合開發環境（Editor / IDE)

編輯程式碼，C 語言的原始碼，副檔名為 .c，標頭檔副檔名為 .h

例如 Visual Studio、 DEV-C++、vim ，vim 是大部分系統都有內建的編輯器。它可以直接在 terminal 介面裡面操作，當你只是需要快速修改某個檔案，或是握一些簡單的編輯又不想開龐大的 IDE 時，vim 是個非常方便的選擇。

IDE 是一種輔助開發人員開發軟體的應用軟體，通常包括程式語言編輯器、自動構建工具、通常還包括除錯器。有些 IDE 包含編譯器 / 直譯器，如微軟的 Microsoft Visual Studio，有些則不包含，如 Eclipse、SharpDevelop 等。

預處理器 (Preprocessor)

預先處理原始碼，將須引入的檔案或是 macro 展開

以 C 語言為例，通常以 '#' 開頭作為預處理的指令

組合語言 (Assembly)

副檔名為 .s, .asm

屬於低階程式語言會針對不同電腦架構及作業系統而有不同，會需要這樣是因為對於不同的架構可以有不同的優化方式，並且可以使高階語言有可攜性，代表高階語言不需要為了不同的環境寫不同的程式碼，而是透過編譯器將程式碼轉成對應環境的組合語言。

常見有 x86-64、ARM、MIPS

組譯器 (Assembler)

將低階語言所寫的程式翻譯成目的檔

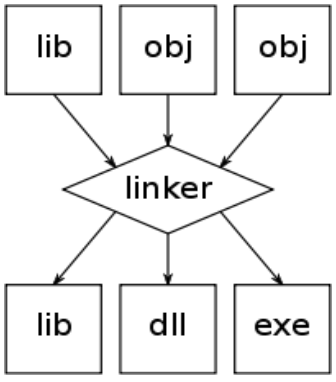
目的碼 (Object codes)

目的檔的副檔名為 .o

目的碼通常是 CPU 可以直接執行的機器碼或是暫存器傳遞語言，

連結器 (Linker)

將多個目標檔或靜態函式庫合併成一個可執行檔或函式庫的工具



載入器 (Loader)

