

語言處理技術

Language Processing

編譯器 / 計算語言學 / 正規表達式 / 機器翻譯 / 交談系統

使用 C 語言實作

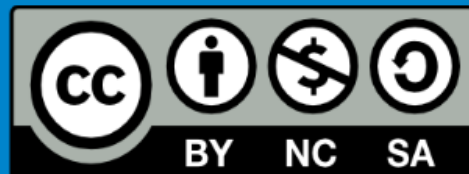
自然語言	{	處理
程式語言		轉換
標記語言		理解

字元	{	語法
詞彙		{
語句		
文章		

開放公益出版：結合創作共用與慈善公益—讀書做善事、寫書做公益

參考價：NT 100 元，如果您喜歡本書籍，請將書款捐贈公益團體

網址：<http://ccckmit.github.com/BookLanguageProcessing/>



授權聲明

本書主要內容來自開放授權的書籍與文件，因此是一種開放授權的衍生著作。書中的許多文字與圖片來自維基百科，採用創作共用的「[姓名標示、非商業性、相同方式分享](#)」的授權，以下是維基百科的授權網址。

若您想要修改本書產生衍生著作時，至少應該遵守下列授權條件：

1. 標示原作者姓名
2. 不可以作為商業用途。
3. 採用「姓名標示-非商業性-相同方式分享」的方式公開衍生著作。
4. 請勿移除公益捐款訊息，讓愛心得以持續發揮！

陳鍾誠 2012/9/14 於 金門大學 資訊工程系

關於 - 開放公益出版品

開放公益出版是一個結合「開放原始碼與公益捐款活動」的雜誌，試圖透過開放原始碼與創作共用的理念，將出版行為與公益募款結合在一起，其基本精神是「讀書做善事、寫書做公益」。

對於讀者而言：當您讀到一本好的公益出版品時，若您的經濟能力許可，請將原本用來購買這本雜誌的錢改捐給慈善機構，幫助我們的社會。

公益出版品上通常標有參考價格，如果您喜歡該出版品，請將書款捐贈公益團體。例如可捐贈給「羅慧夫顱顏基金會 彰化銀行(009) 帳號：5234-01-41778-800」。

讀公益書籍或雜誌不需要有壓力，您不需要每讀一本就急著去捐款，您可以讀了十本再捐，或者使用固定的月捐款方式，當成是雜誌訂閱費，或者是季捐款、一年捐一次等都 OK！甚至是單純當個讀者我們也都很歡迎！

對於作者而言：您可以透過撰寫公益書籍或雜誌，幫助社會上需要幫助的人，透過您的出版品，可以讓很多公益團體得以有足夠的經費做善事。

非常歡迎部落客、創作共用與開放原始碼社群的朋友們，將您的作品加上公益捐款的建議，讓更多的公益團體得到幫助，讓愛心散播出去！

序

年輕的時候，我總夢想著有一天能做出一台可以聽懂我的話，看懂我文章的電腦，於是花了二十年的時間，去追尋這個電腦夢。

我在學校、甚至在中研院擔任研究助理的時候，都在做著這個看來不切實際的夢。

直到現在，這個夢果然還沒有實現，還是那麼的不切實際。而我，則從二十歲的翩翩少年變成了四十三歲佈滿白髮的蒼老中年人。

或許，這個夢永遠都不會被實現了！

但是，我們所投注的青春與歲月，總該留下些甚麼吧！¹

陳鍾誠 2012/10/19 於 金門大學

1 本電子書與全部範例程式碼可於 github 網站下載，網址為：<http://ccckmit.github.com/BookLanguageProcessing/>

內容目錄

第 1 章 簡介.....	11
1.1. 語言的種類.....	11
1.1.1. 自然語言.....	13
1.1.2. 標記語言.....	15
1.1.3. 人造語言.....	23
1.2. 語言的層次.....	26
1.3. 語言的處理.....	27
1.4. 語言的意義.....	30
第 2 章 語言生成.....	33
2.1. 生成語法.....	33
2.1.1. 產生自然語言.....	34
2.1.2. 產生程式語言.....	39
2.2. 語法的層次.....	40
2.3. 實作：生成語法.....	45

2.3.1.基礎函式庫.....	45
2.3.2.生成 Type 3 正規語言.....	46
2.3.2.1.生成整數.....	46
2.3.2.2.生成浮點數.....	48
2.3.3.生成 Type 2 上下文無關語言.....	50
2.3.3.1.生成程式語言 (運算式).....	50
2.3.3.2.生成自然語言 (英文版).....	52
2.3.3.3.生成自然語言 / 中文版.....	54
第 3 章 字元編碼.....	57
3.1.英文 – ASCII.....	57
3.2.中文 – 繁體 Big5.....	59
3.3.多國語言 – Unicode.....	65
3.4.實作：簡繁體互轉.....	75
第 4 章 詞彙與詞典.....	80
4.1.詞彙的處理.....	80
4.2.英文詞彙掃描.....	81
4.3.詞典 – CC-CEDict.....	83

4.4. 中文的詞彙掃描 – 斷詞.....	89
第 5 章 語法剖析.....	90
5.1. 剖析器.....	90
5.2. 剖析程式語言.....	92
5.3. 剖析自然語言.....	94
5.4. 剖析標記語言.....	98
第 6 章 語意辨識.....	100
6.1. 何謂語意？.....	100
6.2. 程式的語意.....	100
6.2.1. 詞彙.....	101
6.2.2. 語句.....	102
6.2.3. 文章層次.....	103
6.3. 自然語言的語意.....	104
6.3.1. 語句的意義.....	104
6.3.2. 格狀語法 (Case Grammar).....	105
6.3.3. 欄位填充機制.....	107
6.3.4. 文章與劇本 (Script).....	108

6.3.5.程式實作.....	111
6.3.5.1.格變語法 case.c.....	111
6.3.5.2.欄位填充機制.....	112
6.3.5.3.劇本比對.....	118
第 7 章 語法比對.....	123
7.1.字串比對.....	123
7.2.正規表達式.....	127
第 8 章 自然語言的處理.....	132
8.1.自然語言.....	132
8.2.規則比對法.....	133
8.2.1.Eliza 交談系統 – 理解程度很淺.....	133
8.2.2.Baseball 棒球問答系統 – 理解程度較深.....	138
8.3.機率統計法.....	143
8.3.1.學習未知詞.....	143
8.4.研究資源.....	147
8.5.習題.....	147
第 9 章 機器翻譯.....	149

9.1.簡介.....	149
9.2.規則式翻譯.....	153
9.2.1.解歧義問題 (Word Sense Disambiguation)	157
9.2.2.語法剖析的問題 (Parsing).....	157
9.2.3.樹狀結構轉換的問題 (Structure Transformation).....	158
9.2.4.目標語句合成的問題 (Sentence Generation).....	158
9.2.5.採用規則的翻譯程式 (C 語言實作).....	158
9.3.統計式翻譯.....	168
9.3.1.雙語文章的建構問題 (Corpus Construction).....	169
9.3.2.雙語語句的對齊問題 (Sentence Alignment).....	169
9.3.3.雙語詞彙的對齊問題 (Word Alignment).....	169
9.3.4.優化問題 (Optimization).....	170
9.4.結語.....	175

第1章 簡介

1.1. 語言的種類

談到語言，一般人會想到「英文、中文、法語、日語」等等語言，這些語言是在人類歷史發展的過程當中逐漸被建構出來的，沒有任何人可以說他「設計出了中文」、「發明了英文」或「創造了法語」。

這種由歷史過程衍生出來的語言稱為「自然語言」，在資訊工程的領域，這是「自然語言處理」這個學科所必須面對與研究的課題。但是在一般領域，「自然語言」則是「語言學」所討論的議題。

自然語言並非刻意「設計出來的」，但是、有些語言卻是由某個人從無到有所創造設計出來的，像是「C 語言、Python、JavaScript、Ruby、Perl」等程式語言，我們都可以查到其發明人是誰？這類的語言稱為人造語言。這種語言通常有非常明確固定的語法，我們可以透過電腦程式去「解釋」這些語言的語法，然後做出對應的動作。

除了「自然語言」與「人造語言」之外，還有一種語言，是將兩者混合所形成的語言，這種語言通常稱為標記語言，像是「HTML、XML、維基語言、Markdown、ReStructuredText」等，這類的語言乃是在自然語言上進行一些標記，以便讓程式可以透過標記進行對應的動作，像是加上粗體、加上超連結、

顯示成表格、或者單純只是標記某個區塊的特性，像是 XML 的標記就是如此。

「計算語言學」就是研究如何透過電腦這個計算工具，用程式處理「自然語言、人造語言與標記語言」的一門學問。

傳統上、在資訊科學領域中，「計算語言學」幾乎是「自然語言處理」的同義詞，因為這是一個非常困難的研究領域，而且有很強的實用價值。像是「自然語言理解、人機對話系統、語音辨識、機器翻譯」等問題，都經常令資訊領域的研究者著迷，而投入一生的精力去研發能解決這些問題的程式。

但是在本書中，我們關注的對象不再只是自然語言，而是擴充到人造語言與標記語言上。因為在實務上，目前程式通常只能處理並理解「人造語言與標記語言」，很難對「自然語言」做到很好的理解。所以我們將「計算語言學」的領域擴充到這兩類語言上，讓讀者也能學到這方面的知識與技術。

因此、本書的範圍涵蓋面比「自然語言處理」這個學科更廣，因為納入了「程式語言、編譯器、解譯器設計、正規語言、正規表達式、全文檢索、維基語言的格式與轉換方法」等議題。

這種做法無疑會讓本書對「自然語言處理」部分的内容減少，而且研究深度降低。但是相對的也會讓本書的實用性大增，讓程式設計者能用這些語言處理知識去設計出更好的軟體系統，而這也正是本書希望能達到的目標。

1.1.1. 自然語言

常見的自然語言通常與國家種族有關，像是「英文、中文、日文」等等，以下是筆者從維基百科取出有關「Norm Chomsky」這位「生成語法」奠基者的描述，您應該可以大致感覺到到這三種語言的相異點，但是在這些表面的相異點背後，也隱含了某些相似點與共同點。這些共同點包含樹狀的語法結構，以及語意上深層的同義結構等等。

英文	中文	日文
Avram Noam Chomsky (/ˈnoʊm ˈtʃɒmski/; born December 7, 1928) is an American linguist, philosopher,[6][7] cognitive scientist, logician,[8][9] historian, political critic, and activist. He is an Institute Professor and Professor (Emeritus) in the Department of Linguistics & Philosophy at MIT, where he has worked for over 50 years.[10] In addition to his work in linguistics, he has written on war,	艾弗拉姆·諾姆·杭士基博士（Avram Noam Chomsky，1928年12月7日－），或譯作「荷姆斯基」、「喬姆斯基」，是麻省理工學院語言學的榮譽退休教授。杭士基的生成語法被認為是20世紀理論語言學研究上的重要貢獻。他對伯爾赫斯·弗雷德里克·斯金納所著《口語行為》的評論，也有助於發動心理學的認知革命，挑戰1950年代研究人類行為和語言方式中佔主導地位的行為主義。他所採用以	エイヴラム・ノーム・チョムスキー（英語：Avram Noam Chomsky、1928年12月7日－）は、アメリカ合衆国の言語学者、哲学者、思想家。マサチューセッツ工科大学教授。言語学者・教育学者キャロル・チョムスキーは彼の妻である。 チョムスキーは1928年にフィラデルフィアのユダヤ

<p>politics, and mass media, and is the author of over 100 books.[11] According to the Arts and Humanities Citation Index in 1992, Chomsky was cited as a source more often than any other living scholar from 1980 to 1992, and was the eighth most cited source overall.[12][13][14][15] He has been described as a prominent cultural figure, and he was voted the "world's top public intellectual" in a 2005 poll.[16] [17]</p>	<p>自然為本來研究語言的方法也大大地影響了語言和心智的哲學研究。他的另一大成就是建立了杭士基層級：根據文法生成力不同而對形式語言做的分類。杭士基還因他對政治的熱忱而著名，尤其是他對美國和其它國家政府的批評。從 1960 年評論越南戰爭以來，他的媒體和政治評論便越來越著名。一般認為他是活躍在美國政壇左派的主要知識分子。杭士基把自己歸為自由意志社會主義者，並且是無政府工團主義的同情者。據藝術和人文引文索引說，在 1980 年到 1992 年，杭士基是被文獻引用數最多的健在學者，並是有史以來被引用數第八多的學者。</p>	<p>系家庭に生まれた。 1949 年、アイビーリーグの一つであるペンシルベニア大学を卒業、学士号取得する。1951 年にペンシルベニア大学大学院修士課程修了、そして 1955 年、ペンシルベニア大学大学院博士課程を修了し、言語学の博士号を取得した。</p>
--	---	---

1.1.2. 標記語言

如果我們把上述維基百科的文章，切換到編輯模式以檢視原文，那麼您將會看到如下的「維基語法」，這種與法通常被稱為 MediaWiki 的語法，因為這是 MediaWiki 基金會所採用的維基語法。

英文	中文	日文
<pre>{{Redirect Chomsky}} {{pp-semi-blp small=yes}} {{pp-move-indef}} {{Infobox philosopher<!-- Philosopher category --> region = [[Western philosophy]] era = [[20th-century philosophy 20th]] / [[21st-century philosophy]] color = #B0C4DE name = Noam Chomsky other_names = Avram Noam Chomsky image = Chomsky.jpg image_size = 230px caption = Visiting [[Vancouver]],</pre>	<pre>{{nofootnotes time=2010-11-27T22:03:59+00:00}} {{noteTA T=zh-hans:诺姆·乔姆斯基; zh-hant:諾姆·杭士基; 1=zh-hans:乔姆斯基; zh-hant:杭士基; }} [[File:Noam chomsky cropped.jpg 200px right thumb 诺姆·乔姆斯基]]</pre>	<pre>{{Infobox_哲学者 <!-- 分野 --> 地域 = 西洋哲学 時代 = 現代哲学 color = #B0C4DE <!-- 画像 --> image_name = Chomsky.jpg image_caption = <!-- 人物情報 --> 名前 = エイヴラム・ノーム・チョムスキー 生年月日 = {{生年月日と年齢 1928 12 7}} 没年月日 =</pre>

<p>... "Avram Noam Chomsky" ({{IPAc-en 'n ou m _ 'tʃ ɒ m s k i}}}; born December 7, 1928) is an American [[Linguistics linguist]], [[Philosophy philosopher]],<ref name="szabo">[http://chomsky.i nfo/bios/2004----.htm "Noam Chomsky"], by Zoltán Gendler Szabó, in "Dictionary of Modern American Philosophers, 1860– 1960", ed. Ernest Lepore (2004). "Chomsky's intellectual life had been divided between his work in linguistics and his political activism, philosophy coming as a distant third. Nonetheless, his influence among analytic philosophers has been enormous because of three factors. First, Chomsky contributed</p>	<p>"艾弗拉姆·诺姆·乔姆斯基"博士 （Avram Noam Chomsky，{{bd 1928 年 12 月 7 日}}），或譯作“荷 姆斯基”-{{zh-hans:，台湾常译作“ 杭士基”；zh-hant:、「喬姆斯 基」；}-，是[[麻省理工学院]][[语言 学]]的荣誉退休教授。乔姆斯基的 [[生成语法]]被认为是[[20 世纪]][[理 论语言学]]研究上的重要贡献。他 對[[伯尔赫斯·弗雷德里克·斯金纳]] 所著《[[口語行为]]》的評論，也有 助於发动[[心理学]]的[[认知革命]]， 挑战[[1950 年代]]研究[[人類行為]] 和[[语言]]方式中占主导地位的[[行 为主义]]。他所採用以自然為本來 研究语言的方法也大大地影響了语 言和心智的[[哲学]]研究。他的另一</p>	<p>... }}} "エイヴラム・ノーム・チ ョムスキー"（[[英語]]： Avram Noam Chomsky、 [[1928 年]][[12 月 7 日]] - ）は、[[アメリカ合衆国]] の[[言語学者]]、[[哲学者]]、 [[思想家]]。 [[マサチューセッツ工科大 学]][[教授]]。 言語学者・教育学者[[キャ ロル・チョムスキー]]は彼 の妻である。 == 人物 == チョムスキーは 1928 年に [[フィラデルフィア]]の[[ユ ダヤ]]系家庭に生まれた。 1949 年、[[アイビーリー グ]]の一つである[[ペンシ</p>
---	---	---

substantially to a major methodological shift in the human sciences, turning away from the prevailing empiricism of the middle of the twentieth century: behaviorism in psychology, structuralism in linguistics and positivism in philosophy. Second, his groundbreaking books on syntax (Chomsky (1957, 1965))

大成就是建立了[[乔姆斯基层级]]：根据文法[[生成力]]不同而对[[形式语言]]做的分类。乔姆斯基还因他对[[政治]]的热忱而著名，尤其是对[[美国]]和其它[[国家]][[政府]]的批评。從 1960 年評論[[越南戰爭]]以來，他的媒體和政治評論便越來越著名。一般认为他是活跃在美国政坛[[左派]]的主要[[知识分子]]。乔姆斯基把自己归为[[自由意志社會主義]]者，并且是[[无政府工团主义]]的同情者。据[[艺术和人文引文索引]]说，在[[1980 年]]到[[1992 年]]，乔姆斯基是被文献引用数最多的健在[[学者]]，并是有史以来被引用数第八多的學者。

ルベニア大学]]を卒業、[[学士]]号取得する。[[1951 年]]に[[ペンシルベニア大学]][[大学院]][[修士]]課程修了、そして[[1955 年]]、[[ペンシルベニア大学]][[大学院]]博士課程を修了し、[[言語学]]の[[博士号]]を取得した。

維基百科

自由的百科全书

首页

分類索引

特色内容

新闻动态

最近更改

随机条目

▼ 帮助

帮助

维基社群

方针与指引

互助客栈

询问处

字词转换

IRC即时聊天

联系我们

关于维基百科

资助维基百科

► 工具

▼ 其他语言

Aragonés

العربية

مصرى

Asturianu

Azərbaycanca

Žemaitėška

Беларуская

諾姆·杭士基

[编辑]

维基百科，自由的百科全书

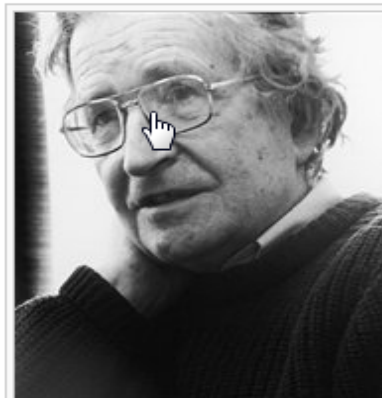


本條目由於缺少**內文腳註**，部分資訊的來源仍然不明確。*（2010年11月27日）*

請通過加入合適的內文腳註來改善這篇條目。

汉语 ▼

艾弗拉姆·諾姆·杭士基博士（Avram Noam Chomsky，1928年12月7日－），或譯作「**荷姆斯基**」、「**喬姆斯基**」，是**麻省理工學院語言學**的榮譽退休教授。杭士基的**生成語法**被認為是**20世紀理論語言學**研究上的重要貢獻。他對**伯爾赫斯·弗雷德里克斯金納**所著《**口語行為**》的評論，也有助於發動**心理學的認知革命**，挑戰1950年代研究**人類行為**和**語言**方式中佔主導地位的**行為主義**。他所採用以自然為本來研究語言的方法也大大地影響了語言和心智的**哲學**研究。他的另一大成就是建立了**杭士基層級**：根據文法**生成力**不同而對**形式語言**做的分類。杭士基還因他對**政治**的熱忱而著名，尤其是他對**美國**和其它**國家政府**的批評。從1960年評論**越南戰爭**以來，他的媒體和政治評論便越來越著名。一般認為他是活躍在美國政壇**左派**的主要**知識分子**。杭士基把自己歸為**自由意志社會主義者**，並且是**無政府工團主義**的同情者。據**藝術和人文引文索引**說，在1980年到1992年，杭士基是被文獻引用數最多的健在**學者**，並是有史以來被引用數第八多的學者。



諾姆·杭士基



維基語法的種類其實有很多，除了 MediaWiki 之外、Markdown、ReStructuredText 等都是常見的維基語法。筆者經常使用的 wikidot 維基網誌也有一套自己的語法，通常也就直接稱為 wikidot 語法，wikidot 語法語 MediaWiki 有些類似，卻也有許多不同點。

維基語法或許不能算是標記語言當中的典型，真正最為人所知的標記語言是 HTML 與 XML，讓我們來看看幾個關於 Chomsky 的範例。

HTML	XML
<pre><html> <body> <title>Noam Chromsky</title> <table> <tr><th>人物簡介</th><th>照片</th></tr> <tr><td> 艾弗拉姆·諾姆·杭士基博士（Avram Noam Chomsky，1928 年 12 月 7 日－）， 或譯作「荷姆斯基」、「喬姆斯基」，是麻省理工 學院語言學的榮譽退休教授。 杭士基的生成語法被認為是 20 世紀理論語言學研究 上的重要貢獻。</pre>	<pre><xml> <people name="Noam Chromsky"> <introduction> 艾弗拉姆·諾姆·杭士基博士（Avram Noam Chomsky，1928 年 12 月 7 日－）， 或譯作「荷姆斯基」、「喬姆斯基」，是麻省理 工學院語言學的榮譽退休教授。 杭士基的生成語法被認為是 20 世紀理論語言學 研究上的重要貢獻。 ... 據藝術和人文引文索引說，在 1980 年到 1992 年， 杭士基是被文獻引用數最多的</pre>

...
據藝術和人文引文索引說，在 1980 年到 1992 年，
杭士基是被文獻引用數最多的
健在學者，並是有史以來被引用數第八多的學者。

...
</td><td></td></tr>
</body></html>

健在學者，並是有史以來被引用數第八多的學者。

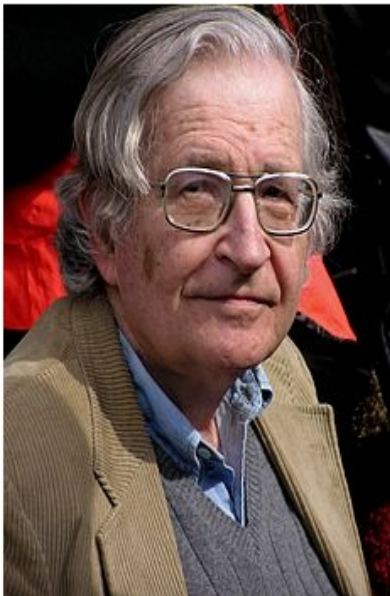
...
</introduction>
<image url="Chomsky.jpg"/>
</xml>

file:///D:/ccc/clboo ☆ ▼ ↺ Google ↻

人物簡介

艾弗拉姆·諾姆·杭士基博士 (Avram Noam Chomsky, 1928年12月7日－)，或譯作「荷姆斯基」、「喬姆斯基」，是麻省理工學院語言學的榮譽退休教授。杭士基的生成語法被認為是20世紀理論語言學研究上的重要貢獻。... 據藝術和人文引文索引說，在1980年到1992年，杭士基是被文獻引用數最多的健在學者，並是有史以來被引用數第八多的學者。...