是作業系統的一部份，用於把程式和動態函式庫的指令載入到記憶體

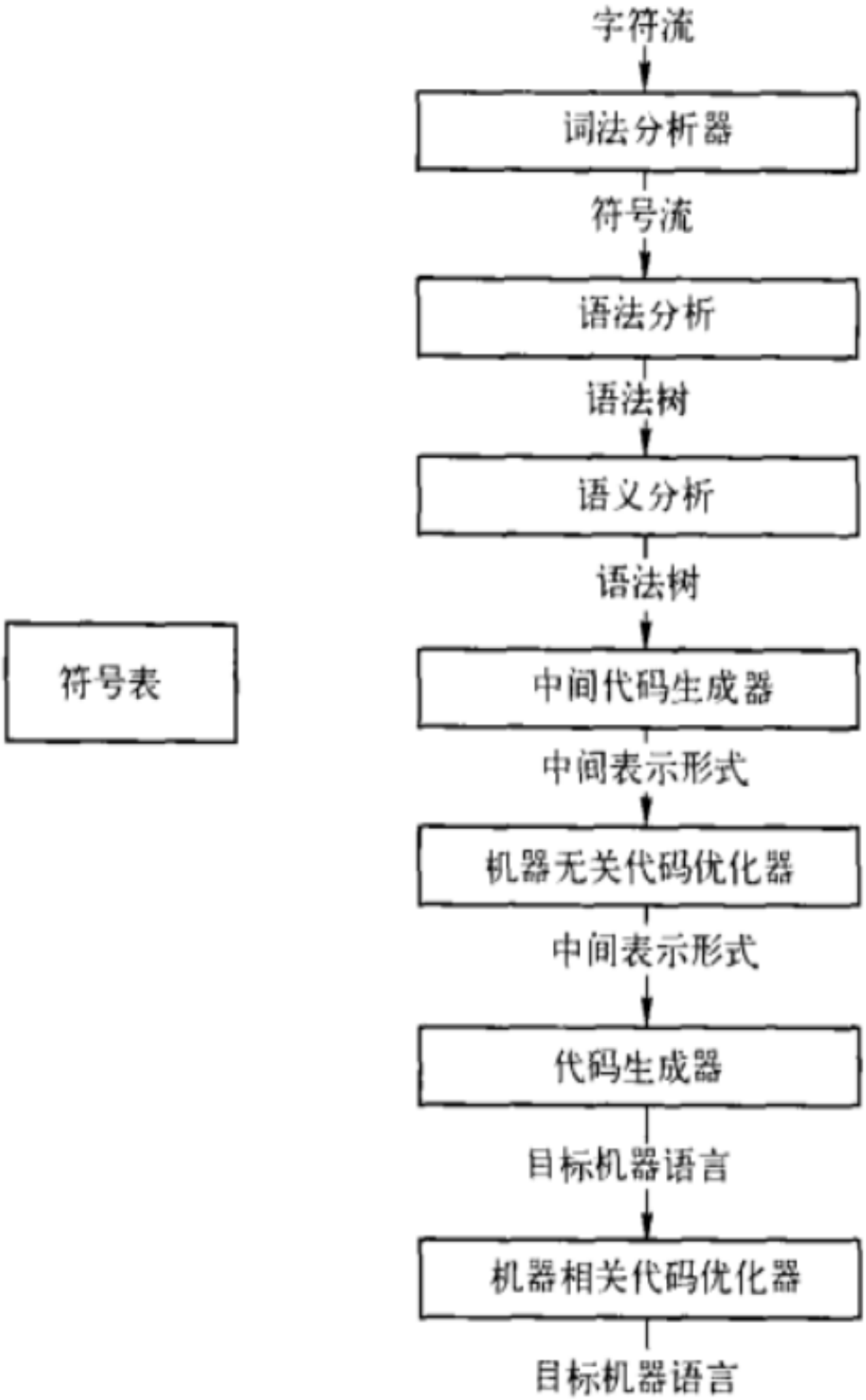
中等待 CPU 執行，當載入完成之後，作業系統會將控制權交給載入的程式碼，讓它開始運作

CPU (Central Processing Unit)