照片



此 XML 未包含顯示用的樣式資訊。下面以文件的節點樹狀方式顯示。

```
- <xml>
- <people name="Noam Chomsky">
- <introduction>
  艾弗拉姆·諾姆·杭士基博士 (Avram Noam Chomsky, 1928年12月7日－)，或譯作「荷姆斯基」、「喬姆斯基」，是麻省理工學院語言學的榮譽退休教授。杭士基的生成語法被認為是20世紀理論語言學研究上的重要貢獻。... 據藝術和人文引文索引說，在1980年到1992年，杭士基是被文獻引用數最多的健在學者，並是有史以來被引用數第八多的學者。...
</introduction>
  <image url="Chomsky.jpg"/>
</people>
</xml>
```

在以上範例中，您可以很清楚的看到 HTML 與 XML 兩者之間的差別，雖然都是標記語言，HTML 顯

然有點像在進行排版工作，而 XML 則單純只是用標記將內容框住的樹狀結構文件。

像 XML 這樣的標記語言雖然已經具有樹狀的結構化訊息了，但是由於內文經常還是自然語言，因此真正要用程式去理解語意內容時，仍然具有相當大的障礙。

雖然好的標記能夠提供程式不少的幫助²，但是距離「完全理解」通常還有一段不小的目標。在這裡，所謂的「自然語言完全理解」其實是個很令人爭議的說法，因為何謂完全理解是一個難以定義的詞彙。

但是我想也沒有其他更傳神的說法了，筆者所說的「程式對自然語言完全理解」，就是像「解譯器對程式語言完全理解那樣」，可以精確的進程式語言所描述的動作。舉例而言，假如有一句話說：

「顯示 Yahoo 首頁！」輸入到那個「具備完全理解能力」的程式當中時，程式就會將您的瀏覽器打開並顯示 Yahoo 的首頁，這樣就是對該語句的完全理解了³。

2 筆者的博士論文：「基於欄位填充機制的 XML 文件檢索方法」正是利用語意標記為程式提供一個支點，在這個支點上進行檢索方法的改進，其中也有一種根據 XML 自動建構出知識框架的方法，其實說穿了只是針對每個標記欄位進行詞彙統計學習的動作罷了，該論文網址位於 – <http://ccckmit.wikidot.com/local--files/re:paper/PhdThesis.pdf>。

3 請注意，在此我們對「自然語言完全理解」之程式的定義，與圖靈 (Turing) 對「具有智慧的機器」之定義不同，圖靈說如果我們可以建造出一台機器可以與人用文字的方式交談，然後讓人無法辨別對方是一台機器的話，那就代表該程式具有人的智慧。但是這種定義方式後來造成很多問題，像是 Eliza 這個 1960 年代就被發展出來的程式就成功的騙過了很多人，讓人以為那是真人在與他聊天，這種立基於「欺騙行為」上面的定義是有缺陷的，因此我們改採力基於「忠實執行」的方式來定義「自然語言完全理解」之程式。

同樣的，如果使用者下了一個命令說「幫我找一個女朋友！」，此時那個「具備完全理解能力」的程式應該做甚麼樣的動作呢？

當然、假如那個程式做不到，那就只能回答：「抱歉！我做不到」。甚至再補上兩句：「我自己都沒有女朋友了，不過，我可以幫你上 **facebook** 請求加入一些女的好友，您覺得如何呢？」。

1.1.3. 人造語言

人造語言的種類很多，但大部分都是程式類的語言，像是高階語言 (像是 C、Ruby、Python)、組合語言 (像是 x86、ARM、CPU0 的組合語言)、還有高階語言在翻譯成組合語言之前通常會經過某種中介語言等等，以下是一些人造語言的範例。

C 語言	中介語言	組合語言
<pre>int sum(int n) { int s = 0; for (i=1; i<=n; i++) { s = s + i; } return s; }</pre>	<pre>= 0 sum = 1 i FOR0: CMP i 10 J > _FOR0 + sum i T0 = T0 sum</pre>	<pre>LDI R1, 0 ST R1, sum LDI R2, 0 ST R2, i LDI R3, 1 LDI R4, 10 CMP R2, R4</pre>

}	<pre> + i 1 i J FOR0 _FOR0: RET sum </pre>	<pre> JGT _FOR0 ADD R1, R1, R2 ADD R2, R2, R3 JMP _FOR0 ST R1, sum RET i: RESW 1 sum: RESW 1 </pre>
---	--	---

程式語言不只可以用來寫程式，有些程式語言具有很特殊的用途，像是 Verilog/VHDL 就可以用來設計數位電路，而 SPICE 則可以用來設計類比電路。以下是這些語言的幾個範例。

Verilog	VHDL	SPICE
<pre> module fulladder (input a, b, c_in, output sum, c_out); wire s1, c1, c2; </pre>	<pre> ENTITY fulladder IS PORT (Cin, x, y : IN STD_LOGIC ; s, Cout: OUT STD_LOGIC) ; </pre>	<pre> RC LOW-PASS FILTER * VS 1 0 AC 1 SIN(0VOFF 1VPEAK 2KHZ) </pre>

xor g1(s1, a, b); xor g2(sum, s1, c_in); and g3(c1, a,b); and g4(c2, s1, c_in) ; or g5(c_out, c2, c1) ; endmodule	END fulladder ; ARCHITECTURE beh OF fulladder IS BEGIN s <= x XOR y XOR Cin ; Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y) ; END beh ;	R1 1 2 1K C1 2 0 0.032UF * ANALYSIS .AC DEC 5 10 10MEG .TRAN 5US 500US * VIEW RESULTS .PRINT AC VM(2) VP(2) .PRINT TRAN V(1) V(2) .PROBE .END
--	---	--

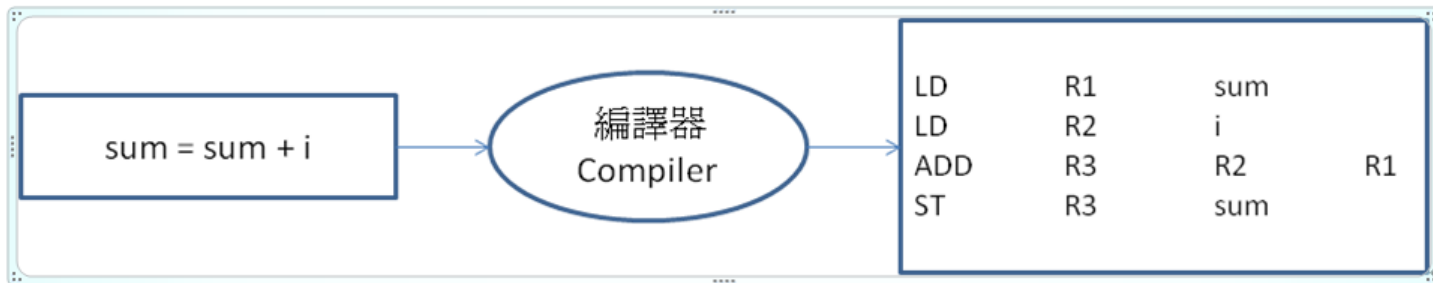
1.2. 語言的層次

不論是以上哪些語言，幾乎都具有「詞彙、語句、文章」等三個層次，以下是這幾個層次的範例。

	英文	中文	C	組合語言
詞彙	books, read, John	讀, 書, 約翰	int , for, {, (, i, sum	LD, ST, R1, sum, i, RESW
語句	John read books.	約翰讀書	int s = 0;	LDI R1, 0
文章	John read books. He is a fens of Harry Porter. He read all books about Harry Porter.	約翰常常讀書， 他是哈利波特 迷，所有關於 的哈利波特的 書他幾乎都讀 過。	int sum(int n) { int s = 0; for (i=1; i<=n; i++) { s = s + i; } return s; }	LDI R1, 0 ST R1, sum LDI R2, 0 ST R2, i ... RET i: RESW 1 sum:RESW 1

1.3. 語言的處理

電腦對於語言文字的處理方法，在程式語言上面已經相當得成熟，編譯器可以很容易的將高階語言程式轉換成組合語言或機器碼，以下是編譯器將高階程式轉為組合語言的一個範例。



另外、解譯器也很容易的可以執行程式，甚至也可以設計出一種稱為 **Just in time (JIT) Compiler** 的「編解譯混合技術」，在解譯過程中以編譯的方式去執行程式，這樣可以讓執行的速度變得更快，像是 Java 的 JVM 與 Google 的 JavaScript V8 JIT 引擎，都是採用這種混合模式製作的。

但是對於「自然語言的處理」而言，目前的電腦技術就顯得相當力不從心了，這點各位可以從「Google 翻譯」的品質就可以發現。Google 翻譯可以說是做得非常好的翻譯程式，但是當您輸入得是一整段文章時，通常 Google 翻譯出來的品質仍然是慘不忍睹的。以下是我們將維基百科中有關 Chomsky 的第一段文章輸入到 Google 翻譯中所呈現的結果，您可以看到一開始好像翻的不錯，後來就有一點落差了。

翻譯

原文是 英文 ▾



翻譯成 中文(繁體) ▾

翻譯

中文 英文 日文 偵測語言

Avram Noam Chomsky (/ˈnoʊm ˈtʃɒmski/; born December 7, 1928) ×
is an American linguist, philosopher,[6][7] cognitive scientist,
logician,[8][9] historian, political critic, and activist. He is an Institute
Professor and Professor (Emeritus) in the Department of
Linguistics & Philosophy at MIT, where he has worked for over 50
years.[10] In addition to his work in linguistics, he has written on
war, politics, and mass media, and is the author of over 100 books.
[11] According to the Arts and Humanities Citation Index in 1992,
Chomsky was cited as a source more often than any other living
scholar from 1980 to 1992, and was the eighth most cited source
overall.[12][13][14][15] He has been described as a prominent
cultural figure, and he was voted the "world's top public intellectual"
in a 2005 poll.[16][17]

英文 中文(繁體) 中文(簡體)

，阿夫拉姆諾姆 喬姆斯基 (ˈnoʊmtʃɒmski的/，出生於1928年12月7
日) 是美國語言學家，哲學家，[6] [7]認知科學家，邏輯學
家，[8][9]歷史學家，政治評論家和社會活動家。他是學院教授，名
譽教授在麻省理工學院語言學與哲學系，他在那裡已經工作了超過
50年。[10]此外，他的工作在語言學上，他寫了戰爭，政治和質
量媒體，是作者的100本書。[11]根據1992年藝術與人文引文索引，
喬姆斯基被引用的來源往往比任何其他生物的學者1980年至
1992，並且是第八次被引用最多的來源整體[12] [13] [14] [15]他被描
述為一個傑出的文化人物，他在2005年的調查中被評為“世界上頂
尖的公共知識分子”。[16][17]

新! 按一下上方的文字，即可編輯及查看其他翻譯。 [關閉](#)

而以下這段則是落差更大了。

番羽譯

原文是 英文 ▾



翻譯成 中文(繁體) ▾

翻譯

中文 英文 日文 偵測語言

After the publication of his first books on linguistics, Chomsky became a prominent critic of the Vietnam War, and since then has continued to publish books of political criticism. He has become well known for his critiques of U.S. foreign policy,[23] state capitalism[24][25] and the mainstream news media. His media criticism has included Manufacturing Consent: The Political Economy of the Mass Media (1988), co-written with Edward S. Herman, an analysis articulating the propaganda model theory for examining the media. He describes his views as "fairly traditional anarchist ones, with origins in the Enlightenment and classical liberalism", [26] and sometimes identifies with anarcho-syndicalism and libertarian socialism.



英文 中文(繁體) 中文(簡體)

喬姆斯基語言學他的第一本書出版後，成為了著名評論家的越南戰爭，此後不斷出版的書籍的政治批評。他已經成為眾所周知的美國的外交政策，[23]國家資本主義[24] [25]和主流新聞媒體對他的批評。他的媒體的批評包括：製造同意：政治經濟學的大眾傳播媒介（1988），寫與愛德華 赫爾曼，分析闡明理論研究媒體的宣傳模式。他描述了自己的看法，"無政府主義者的相當傳統，起源於啓蒙運動和古典自由主義"，[26]，有時識別與無政府工團主義和自由主義的社會主義。



新! 按一下上方的文字，即可編輯及查看其他翻譯。[關閉](#)

對於標記語言處理的品質，則與標記語言的類型有關，也與處理的目標很有關係。舉例而言，程式可以很容易的將 `wiki` 或 `XML` 標記語言轉換為 `HTML` 呈現出來，這只要用「字串比對」與「正規表達式」就可以輕易的做到。

但如果程式處理的對象是標記語言中的文字，那處理的品質通常會與標記所提供的資訊有關。舉例而言，如果要將英文的 `wiki` 文件翻譯成中文，那困難度就不亞於「自然語言的翻譯」了。但是如果要對某種具有良好 `XML` 語意標記的文件進行「文件摘要」的動作，那程式通常可以做得還不錯。

1.4. 語言的意義

如前所述，我們在「自然語言完全理解」的定義中，採用了類似「解譯器對程式語言完全理解那樣」的定義，也就是一但程式能精確執行使用者的命令或請求，那就算「程式理解了使用者的自然語言」。

但是即使有這樣的定義，那麼語言的意義應該表達成甚麼結構呢？需要表達成像語法理論中的語法樹結構嗎？還是需要加上不同的結構？或者是採用與樹狀結構完全不同的架構呢？

先讓我們看看程式語言的例子，對電腦而言，解譯器、編譯器都是可以「執行程式語言」的，因此符合了「程式理解語言」的定義。

那麼，編譯器與解譯器是如何執行程式語言的呢？

在編譯器中，程式語言被翻譯成一串機器指令，然後再交給 CPU 去執行。

在解譯器當中，程式語言經過語法剖析成樹狀結構後，就可以直接解譯執行，解譯時除了使用語法樹之外，還必須建立一些「變數」的結構，然後在執行某個語法或指令時，透過修改變數模擬指令所要求的動作。

於是，我們可以將那些低階的指令串，視為「程式語言」的語意。

但是，「自然語言」的語意又是甚麼呢？我們可以將「自然語言」翻譯成甚麼「指令動作」呢？

先讓我們看看電腦的能力所在，電腦能執行的不過就是 CPU 指令集所描述的指令，因此若我們將「自然語言」翻譯成「CPU 指令串」，那就完成了理解的動作⁴。

但是這樣的翻譯過程顯然太過遙遠，我們很難以想像如何將自然語言翻譯成「CPU 指令串」，但是我們比較能想像的是「自然語言」如何被翻譯成「一連串的动作」。

舉例而言，如果使用者說：「幫我把 facebook 打開，然後念出今天的留言給我聽」，這時候電腦將這句話翻譯成以下動作。

1. 打開瀏覽器

4 不過到底程式理解是正確還錯誤，就得看這些指令串執行的結果，對使用者而言是否有意義了。

2. 連結到 <http://www.facebook.com/>
3. 輸入使用者的帳號密碼登入。
4. 登入後進入訊息畫面。
5. 將訊息畫面上的文字轉成語音輸出。

如果採用這種將「自然語言」翻譯成「一連串動作」的角度看，那麼「程式理解語言」就是「程式將語言轉換成一連串的動作，而這些動作符合使用者的請求」。

第2章 語言生成

2.1. 生成語法

不管是程式語言或自然語言，都可以用語法進行描述。但不同的是，程式語言的語法是設計者自行定義的，所以凡是不符合語法的程式都會直接被視為錯誤的程式。而在處理自然語言的時候，凡是文章中的句子，不管合不合語法，你的程式都要能夠處理。這是自然語言處理之所以困難的原因之一。

規則比對法是早期的自然語言研究重心，這個方法企圖利用語言知識庫處理機器翻譯的問題。其主要的核心理念來源於 Chomsky 的生成語法路線，Chomsky 考察語言的結構後，提出一套使用語法規則描述語句結構的方法，稱為『生成語法』。許多學習外語的人都曾經學習過文法的概念，Chomsky 所謂的語法與這些文法其實是類似的。

先讓我們來看一個很簡單的生成語法範例，如下所示。

(a) BNF 語法↵	(b) 生成的語言↵
$S = A B$ ↵ $A = 'a' \mid 'b'$ ↵ $B = 'c' \mid 'd'$ ↵	$L = \{ac, ad, bc, bd\}$ ↵

以下就讓我們來看看「語法」在「程式語言」與「自然語言」中的角色各自為何吧！

2.1.1. 產生自然語言

我們也可以用類似的語法描述自然語言，以下是一個範例。

(a) BNF 語法↵	(b) 生成的語言↵
$S = N ' ' V$ ↵ $A = 'John' \mid 'Mary'$ ↵ $B = 'eats' \mid 'talks'$ ↵	$L = \{John\ eats, John\ talks, \downarrow$ $Mary\ eats, Mary\ talks\}$ ↵

當然、語法規則可能會越來越多，舉例而言，如果我們使用下列語法描述自然語言，就可以生成比較

複雜一點的語句。

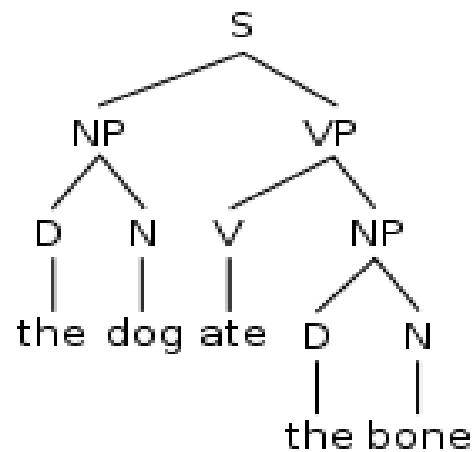
(a) BNF 語法

$S = NP VP$

$NP = D N$

$VP = V NP$

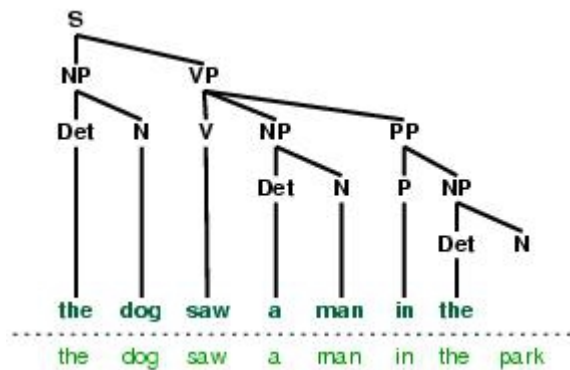
(b) 語言的剖析範例



接著，讓我們來看看一組較為完整的英語語法規則。

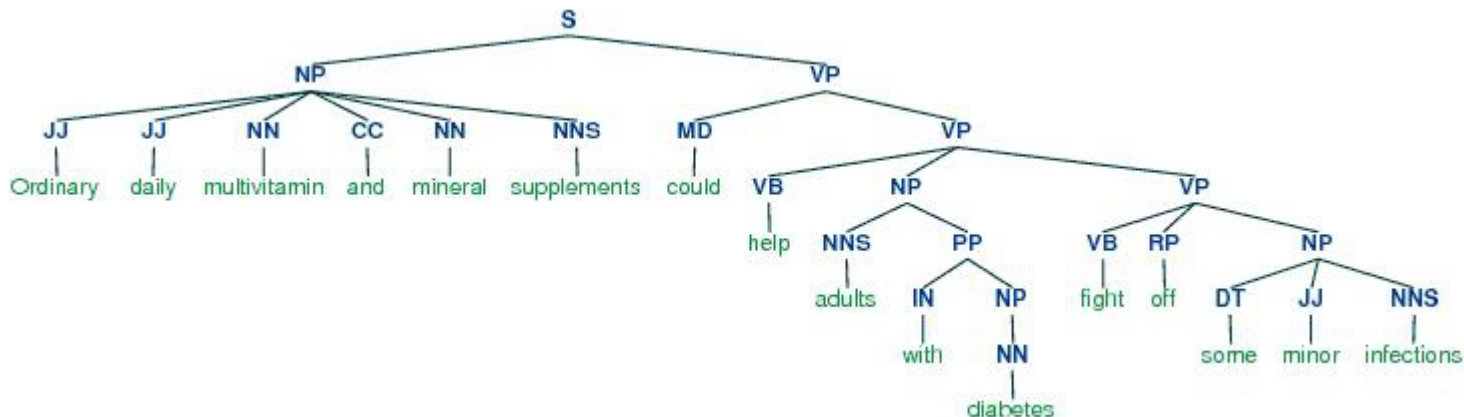
規則	說明
$S \Rightarrow NP VP$	句子 = 名詞子句 接 動詞子句
$NP \Rightarrow Det Adj^* N PP^*$	名詞子句 = 定詞 接 名詞
$VP \Rightarrow V (NP PP^*)$	動詞子句 = 動詞 接 名詞子句 接副詞子句
$PP \Rightarrow P NP$	副詞子句 = 副詞 接 名詞子句

根據這樣的規則，我們就可以將 『The dog saw a man in the park.』 這句話，剖析成下列的語法樹。



圖、簡單的語法樹的範例

甚至，對於更複雜的句子，像是『Ordinary daily multivitamin and mineral supplements could help adults with diabetes fight off some minor infections.』，也可能轉換成下列的剖析樹。



圖、複雜的語法樹的範例

一般來說，目前的自然語言剖析技術並沒有辦法將所有句子都轉換成完整的樹狀結構，通常只有 60%-70% 左右的成功率而已。因此，有時會採用部分剖析樹直接使用，而非一定要完全剖析成功。

2.1.2. 產生程式語言

當然、我們也可以用 BNF 語法描述程式語言，以下是一個用 BNF 描述運算式的範例。

(a) BNF 語法↵	(b) 語言的實際範例↵
$E = N \mid E [+ - * /] E$ $N = [0-9]^+$	3 $3 + 5$ $3 + 5 * 8 - 4 / 6$

先讓我們看看一個程式語言語法的範例，以下規則描述了一個數學運算式的語法，其中的 E 代表運算式 (Expression)，而 T 代表運算式中的一個基本項 (Term)。

$E = T \mid T [+ - * /] E$ 這條規則代表說 E 可以生成 T，或者先生成 T 之後緊跟著 + - * / 其中的一個字元，然後再遞回性的生成 E。

同樣的、 $T = [0-9]$ 代表 T 可以生成 0 到 9 當中的一個數字，或者生成 「(」 符號後再生成 E 接著生成 「)」 符號。

$$E = T \mid T [+ - * /] E$$
$$T = [0-9] \mid (E)$$

以下是符合該語法與不符合該語法的一些語句範例。

符合語法的範例

4
2+5
2-1/(2)-(9*0)
8+(5-7*((6)*((2))* (7)*8+4+(6))/(6-1)-(9))

不符合語法的範例

x+y
3+) 8
(3-*8)/4

2.2. 語法的層次

那麼、生程與法的描述能力到底如何呢？Chomsky 提出了一個很完整的答案，稱為 Chomsky Hierarchy (喬姆斯基語言階層)，如下所示。

Grammar	Languages	Automaton	Production rules (constraints)
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$ (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$

上圖中、 α, β, γ 可以是任何語句 (含終端與非終端符號)，而 A, B 代表只能是非終端符號，a 代表只能是終端符號。其中 Type-0 的語言描述力是最強的，基本上任何的語法規則您都可以撰寫而毫無限制，這種語言所能描述的語言稱為「遞歸可枚舉語言」 (Recursive Enumerable, RE)，這種語言的能力在計算理論上可以對應到圖靈機，也就相當於是一台記憶體空間沒有上限的電腦所展現的能力。

Type1 語言的語法有點限制，因為每個規則的左邊至少要有一個非終端項目 A，但其前後可以連接任意規則，這種語法所能描述的語言稱為「對上下文敏感的語言」 (Context-Sensitive)，因為 α 可以決定之後到底是否要接 $A\beta$ ，所以前後文之間是有關係的，因此才叫做「對上下文敏感的語言」。這種語言在計算理論上可以對應到「線性有界的非決定性圖靈機」，也就是一台「記憶體有限的電腦」。

Type 2 語言的語法限制更大，因為規則左邊只能有一個非終端項目 (以 A 代表)，規則右邊則沒有限制，

這種語言被稱為「上下文無關的語言」(Context Free)，在計算理論上可以對應到「非決定性的堆疊機」(non-deterministic pushdown automaton)。

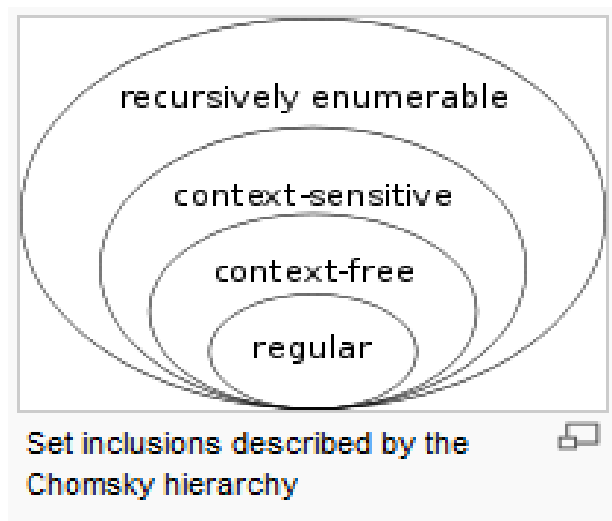
幾乎所有的程式語言都屬於 Type 2 的「上下文無關的語言」，這樣的程式語言處理起來也會比較簡單⁵。但是雖然程式語言在語法上通常屬於 Type 2 的「上下文無關」的語言，但其變數的型態與宣告等語意結構，卻是與上下文有關的，只是這個結構的處理通常是在「語意」層次去處理，而非在語法層次去處理的。

我們所常用的自然語言語法，像是「英文文法」、「中文文法」通常也是屬於 Type 2 的語法，但是自然語言中的「指稱」，像是「你、我、他」，以及許多「語意」上的問題，也都是與上下文有關的，無法單純以 Type 2 的語法處理這些特性。

Type 3 的語法限制是最多的，其規則的左右兩邊都最多只能有一個非終端項目 (以 A, B 表示)，而且右端的終端項目 (以 a 表示) 只能放在非終端項目 B 的前面。這種語言稱為「正規式」(Regular)，可以用程式設計中常用的「正規表達式」(Regular Expression) 表示，對應到計算理論中的有限狀態機 (Finite State Automaton)。

5 事實上程式語言並非完全與上下文無關的，只是 BNF 語法是與上下文無關的語言而已，但在「變數宣告、型態相容性、物件、結構、函數宣告」等項目上，程式的區塊是與上下文有關的。但是這個相關性並不需要在語法的層次解決，而是留待語意層次再去解決就可以了，所以我們通常還是用 Type 2 的語法去描述程式語言。

下圖是這些語言之間的包含關係，也就是 $Type3 \subset Type2 \subset Type1 \subset Type0$ 。



Type2 所不能處理的語言當中，有個最著名的範例是 $a^n b^n c^n$ ，由於這當中 abc 三個字母必須按照順序各出現 n 次，而 Type2 的與上下文無關語法，無法記憶到底已經產生了幾個 a^n ，所以也就無法產生出這樣的語言了。

以下的 Type 1 語法可以生成 $a^n b^n c^n$ 語言，只是結構相對複雜而已，以下是一個可以生成 aaabbbccc 的案例：

可生成 $a^n b^n c^n$ 語言的語法

1. $S \rightarrow aSBC$
2. $S \rightarrow aBC$
3. $CB \rightarrow HB$
4. $HB \rightarrow HC$
5. $HC \rightarrow BC$
6. $aB \rightarrow ab$
7. $bB \rightarrow bb$
8. $bC \rightarrow bc$
9. $cC \rightarrow cc$

生成 aaabbbccc 的範例

$$\begin{aligned}
 &S \\
 &\Rightarrow_1 aSBC \\
 &\Rightarrow_1 aa\mathbf{S}BCBC \\
 &\Rightarrow_2 aaa\mathbf{B}C\mathbf{B}C\mathbf{B}C \\
 &\Rightarrow_3 aaa\mathbf{B}\mathbf{H}\mathbf{B}C\mathbf{B}C \\
 &\Rightarrow_4 aaa\mathbf{B}\mathbf{H}\mathbf{C}\mathbf{C}\mathbf{B}C \\
 &\Rightarrow_5 aaa\mathbf{B}\mathbf{B}\mathbf{C}\mathbf{C}\mathbf{B}C \\
 &\Rightarrow_3 aaa\mathbf{B}\mathbf{B}\mathbf{C}\mathbf{H}\mathbf{B}C \\
 &\Rightarrow_4 aaa\mathbf{B}\mathbf{B}\mathbf{C}\mathbf{H}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_5 aaa\mathbf{B}\mathbf{B}\mathbf{C}\mathbf{B}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_3 aaa\mathbf{B}\mathbf{B}\mathbf{H}\mathbf{B}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_4 aaa\mathbf{B}\mathbf{B}\mathbf{H}\mathbf{C}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_5 aaa\mathbf{B}\mathbf{B}\mathbf{B}\mathbf{C}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_6 aa\mathbf{a}\mathbf{b}\mathbf{B}\mathbf{B}\mathbf{C}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_7 aa\mathbf{a}\mathbf{b}\mathbf{b}\mathbf{B}\mathbf{C}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_7 aa\mathbf{a}\mathbf{b}\mathbf{b}\mathbf{b}\mathbf{C}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_8 aa\mathbf{a}\mathbf{b}\mathbf{b}\mathbf{b}\mathbf{c}\mathbf{C}\mathbf{C} \\
 &\Rightarrow_9 aa\mathbf{a}\mathbf{b}\mathbf{b}\mathbf{b}\mathbf{c}\mathbf{c}\mathbf{C} \\
 &\Rightarrow_9 aa\mathbf{a}\mathbf{b}\mathbf{b}\mathbf{b}\mathbf{c}\mathbf{c}\mathbf{c}
 \end{aligned}$$

2.3. 實作：生成語法

2.3.1. 基礎函式庫

```
#include <stdio.h>

// int randInt(int n):隨機傳回一個小於 n 的整數 (0,1,2..., n-1)
// 用法:randInt(5) 會傳回 0, 1, 2, 3, 4 其中之一
int randInt(int n) { // 隨機傳回一個小於 n 的整數 (0,1,2..., n-1)
    return rand() % n;
}

// int randChar(char *set):隨機傳回 set 中的一個字元
// 用法:randChar("0123456789") 會傳回一個隨機的數字
char randChar(char *set) { // 隨機傳回 set 中的一個字元
    int len = strlen(set);
    int i = rand()%len;
    return set[i];
}

// int randStr(char *set, int size):隨機傳回 set 中的一個字元
// 用法: char *n[] = {"dog", "cat"}; randStr(n, 2)
//      會傳回 "dog", "cat" 其中之一。
char *randStr(char *set[], int size) { // 隨機傳回 set 中的一個字串
    int i = rand()%size;
    return set[i];
}
```

```
}
```

2.3.2. 生成 Type 3 正規語言

Type 2 的語法限制一般寫成： $A \rightarrow a$ or $A \rightarrow aB$ ，這種語法稱為 right regular grammar。

但事實上，Type 2 也可以寫成 left regular grammar 的語法限制如： $A \rightarrow a$ or $A \rightarrow Ba$ 。

2.3.2.1. 生成整數

以下是一個採用 right regular grammar 的程式範例，用來生成整數 (Integer)。

執行結果：

```
G:\code\cl>gcc gint.c -o gint
```

```
G:\code\cl>gint
```

```
1498
```

```
2511571221871
```

97
532313
727
39
60804
60
063
94661

程式：gint.c

```
#include "rlib.c"
// === BNF Grammar =====
// N = [0-9] N | [0-9]
int main(int argc, char * argv[]) {
    int i;
    for (i=0; i<10; i++) {
        N();
        printf("\n");
    }
}

int N() {
```

```
    printf("%c", randChar("0123456789"));  
    if (randInt(10) < 8)  
        N();  
}
```

2.3.2.2. 生成浮點數

執行結果：

```
D:\ccc102\CL>gcc gfloat.c -o gfloat
```

```
D:\ccc102\CL>gfloat
```

1.0

4

251.12

64326.71

97.413418

47

39.52

62859018.4661

4.3

8.939

程式：gfloat.c

```
#include "rlib.c"
// === BNF Grammar =====
// F = [0-9] F | [0-9] H | [0-9]
// H = . T
// T = [0-9] T | [0-9]
int main(int argc, char * argv[]) {
    int i;
    for (i=0; i<10; i++) {
        F();
        printf("\n");
    }
}

int F() {
    printf("%c", randChar("0123456789"));
    if (randInt(10) < 7)
        F();
    else if (randInt(10) < 8)
        H();
}
```

```
int H() { printf("."); T(); }

int T() {
    printf("%c", randChar("0123456789"));
    if (randInt(10) < 7)
        T();
}
```

2.3.3. 生成 Type 2 上下文無關語言

Type 2 的語法限制形式為： $A \rightarrow \gamma$ ，這種與法很重要，以下我們將用幾個範例說明這種與法的用途。

2.3.3.1. 生成程式語言 (運算式)

執行結果

```
D:\ccc102\CL>gcc gexp.c -o gexp
```

```
D:\ccc102\CL>gexp
(9+(5-(5*1)+2*8/1))*7
```

4
2-1/(2)-(9*0)
(2)+5
(6-(4)*6)+((7)*9)-9*7*1-0-7
0-7
1
0-0/((3))-(8)
6*(2/((5)-1+6-(1))*(((8))/0))-9+0)-8
8+(5-7*((6)*((2))* (7)*8+4+(6))/(6-1)-(9))

原始程式：gexp.c

```
#include "rlib.c"

// === BNF Grammar =====
// E = T [+-*/] E | T
// T = [0-9] | (E)

int main(int argc, char * argv[]) {
    int i;
    for (i=0; i<10; i++) {
        E();
        printf("\n");
    }
}
```

```

    }
}

int E() {
    if (randInt(10) < 5) {
        T(); printf("%c", randChar("+-*/" )); E();
    } else {
        T();
    }
}

int T() {
    if (randInt(10) < 7) {
        printf("%c", randChar("0123456789"));
    } else {
        printf("("); E(); printf(")");
    }
}

```

2.3.3.2. 生成自然語言 (英文版)

執行結果

```
D:\ccc102\CL>gcc genglish.c -o genglish
```

```
D:\ccc102\CL>genglish
```

the cat chase a cat

a dog chase a cat

the cat eat the bone

the bone chase the dog

the dog chase the bone

a dog chase a cat

the dog eat a dog

a cat eat the dog

the dog chase the dog

a dog eat the bone

程式：genglish.c

```
#include "rlib.c"
```

```
// === BNF Grammar =====
```

```
// S = NP VP
```

```
// NP = D N
```

```
// VP = V NP
```

```
int main(int argc, char * argv[]) {
```

```
    int i;
```

```

    for (i=0; i<10; i++) {
        S();
        printf("\n");
    }
}

int S() { NP(); VP(); }

int NP() { D(); N(); }

int VP() { V(); NP(); }

char *d[] = {"a", "the"};
int D() { printf("%s ", randStr(d, 2)); }

char *n[] = {"dog", "bone", "cat"};
int N() { printf("%s ", randStr(n, 3)); }

char *v[] = {"chase", "eat"};
int V() { printf("%s ", randStr(v, 2)); }

```

2.3.3.3. 生成自然語言 / 中文版

執行結果：

```
D:\ccc102\CL>gcc gchinese.c -o gchinese
```

```
D:\ccc102\CL>gchinese
```

```
這隻 貓 追 那隻 貓  
那隻 狗 追 那隻 貓  
這隻 貓 吃 一隻 骨頭  
這隻 骨頭 追 一隻 狗  
一隻 狗 追 一隻 骨頭  
那隻 狗 追 這隻 貓  
這隻 狗 吃 一隻 狗  
那隻 貓 吃 那隻 狗  
這隻 狗 追 一隻 狗  
這隻 狗 吃 這隻 骨頭
```

程式：gchinese.c

```
#include "rlib.c"  
  
// === BNF Grammar =====  
// S = NP VP  
// NP = D N
```

```
// VP = V NP

int main(int argc, char * argv[]) {
    int i;
    for (i=0; i<10; i++) {
        S();
        printf("\n");
    }
}

int S() { NP(); VP(); }

int NP() { D(); N(); }

int VP() { V(); NP(); }

char *d[] = {"一隻", "那隻", "這隻"};
int D() { printf("%s ", randStr(d, 3)); }

char *n[] = {"狗", "骨頭", "貓"};
int N() { printf("%s ", randStr(n, 3)); }

char *v[] = {"追", "吃"};
int V() { printf("%s ", randStr(v, 2)); }
```


第3章 字元編碼

3.1. 英文 - ASCII

英文的電腦編碼系統主要以 ASCII (American Standard Code for Information Interchange, 美國資訊交換標準代碼) 字元編碼方法為核心，這是在電腦剛開始發展時就定義好的一套 7 位元字母編碼系統，這個系統可以將英文的大小寫與鍵盤上的主要符號都編列到 7 個位元的碼中。

下圖是 1968 年制定的 ASCII 編碼表。

b_7 $\xrightarrow{\hspace{1.5cm}}$ b_6 $\xrightarrow{\hspace{1.5cm}}$ b_5 $\xrightarrow{\hspace{1.5cm}}$ Bits						0	0	0	0	1	1	1	1
						0	0	1	0	1	0	1	1
	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	<div> <div>Column →</div> <div>Row ↓</div> </div>	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	\$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(8	H	X	h	x
	1	0	0	1	9	HT	EM)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	
	1	1	0	1	13	CR	GS	-	=	M]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

現今所使用的 ASCII 編碼，通常採用 8 個位元為一組的 byte 編碼方式，這種方式下 0~127 按照原先的 ASCII 編碼，而 128~255 的部分則沒有使用到。後來很多國家就利用這個未使用到的部份去延伸出自己的編碼方式，向是台灣的 Big5 大五碼，與中國的 GB2312 等都是如此。

3.2. 中文 - 繁體 Big5

Big5 是台灣電腦發展早期由廠商與資策會所共同制定出的編碼方式，製定於 1983 年。Big5 是一種雙位元組的編碼系統，是利用 ASCII 的 127 (0x7F) 之後的未用區域當作第一個 byte (首碼)，然後與第二個 byte (次碼) 搭配所形成的編碼系統。

理論上，為了避免與 ASCII 衝碼，首碼可以從 128 (0x80) 開始，但事實上 Big5 的首碼位於 0xA1~0xFE 之間，而次碼則位於 0x40~0x7E 之間。

常用字首碼主要範圍位於 0xA1~0xC6，而次常用字的首碼範圍位於 0x94~0xF9。較精確的編碼範圍如下表所示。

0x8140-0xA0FE	保留給使用者自定義字元（造字區）
0xA140-0xA3BF	標點符號、希臘字母及特殊符號，包括在0xA259-0xA261，安放了九個計量用漢字：尅尅尅尅尅尅尅尅。
0xA3C0-0xA3FE	保留。此區沒有開放作造字區用。
0xA440-0xC67E	常用漢字，先按筆劃再按部首排序。
0xC6A1-0xC8FE	保留給使用者自定義字元（造字區）
0xC940-0xF9D5	次常用漢字，亦是先按筆劃再按部首排序。
0xF9D6-0xFEFE	保留給使用者自定義字元（造字區）

如果您是在以 big5 為主的作業系統之下（例如 windows 繁體中文版），要列出 big5 的字元相當簡單，我們可以採用下列程式印出主要的 big5 編碼字元。⁶

檔案：big5dump.c

```
#include <stdio.h>
```

6 請注意這個程式由於過度簡單，並沒有將未編碼區完全濾除，因此會把一些未編碼字元也印出，所以使用者會看到一部分的亂碼。

```

#define BYTE unsigned char

int main() {
    BYTE h, t;
    for (h=0xA1; h<=0xF9; h++) { // Big5 首碼範圍
        printf("\n");
        for (t=0x00; t<0xFF; t++) { // Big5 次碼範圍
            printf("%c%c %2x%2x", h, t, h, t);
            if (t % 8 == 0) printf("\n");
        }
    }
}

```

執行結果片段：

```
G:\code\cl>gcc big5dump.c -o big5dump
```

```
G:\code\cl>big5dump
```

```
...
```

```
? f129? f12a? f12b? f12c? f12d? f12e? f12f? f130
```

```
? f131? f132? f133? f134? f135? f136? f137? f138
```

```
? f139? f13a? f13b? f13c? f13d? f13e? f13f 蹠 f140
```

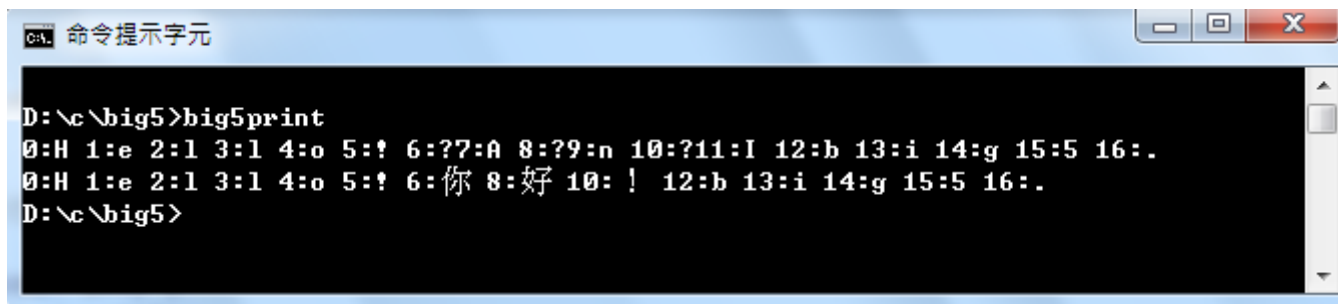
```
蹠 f141 蹠 f142 蹠 f143 蹠 f144 蹠 f145 輓 f146 輓 f147 輓 f148
```

𨮒 f149 𨮓 f14a 𨮔 f14b 𨮕 f14c 𨮖 f14d 𨮗 f14e 𨮘 f14f 𨮙 f150
 𨮚 f151 𨮛 f152 𨮜 f153 𨮝 f154 𨮞 f155 𨮟 f156 𨮠 f157 𨮡 f158
 𨮢 f159 𨮣 f15a 𨮤 f15b 𨮥 f15c 𨮦 f15d 𨮧 f15e 𨮨 f15f 𨮩 f160
 𨮪 f161 𨮫 f162 𨮬 f163 𨮭 f164 𨮮 f165 𨮯 f166 𨮰 f167 𨮱 f168
 𨮲 f169 𨮳 f16a 𨮴 f16b 𨮵 f16c 𨮶 f16d 𨮷 f16e 𨮸 f16f 𨮹 f170
 𨺀 f171 𨺁 f172 𨺂 f173 𨺃 f174 𨺄 f175 𨺅 f176 𨺆 f177 𨺇 f178
 𨺈 f179 𨺉 f17a 𨺊 f17b 𨺋 f17c 𨺌 f17d 𨺍 f17e? f17f? f180
 ? f181? f182? f183? f184? f185? f186? f187? f188
 ? f189? f18a? f18b? f18c? f18d? f18e? f18f? f190
 ? f191? f192? f193? f194? f195? f196? f197? f198
 ? f199? f19a? f19b? f19c? f19d? f19e? f19f? f1a0
 𨺏 f1a1 𨺐 f1a2 𨺑 f1a3 𨺒 f1a4 𨺓 f1a5 𨺔 f1a6 𨺕 f1a7 𨺖 f1a8
 𨺗 f1a9 𨺘 f1aa 𨺙 f1ab 𨺚 f1ac 𨺛 f1ad 𨺜 f1ae 𨺝 f1af 𨺞 f1b0
 𨺟 f1b1 𨺠 f1b2 𨺡 f1b3 𨺢 f1b4 𨺣 f1b5 𨺤 f1b6 𨺥 f1b7 𨺦 f1b8
 𨺧 f1b9 𨺨 f1ba 𨺩 f1bb 𨺪 f1bc 𨺫 f1bd 𨺬 f1be 𨺭 f1bf 𨺮 f1c0
 𨺯 f1c1 𨺰 f1c2 𨺱 f1c3 𨺲 f1c4 𨺳 f1c5 𨺴 f1c6 𨺵 f1c7 𨺶 f1c8
 𨺷 f1c9 𨺸 f1ca 𨺹 f1cb 𨺺 f1cc 𨺻 f1cd 𨺼 f1ce 𨺽 f1cf 𨺾 f1d0
 𨺿 f1d1 𨻀 f1d2 𨻁 f1d3 𨻂 f1d4 𨻃 f1d5 𨻄 f1d6 𨻅 f1d7 𨻆 f1d8
 ...

由於 big 5 的設計是利用 ASCII 的後段進行編碼，以便從 1 byte 的 ASCII 中延伸出 2byte 的 big5 編

碼。所以對於一個中英夾雜的字串而言，ASCII 0~127 範圍的碼必須解讀為 1byte 字元，而 128 之後的 big5 範圍區則必須採用 2byte 的方式解讀，更精確的說，當首碼位於 0xA1 到 0xF9 之間時，必須採用 2byte 的方式解讀。

為了說明 big5 的處理技巧，我們撰寫了一個程式將 big5 字串 s="Hello!你好！big5." 列印出來。其執行結果如下圖所示。



```
D:\c\big5>big5print
0:H 1:e 2:l 3:l 4:o 5:! 6:??7:A 8:?9:n 10:?11:l 12:b 13:i 14:g 15:5 16:..
0:H 1:e 2:l 3:l 4:o 5:! 6:你 8:好 10:！ 12:b 13:i 14:g 15:5 16:..
D:\c\big5>
```

以上執行結果的第一個輸出行，代表的是我們將字串 s 以單位元組的方式解讀所得到的列印結果，您可以看到其中 6~12 的部分印出了一些 ? 代表系統無法解讀的亂碼，但是第二行的部份我們根據首碼決定到底要用 1byte 或 2byte 的解讀方式印出，結果就可以正確的印出「6:你 8:好 10:！」這部分的中文字。其解讀方式大致如下圖所示。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
單一 byte 印出	H	e	l	l	o	!	?	A	?	n	?	I	b	i	g	5	.
根據首碼解讀	H	e	l	l	o	!	你		好		!		b	i	g	5	.