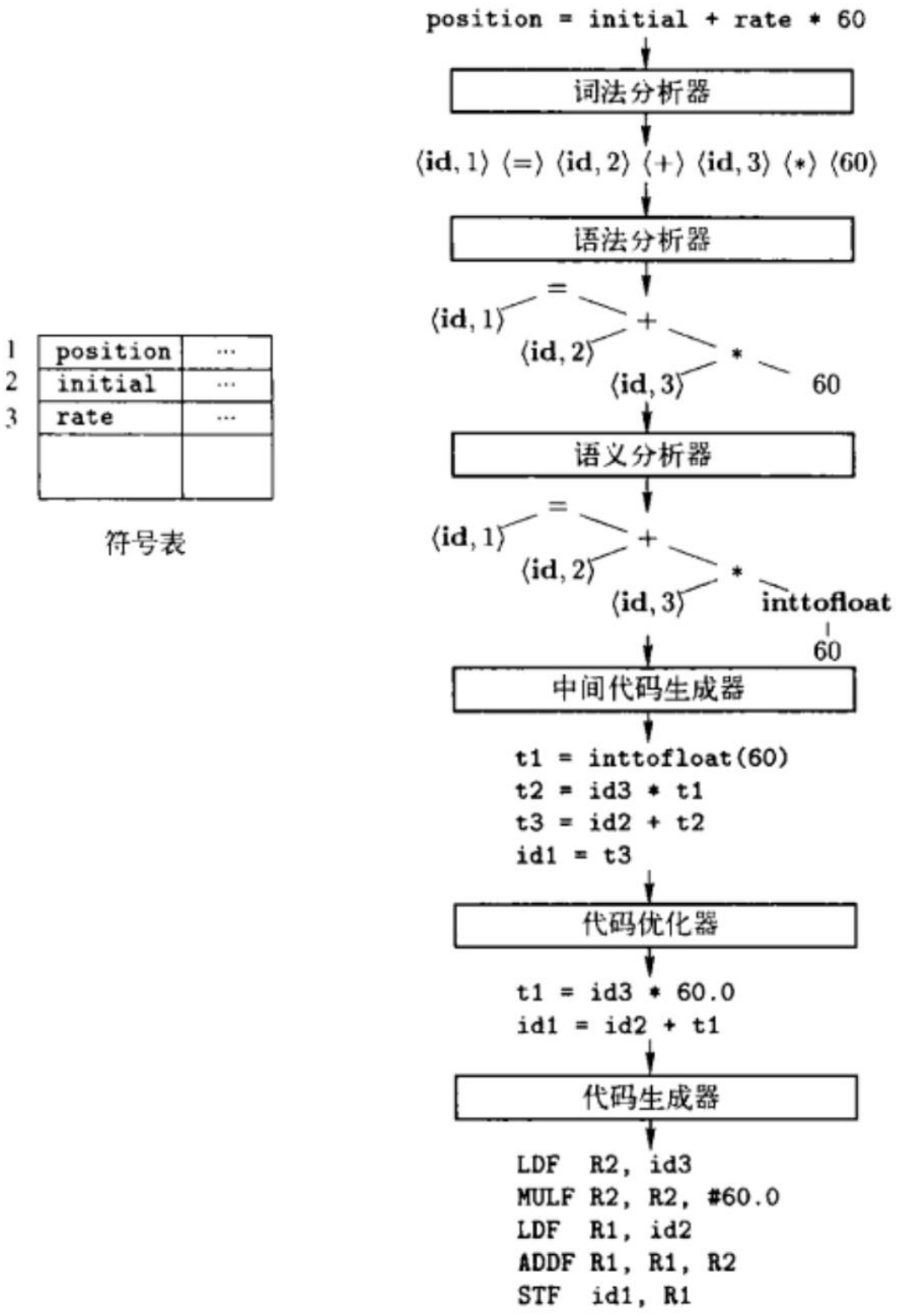
對載入的指令進行運算或儲存等操作

三、 原理流程

編譯主要是把語言轉成組合語言**可以分為六個步驟，為詞彙掃描、語法剖析、語意分析、中間碼產生、最佳化、組合語言產生。**


字符流
词法分析器
符号流
语法分析
语法树
语义分析
语法树
符号表
中间代码生成器
中间表示形式
机器无关代码优化器
中间表示形式
代码生成器
目标机器语言
机器相关代码优化器
目标机器语言

◆ 編譯器的各個步驟



1	position	···
2	initial	···
3	rate	···

符号表

◆ 指定述句的翻譯

1. **詞彙掃描：**

詞彙掃描是編譯的第一步。主要任務是讀入源程式的輸入字元（將程式碼一個字元一個字元的讀入），將其組成詞素（一個字元序列），生成並輸出一個詞法單元序列。對於每個詞素， 詞彙分析器產生如下形式的詞彙單位（token）作為輸出： 〈token-name, attribute-value〉 這個 token 傳給語法剖析。

在這個 toke 中，其中 token-name 表示詞彙掃描過程中的抽象符號，類似於為該詞彙單元賦予唯一 id，而 attribute-value 指向符號表中關於這個 token 的條目 （entry）。符號表條目的資訊會由語意分析和程式碼產生步驟使用。

例如，假設原始程式包含如下的指定述句：

position = initial + rate 60 （1.1）

這個指定述句中的字元可以組合成如下詞素，並對映成如下 token。這些 token 將傳給語法剖析階段：

(1) position 是詞素，被對映成 token〈id, 1〉，其中 id 是抽象符號識別字的縮寫，而 1 指向符號表中 position 對

應的條目。識別字對應的符號表條目存放該識別字有關的 資訊，例如它的名稱和類型。

(2) 指定符號 = 是對映成詞彙單位〈=〉的詞素。因為這個 token 不需要屬性值，所以我們省略了第二個部分。也可以使用 assign 這樣的抽象符號作為 token 的名稱，但是為了標記上的方便，我們選擇使用詞素本身作為抽象符號的名稱。