該程式的原始碼如下：

```
#include <stdio.h>

#define BYTE unsigned char

int main() {
    char s[] = "Hello!你好!big5.";
    int i;
    for (i=0; i<strlen(s); i++) {
        printf("%d:%c ", i, s[i]);
    }
    printf("\n");
    for (i=0; i<strlen(s); ) {
        BYTE b=(BYTE) s[i];
        if (b>=0xA1 && b<=0xF9) {
            printf("%d:%c%c ", i, s[i], s[i+1]);
            i+=2;
        }
    }
}
```



```
        else {  
            printf("%d:%c ", i, s[i]);  
            i++;  
        }  
    }  
}
```

3.3. 多國語言 - Unicode

Unicode⁷ 是一種廣納全世界各種文字的編碼方式，目前最新的版本為第六版，已收入了超過十萬個字元。Unicode 涵蓋的資料除了視覺上的字形、編碼方法、標準的字元編碼外，還包含了字元特性，如大小寫字母。

Unicode 發展是由 (The Unicode Consortium) 所負責制定的標準，致力於讓 Unicode 方案取代既有的字元編碼方案。因為既有的方案往往僅有有限的空間，亦不適用於多語環境。

Unicode 備受認可，並廣泛地應用於電腦軟體的國際化與在地化過程。有很多新科技，例如 HTML、XML、Java 程式語言，以及許多作業系統都採用 Unicode 編碼。

⁷ <http://zh.wikipedia.org/wiki/Unicode>

Unicode 依隨著通用字符集 UCS (Universal Character Set⁸) 的標準而發展，UCS 規定了每個字的圖形樣式以及對應代碼。

目前實際應用較多的 Unicode 版本為 UCS-2，使用兩個 byte (16 位元) 的編碼方式。這樣理論上一共最多可以表示 65536 個字元。但是仍然無法容納所有的各國字元，因此還有包含更廣的 UCS-4 版本採用 4 個 byte 的編碼方式，理論上可容納 40 億個字元 (應該可以完全涵蓋所有國家的所有字才對，但這是個太龐大的工程)。

Unicode 的字元代碼雖然是固定的，但是兩個 UCS-2 每個字佔 2 bytes 的空間，而 UCS-4 則每個字佔據 4-bytes 的空間則是讓許多人難以接受的，因為這會耗費過多的記憶體、硬碟空間與網路傳輸頻寬，因此 UCS-2 與 UCS-4 都可以採用變動長度的編碼方式，稱為 UTF (Unicode Transformation Format)。

其中 UTF-8 是採用 1 byte 代表英文區域的編碼方式，有點類似 big5 的設計想法，都是從 ASCII 的 128-255 區域延伸出來的編碼法。

而 UTF-16 則主要對應了 UCS-2 的編碼方式，但事實上 UTF-16 還可以區分為 UTF-16 LE 與 UTF-16 BE 兩種編法⁹，兩者的 byte 順序不太相同。

8 <http://zh.wikipedia.org/wiki/UCS>

9 LE 代表 Little Endian，BE 代表 Big Endian 的位元順序。

在 C 語言當中，要採用 Unicode 編碼方式處理並輸出字串，可以採用 `wchar_t` 這種寬字元形態來表示 unicode 字元，並用 `wchar_t` 的指標或陣列來表示 unicode 字串，以下是一個在使用 unicode 進行輸出的 C 語言範例。(注意：這個程式必須儲存成 UTF-8 檔首無 BOM 的格式，才能用 gcc 正確編譯)

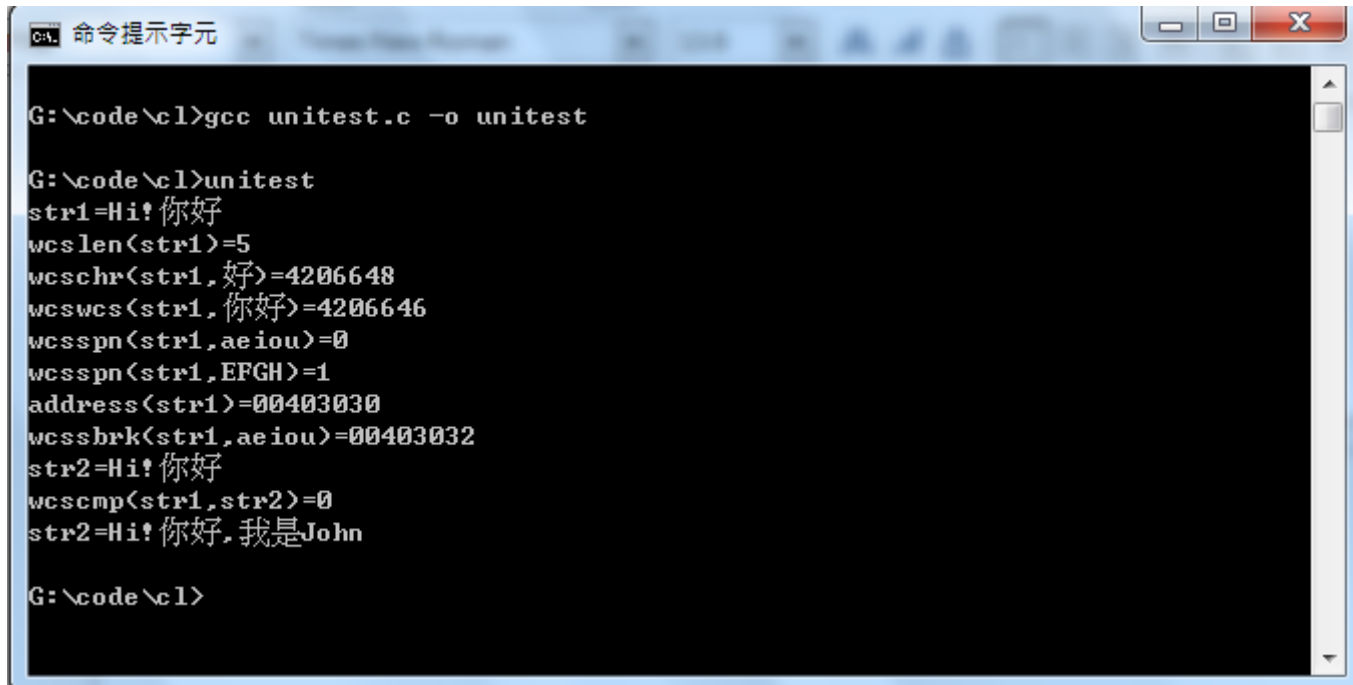
檔案：unittest.c

```
#include <stdio.h>
#include <locale.h>

int main(void)
{
    if (!setlocale(LC_CTYPE, "")) {
        fprintf(stderr, "Error:Please check LANG, LC_CTYPE, LC_ALL.\n");
        return 1;
    }
    wchar_t *str1=L"Hi!你好"; // 輸出結果 (範例)
    printf("str1=%ls\n", str1); // str1=Hi!你好
    printf("wcslen(str1)=%d\n", wcslen(str1)); // wcslen(str1)=5
    printf("wcschr(str1,%lc)=%d\n", L'好', wcschr(str1, L'好')); // wcschr(str1,好)=4206648
    printf("wcswcs(str1,%ls)=%d\n", L"你好", wcsstr(str1, L"你好")); // wcswcs(str1,你好)=4206646
    printf("wcsspncpy(str1,aeiou)=%d\n", wcsspncpy(str1, L"aeiou")); // wcsspncpy(str1,aeiou)=0
    printf("wcsspncpy(str1,EFGH)=%d\n", wcsspncpy(str1, L"EFGH")); // wcsspncpy(str1,EFGH)=1
    printf("address(str1)=%p\n", str1); // address(str1)=00403030
    printf("wcspbrk(str1,aeiou)=%p\n", wcspbrk(str1, L"aeiou")); //
```

```
wcssbrk(str1,aeiou)=00403032
wchar_t str2[20];
wcscpy(str2, str1);
printf("str2=%ls\n", str2); // str2=Hi!你好
printf("wcscmp(str1,str2)=%d\n", wcscmp(str1, str2)); // wcscmp(str1,str2)=0
wscat(str2, L",我是 John");
printf("str2=%ls\n", str2); // str2=Hi!你好,我是 John
return 0;
}
```

上述程式的執行輸出結果如下：



```
命令提示字元

G:\code\cl>gcc unittest.c -o unittest

G:\code\cl>unittest
str1=Hi! 你好
wcslen(str1)=5
wcschr(str1,好)=4206648
wcs wcs(str1,你好)=4206646
wcssp n(str1,aeiou)=0
wcssp n(str1,EFGH)=1
address(str1)=00403030
wc s s b r k(str1,aeiou)=00403032
str2=Hi! 你好
wcscmp(str1,str2)=0
str2=Hi! 你好,我是John

G:\code\cl>
```

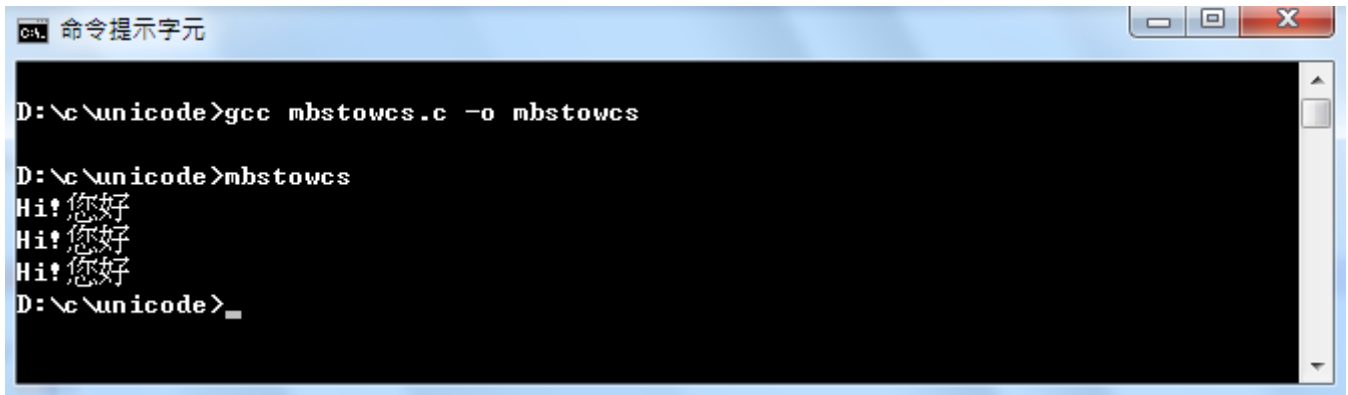
有時，我們為了陣列存取方便，避免 1byte 與 2byte 的複雜判斷，會需要將含有中文的字串先轉換為寬字串 (`wchar_t*`) 進行逐字處理，然後處理完後再將寬字串轉換回一般字串 (`char*`)，此時就可以用寬窄字串轉換的函數進行這項轉換工作，以下是一個寬窄字串轉換的程式範例。

檔案：mbstowcs.c

```
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    char *str = "Hi!您好";
    wchar_t wstr[10];
    char str2[10];
    mbstowcs(wstr, str, 10);
    printf("%s\n", str);
    printf("%ls\n", wstr);
    wcstombs(str2, wstr, 10);
    printf("%s", str2);
}
```

執行結果



```
命令提示字元

D:\c\unicode>gcc mbstowcs.c -o mbstowcs

D:\c\unicode>mbstowcs
Hi! 您好
Hi! 您好
Hi! 您好
D:\c\unicode>_
```

當您想使用函數去處理 Unicode 表示法的字串時，必須使用特別為 `wchar_t` 設計的字串函數，而非一般為 `char` 所設計的 ANSI 字串函數，幾乎每個 ANSI 字串函數都有對應的 Unicode 字串函數，命名的方式是用 `wcs` 取代 `str`，或者加上一個 `w` 字代表寬字元，以下是 ANSI 字串函數與 Unicode 字串函數的對應表。

功能	窄字元	寬字元	說明
長度	<code>strlen()</code>	<code>wcslen()</code>	字串長度
連接	<code>strcat()</code>	<code>wcscat()</code>	字串連接
比較	<code>strcmp()</code>	<code>wcscmp()</code>	字串比較

比較	strcoll()	wscoll()	字串比較 (不分大小寫)
複製	strcpy()	wcscpy()	字串複製
尋找	strchr()	wcschr()	尋找字元
尋找	strstr()	wcswcs()	尋找字串
分割	strtok()	wcstok()	字串分割
比對	strcspn()	wcscspn()	傳回字串中第一個符合字元集的位置
比對	strpbrk()	wcspbrk()	傳回字串中第一個符合字元集的指標
轉換	strxfrm()	wcsxfrm()	根據區域設定 locale() 轉換字元集

在 C 語言當中，如果要讀入一個 UTF-16 的檔案，也同樣必須使用 `wchar_t` 的函數，像是 `fgetwc()` 或 `fgetws()`，但是由於 UTF-16 檔案前面可能會有用來供辨認檔案形態的「魔數」存在，因此必須先將「魔數」吃掉之後才能開始讀取。同樣的，如果要寫入一個 UTF-16 的檔案，也同樣必須先寫入「魔數」之後，再開始寫入 `wchar_t` 形態的正文。

以下是筆者撰寫的一個 UTF-16 檔案處理函式庫 `unicode.c`。

```
#include <locale.h>
#include <stdio.h>
```



```

int uinit() {
    if (!setlocale(LC_CTYPE, "")) {
        fprintf(stderr, "setlocale() 失敗，請檢查 LANG, LC_CTYPE, LC_ALL.\n");
        return 1;
    }
}

FILE *uopen(char *fname, char *mode) {
    FILE* file=fopen(fname, mode);
    wchar_t magic = 0xFEFF; // 魔數為 0xFEFF
    if (mode[0] == 'r') {
        wchar_t head = fgetwc(file);
        if (head != magic)
            return NULL;
    } else {
        fputwc(magic, file); // 輸出魔數到檔案
    }
    return file;
}

wchar_t* ureadtext(char *fileName) {
    FILE *file = uopen(fileName, "rb");
    fseek(file, 0, SEEK_END);
    long size = ftell(file);
    rewind(file);
    wchar_t *text = (wchar_t*) malloc(size+sizeof(wchar_t));

```

```

    int len = size/sizeof(wchar_t);
    fread(text, sizeof(wchar_t), len, file);
    fclose(file);
    text[len] = L'\0';
    return text;
}

void uwritetext(char *fileName, wchar_t *text) {
    FILE *file = uopen(fileName, "wb");
    fwrite(text, sizeof(wchar_t), wcslen(text)+1, file);
}

```

利用這個函式庫，我們可以撰寫以下程式以讀取 **unicode** 編碼的檔案並印出：(注意：由於筆者並沒有處理 UTF8 格式，所已輸入檔只能是檔首有魔數的 UTF-16 檔案)

檔案：uniread.c

```

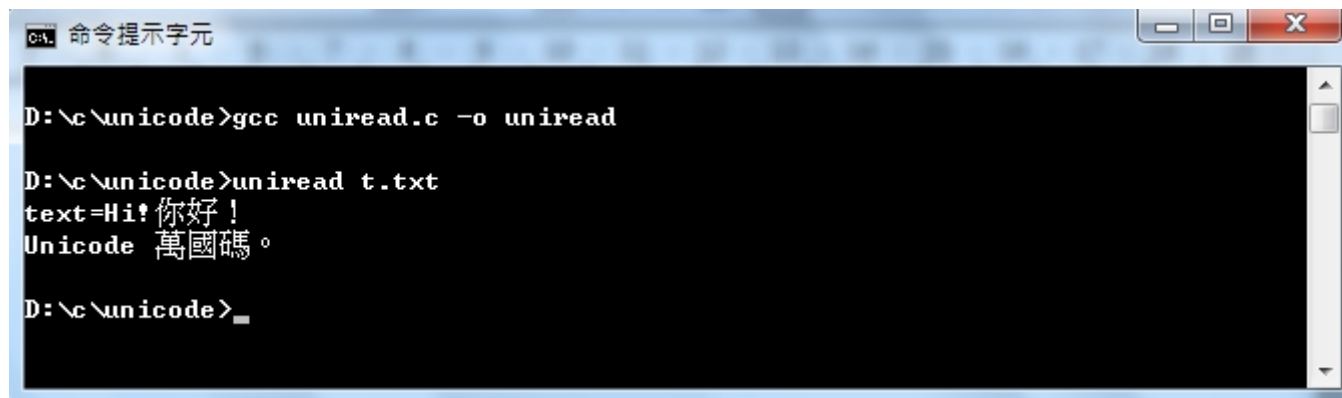
#include "unicode.c"

int main(int argc, char *argv[]) {
    uinit();
    wchar_t *text = ureadtext(argv[1]);
    wprintf(L"text=%s", text);
}

```

```
}
```

執行結果：



```
命令提示字元

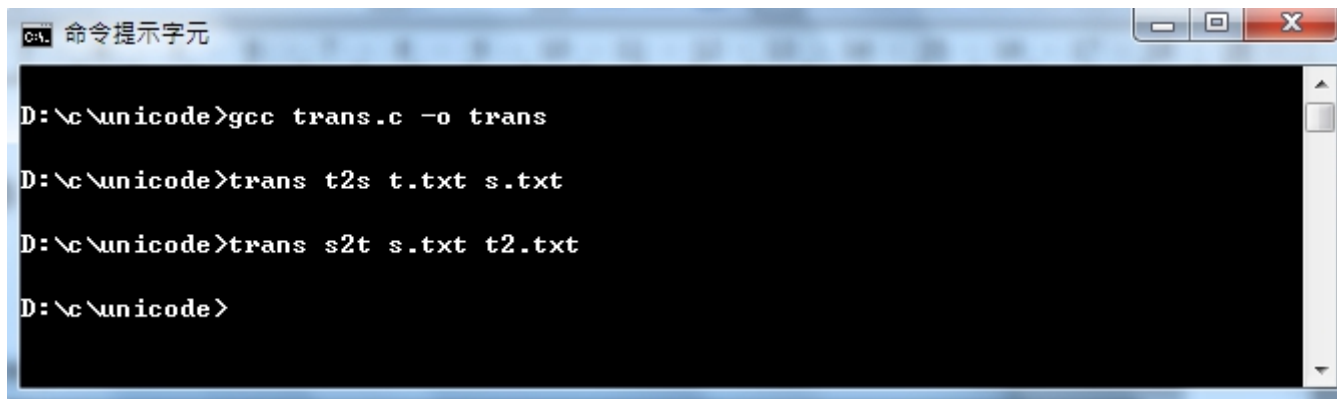
D:\c\unicode>gcc uniread.c -o uniread

D:\c\unicode>uniread t.txt
text=Hi! 你好!
Unicode 萬國碼。

D:\c\unicode>_
```

3.4. 實作：簡繁體互轉

編譯執行



```
命令提示字元

D:\c\unicode>gcc trans.c -o trans

D:\c\unicode>trans t2s t.txt s.txt

D:\c\unicode>trans s2t s.txt t2.txt

D:\c\unicode>
```

執行結果：

t.txt	s.txt	t2.txt
Hi!你好! Unicode 萬國碼。	Hi!你好! Unicode 万国码。	Hi!你好! Unicode 萬國碼。

簡繁轉換表：位於網址 <https://dl.dropbox.com/u/101584453/linguistics/c/unicode/t2s.txt>

轉換程式：trans.c

```
#include "unicode.c"

#define UMAX 65536 // UTF16 的字最多 65536 個。
wchar_t zmap[UMAX] /*繁轉簡表格*/, smap[UMAX] /*簡轉繁表格*/;

int main(int argc, char *argv[]) {
    uinit(); // 初始化
    tsmap(); // 設定簡繁轉換表
    if (argc != 4) help(); // 檢查參數
    if (strcmp(argv[1], "t2s")==0) // 如果是 t2s，繁轉簡
        translate(zmap, argv[2], argv[3]);
    else if (strcmp(argv[1], "s2t")==0) // 如果是 s2t，簡轉繁
        translate(smap, argv[2], argv[3]);
    else // 否則顯示 help()
        help();
}

int help() { // 顯示使用方法
    printf("trans <mode> <fromFile> <toFile>\n"
           "<mode>={z2s or s2z}\n");
}

// 根據 map 進行轉換。 zmap: 繁轉簡, smap: 簡轉繁。
int translate(wchar_t *map, char *inFileName, char *outFileName) {
    FILE* inFile=fopen(inFileName, "rb"); // 開啟輸入檔 (UTF-16 格式)
```

```

FILE* outFile=fopen(outFileName, "wb"); // 建立輸出檔 (UTF-16 格式)
while(1) { // 一直讀
    wchar_t c = fgetwc(inFile); // 讀一個字元
    if (feof(inFile)) // 如果已經到檔尾，跳開並結束
        break;
    if (map[c] == 0) // 如果字元的轉換不存在
        fputwc(c, outFile); // 直接將字元輸出
    else
        fputwc(map[c], outFile); // 否則將轉換後的字元輸出
}

fclose(inFile); // 關閉輸入檔
fclose(outFile); // 關閉輸出檔
}

// 載入簡繁轉換表
int tsmap() {
    memset(zmap, 0, sizeof(wchar_t)*UMAX); // 清空繁轉簡表格
    memset(smap, 0, sizeof(wchar_t)*UMAX); // 清空簡轉繁表格
    FILE* inFile=fopen("t2s.txt", "rb"); // 開啟繁簡對照表 (UTF-16 格式)
    while(!feof(inFile)) { // 一直讀到檔案結束
        wchar_t zs[10];
        if (fgetws(zs, 10, inFile)== NULL) // 讀入 (繁, 簡)
            break;
        wchar_t z = zs[0], s=zs[1]; // 設定 (z=繁, s=簡)
        if (z!=s) {
            zmap[z] = s; // zmap[繁]=簡
            smap[s] = z; // smap[簡]=繁
        }
    }
}

```

```
        }  
    }  
    fclose(inFile); // 關閉繁簡對照表  
}
```

第4章 詞彙與詞典

4.1. 詞彙的處理

不論是自然語言或程式語言，其處理大致上都可分為「字元、詞彙、語句、文章」等四個層次，本章所關注的焦點是詞彙的處理。

對於英文而言，詞彙之間通常會用空白分隔 (例如：This is a book.)，這使得英文的詞彙處理相對簡單。但是對於中文而言，詞彙與詞彙之間是沒有空白分隔的，於是將文章切分成一個一個的詞彙，通常就必須仰賴「詞典」了。

詞彙的處理是一個相對簡單的工作。當然，凡事都有例外，詞彙的處理有時也會有點困難。例如英文中也會有些詞彙裏夾雜著標點符號，例如：Mr. Jamie 當中的「Mr.」，這個字是個縮寫，英文當中有很多縮寫，這些縮寫有時會包含標點符號，這時就會比較難處理。

中文詞彙的處理由於依賴詞典，當某些詞彙在詞典當中不存在時，就會發生詞彙切分錯誤的情況，例如「人名、地名、公司名稱、專有名詞」等等，這些詞彙在「自然語言處理」當中統稱為「未知詞」，其實應該稱為「未收錄詞」才對，通常這些詞彙會被切分成數個單一字元，當然這會造成後續處理的

一些困擾。但是我們幾乎不可能造出一個蒐錄全世界所有中文詞彙的詞典，因此這個困擾一直存在，不過有些程式可以透過統計的方式學出詞彙，這可以解決部分問題，但是也會造成其他問題，例如學出來的詞彙不是一個詞，或者是某些詞彙沒有被統計法判斷為一個詞等等。

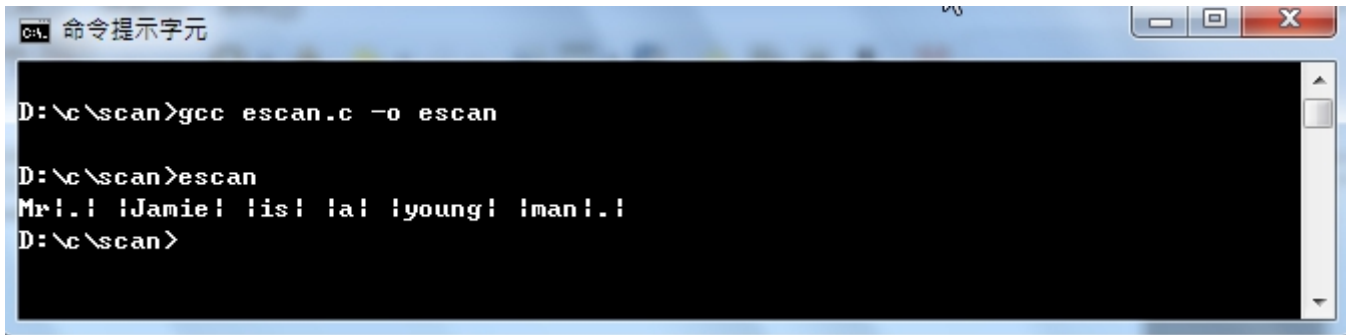
在本章中，我們將討論如何對「英文、中文、程式」進行詞彙切分的工作，並且研究詞典在自然語言處理程式當中的使用方法。

4.2. 英文詞彙掃描

英文詞彙的掃描，如果不去考慮一些特殊的例外，那麼會是很簡單的，只要遵照下列規則就能完成。

1. 碰到英文字母時持續掃描，直到不是英文字母為止。
2. 非英文字母的情況都傳回單一字元。

以下是我們用 C 寫的一個英文詞彙掃描程式 `escan.c`，其執行結果如下：



```
D:\c\scan>gcc escan.c -o escan

D:\c\scan>escan
Mr!.! !Jamie! !is! !a! !young! !man!.!
D:\c\scan>
```

程式：escan.c¹⁰

```
#include <stdio.h>

char text[] = "Mr. Jamie is a young man.";

char *next(char *text, int *idx, char *token) { // 取得下一個 token
    int i = *idx;
    if (text[i] == '\0') return NULL; // 已到尾端
    while (isalpha(text[i])) { // 忽略空白
```

10 程式位於：<https://dl.dropbox.com/u/101584453/Linguistics/c/scan/escan.c>

```

        i++;
    }
    if (i == *idx) i++;        // 如果不是英文字母，取單一字元。
    int len = i - (*idx);
    strncpy(token, &text[*idx], len);
    token[len] = '\0';
    *idx = i; // 前進到下一個字開頭
    return token;
}

int scan(char *text) { // 掃描器的主要功能
    int ti=0;
    char token[100];
    while (next(text, &ti, token) != NULL) { // 不斷掃描
        printf("%s|", token);
    }
}

int main(int argc, char * argv[]) { // 掃描器主程式
    scan(text);
}

```

4.3. 詞典 – CC-CEDict

電子詞典在「中文自然語言處理」當中可以說是關鍵性的基礎建設，像是「翻譯技術」就必須仰賴詞典，否則很難進行。本節將以一個創作共用 CC 授權的詞典檔 CC-CEDict 為範例，說明詞典的處理與使用方式。

CC-CEDict 詞典¹¹ 是一本採用創作共用的「姓名標示、相同方式分享」授權的詞典，包含六萬多個詞彙，詞典的內容含有「繁體中文、簡體中文、英文」等欄位，以下是詞典中的一小段範例。

```
...
牛皮 牛皮 [niu2 pi2] /(n) cowskin/leather/
牛皮癬 牛皮癬 [niu2 pi2 xian3] /psoriasis/
牛皮菜 牛皮菜 [niu2 pi2 cai4] /chard (Beta vulgaris), a foliage beet/
牛磺酸 牛磺酸 [niu2 huang2 suan1] /taurine/
牛肉 牛肉 [niu2 rou4] /beef/
牛肉拉麵 牛肉拉面 [niu2 rou4 la1 mian4] /ramen (pulled noodles) with beef/
牛肉炒麵 牛肉炒面 [niu2 rou4 chao3 mian4] /stir-fried noodles with beef/
牛肉芹菜 牛肉芹菜 [niu2 rou4 qin2 cai4] /beef and celery/
```

¹¹ CC-CEDict 的官網 – <http://www.mdbg.net/chindict/chindict.php?page=cedict>

筆者儲存版本 – <http://ccckmit.wikidot.com/data:cedict>

```
牛肉面 牛肉面 [niu2 rou4 mian4] /beef noodle soup/  
牛膝 牛膝 [niu2 xi1] /Achyranthes bidentata (root used in Chinese medicine)/  
牛至 牛至 [niu2 zhi4] /oregano/marjoram/  
牛蒡 牛蒡 [niu2 pang2] /burdock/  
...
```

這本詞典雖然已經編輯的相當好了，但是還是會有少數缺點，像是極少數繁體字仍然沿用簡體版本，例如以上範例中的「牛肉面 牛肉面 [niu2 rou4 mian4] /beef noodle soup/」這一條，其中的第一個「面」字應該改為「麵」會比較符合繁體的用法，不過大致上這本詞典已經擁有相當好的品質了。

當我們想要在程式中使用這樣的詞典時，必須將詞典讀入之後進行簡單的格式剖析，然後將欄位取出。筆者寫了一個程式 `cedictparse.c` 以便完成這項工作¹²，程式碼如下：

```
#include <stdio.h>  
#include <locale.h>
```

12 程式位於：<http://dl.dropbox.com/u/101584453/linguistics/c/cedict/cedictparse.c>

```
#define LEN 512
```

```
int main(int argc, char *argv[]) {  
    uinit(); // 初始化  
    cedict(); // 設定簡繁轉換表  
}
```

```
// 載入 CC-CEDICT
```

```
int cedict() {  
    FILE* inFile=uopen("cedict_ts.u16le", "rb");// 開啟 CC-CEDICT  
    FILE* outFile=uopen("cedict_log.u16le", "wb");  
    while(!feof(inFile)) { // 一直讀到檔案結束  
        wchar_t line[LEN], head[LEN], tail[LEN], tc[LEN], sc[LEN], pr[LEN], en[LEN];  
        if (fgetws(line, LEN, inFile)== NULL) // 讀入 (繁, 簡)  
            break;  
        if (line[0] == '#') continue;  
        fwprintf(outFile, L"line=%ls", line);  
        swscanf(line, L"%s %s [%[^]]] /^[^/]", tc, sc, pr, en);  
        fwprintf(outFile, L"tc=%ls sc=%ls pr=%ls en=%ls\n\n", tc, sc, pr, en);  
    }
```

```
}  
fclose(inFile); // 關閉輸入檔 CC-CEDICT  
fclose(outFile); // 關閉 log 輸出檔  
}
```

以上程式中我們利用 `swscanf(line, L"%s %s [%[^]]] /^[^/]", tc, sc, pr, en);` 這一行，剖析 CC-CEDICT 中的一個輸入行，將「繁體 `tc`，簡體 `sc`，發音 `pr` 與英文 `en`」等欄位取出來¹³，以下是該程式的輸出 `log` 檔的一部分結果。

```
line=牛皮 牛皮 [niu2 pi2] /(n) cowskin/leather/  
tc=牛皮 sc=牛皮 pr=niu2 pi2 en=(n) cowskin
```

```
line=牛皮癬 牛皮癬 [niu2 pi2 xian3] /psoriasis/  
tc=牛皮癬 sc=牛皮癬 pr=niu2 pi2 xian3 en=psoriasis
```

```
line=牛皮菜 牛皮菜 [niu2 pi2 cai4] /chard (Beta vulgaris), a foliage beet/  
tc=牛皮菜 sc=牛皮菜 pr=niu2 pi2 cai4 en=chard (Beta vulgaris), a foliage beet
```

13 英文的部份我們只取第一個解釋。

line=牛磺酸 牛磺酸 [niu2 huang2 suan1] /taurine/
tc=牛磺酸 sc=牛磺酸 pr=niu2 huang2 suan1 en=taurine

line=牛肉 牛肉 [niu2 rou4] /beef/
tc=牛肉 sc=牛肉 pr=niu2 rou4 en=beef

line=牛肉拉麵 牛肉拉面 [niu2 rou4 la1 mian4] /ramen (pulled noodles) with beef/
tc=牛肉拉麵 sc=牛肉拉面 pr=niu2 rou4 la1 mian4 en=ramen (pulled noodles) with beef

line=牛肉炒麵 牛肉炒面 [niu2 rou4 chao3 mian4] /stir-fried noodles with beef/
tc=牛肉炒麵 sc=牛肉炒面 pr=niu2 rou4 chao3 mian4 en=stir-fried noodles with beef

line=牛肉芹菜 牛肉芹菜 [niu2 rou4 qin2 cai4] /beef and celery/
tc=牛肉芹菜 sc=牛肉芹菜 pr=niu2 rou4 qin2 cai4 en=beef and celery

line=牛肉面 牛肉面 [niu2 rou4 mian4] /beef noodle soup/
tc=牛肉面 sc=牛肉面 pr=niu2 rou4 mian4 en=beef noodle soup

line=牛膝 牛膝 [niu2 xi1] /Achyranthes bidentata (root used in Chinese medicine)/
tc=牛膝 sc=牛膝 pr=niu2 xi1 en=Achyranthes bidentata (root used in Chinese medicine)

line=牛至 牛至 [niu2 zhi4] /oregano/marjoram/
tc=牛至 sc=牛至 pr=niu2 zhi4 en=oregano


```
line=牛蒡 牛蒡 [niu2 pang2] /burdock/  
tc=牛蒡 sc=牛蒡 pr=niu2 pang2 en=burdock
```

我們可以看到以上的程式剖析結果是正確無誤的。

4.4. 中文的詞彙掃描 - 斷詞

由於中文不像英文有利用空白進行詞彙分隔，因此中文的詞彙掃描比英文稍為複雜一些。通常我們必須使用詞典才能完成中文的掃描動作。

當我們使用詞典進行中文詞彙掃描時，有一個簡單卻重要的技巧是「長詞優先」。舉例而言，假如輸入的語句是「黃媽媽愛吃牛肉拉麵」，那麼整個句字經過詞彙掃描程式後的輸出可能會有下列結果。

1. 黃，媽媽，愛，吃，牛肉，拉麵
2. 黃，媽，媽，愛，吃，牛，肉，拉，麵
3. 黃媽媽，愛，吃，牛肉拉麵

在以上三個結果中，如果要評判好壞，我們會認為取得的詞彙越長越好，這個法則就稱為「長詞優先」法則¹⁴。

因此，若根據長詞優先法則，則應該選擇以上的第 3 個斷詞結果。

14 當然這必須視應用而定，像是在「翻譯、檢索、注音轉國字」等問題上，長詞優先的法則就很適用。

第5章 語法剖析

5.1. 剖析器

當我們能正確的取得詞彙之後，就可以站在詞彙的基礎上進行語法剖析，撰寫出剖析器程式。

「語法剖析」的方法仍然是依靠生成語法，但與前述「語言生成」的方向相反。「語言生成」是從語法開始，逐步的用非終端項目生成終端項目，以便產生完整的語句。但是語法剖析則是從現有的語句開始，逐步的透過語法比對，從終端項目組合出非終端項目，逐步的從下而上組合出整個語法樹。

在本章中，我們將分別以「程式語言」、「自然語言」、「標記語言」為例，說明語法剖析的方法與其應用。

首先，讓我們來看看後續進行剖析所需要的一個基本程式檔：`parselib.c`

```
#include <stdio.h>
#define BOOL int
char *text, token[100];
int ti = 0;
#define ch text[ti]
```

```
#define cmember(c,set) (c != '\0' && strchr(set,c))
```

```
void cnext() {  
    printf("%c", ch);  
    ti++;  
}
```

```
void init(char *s) {  
    text = s;  
    token[0]='\0';  
}
```

```
char *scan(char *format) {  
    if (sscanf(&text[ti], format, token) > 0) {  
        ti += strlen(token);  
        return token;  
    } else {  
        printf("scan fail!");  
        exit(1);  
        return NULL;  
    }  
}
```

```
BOOL next(char *format, char *options) {
```

```
scan(format);  
return (strstr(options, token)>=0);  
}
```

5.2. 剖析程式語言

執行結果：

```
ccckmit@CCCKMIT-PC /g/code/cl/ch5 (master)
```

```
$ gcc parseexp.c -o parseexp
```

```
ccckmit@CCCKMIT-PC /g/code/cl/ch5 (master)
```

```
$ parseexp
```

```
<E>T(3)*<E>(<E>T(5)+<E>T(8)</E></E>)</E></E>
```

程式：parseexp.c

```
#include "parselib.c"
```

```
// === BNF Grammar =====
```

```
// E = T [+*/] E | T
```

```
// T = [0-9]+ | (E)
```

```
int main(int argc, char * argv[]) {
```

```
    init("3*(5+8)");
    E();
}

int E() {
    printf("<E>");
    T();
    if (cmember(ch, "[+-*/]")) {
        cnext();
        E();
    }
    printf("</E>");
}

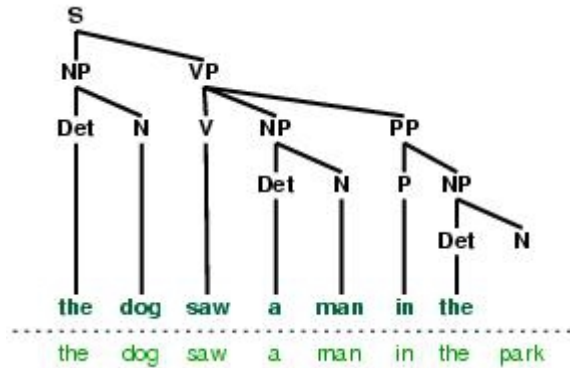
int T() {
    if (cmember(ch, "(")) {
        cnext();
        E();
        cnext();
    } else {
        scan("%[0-9]");
        printf("T(%s)", token);
    }
}
```

5.3. 剖析自然語言

規則比對法是早期的自然語言研究重心，這個方法企圖利用語言知識庫處理機器翻譯的問題。其主要的核心理念來源於 Chromsky 的生成語法路線，Chromsky 考察語言的結構後，提出一套使用語法規則描述語句結構的方法，稱為生成語法。許多學習外語的人都曾經學習過文法的概念，Chromsky 所謂的語法與這些文法其實是類似的。舉例而言，以下是一組簡單的生成語法規則。

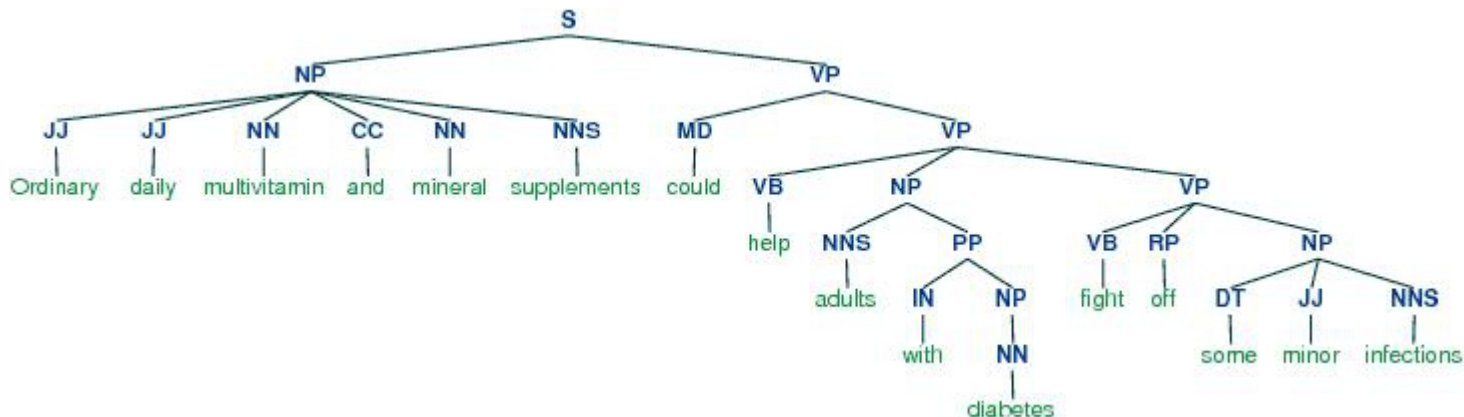
規則	說明
$S \Rightarrow NP VP$	句子 = 名詞子句 接 動詞子句
$NP \Rightarrow Det Adj^* N PP^*$	名詞子句 = 定詞 接 名詞
$VP \Rightarrow V (NP PP^*)$	動詞子句 = 動詞 接 名詞子句 接副詞子句
$PP \Rightarrow P NP$	副詞子句 = 副詞 接 名詞子句

根據這樣的規則，我們就可以將 『The dog saw a man in the park.』 這句話，剖析成下列的語法樹。



圖、簡單的語法樹的範例

甚至，對於更複雜的句子，像是『Ordinary daily multivitamin and mineral supplements could help adults with diabetes fight off some minor infections.』，也可能轉換成下列的剖析樹。



圖、複雜的語法樹的範例

一般來說，目前的自然語言剖析技術並沒有辦法將所有句子都轉換成完整的樹狀結構，通常只有 60%-70% 左右的成功率而已。因此，有時會採用部分剖析樹直接使用，而非一定要完全剖析成功。

執行結果：

```
$ gcc parseenglish.c -o parseenglish
```

```
ccckmit@CCCKMIT-PC /g/code/cl/ch5 (master)
```

```
$ parseenglish
```

```
<S><NP>D(a)N(dog)</NP><VP>V(eat)<NP>D(a)N(cat)</NP></VP></S>
```

程式：parseenglish.c

```
#include "parselib.c"
// === BNF Grammar =====
// S = NP VP
// NP = D N
// VP = V NP
int main(int argc, char * argv[]) {
    init("a dog eat a cat ");
    S();
}

int S() { printf("<S>"); NP(); VP(); printf("</S>"); }

int NP() { printf("<NP>"); D(); N(); printf("</NP>"); }

int VP() { printf("<VP>"); V(); NP(); printf("</VP>"); }

int D() {
    next("%[a-z] ", "|the|a|");
    printf("D(%s)", token);
}
```

```

    scan("%[ ]");
}

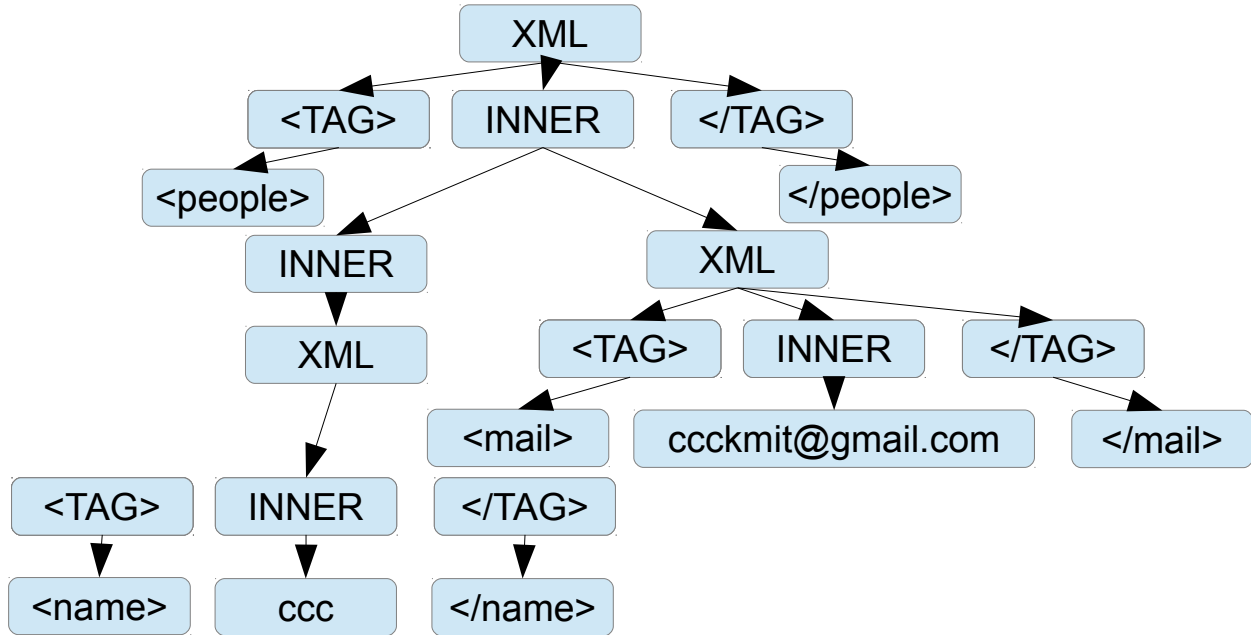
int N() {
    next("%[a-z] ", "|dog|bone|cat|");
    printf("N(%s)", token);
    scan("%[ ]");
}

int V() {
    next("%[a-z] ", "|chase|eat|");
    printf("V(%s)", token);
    scan("%[ ]");
}

```

5.4. 剖析標記語言

簡化的 XML 的語法	XML 範例
XML = <TAG> INNER </TAG> TAG = [0-9a-zA-Z]* INNER = INNER XML [^<]*	<pre> <people> <name>ccc</name> <mail>ccckmit@gmail.com</mail> </people> </pre>



第6章 語意辨識

6.1. 何謂語意？

在語言處理的領域，最難定義的大概就是語意了。我們可以清楚的理解何謂詞彙、何謂語法，但是卻很難清楚的「定義」某個文章的語意。

在第一章當中，我們曾經用「程式將語言轉換成一連串的動作，而這些動作符合使用者的請求」來定義一個程式是否理解了使用者的語句。但是到底使用者的請求是甚麼呢？在這一章中，我們同樣會將語言分成「程式語言、自然語言、標記語言」等三類，來說明這些語言的語意，然後再開始探討如何辨識語意的方法。

讓我們從語法最固定的「程式語言」開始，看看語意到底是甚麼？

6.2. 程式的語意

對於程式語言而言，其語意是非常清楚的，但是我們卻通常不知道這些東西稱為語意。

當我們將一段程式編譯成二進位的機器碼，然後將機器碼交給電腦執行，這些機器碼就代表了程式的語意。當然、機器碼很難理解，事實上這些機器碼對應到的也就是電腦的組合語言，因此我們說某個 C 語言程式的語意，其實就是將該程式編譯後輸出的組合語言。

那麼、組合語言的語意到底是甚麼呢？事實上、組合語言只是一堆指揮 CPU 如何動作的指令罷了，因此、組合語言的語意其實就是 CPU 的動作。

從這個觀點來看，我們可以很清楚的理解程式語言的語意，讓我們更進一步的仔細看看程式語言的語意究竟是甚麼意思？

6.2.1. 詞彙

在程式語言當中，詞彙大致可以分為幾類，第一類是關鍵字，像是 `if`, `for`, `while`, `struct`, `class`, `union`, ...，第二類是變數名稱，像是 `x`, `y`, `i`, `j`, `f`,，第三類是形態，像是 `int`, `object`, `string`, ... 等類型，第四類是符號，像是 `{`, `}`, `[`, `]`, `(`, `)`, `>`, `<`, `==`, `!=`, `&&`, `||`, 等等。

第二類的變數名稱的語意非常的清楚，代表的就是該變數的記憶體位址，或者以物件導向的觀點，可以視為對應到該變數的物件。

第三類的形態則是描述某變數結構的詞彙，例如 `int x;` 代表 `x` 是一個 `int` 形態的整數。因此形態關鍵字其實是用來指定某變數的形態。

第一類的關鍵字，還可以進一步分成幾類，一種是程序性的關鍵字，像是 `if`, `for`, `while` 等，另一種是結構性的關鍵字，像是 `struct`, `class`, `union` 等。

程序性的關鍵字通常用來描述某種控制流程，像是 `if` 的語法為 `if EXP BLOCK ELSE BLOCK`，其意義是當 `EXP` 為 `true` 時，就執行第一個 `BLOCK`，否則就執行第二個 `BLOCK`。而 `while` 的語法為 `while EXP BLOCK`，其語意識當 `EXP` 為 `true` 時，持續執行 `BLOCK`，直到 `EXP` 為 `false` 時才跳出。