(3)initial 是對映成 token〈id,2〉的詞素，其中 2 指向 initial 對應的符號表條目。

(4)+是對映成 token〈+〉的詞素。

(5)rate 是對映成 token〈id, 3〉的詞素，其中 3 指向 rate 對應的符號表條目。

(6)*是對映成詞彙單位〈*〉的詞素。

(7)60 是對映成 token〈60〉的詞素。

分隔詞素的空格會被詞彙分析器忽略掉。

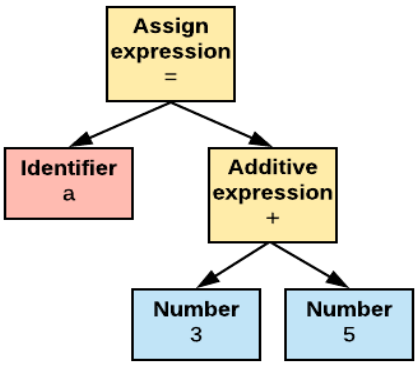
是經過詞彙分析之後，指定述句 1.1 表示成如下的 token：

〈id,1〉〈=〉〈id,2〉〈+〉〈id,3〉〈*〉〈60〉（1.2）

在此述句，token 名稱 =、+和 分別是表示指定、加法運算子、乘法運算子的抽象符號。

2. 語法剖析：

語法分析使用詞法分析產生的詞法單元的第一個分量來建立樹形中間表示。從而給出詞法單元流的語法結構，常用表示方法為語法樹。樹的內部節點表示一個運算，此節點的子節點表示運算的分量。在編譯器的後續階段將會使用該語法樹來分析程式，並生成目標程式。詞彙單位串流對應的語法樹顯示為語法分析器的輸出。



這棵樹顯示了指定述句

Position = initial + rate 60

中各個運算的執行順序。這棵樹有標號為的內部節點，是它的左子節點，整數 60 是它的右子節點。節點表示識別字

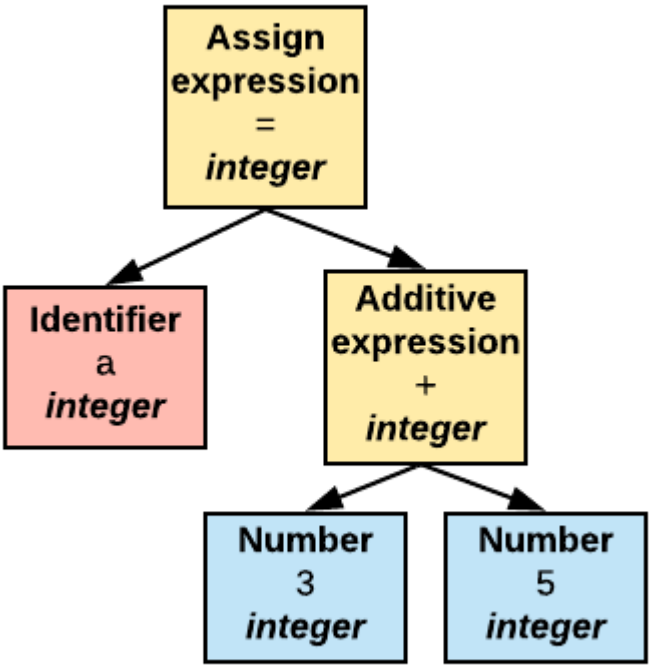
rate。標號為的節點指明了我們必須首先把 rate 的值乘上 60。標號為 + 的節點表明我們必須把相乘的結果加上 initial 的值。這棵樹的根節點的標號為 =， 它表明我們必須把相加的結果儲存到識別字 position 對應的位置。這個運算順序和一般的算術規 則相同，即乘法的優先順序高於加法，因此乘法應該在加法之前計算。