而結構性的關鍵字，是用來描述某個資料結構或物件的組織方式，例如 `typedef struct { float x, y; } Point;` 代表有一個結構稱為 `Point`，其內容有 `x`, `y` 兩個浮點數。

第四類的符號，其作用與關鍵字類似，只是以比較短的形式呈現，其中 `>=`, `!=`, `<`, 等符號比較像程序性的關鍵字，而 `{}`, `[]`, `()`, 等則比較項結構性的關鍵字。

6.2.2. 語句

在語句層次，程式語言的意義通常代表一個動作，舉例而言，像是 $x=3$ 這個語句所代表的是將 x 內容指定為 3，而 $y=x+5$ 則是將 $x+5$ 的結果指定給 y 。

而 $(x > y)$ 這個語句則是將 x 與 y 進行大小的比較，然後將比較結果是真或假傳回。

6.2.3. 文章層次

在文章的層次，程式語言的意義通常是某種動作的流程，舉例而言，對於控制邏輯 if 語句 $\text{if}(x>y) \ t = x; \text{else } t = y;$ 這個段落而言，其意義代表的是「如果 x 比 y 大，那就將 t 設定為 x ，否則就將 t 設定為 y 」。

或許有人會覺得，我只不過用中文解釋了一遍該程式的意義而已，這樣就算程式的語意嗎？但事實上，我們除了用中文解釋，我也可以將同樣這件事用「組合語言」來解釋，如下所示。

原始程式	語意 (自然語言版)	語意 (組合語言版)
<pre>if(x>y) t = x; else t = y;</pre>	<p>如果 x 比 y 大</p> <p>那就將 t 設定為 x</p> <p>否則</p>	<pre>CMP x, y JLE ELSE LD t, x JMP EXIT ELSE: LD t, y</pre>

	就將 t 設定為 y	EXIT:
--	------------	-------

事實上，這個程式的語意與用甚麼語言表達無關，因為真正的語意是那些動作，而非語言本身。

6.3. 自然語言的語意

剖析法是語言理論的重頭戲。但是，即使我們利用與法理論完成了剖析，仍然不足以建構有價值的應用，許多有價值的應用，像是機器翻譯系統、問答系統、交談式系統、智慧型檢索等應用，都需要進一步抓住語句的意義，才能有應用的價值。

這時，我們需要某種語義理論，才能讓程式從語法形式的表像中，得知語句所表達的意義。但是，甚麼是語句的意義呢？電腦又如何能理解語句的意義呢？這個問題是自然語言當中最重要，但卻也最神祕難解的議題。在語言學與人工智慧上，都仍然是個謎。

6.3.1. 語句的意義

讓我們暫且撇開語句意義的爭議不談，先看看前輩們如何處理意義問題。在語法理論當中，有一個較為特別且受注目的語法學派，這個學派並不使用生成語法進行剖析，而是採用意義導向式的語法，直

接支解句子當中的元素。這個學派沒有固定的名稱，其語法通常稱為 **Case Grammar** (中文稱為格狀語法、或格變語法)、**Semantic Role** (語意角色語法)、或 **Conceptual Dependency** (概念依存語法)。在本書當中，我們以格狀語法統稱之。

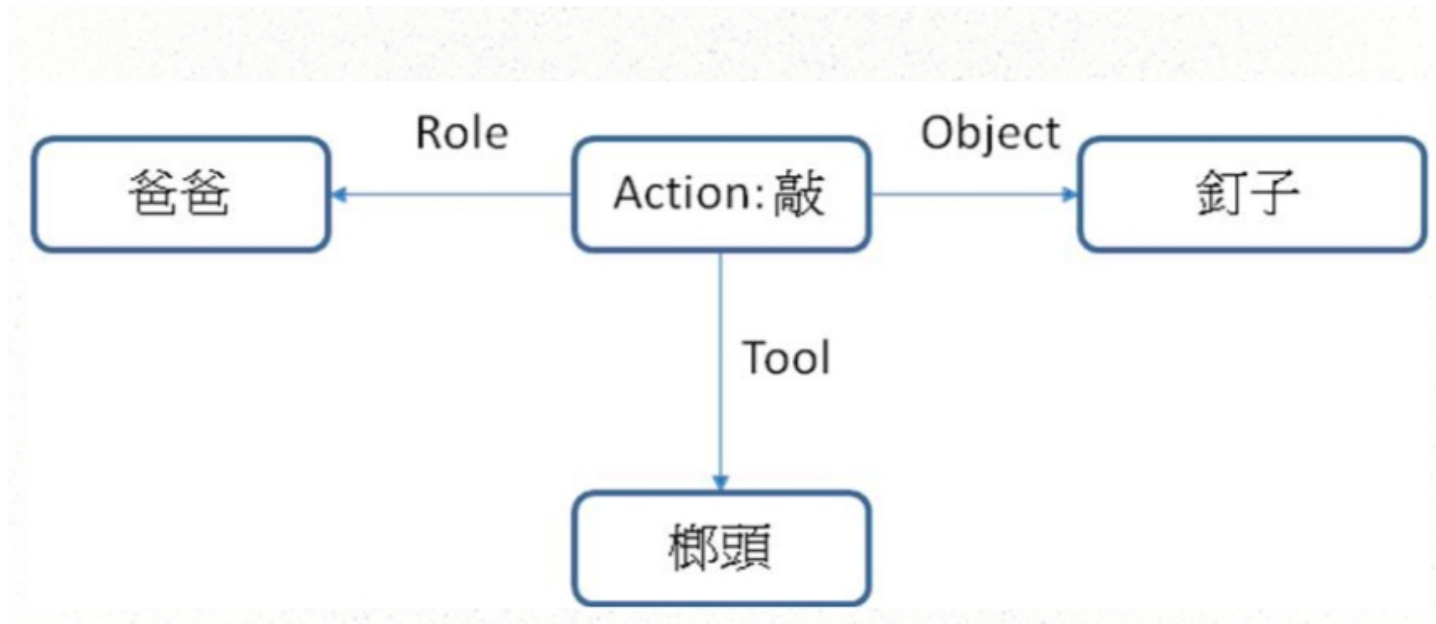
6.3.2. 格狀語法 (Case Grammar)

格狀語法通常不需要先進行剖析，程式會跳過剖析的階段，直接採用物件導向式的方式，根據語意規則比對的方式進行理解。

格狀語法是以角色與動作為核心的語法，例如，主角通常稱為 **Role**，動作通常稱為 **Action**，物體通常稱為 **Object**，而工具稱為 **Tools**。如此，以下的語法規則 1 就非常的具有語意上的抓取能力。

語法規則 1：Role Action Object by Tools

舉例而言，『爸爸用榔頭敲釘子』這個句子經過『理解』之後，就會建構出下列圖狀結構。



圖、語義結構的範例

很明顯的，這樣的語法會以動詞 Action 為核心，而主角 Role、工具 Tool 與目標物 Object 都是該動作

上的一個角色。這種強調動作語意的語法結構，對語意的『理解』會比生成語法直接。因此，通常被用在交談式系統的設計上，以避開剖析的困難，直接處理語意問題。

6.3.3. 欄位填充機制

其實，格狀語法感覺並不像是一種語法，而是一種圖形表達結構。為了將語句對應到格狀語法，必須採用某種方法，可以將詞彙填入到這些格子 (Case) 當中，這種方法就稱為欄位填充機制。

以下是一個格狀語法填充的範例，讓我們先看看使用的語法規則。

語法規則：Role(動物) Action(咬) Object(動物 or 物品) by Tools(牙齒)

語句範例：那隻狗咬了穿紅衣服的女人，牙齒血淋淋的！

字典紀錄：狗:Animal, 人:People, People:Animal

屬性比對：那隻「狗 (Animal)」 「咬」了穿紅衣服的女「人 (People:Animal)」，牙齒血淋淋的！

標示欄位：那隻「Role:狗 (Animal)」 「Action:咬」了穿紅衣服的女「Object:人 (People:Animal)」，牙齒血淋淋的！

填充結果：Role(動物:狗) Action(咬) Object(動物:人) by Tools(牙齒?)

如果可以搭配與法剖析的機制，那麼就可能更進一步的將填充結果提升到名詞子句 (NP) 的層次，得到：

填充結果：Role(動物:那隻狗) Action(咬了) Object(動物:人:穿紅衣服的女人) by Tools(牙齒?)

於是如果我們對電腦進行閱讀測驗考試，問電腦「那隻狗咬了誰？」時，電腦就有可能回答「穿紅衣服的女人」。而詢問「誰咬了穿紅衣服的女人」的時候，電腦也有辦法回答「是那隻狗」。甚至是詢問「那隻狗用甚麼咬了女人」時，電腦也可以根據欄位填充結果，回答「應該是用牙齒」。

6.3.4. 文章與劇本 (Script)

以上的格狀規則「Role Action Object by Tools」可以適用於很多的語句，但是卻很難描述更上層的複雜概念，像是「故事、小說、新聞、天氣預報」等等，因此研究者發展出了一種處理更高層語意結構的

概念，稱為劇本 (Script)¹⁵¹⁶¹⁷。

那麼，甚麼是劇本呢？讓我們以「結婚」為例子，來看看一個典型的劇本。

男人 愛 女人 => 男人 女人 交往 => 男人 女人 結婚 => 男人 女人 交配 => 女人 生 小孩 => 小孩長大 => 外遇[男人 (交配愛) 另一個女人] => 女人 發現 這件事 => 男人 女人 離婚

像這樣的故事我們經常聽到，連續劇與電影裏也經常上演，真實生活更是有無數的案例存在，所以當我們想要讓電腦「理解」這種故事的時候，最簡單的方法就是先準備好這樣的劇本，然後當電腦一邊解析文章的時候，就一邊比對文章的語句，看看目前進展到哪一個階段，每個人所扮演的角色各自為何者等等。

舉例而言，假如電腦正在解析一篇文章，其內容如下：

約翰與瑪莉是大學的同班同學，在大三時約翰愛上了瑪莉，畢業後他們就結婚了，並且生了一個小男孩麥克。

15 Schank, R. & Abelson, R. (1977). Scripts, plans, goals, and understanding: An inquiry into human knowledge structure. Hillsdale, NJ: Lawrence Erlbaum Associates.

16 Schank, R. C. & Abelson, R. P. (1995). Knowledge and memory: The real story. In Wyer, R. S. (Ed.), Knowledge and memory: The real story. Advances in social cognition, 8, 1-85. Schank,

17 <http://www4.ncsu.edu/~kklein/html/narrative.htm>

約翰畢業後進入一家大公司工作，由於表現良好，所以被拔擢為經理，妮可是他的秘書。由於工作上朝夕相處，以致約翰與妮可日久生情。

當瑪莉發現約翰與妮可的外遇事件後，她決定離婚，並且取得了麥克的監護權。而兩人離婚之後不久，約翰與妮可就結婚了。

於是，如果我們使用上述劇本去解析比對這篇文章，就可能得到下列的結果：

約翰與瑪莉是大學的同班同學，在大三時 (約翰：男人) (愛)上了 (瑪莉：女人)，畢業後(他們:男人,女人) 就 (結婚) 了，並且(生)了一個 (小男孩麥克:小孩)。

約翰畢業後進入一家大公司工作，由於表現良好，所以被拔擢為經理，妮可是他的秘書。由於工作上朝夕相處，以致 (約翰：男人) 與 (妮可：女人) 日久生情。

當瑪莉發現(約翰：男人)與(妮可：女人) 的(外遇)事件後，(她:女人) 決定 (離婚)，並且取得了麥克的監護權。而兩人離婚之後不久，約翰與妮可就結婚了。

於是透過這樣的「高層欄位填充機制」，電腦稍微能掌握到故事的發展進度與結構，而這正是劇本 (Script) 所產生的效果。

6.3.5. 程式實作

6.3.5.1. 格變語法 case.c

以下是我們用 C 撰寫的一個程式 case.c 執行的結果

```
D:\c>gcc case.c -o case
```

```
D:\c>case
```

```
爸爸 用 榔頭 敲 釘子
```

```
people:爸爸 knock:敲 object:釘子 hammer:榔頭
```

```
貓 吃 魚
```

```
animal:貓 魚 eat:吃 food:魚
```

```
#include "semantics.c"
```

```
// 格變語法: Role Action Object by Tools
```

```
// 比對範例: semantics("爸爸用榔頭敲釘子") => people:爸爸 knock:敲 object:釘子 hammer:榔頭
```

```
// 比對範例: semantics("貓吃魚") => animal:貓 魚 eat:吃 food:魚
```

```
char *words[] = {"爸爸", "敲", "釘子", "榔頭", "用", "貓", "吃", "魚", NULL}; // 詞典
```

```

char *isList[][2]  // 詞彙與標記的關係串列
= {{ "敲", "knock"}, {"爸爸", "people"}, {"釘子", "object"}, {"榔頭", "hammer"},
   {"貓", "animal"}, {"吃", "eat"}, {"魚", "food"}, {"魚", "animal"}, {NULL, NULL}};
char *cases[][SLOTS] = { // case (格變語法)
    { "animal", "eat", "food", NULL },
    { "people", "knock", "object", "hammer"},
    { NULL, NULL, NULL, NULL }
};

int main() {
    semantics("爸爸用榔頭敲釘子"); // 語意解析範例 1
    semantics("貓吃魚");           // 語意解析範例 2
}

```

6.3.5.2. 欄位填充機制

程式：semantics.c

```

#include <stdio.h>
#include <assert.h>

#define NIL -1 // 找不到時傳回 NIL
#define SLOTS 4 // 一個 case 的最多 slot 數量
#define FILLS 4 // 每個 SLOT 最多填入詞彙數
#define TOKENS 1000 // 一個句子中最多詞彙上限
#define SMAX 100000 // 字串表容量

char *empty=""; // 空字串
char *slots[SLOTS][FILLS]; // 每個欄位的填充值（可填多個，最多 FILLS 個）
int fills[SLOTS]; // 每個欄位的填充個數
char *tokens[TOKENS]; // 詞彙串列
char strTable[SMAX]; // 字串表
int strTop = 0; // 字串表大小
extern char *words[]; // 詞彙串列
extern char *isList[][2]; // （詞彙，標記）配對串列
extern char *cases[][SLOTS]; // 格變語法串列

// 將語句斷詞成詞彙串列
int tokenize(char *str, char *tokens[]) {
    int i, ti=0;

```

```

    for (i=0; i<strlen(str); ) {
        int wi = wordFind(&str[i]);
        char *ch;
        if (wi == NIL) {
            ch = &strTable[strTop];
            sprintf(ch, "%c%c", str[i], str[i+1]);
            strTop += 3;
        }
        assert(ti < TOKENS);
        tokens[ti] = (wi==NIL)?ch:words[wi];
        i+= strlen(tokens[ti]);
        ti++;
    }
    tokens[ti] = NULL;
}

// 尋找詞典中是否有這個詞彙
int wordFind(char *str) {
    int i;
    for (i=0; words[i]!=NULL; i++) {
        if (strncmp(str, words[i], strlen(words[i]))==0) {
            return i;
        }
    }
    return NIL;
}

```

```
        }  
    }  
    return NIL;  
}
```

// 印出詞彙陣列

```
int wordsPrint(char *words[]) {  
    int i;  
    for (i=0; words[i] != NULL; i++) {  
        printf("%s ", words[i]);  
    }  
    printf("\n");  
}
```

// 欄位填充的主要演算法

```
double caseFill(char *tokens[], char *fields[SLOTS], char *slots[SLOTS][FILLS]) {  
    int ti, si, fi, score=0;  
    for (si=0; si<SLOTS; si++)  
        fills[si] = 0;  
    for (ti=0; tokens[ti]!=NULL; ti++) {  
        for (si=0; si<SLOTS; si++) {  
            if (pairFind(tokens[ti], fields[si], isList)!=NIL) {
```

```

        assert(fills[si] < FILLS);
        slots[si][fills[si]++] = tokens[ti];
    }
}

for (si=0; si<SLOTS; si++)
    score += fills[si];
return score;
}

```

// 尋找出最好的填充規則

```

int caseBest() {
    int ci, best=0;
    double bestScore=0.0;
    for (ci=0; cases[ci][0] != NULL; ci++) {
        double score = caseFill(tokens, cases[ci], slots);
        if (score > bestScore) {
            best = ci;
            bestScore = score;
        }
    }
    return best;
}

```

```

}

// 印出填充的情況
int casePrint(char *fields[SLOTS], char *slots[SLOTS][FILLS]) {
    int si, fi;
    for (si=0; si<SLOTS; si++) {
        if (fields[si] != NULL) {
            printf("%s:", fields[si]);
            for (fi=0; fi<fills[si]; fi++)
                printf("%s ", slots[si][fi]);
        }
    }
    printf("\n");
}

```

// 尋找 (child, parent) 配對是否存在 pairList 當中。

```

int pairFind(char *child, char *parent, char *pairList[][2]) {
    if (child == NULL || parent == NULL) return NIL;
    int pi=0;
    for (pi=0; pairList[pi][0] != NULL; pi++) {
        if (strcmp(child, pairList[pi][0])==0
            && strcmp(parent, pairList[pi][1])==0)

```

```

        return pi;
    }
    return NIL;
}

// 語意解析：以欄位填充方法找出最佳填充的 case，然後印出填充情況。
int semantics(char *str) {
    tokenize(str, tokens); // 斷詞
    wordsPrint(tokens);    // 印出斷詞
    int best = caseBest(); // 找出最佳填充
    double score = caseFill(tokens, cases[best], slots); // 重填一次最佳填充
    casePrint(cases[best], slots); // 印出最佳填充
    printf("\n");
}

```

6.3.5.3. 劇本比對

執行結果：

```
D:\c\semantics>gcc script.c -o script
```


D:\c\semantics>script

約翰 與 瑪莉 是 大學 的 同 班 同 學

man:約翰 love:woman:瑪莉

在 大 三 時 約翰 愛 上 了 瑪莉

man:約翰 love:愛 woman:瑪莉

畢 業 後 他們 就 結 婚 了

man:他們 marry:結婚 woman:他們

並 且 生 了 一 個 小 男 孩 麥克

woman:born:生 baby:麥克

約翰 畢 業 後 進 入 一 家 大 公 司 工 作

man:約翰 love:woman:

由 於 表 現 良 好 ， 所 以 被 拔 擢 為 經 理

man:love:woman:

妮 可 是 他 的 秘 書

man:他 love:woman:妮可

由於工作上朝夕相處，以致約翰與妮可日久生情

man:約翰 love:日久生情 woman:妮可

當瑪莉發現約翰與妮可的外遇事件後

man:約翰 woman:瑪莉 妮可 extramarital:外遇

她決定離婚

man:divorce:離婚 woman:她

並且取得了麥克的監護權。

man:麥克 love:woman:

而兩人離婚之後不久

man:divorce:離婚 woman:

約翰與妮可就結婚了。

man:約翰 marry:結婚 woman:妮可

程式：script.c

```
#include "semantics.c"
```

// 劇本比對：使用格變語法

```
char *sentences[] =
```

```
{ "約翰與瑪莉是大學的同班同學", "在大三時約翰愛上了瑪莉", "畢業後他們就結婚了",  
  "並且生了一個小男孩麥克", "約翰畢業後進入一家大公司工作", "由於表現良好，所以被拔擢為經理",  
  "妮可是他的秘書", "由於工作上朝夕相處，以致約翰與妮可日久生情", "當瑪莉發現約翰與妮可的外遇事件後",  
  "她決定離婚", "並且取得了麥克的監護權。", "而兩人離婚之後不久", "約翰與妮可就結婚了。";
```

```
char *words[] = {"日久生情", "約翰", "瑪莉", "妮可", "麥克", "結婚", "他們", "他們", "外遇",  
  "離婚", "愛", "生", NULL}; // 詞典
```

```
char *isList[][2] // 詞彙與標記的關係串列
```

```
= { {"日久生情", "love"}, {"約翰", "man"}, {"瑪莉", "woman"}, {"妮可", "woman"},  
    {"麥克", "man"}, {"麥克", "baby"}, {"他們", "man"}, {"他們", "woman"},  
    {"外遇", "extramarital"}, {"結婚", "marry"}, {"離婚", "divorce"}, {"愛", "love"},  
    {"生", "born"}, {"他", "man"}, {"她", "woman"}, {NULL, NULL}};
```

```
char *cases[][SLOTS] = { // case (格變語法)
```

```
    { "man", "love", "woman", NULL },           // 男人 愛 女人  
    { "man", "date", "woman", NULL },           // 男人 女人 交往  
    { "man", "marry", "woman", NULL },          // 男人 女人 結婚  
    { "man", "sex", "woman", NULL },            // 男人 女人 交配  
    { "woman", "born", "baby", NULL },          // 女人 生 小孩
```

```
    { "baby", "growup", NULL, NULL},           // 小孩 長大
    { "man", "woman", "extramarital", NULL}, // 外遇[男人 (交配|愛) 另一個女人]
    { "man", "divorce", "woman", NULL}, // 男人 女人 離婚
    { NULL, NULL, NULL, NULL }
};

int main() {
    int i;
    for (i=0; sentences[i] != NULL; i++)
        semantics(sentences[i]);
}
```

第7章 語法比對

7.1. 字串比對

字串比對是語言處理當中一個技術相對成熟的工具，很多比較明確的語言處理問題，都可以用字串比對的方是得到不錯的解法。

但是，字串比對方法的能力也有不小的限制，例如不具被抽象化的能力，不具備遞迴式語法的比對能力等等。因此字串比對只能在一些較為明確、不需推理與不需多層次連結的情況下才有辦法進行。

字串比對的方法當中，最簡單的一種當屬於尋找子字串的工作了，以下函數原型分別是 **Java** 與 **C/C++** 的尋找子字串函數。

```
Java : int indexOf(String str)
```

```
C/C++ : char * strstr (char * str1, char * str2 );
```

舉例而言，當我們想知道一個字串中是否有「父親、母親」等詞彙時，我們可以用下列程式進行判斷。

```
if (strstr(str, "父親") || strstr(str, "母親"))
    ....
else
    ....
```

事實上，1960 年代就發展出來的著名交談系統 **Eliza** 就是利用類似這樣的字串比對方法所做的，這種簡單方法的能力有時超越我們的想像。

這樣的方法也可用來抽取文件中的某些欄位，例如以下的程式就可以抽取 xml 文件中標記內的文字。

程式：xmlInner.c

```
#include <stdio.h>

char *innerText(char *inner, char *pText, char *beginMark, char *endMark) {
    char *beginStart = strstr(pText, beginMark);
    if (beginStart == NULL) return NULL;
    char *beginEnd = beginStart + strlen(beginMark);
    char *endStart = strstr(beginEnd, endMark);
```

```

    if (endStart < 0) return NULL;
    int len = endStart-beginEnd;
    strncpy(inner, beginEnd, len);
    inner[len] = '\0';
}

int main() {
    char xml[] = "<people name=\"陳鍾誠\" sex=\"男\">"
        "<age>43</age>"
        "<hometown>金門縣</hometown>"
        "</people>";
    char name[30], sex[10], age[10], hometown[30];
    innerText(name, xml, "name=\"", "\"");
    printf("name=%s\n", name);
    innerText(sex, xml, "sex=\"", "\"");
    printf("sex=%s\n", sex);
    innerText(age, xml, "<age>", "</age>");
    printf("age=%s\n", age);
    innerText(hometown, xml, "<hometown>", "</hometown>");
    printf("hometown=%s\n", hometown);
}

```

執行結果：

```
D:\ccc102\CL>gcc xmlInner.c -o xmlInner
```

```
D:\ccc102\CL>xmlInner
```

```
name=陳鍾誠
```

```
sex=男
```

```
age=43
```

```
hometown=金門縣
```

如果我們想要比對的樣式較為複雜，就會用到類似「正規表達式」的比對方法。許多程式語言都有內建正規表達式的函式庫，像是 Perl, JavaScript, Java, C#, Python,等。

C 語言雖然也有一些函式庫可以處理正規表達式 (像是 glibc 當中的 regex.h 就是)，但是這樣的函式庫並不屬於標準函式庫。不過，在標準 C 函式庫中的 `sscanf` 其實就具備了類似正規表達式的功能，因此理解正規表達式也是學會 `sscanf` 這種字串比對函數的關鍵，讓我們先來看看何謂正規表達式？

7.2. 正規表達式

正規語法 (Regular Grammar) 是一種相當簡單的語法，這種語法被 Perl 語言成功的用於字串比對，接著成為重要的程式設計工具。此種標準的正規語法後來被稱為正則表達式 (Regular Expression)。目前，大部分的語言都已納入正則表達式的函式庫，正規表達式可以說是程式設計師必定要瞭解的工具，也就是常識的一部分。系統程式設計師更應該要瞭解正則表達式，因為正規語法是程式語言當中，用來描述基本詞彙 (Vocabulary)，並據以建構詞彙掃描器 (Lexer) 的基礎語法，Lexer 是編譯器的基本元件之一。

假如我們要用正則表達式描述整數數字，那麼，可以用 `[0123456789]+` 這個表達式，其中的中括號 `[` 與 `]` 會框住一群字元，用來代表字元群，加號 `+` 所代表的是重複 1 次或以上，因此，該表達式就可以描述像 `3702451` 這樣的數字。然而，在正則表達式中，為了更方便撰寫，於是允許用 `[0-9]+` 這樣的式子表達同樣的概念，其中的 `0-9` 其實就代表了 `0123456789` 等字元，這是一種簡便的縮寫法。甚至，可以再度縮短後以 `[\d]+` 代表，其中的 `\d` 就代表數字所成的字元集合。

利用範例學習是理解正則表達式的有效方法，表格 1 就顯示了一些具有代表性的正則表達式範例。

表格 1. 正則表達式的範例

語法	正則表達式	範例
整數	[0-9]+	3704
有小數點的實數	[0-9]+\.[0-9]+	7.93
英文詞彙	[A-Za-z]+	Code
變數名稱	[A-Za-z_][A-Za-z0-9_]*	_counter
Email	[a-zA-Z0-9_]+@[a-zA-Z0-9\._]+	ccc@kmit.edu.tw
URL	http://[a-zA-Z0-9\._/]+	http://ccc.kmit.edu.tw/mybook/

為了協助讀者理解這些範例，我們有必要對範例中的一些正則表達式符號進行說明。

在實數的範例中，使用 \. 代表小數點符號 .，不熟悉正則表達式的讀者一定覺得奇怪，為何要加上斜線符號 \ 呢？這是因為在正則表達式當中，有許多符號具有特殊意義，例如點符號 . 是用來表示任意字元的，星號 * 是代表 0 次或以上，加號 + 代表一次或以上，在正則表達式當中，有許多這類的特殊

字元，因此用斜線 \ 代表跳出字元，就像 C 語言當中 printf 函數內的用途一樣。因此，當我們看到 \ 符號時，必須繼續向後看，才能知道其所代表的意義。

接著，讓我們用 C 語言的 sscanf 來示範如何使用這種「類正規表達式」函數來抽取文章中的樣式，以下是一個範例。

程式：sscanf.c

```
#include <stdio.h>

int main() {
    char name[20], tel[50], field[20], areaCode[20], code[20];
    int age;
    sscanf("name:john age:40 tel:082-313530", "%s", name);
    printf("%s\n", name);
    sscanf("name:john age:40 tel:082-313530", "%8s", name);
    printf("%s\n", name);
    sscanf("name:john age:40 tel:082-313530", "%[^:]", name);
    printf("%s\n", name);
    sscanf("name:john age:40 tel:082-313530", "%[^:]:%s", field, name);
    printf("%s %s\n", field, name);
    sscanf("name:john age:40 tel:082-313530", "name:%s age:%d tel:%s", name, &age, tel);
```

```
printf("%s %d %s\n", name, age, tel);
sscanf("name:john age:40 tel:082-313530", "%*[^:]:%s %*[^:]:%d %*[^:]:%s", name, &age, tel);
printf("%s %d %s\n", name, age, tel);

char protocol[10], site[50], path[50];
sscanf("http://ccckmit.wikidot.com/cp/list/hello.txt",
      "%*[^:]:%*2[/][^/]/%[a-zA-Z0-9._/-]",
      protocol, site, path);
printf("protocol=%s site=%s path=%s\n", protocol, site, path);
return 1;
}
```

執行結果：

```
D:\oc>gcc sscanf.c -o sscanf
```

```
D:\oc>sscanf
```

```
name:john
```

```
name:joh
```

```
name
```

```
name john
```

```
john 40 082-313530
```

john 40 082-313530

protocol=http site=ccckmit.wikidot.com path=cp/list/hello.txt

第8章 自然語言的處理

8.1. 自然語言

自然語言處理是人工智慧的重要子領域，主要研究如何用電腦處理像中文、英文等自然語言，特別是表現出與智慧相關的語言行為，像是交談系統、機器翻譯、自然語言理解等。

自然語言處理的終極目標是要讓電腦可以理解人類的話語，並且進行流暢的對話，但是這個目標顯然太過高遠，目前看不到任何完整解決的可能性。但是如果只是要欺騙人類，讓某些人以為電腦真的在與他進行對話，這倒是不太困難，著名的 *Eliza* 程式就曾經達成這樣的目標。

自然語言處理的方法主要分為統計式與規則式的兩種方法。在 *Web* 發明以前，規則式的方法較受到重視，但在 *Web* 發明之後，由於語料的大量增加，導致統計式的方法發展迅速。另外，像是神經網路式的自然語言處理方法，雖然有少數人進行研究，但是還沒有形成一個完整的體系。

8.2. 規則比對法

不論是採用生成語法或格狀語意，最後電腦都得將「自然語言語句」與「某種規則」進行比對，才能將語句填入到對應的規則當中，如果這種規則是用「人工撰寫」的，那麼這樣的方法就稱為「規則比對法」，這種方法就好像「專家系統」當中的邏輯推理引擎一樣。

8.2.1. Eliza 交談系統 - 理解程度很淺

```
G:\c\ChatBot>gcc ChatBot.c -o ChatBot
```

```
G:\c\ChatBot>ChatBot
```

您好，我的朋友! 有心事嗎？

>> 你是誰

你認為我是誰

>> 電腦嗎？

你說的電腦是指我嗎？

>> 是指你阿！

別說我了，談談你吧！

>> 今天真糟糕，運氣不好

為何不好？

>> 我的好朋友不理我了

你的好朋友不理你了?

>> 他生氣了

請繼續說下去

>> 我們吵架了

你為何們吵架了

>> 為了一包泡麵

想多聊一聊嗎

>> 不聊了

為何不聊了?

>> 因為心情不好

為何不好?

>> 再見

祝福你

程式：ChatBot.c

```
#include <stdio.h>
#include <time.h>
#define LEN 512
// 注意，本程式用 big5 編碼，會有「許蓋功」的衝碼問題，請勿使用「許蓋功」等字。
#define QMAX 10
#define AMAX 20
char *empty="";
```



```

typedef struct {
    char *q[QMAX];
    char *a[AMAX];
} QA;

QA qa[]={
{ .q={"謝謝", NULL}, .a={"不客氣!", NULL} },
{ .q={"對不起", "抱歉", "不好意思", NULL}, .a={"別說抱歉!", "別客氣，儘管說!", NULL} },
{ .q={"可否", "可不可以", NULL}, .a={"你確定想%s", NULL} },
{ .q={"我的", NULL}, .a={"你的%s?", NULL} },
{ .q={"我", NULL}, .a={"你為何%s", NULL} },
{ .q={"你是", NULL}, .a={"你認為我是%s", NULL} },
{ .q={"認為", "以為", NULL}, .a={"為何說%s?", NULL} },
{ .q={"感覺", NULL}, .a={"常有這種感覺嗎?", NULL} },
{ .q={"為何不", NULL}, .a={"你希望我%s", NULL} },
{ .q={"是否", NULL}, .a={"為何想知道是否%s", NULL} },
{ .q={"不能", NULL}, .a={"為何不能%s?", "你試過了嗎?", NULL} },
{ .q={"我是", NULL}, .a={"你好，久仰久仰!", NULL} },
{ .q={"甚麼", "什麼", "何時", "誰", "哪裡", "如何", "為何", "因何", NULL},
.a={"為何這樣問?", "為何你對這問題有興趣?", "你認為答案是甚麼呢?", NULL} },
{ .q={"原因", NULL}, .a={"這是真正的原因嗎?", "這是真正的原因嗎?", NULL} },
{ .q={"理由", NULL}, .a={"這說明了甚麼呢?", "還有其他理由嗎?", NULL} },
{ .q={"你好", "嗨", "您好", NULL}, .a={"你好，有甚麼問題嗎?", NULL} },
{ .q={"或者", NULL}, .a={"你好像不太確定?", NULL} },
{ .q={"你是", NULL}, .a={"你認為我是%s", NULL} },
{ .q={"不曉得", "不知道", NULL}, .a={"為何不知道?", "在想想看，有沒有甚麼可能性?", NULL} },

```

```

{ .q={"不想","不希望", NULL}, .a={"有沒有甚麼辦法呢?","為何不想%s?","那你希望怎樣呢?", NULL} },
{ .q={"想","希望", NULL}, .a={"為何想%s?","真的想%s?","那就去做阿?","為何不呢?", NULL} },
{ .q={"不", NULL}, .a={"為何不%s?", NULL} },
{ .q={"請", NULL}, .a={"我該如何%s","你想要我%s", NULL} },
{ .q={"你", NULL}, .a={"你真的是在說我嗎?","別說我了，談談你吧!","為何這麼關心我%s","你自己%s",
NULL} },
{ .q={"總是","常常", NULL}, .a={"能不能具體說明呢?","何時?", NULL} },
{ .q={"像", NULL}, .a={"有多像?","哪裡像?", NULL} },
{ .q={"對", NULL}, .a={"你確定嗎?","我了解!", NULL} },
{ .q={"朋友", NULL}, .a={"多告訴我一些有關他的事吧!","你認識他多久了呢?", NULL} },
{ .q={"電腦", NULL}, .a={"你說的電腦是指我嗎?", NULL} },
{ .q={"難過", NULL}, .a={"別想它了","別難過","別想那麼多了","事情總是會解決的 ", NULL} },
{ .q={"高興", NULL}, .a={"不錯丫","太棒了","這樣很好丫", NULL} },
{ .q={"是阿", "是的", NULL}, .a={"甚麼事呢?","我可以幫助你嗎?","我希望我能幫得上忙!", NULL} },
{ .q={"電腦", NULL}, .a={"你說的電腦是指我嗎?", NULL} },
{ .q={NULL}, .a={"我了解","我能理解","還有問題嗎 ?","請繼續說下去","可以說的更詳細一點嗎?",
"這樣喔! 我知道!","然後呢? 發生甚麼事?","再來呢? 可以多說一些嗎","接下來呢? ","可以多告訴我一些嗎?",
"多談談有關你的事，好嗎?","祝福你","想多聊一聊嗎", NULL} }
};

```

```

void replace(char *source, char *target, char *from, char *to) {
    char *match = strstr(source, from);
    if (!match)
        strcpy(target, source);
    else {
        int len = match-source;
        strncpy(target, source, len);
    }
}

```

```

        target[len] = '\\0';
        sprintf(target+len, "%s%s", to, match+strlen(from));
    }
}

void delay(unsigned int secs) {
    time_t retTime = time(0) + secs;    // Get finishing time.
    while (time(0) < retTime);           // Loop until it arrives.
}

void answer(char *question) {
    int i, qi, ai, acount;
    char *tail=empty;
    for (i=0; qa[i].q[0]!=NULL; i++) {
        for (qi=0; qa[i].q[qi]!=NULL; qi++) {
            char *q = qa[i].q[qi];
            char *match = strstr(question, q);
            if (match != NULL) {
                tail = match+strlen(q);
                printf("match Q:%s tail:%s\\n", q, tail);
                goto Found;
            }
        }
    }
}

```

Found:

```

    for (account = 0; qa[i].a[account] != NULL; account++);
    ai = rand()%account;
    delay(5+rand()%10);
    char youTail[LEN];
    replace(tail, youTail, "我", "你");
    printf(qa[i].a[ai], youTail);
    printf("\n");
}

int main() {
    char question[LEN];
    printf("您好，我的朋友！ 有心事嗎 ?\n");
    do {
        printf(">> ");
        gets(question);
        answer(question);
    } while (!strcmp(question, "再見")==0);
}

```

8.2.2. Baseball 棒球問答系統 - 理解程度較深

Baseball 是一個專門用來回答美國棒球紀錄之問題的交談系統¹⁸，包含一個有關美國棒球運動比賽紀錄

¹⁸ Baseball: An Automatic Question Answerer, B. Green, A. Wolf, C. Chomsky, and K. Laughery. Computers and Thought, Massachusetts: AAAI Press, (1963)

的資料庫，並根據這個資料庫所記載的資料回答使用者的問題，其方法為格位填充法，由於問答內容限定在一個資料庫中，因此、使用的格位只要包含這些欄位即可。

以下是其資料庫的一些記錄，我們用這個表格來說明此系統的運作過程。

Place	Month	Day	Game	Winner/Score	Loser/Score
Cleveland	July	6	95	White Sox / 2	Indian / 0
Boston	July	7	96	Red Sox /5	Yankees / 3
Detroit	July	7	97	Tiger / 10	Athletics / 2
Boston	July	15	98	Yankees/7	Red Sox/4

BASEBALL 系統提出一個稱為規格串列(Specification list) 的資料結構以便進行格位填充，下表是這個系統根據每個問題所建立的規格串列表：

Question	Specification List
Where did the Red Sox play on July 7 ?	Team = Red Sox Place = 7 Month = July Day = 7
What team won 10 games in July ?	Team(wining) = ? Game(number_of)=10 Month = July

問題是要如何建立規格串列呢？其方法是採用字典查詢，並對每一個字訂定其語意。

以下是一些字及其語意的對應關係表

Word	Semantic
Team	Team = (blank)
Red Sox	Team = Red Sox
Who	Team = ?
Winning	Adj : Winning
Boston	Place = Boston Or Team = Red Sox
The	No meaning
Did	No meaning
...	

以下是其主要流程，我們將以 How many games did the Yankee play in July ? 為例，解釋每一個步驟的動

作：

1. Question Read in : How many games did the Yankee play in July ?

2. Dictionary Look-up

Word	Semantic
How many	Adj : Number_of = ?
games	Game = (blank)
Did	No meaning
The	No meaning
Yankee	Team = Yankee
Play	No meaning
In	No meaning
July	Month = July

3. Syntax (形成短語 phrase，以便建構出 modifier, 例如上述的 winning)

[How many games] did [the Yankees] play (in [July]) ?

[X] : 代表 noun phrase

(Y) : 代表 adverbial phrase 或 prepositional phrase

4. Content Analysis(根據字典的語意組合正確的規格串列, 例如上述的 Boston 到底是那個意義呢?)

Phrase	Semantic
[How many games]	Game(Number_of) = ?
Did	No meaning
[the Yankees]	Team = Yankee
Play	No meaning
(in July)	Month = July

5. Retrieve records(從資料庫中取出記錄)

Place	Month	Day	Game	Winner/Score	Loser/Score
Boston	July	7	96	Red Sox /5	Yankees / 3
Boston	July	15	98	Yankees/7	Red Sox/4

6. Response (直接顯示擷取出的紀錄)

Game(Number_of) = ? => count(Game) = ?

回答： Two Games .

8.3. 機率統計法

8.3.1. 學習未知詞

在英文當中，由於有空白與標點當作詞彙分界線，所以那些不在字典當中的詞彙，很容易被辨認出來，視為未知詞，只是該詞彙的語義無法被界定而已。

但是在中文當中，沒有空白作為分界，因此「哪些字串算是詞彙，哪些不是詞彙？」就成了個問題，對於這個問題，可以使用統計的方法，得到不錯的處理結果。

簡立峰等人曾提出可使用 **PatTree** 計算詞彙左右自由度，以便抽取未知詞的方法¹⁹。該方法在詞彙出現很多次的情況，通常表現得很好，但是對於詞彙只出現一次的情況，就無能為力了，這也是統計方法通常會遭遇到的問題，統計樣本不夠的時候，統計就失去了效力。

使用左右自由度判斷詞彙邊界的方法，直覺上其實很簡單，首先讓我們看看一些語句

◦ 老張牛肉麵很好吃，	◦ 我們昨晚去吃牛肉麵。	◦ 媽媽買了一斤牛肉，用來煮牛肉湯。
◦ 牛肉麵是中國美食。	◦ 老李得了台北牛肉麵節的首獎。	◦ 今晚的牛肉湯很好喝。
樓下的老張牛肉麵遠近馳名	我們昨晚去老張牛肉麵吃晚餐。	一起去吃老張牛肉麵吧。

於是我們可以發現「牛肉麵、牛肉、牛肉麵節、張牛、老張牛肉麵、張牛肉麵」等字串的左右接情況分別如下。

19 Chien, Lee-Feng "PAT Tree-Based Keyword Extraction for Chinese Information Retrieval," The ACM SIGIR Conference, Philadelphia, USA, 1997, pp. 50-58. – <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.102.2087>

詞彙	前後文	左接詞彙/次數	右接詞彙/次數	判斷
牛肉麵	張牛肉麵很 吃牛肉麵。 。牛肉麵是 北牛肉麵節 張牛肉麵遠 張牛肉麵吃 張牛肉麵吧	張:4, 吃:1, 。:1, 北:1	很:1, 。:1, 是:1, 節:1, 遠:1, 吃:1, 吧:1	左右自由度都夠大，應該是詞彙
牛肉	張牛肉麵很 吃牛肉麵。 。牛肉麵是 北牛肉麵節 煮牛肉湯。 的牛肉湯很 張牛肉麵遠 張牛肉麵吃 張牛肉麵吧	張:4, 吃:1, 。:1, 北:1, 煮:1, 的:1	麵:7, 是:1, 湯:2	左右自由度都夠大，應該是詞彙
牛肉麵節	北牛肉麵節的	北:1	的:1	只出現一次，無法判斷是否為詞彙
張牛	老張牛肉 :4	老:4	肉:4	左右都不自由，應該不是詞彙

老張牛肉麵	◦老張牛肉麵很 的老張牛肉麵遠 去老張牛肉麵吃 吃老張牛肉麵吧	◦:1, 的:1, 去:1, 吃:1	很:1, 遠:1, 吃:1, 吧:1	左右自由度都夠大，應該是詞彙
張牛肉麵	老張牛肉麵很 老張牛肉麵遠 老張牛肉麵吃 老張牛肉麵吧	老:4	很:1, 遠:1, 吃:1, 吧:1	左邊不自由，右邊自由，應該是詞尾，但不是完整的詞彙
老張牛肉	◦老張牛肉麵 的老張牛肉麵 去老張牛肉麵 吃老張牛肉麵	◦:1, 的:1, 去:1, 吃:1	麵:4	左邊自由，右邊不自由，應該是詞首，但不是完整的詞彙

左右自由度的方法，就是用以上的統計，進行左右自由度的計算，以便判斷哪些是詞彙，哪些不是詞彙的方法，這是統計在自然語言處理上的一個典型應用²⁰。

20 本書附錄 A 包含一個以 C# 撰寫的詞彙抽取程式，就是採用這種左右自由度計算的方法實作的，由於 C 語言沒有內建的雜湊表物件，因此作者改用 C# 撰寫。

8.4. 研究資源

自然語言領域可以分享的不是只有程式，包含字典、語料庫、平行語料庫等都是可以分享的資源，目前已經有不少字典資源可以在網路上取得，像是 WordNet 就是一個不錯的詞彙網路，可以供自然語言研究者免費使用，CEDICT 是一個相當好的漢英雙語詞典。但是在中文領域，平行語料庫的資源就相對稀少，還好有些網路資源可以彌補這個現象。舉例而言，我們可以使用維基百科作為平行語料庫的基礎，因為維基百科當中有許多文章是從英文被翻譯成中文的，只要利用程式將這些語句進行對齊動作，就可以得到大量的平行語料庫，進而進行統計式翻譯系統的建構。

筆者衷心的希望自然語言的研究人員，能進一步的將手上的語料庫釋放出來，讓大家能共享這些資源，以加快學術的進步速度，讓自然語言處理的研究進行的更為順利。

8.5. 習題

1. 您有沒有辦法寫一個程式理解小學數學的句子，然後自動計算出答案呢？

以下是一些小學數學習題的範例。

小珍有 5 顆蘋果，吃掉 2 顆，剩下幾顆？

9 根骨頭，分給小花 3 根，還有幾根？

5 個人排隊等公車，又來了 3 個人，一共有幾個人在排隊等公車？

第9章 機器翻譯

9.1. 簡介

機器翻譯是自然語言處理的問題當中，最受注目的問題之一。這個問題的解決可以大大的促進知識的流通。最直接的好處是讓英文不好的人也可以看任何的英文書或文章。人類將不再需要花費這麼多的腦力進行翻譯工作，資訊的傳播將又即時又精準。

除此之外，只要解決了機器翻譯問題，許多價值頗高的衍生應用就會出現。例如，搭配語音辨識、機器翻譯與語音合成三項技術，就可以作出同步口譯軟體。

然而，機器翻譯卻又是極為困難的一個主題，牽涉到自然語言理解這個超難的領域。我之所以稱自然語言理解為超難的領域，是因為自己曾經夢想著能在此領域作出重要的貢獻，投入了五年的光陰。但是、五年的光陰卻沒有留下甚麼成績，只體會到一點 - 『這不是當今科學技術所能解決的問題』。除非，在可見的未來，有重大的技術突破，否則，這個領域可能到筆者生命結束都沒有辦法達到實用的地步。

即便筆者採取了比較悲觀的角度，然而，從網際網路出現以來，網路上的大量語料成了機器翻譯技術

的良好實驗場所。筆者便相當希望投入時間，將維基百科上的雙語語料抽取出來，以便供自己與其他人進行自然語言實驗時，能有比較的基準。網路上的資料成了機器翻譯技術的踏腳石，或許，這會幫助我們克服資料來源上的困難，讓技術突飛猛進也說不定。

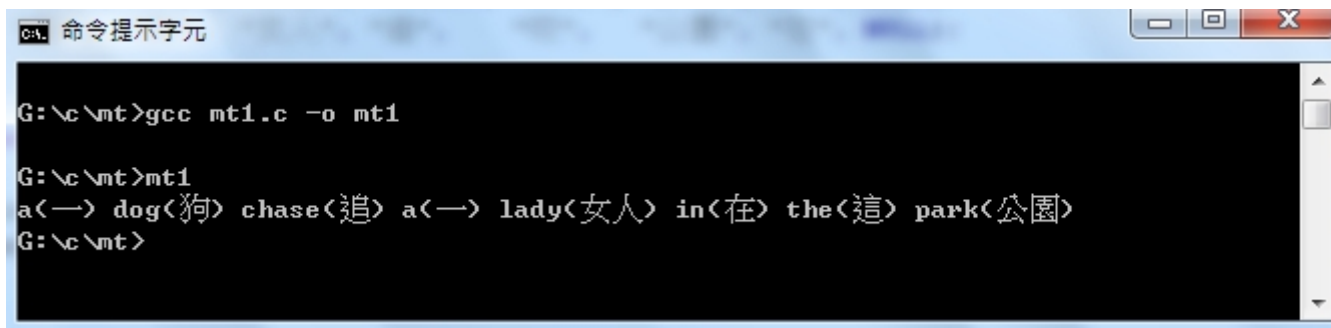
更重要的是，超大型軟體公司，像是 Google 與微軟，競相投入機器翻譯的研究當中，尤其是 Google，更是視機器翻譯為公司進一步擴張的祕密武器。這些資本雄厚的公司投入其中，或許也會帶來一些進步。