3. 語意分析：

接下來要判斷語意的正確性，比如說我們允許兩個常量相乘，但不允許對兩個指標做相乘。而語意又可分成 靜態語意 以及 動態語意 ，靜態語意是編譯器可以在 編譯期分析的，並且回報錯誤，例如：型態的轉換否正確。而動態語意則是在 執行期 相關的，例如：以零當作除數是一個執行期的語意錯誤。經過分析後，我們可以在文法樹上加上語意型別。

假設 position、initial 和 rate 已宣告為浮點數型 別，而詞素 60 本身形成整數。語意分析器的型別檢查程式發現運算子 用於浮點數 rate 編譯系統設計 1-10 和整數 60。在

這種情況下，這個整數可以轉換成浮點數。請注意，語意分析器的輸 出有運算子 inttofloat 的額外節點。inttofloat 明確地把它的整數參數轉換為浮點數。



4. 中間碼產生：

在將源程式翻譯成目標程式的過程中，編譯器會根據需要產生多箇中間表示。如在語法分析和語意分析階段產生的語法樹。在對源程式完成語法和語義分析後，編譯器一般會產生一個低階的類似機器語言的中間表示。該中間表示應該易於生成並且很容易翻譯成目標程式。如三地址程式碼表示會將

源程式轉成一組具有三個運算分量的類組合語言：

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

每一個指令一般具有三個運算分量，每個賦值語句右邊只有一個運算子。之所以如此細化，是因為程式的執行是有先後順序的，比如上圖的乘法應該在加法之前完成，通過指令的順序完成邏輯演算法的運算順序。

5. 最佳化：

程式碼優化是在程式碼優化步驟試圖優化改進中間程式碼，以便生成更好的目的碼，“更好”可能是以更快、更短或者更能耗低為目標。如上述賦值語句，上面的 60 要使用 inttofloat 演算法轉成 float 型別，但顯然如果直接在編譯時刻將 60 一勞永逸的改成 60.0，就可以消除 inttofloat 運算，而在程式碼中 t3 只用於使用一次，且僅用於傳遞值，因此可以直接將 t3 的值賦給 id1。從而減少指令數：

```
t1 = id3 * 60.0
id1 = id2 + t1
```

6. 組合語言產生：

程式碼生成器以原始碼中間表示生成目標語言，如果目標語言是機器程式碼，那就要為每個變數指定暫存器或記憶體位置。程式碼生成的一個至關重要的方面就是合理的分配暫存器以存在變數值。如上述程式碼使用暫存器 R1，R2：

```
LDF  R2,  id3
MULF R2,  R2, #60.0
LDF  R1,  id2
ADDF R1,  R1, R2
STF  id1, R1
```

每個指令的第一個運算元指定了目的位址。各個指令中的 F 告訴我們它處理的是浮點數。上面的程式碼把位址 id3 中的內容載入暫存器 R2，然後將其與浮點常數 60.0 相乘。井號“#”表示 60.0 應該視為直接常數。第三個指令把 id2 移到暫存器 R1，而第四個指令把前面計算得到並存放 在 R2 的值加到 R1。最後，暫存器 R1 的值存放到 id1 的位址。這樣，這些程式碼即正確實作了指定述句（1.1）。

四、 實例

1. Gcc 編譯器

因為它原本只能處理 C 語言。GCC 在釋出後很快地得到擴展，變得可處理 C++。之後也變得可處理 Fortran、Pascal、Objective-C、Java、Ada，Go 與其他語言。

許多作業系統，包括許多類 Unix 系統，如 Linux 及 BSD 家族都採用 GCC 作為標準編譯器。

GCC 原本用 C 開發，後來因為 LLVM、Clang 的崛起，它更快地將開發語言轉換為 C++。許多 C 的愛好者在對 C++一知半解的情況下主觀認定 C++的效能一定會輸給 C，但是 Ian Lance Taylor 給出了不同的意見，並表明 C++不但效能不輸給 C，而且能設計出更好，更容易維護的程式[2][3]。

gcc（GNU C Compiler）是 GNU 推出的功能強大、效能優越的多平臺編譯器。gcc 是可以在多種硬體平臺上編譯出可執行程式的超級編譯器，其執行效率與一般的編譯器相比平均效率要高 20%~30%。

- Linux 下的 gcc

現在弄個 C 語言：

```
[root@localhost tmp]# rm -rf *
[root@localhost tmp]# ls
[root@localhost tmp]# vim one.c
```

使用 vim 編輯器：

保存後查看原始碼：

```
[root@localhost tmp]# cat one.c
#include<stdio.h>

void main(){
        printf("www.phpkhbd.com");
}
[root@localhost tmp]#
```

一開始只有：

```
[root@localhost tmp]# ls
one.c
```

使用 gcc 編譯（它可以編譯 C 並連結 C 庫）

```
[root@localhost tmp]# gcc one.c
[root@localhost tmp]# ls
a.out  one.c
[root@localhost tmp]#
```

可見，成功生成了 a.out

執行 a.out

2. Cygwin

Cygwin 包括了一套庫，該庫在 Win32 系統下實現了 POSIX 系統呼叫的 API；還有一套 GNU 開發工具集（比如 GCC、GDB），這樣可以進行簡單的軟體開發；還有一些 UNIX 系統下的常見程式。2001 年，新增了 X Window System。

另外還有一個名為 MinGW 的庫，可以跟 Windows 原生的 MSVCRT 庫（Windows API）一起工作。MinGW 占用記憶體、硬碟空間都比較少，能夠連結到任意軟體，但它對 POSIX 規範的實現沒有 Cygwin 庫完備。簡單的說，它是讓 Windows 下可以執行 bash、dash、csh 的一個方法，即在 Windows 下可以使用 Linux 終端機的指令。

3. Clang

Clang 是 LLVM 編譯器工具集的前端，目的是輸出代碼對應的抽象語法樹，並將程式碼編譯成 LLVM Bitcode。接著在後端使用 LLVM 編譯成平台相關的機器語言 。Clang 支援 C、C++、Objective C。

在 Clang 語言中，使用 Stmt 來代表 statement。Clang 程式碼的單元皆為語句，語法樹的節點類型就是 Stmt。另外 Clang 的表達

式也是語句的一種，Clang 使用 Expr 來代表 Expression，Expr 本身繼承自 Stmt。節點之下有子節點列表。

Clang 本身效能優異，其生成的 AST 所耗用掉的記憶體僅僅是 GCC 的 20%左右。測試證明 Clang 編譯 Objective-C 代碼時速度為 GCC 的 3 倍，還能針對使用者發生的編譯錯誤準確地給出建議。

五、 編譯器 vs 直譯器

在比較之前，來了解一下直譯器，會一行一行的動態將程式碼直譯為機器碼，並執行。。直譯器像是一位「中間人」，每次執行程式時都要先轉成另一種語言再作執行，因此直譯語言多半以動態語言為主，具有靈活的型別處理，動態生成與程式彈性，但速度會比編譯式語言要慢一些。它不會一次把整個程式轉譯出來，而是每轉譯一行程式敘述就立刻執行，然後再轉譯下一行，再執行，如此不停地進行下去。

差異

直譯器的好處是它消除了編譯整個程式的負擔，程式可以拆分成多個部分來模組化，但這會讓執行時的效率打了折扣。相對

地，編譯器已一次將所有原程式碼翻譯成另一種語言，如機器碼，執行時便無需再依賴編譯器或額外的程式，故而其運行速度比較快。

編譯語言(C、C++、bjective-C、Visual Basic**..)**

用編譯器將程式碼轉換成目標平台專用的執行檔

- 優點

  執行速度快，可與硬體結合。

  可寫較大的程式。

- 缺點

  執行檔可攜性不高。

直譯語言(Basic， JavaScript、Python、Ruby**..)**

用直譯器對原始程式碼一邊讀程式碼一邊執行。

- 優點

  程式可攜性高。

- 缺點

  執行速度慢。

  原始碼必須公開。