如我們在前言當中所敘述的，人工智慧問題可以說是資訊科學界的聖杯 (或者說是魔戒)，而機器翻譯技術可說是人工智慧問題當中最有價值的問題之一可以說是聖杯頂上的寶石。這麼珍貴的東西，值得花一輩子去追尋，不是嗎？讓我們邁向偉大的航道吧。

針對機器翻譯問題，有很多種不同的作法，規則比對式的方法、統計式的方法、範例導向式的方法等等。這些方法各有個的好處，卻也都有各自的盲點。

要做出很棒的機器翻譯系統卻很難，但是，要做出簡單的機器翻譯系統卻並不難，以下是筆者寫的一個簡易翻譯系統。

執行結果



```
G:\c\mt>gcc mt1.c -o mt1

G:\c\mt>mt1
a(一) dog(狗) chase(追) a(一) lady(女人) in(在) the(這) park(公園)

G:\c\mt>
```

程式：mt1.c

```
#include <stdio.h>
#include <string.h>
#define NIL -1
char *ewords[] = {"a", "the", "cat", "dog", "lady", "chase", "bite", "park", "in", NULL};
char *cwords[] = {"一", "這", "貓", "狗", "女人", "追", "咬", "公園", "在", NULL};

// 尋找詞典中是否有這個詞彙
int wordFind(char *words[], char *word) {
    int i;
    for (i=0; words[i]!=NULL; i++) {
```

```

        if (strncmp(word, words[i], strlen(words[i]))==0)
            return i;
    }
    return NIL;
}

void mt(char *s, char *swords[], char *twords[]) {
    char *spliter=",:. ";
    char *sword = strtok(s, spliter);
    while (sword != NULL) {
        int si = wordFind(swords, sword);
        if (si!=NIL) {
            printf("%s(%s) ", swords[si], twords[si]);
        }
        sword = strtok(NULL, spliter);
    }
}

int main(int argc, char * argv[]) {
    char english[] = "a dog chase a lady in the park";
    mt(english, ewords, cwords);
}

```

在以上的翻譯系統當中，我們僅僅簡單的進行詞彙翻譯動作，因此『a dog chase a lady in the park』翻譯出來的結果是『a(一) dog(狗) chase(追) a(一) lady(女人) in(在) the(這) park(公園)』。

如果要製作更好的翻譯系統，就必須考慮語法結構，也應該去處理中文的量詞問題，這就必須依靠翻譯規則與語法剖析才做得到了。

9.2. 規則式翻譯

規則式翻譯就是依靠是語法規則以及語意規則進行翻譯的方法，在前述章節當中，我們已經說明了語法規則的形式，不論是生成語法或格狀語法的規則，都可以用來剖析自然語言的語句。一但剖析完成之後，就可以透過樹狀結構之間的轉換，將樹狀結構轉成目標語言的語法，輸出目標語句。

要利用規則比對的方式進行翻譯，首先必須利用語法規則進行語法剖析，將英文語句轉換成語法樹，以下是一組簡單的語法規則，其中前三條同時適用於中文與英文，第四條則是英文的動詞片語規則，第五條則是中文的動詞片語規則，這兩台在中文與英文的語法上有所差異，因此，分成兩種語文列出。

中文與英文的語法規則

規則	適用語文	說明
S=>NP VP	中英	句子=名詞片語 動詞片語

NP => Det Adj* N PP*	中英	名詞片語=定詞 名詞
PP => P NP	中英	介係詞片語=副詞 名詞片語
VP => V (NP PP*)	英文	動詞片語=動詞 名詞片語 介係詞片語
VP => V (PP*的 NP)	中文	動詞片語=動詞 介係詞片語 名詞片語

舉例而言，針對『I like the girl in red dressing』這個句子，其語法樹可能如下圖所示。

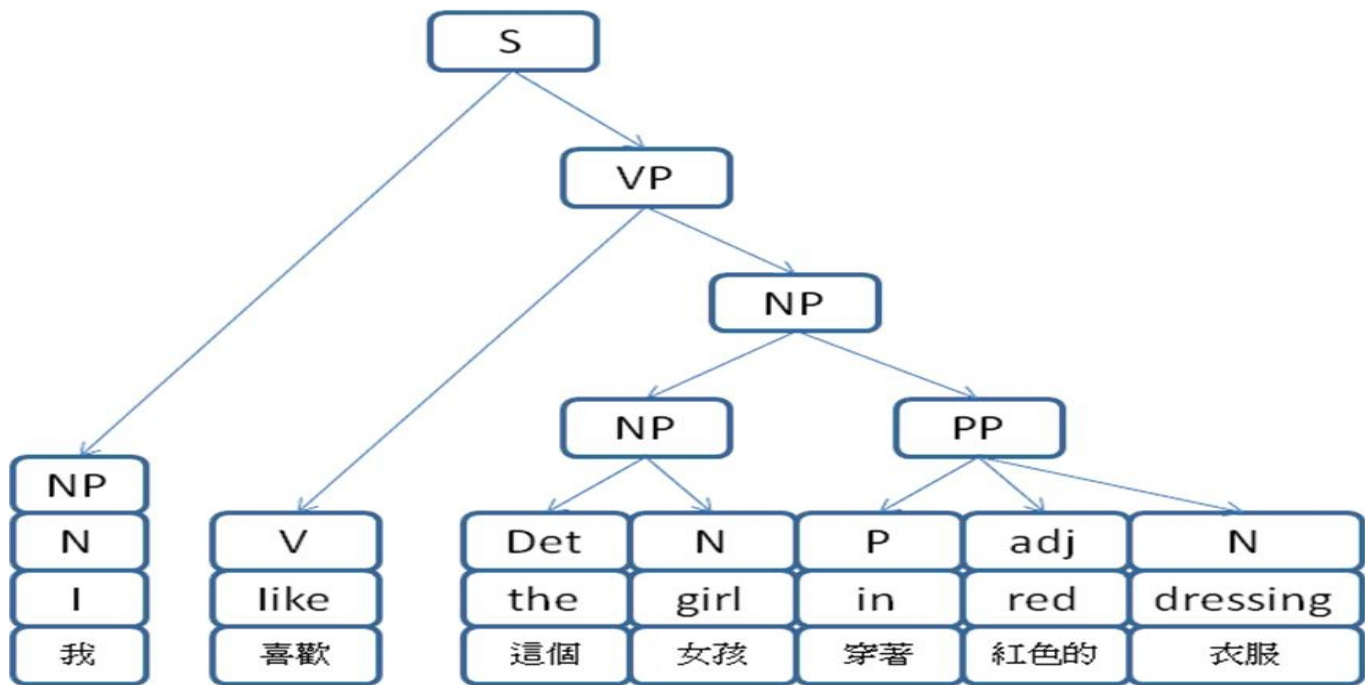


插圖 1: 語句 *I like the girl in red dressing* 的剖析樹

其實，不論語句的剖析樹是否得以完全建立，我們都可以透過語法規則對翻譯語句的好壞進行加減分的動作，辦法是對每一規則都給予一個權重。然後，當規則比對成功時，就進行加減分的動作。當加

減的分數確定之後，就能根據這些分數，進行最佳化的計算，以便找出所有被列舉出的語句當中，最好的翻譯語句。

一旦剖析動作完成之後，就可以利用中文的語法規則，進行結構上的調整，以便輸出。例如，在圖一的剖析樹當中，由於英文動詞片語上的規則 $VP \Rightarrow V (NP PP^*)$ ，在中文上的對應規則為 $VP \Rightarrow V (PP^* \text{ 的 } NP)$ 。因此，生成中文語句時，必須先經過中文語法的調整，這個調整使得圖一當中的 NP 與 VP 會先反轉後再輸出，於是從『NP(這個女孩) PP(穿著紅色的衣服)』轉換為『PP(穿著紅色的衣服) 的 NP (這個女孩)』。於是，『I like the girl in red dressing』這個句子就被翻譯成了『我喜歡穿著紅色的衣服的這個女孩』。

當然，這句話在中文上仍有些小問題，太多『的』這個字在中文裡是不好的用法。因此，必須再經過修飾規則才能將結果翻譯得更好，像是變成『我喜歡穿著紅色衣服的這個女孩』。但是，這距離較好的翻譯，像是『我喜歡這個穿著紅衣服的女孩』，仍有一段距離。這有可能靠著更精細的語法規則達成。

以往，採用規則比對方式的人，可能沒有使用任何的統計資訊，於是採用人為給定規則權重的方式，以進行剖析或詞性標記等動作。然而，由於權重是人為給定的，這使得每條規則的影響力落入主觀的判斷之中，難以科學化。這是規則式方法的主要問題所在，如果能利用統計資訊改進這樣的問題，就能較為客觀。另外，語料庫當中的資訊暨大量又多樣，可以完成某些規則式方法所難以達到的工作，

這也是我們接下來要看的主題，統計式的機器翻譯方式。

直接處理翻譯問題通常太過困難，其結果也難以評判好壞。因此，通常會將機器翻譯問題進一步細分成幾個子問題，分別對這些問題進行研究。

9.2.1. 解歧義問題 (Word Sense Disambiguation)

解歧義 (Word Sense Disambiguation) 問題乃是為了解決一字多義現象而進行的研究。所為一字多義現象，就是翻譯文章中的一個字詞，在字典當中卻有數個對應的意義，導致翻譯時難以抉擇的問題。

舉例而言，像是英文的 Free 一詞，就對應到中文的『自由的、免費的、未被占據的、有空閒時間的、鬆開的、慷慨的、優美的、放肆的』這些形容詞，甚至還有副詞與動詞的用途未列出。那麼，程式如何決定『No free lunch』的 free 到底是指哪一個意義呢？為何 free 在該句中被人類解釋為免費的，而『Free Culture』中的 free 卻被解釋為自由的呢？

9.2.2. 語法剖析的問題 (Parsing)

Parsing 問題通常是從 Norm Chomsky 的語言學理論開始的。由於 Chomsky 的功能語法學派提出的樹狀語法結構理論相當成功，因此，若能將一個句子轉換成樹狀結構，對翻譯的進行將會有所幫助，而語法剖析的目的正是將輸入語言 (例如英文) 轉換成一顆語法樹的動作。

9.2.3. 樹狀結構轉換的問題 (Structure Transformation)

由於中文與英文在語法結構上有相當大的不同。因此，翻譯的難度比拉丁語系的語文之間互轉更為困難。舉例而言，如果我們想將『I don't like the woman in red』這句話，翻譯成中文，如果直接翻譯成『我不喜歡這個女人在紅色』，我想應該沒有人知道在講甚麼。比較適當的翻譯是『我不喜歡那個穿著紅色衣服的女人』。但是，這個翻譯有一些結構上的轉換，電腦如何能正確做到呢？

9.2.4. 目標語句合成的問題 (Sentence Generation)

當原始句子被剖析後，成為樹狀結構，之後再利用結構轉換形成目標語意樹之後，如何能將此語意樹轉換成目標語句，例如，如何將轉為中文的語意樹平坦化，轉換成中文語句，便是英翻中問題的最後一道步驟。

另外，還有詞型變化、語境分析等問題，由於篇幅的關係，在此就不詳述了。

9.2.5. 採用規則的翻譯程式 (C 語言實作)

為了更清楚的說明規則比對的方法，筆者設計了一組更簡化的規則，並利用這組規則撰寫了一個簡易的翻譯系統，並用「a dog chase a lady in the park」這句話來進行翻譯測試，翻譯的結果如下所示：

原文：a dog chase a lady in the park

翻譯：a(一)[隻]dog(狗)in(在)the(這)[個]park(公園)chase(追)a(一)[個]lady(女人)

程式碼：mt2.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

#define NODES 5
#define NIL -1
#define eq(a,b) (strcmp(a,b)==0)

char *text;
int ti = 0;
char empty[] = "";

typedef struct _Node {
    char *type;
    char *value;
```

```
    struct _Node *childs[NODES];  
    int childCount;  
} Node;
```

```
Node nodes[1000];  
int nodeTop = 0;
```

```
Node *stack[100];  
int stackTop = 0;
```

```
char space[512];
```

```
char* spaces(int n) {  
    memset(space, ' ', n);  
    space[n] = '\0';  
    return space;  
}
```

```
Node *nodeNew(char *type) {  
    Node *node = &nodes[nodeTop++];  
    node->type = strdup(type);  
    node->value = empty;  
    node->childCount = 0;  
    memset(node->childs, 0, sizeof(Node*)*NODES);  
    return node;
```

```
}
```

```
Node *push(char *type) {  
    printf("%s+%s\n", spaces(stackTop), type);  
    Node *parent = NULL;  
    Node *node = nodeNew(type);  
    if (stackTop >= 1) {  
        parent = stack[stackTop-1];  
        parent->childs[parent->childCount++] = node;  
    }  
    stack[stackTop++] = node;  
    return node;  
}
```

```
Node* pop(char *type) {  
    stackTop--;  
    Node* node = stack[stackTop];  
    assert(strcmp(node->type, type)==0);  
    printf("%s-%s\n", spaces(stackTop), type);  
    return node;  
}
```

```
int level=0;
```

```
Node* tree(Node *node, int level) {
```

```

printf("%s+%s:%s\n", spaces(level), node->type, node->value);
int i=0;
for (i=0; i<node->childCount; i++)
    tree(node->childs[i], level+1);
printf("%s-%s\n", spaces(level), node->type);
}

```

```

// === BNF Grammar =====

```

```

// S = NP VP PP

```

```

// NP = D N

```

```

// VP = V NP

```

```

Node* match(char *name, char *words) {
    Node *node = push(name);
    char token[100], word[100];
    if (sscanf(&text[ti], "%[a-z] ", token)>0) {
        ti += strlen(token);
        printf("%s%s\n", spaces(stackTop), token);
        node->value = strdup(token);
        sprintf(word, "%s", token);
        assert(strstr(words, word)>=0);
        sscanf(&text[ti], "%[ ]", token);
        ti += strlen(token);
    }
    pop(name);
}

```

```
}
```

```
Node* D() { return match("D", "|the|a|"); }
```

```
Node* N() { return match("N", "|dog|cat|lady|"); }
```

```
Node* V() { return match("V", "|chase|eat|"); }
```

```
Node* P() { return match("P", "|in|on|at|of|"); }
```

```
Node* NP() { push("NP"); D(); N(); return pop("NP"); }
```

```
Node* VP() { push("VP"); V(); NP(); return pop("VP"); }
```

```
Node* PP() { push("PP"); P(); NP(); return pop("PP"); }
```

```
Node* S() { push("S"); NP(); VP(); PP(); return pop("S"); }
```

```
// === BNF Grammar =====
```

```
// 英文語法                      中文語法
```

```
// S = NP VP PP    S = NP PP VP
```

```
// NP = D N                      NP = D Q N
```

```
// VP = V NP                      VP = V NP
```

```
char *animal[] = {"dog", "cat", NULL};
```

```
char *people[] = {"lady", "man", NULL};
char *ewords[] = {"a", "the", "cat", "dog", "lady", "chase", "bite", "park", "in", NULL};
char *cwords[] = {"一", "這", "貓", "狗", "女人", "追", "咬", "公園", "在", NULL};
```

// 尋找詞典中是否有這個詞彙

```
int wordFind(char *words[], char *word) {
    int i;
    for (i=0; words[i]!=NULL; i++) {
        if (strcmp(word, words[i])==0)
            return i;
    }
    return NIL;
}
```

```
void gen(Node *node) {
    if (node->value != empty) {
        int ei = wordFind(ewords, node->value);
        printf("%s(%s)", node->value, cwords[ei]);
    }
    if (eq(node->type, "S")) {
        Node *np=node->childs[0];
        Node *vp=node->childs[1];
        Node *pp=node->childs[2];
        gen(np);
```



```

    gen(pp);
    gen(vp);
} else if (eq(node->type, "NP")) {
    Node *d = node->childs[0];
    Node *n = node->childs[1];
    char *q;
    if (wordFind(animal, n->value)!=NIL)
        q = "隻";
    else
        q = "個";
    gen(d);
    printf("[%s]", q);
    gen(n);
} else {
    int i;
    for (i=0; i<node->childCount; i++)
        gen(node->childs[i]);
}
}

void mt(char *str) {
    text = str;
    printf("===== parse =====\n");
    Node *s = S();
    printf("===== tree =====\n");
}

```

```
tree(s, 0);  
printf("===== gen =====\n");  
gen(s);  
printf("\n");  
}  
  
int main(int argc, char * argv[]) {  
    mt("a dog chase a lady in the park");  
}
```

程式執行畫面如下圖所示：


```
===== tree =====
```

```
+S:
```

```
+NP:
```

```
+D:a
```

```
-D
```

```
+N:dog
```

```
-N
```

```
-NP
```

```
+UP:
```

```
+U:chase
```

```
-U
```

```
+NP:
```

```
+D:a
```

```
-D
```

```
+N:lady
```

```
-N
```

```
-NP
```

```
-UP
```

```
+PP:
```

```
+P:in
```

```
-P
```

```
+NP:
```

```
+D:the
```

```
-D
```

```
+N:park
```

```
-N
```

```
-NP
```

```
-PP
```

```
-S
```

```
===== gen =====
```

```
a<一>[隻]dog<狗>in<在>the<這>[個]park<公園>chase<追>a<一>[個]lady<女人>
```

```
G:\c\mt>
```

9.3. 統計式翻譯

自然語言的規則，通常很難像程式語言一樣能百分之百符合語法，因此是很難掌握的。

在機器翻譯的領域當中，研究方法通常可以分為兩類，規則比對法與語料庫統計法兩種。在網際網路出現之前，由於語料庫的缺乏，許多人採用規則比對法進行研究。然而、在網際網路出現之後，網路上的大量語料，使得語料庫的方法出現了相當重要的成果。統計式翻譯儼然成為機器翻譯的翻譯的新主流。當然，規則法與統計法兩者結合，可能會產生更好的結果，這或許是未來待解決的問題之一。

然而，網路上的資料通常是沒有經過處理的語料，要如何處理這些與料，使這些語料變成統計式語言學可用的語料庫，也變成了一個重要的問題。這個問題衍生出許多不同層次的問題。

首先，在網路上有許多的文章，其中包含了許多翻譯文章，這些翻譯文章往往可以在網路上找到原文。譯文與原文之間如果能很好的對應起來，就可以形成所謂的雙語語料庫。雙語語料庫是統計式翻譯上相當重要的指導老師，這個指導老師可以讓統計式的翻譯程式變強。因此，誰能建構出大量且高品質的雙語語料庫，誰就掌握了統計式翻譯的優勢。

然而，高品質的雙語語料庫，並不容易建構。一個好的雙語語料庫，至少要有三個層次的對應資訊。第一個層次是文章的對應，也就是提供譯文與原文文章間的對應關係。第二個層次是句子的對應，也就是每個句子要和其譯文對應起來。第三個層次是詞彙的對應，也就是每個詞彙必須要與其翻譯詞彙

對應起來。

除此之外，最高品質的雙語語料庫還應包含每個詞彙的詞性標記 (Tagging)，如果連語法的樹狀結構都被標記出來，那就會形成樹狀語料庫 (Tree Bank)。樹狀語料庫可以說是統計式翻譯程式的最佳指導老師，但也是最難以建立的。

9.3.1. 雙語文章的建構問題 (Corpus Construction)

要建構出雙語語料庫，在文章層次上，首先要能找出哪些是翻譯文章，然後找出其對應的原始文章。因此，如果有一個程式能夠自動建構出這樣的對應關係，而且正確又快速的話，那將能在極短的時間內建構出大量的雙語文章庫。這個問題稱為雙語文章的建構問題。

9.3.2. 雙語語句的對齊問題 (Sentence Alignment)

然而，即使有了大量的雙語文章庫，句子與句子之間的對應關係仍然沒有被關連起來。因此，需要有一個程式自動去建立句子間的關係，這個問題稱為雙語語句的對齊問題。

9.3.3. 雙語詞彙的對齊問題 (Word Alignment)

同樣的，有了句子之間的對齊關係之後，仍然需要再次去對齊詞彙語詞彙之間的關係，這個問題就被

稱為雙語詞彙的對齊問題。

然而，機器翻譯上的問題，通常是無法完全區分開的。例如，語句的對應關係如果出了錯誤，詞彙的對應關係就無法正確建構。而現今並沒有能在語句上對齊得很好的方法，正確率能到達 90% 就已經算很好了。而且，這三個層次往往會互相影響，所以很難單獨解決。

同樣的，解歧義、剖析、結構轉換與語句合成的問題，在統計式翻譯上也是互相影響的。因此，有必要形成一個整體性的理論，讓這些資訊的統計互相回饋影響，以整體的考量方式進行最佳化的計算。這或許是統計式翻譯接下來應該要走的道路。

9.3.4. 優化問題 (Optimization)

機器翻譯乃是將一個語言的文章翻譯成另一個語言的文章的問題，其中的來源語言 (source language，簡寫為 *s*) 與目標語言 (target language，簡寫為 *t*) 之間，具有意義上相等的關係 (或者說非常相近)。因此，我們可以將機器翻譯的問題，寫成如方程式 1 所描述的一個抽象過程。這個方程式相當的重要，可以說是機器翻譯的第一定律。

$$bts = mt(s) = \arg \max_t \{score(t|s)\} \quad (\text{Equation 9.1})$$

在上述公式中，*bts* 代表最佳的翻譯語句 (Best Target Sentence)，而 *mt* 則是該機器翻譯程式，可以將

任何輸入的來源語句 s 翻譯成最佳目標語句 bts 。這個過程，乃是經由一個列舉程序，列出所有可能的翻譯語句 $\{t_1, t_2, \dots, t_n\}$ ，然後，從中選取一個最佳的句子，作為 mt 程序所輸出的最佳翻譯語句 bts 。

舉例而言，假如我們想寫出一個將英文翻譯為中文的程式，則來源語言 s 為英文句子 s ，我們會根據這個英文句子找出可能的中文翻譯句 $\{t_1, t_2, \dots, t_n\}$ 。然後，利用評估函數 $score(t|s)$ 評判以 t 作為 s 之翻譯的適當性。最後，選取出最適當的語句 bts 作為 $mt(s)$ 程式的輸出，如此就完成了整個語句翻譯的過程。

或許對許多讀者而言，上述的說明仍然太過抽象，讓我們以一個範例說明此種過程。假如我們希望將英文句子『I like the girl in red dressing』翻譯成中文，首先，透過字典，我們可以查出其中每一個英文詞彙的中文對照詞彙，例如 like 可能對應『愛、喜歡、像、相似』等詞彙，下圖顯示了該語句中每個英文詞對應的候選中文詞的情況。

I	like	the	girl	in	red	dressing
我:0.8	愛: 0.3	這個:0.3	女孩:0.4	在:0.3	紅色:0.3	打扮:0.2
一:0.1	喜歡:0.2	那個:0.2	女兒:0.2	於:0.2	紅色的:0.2	服飾:0.15
...	像:0.1	...	母的:0.1	朝向:0.1	赤字:0.05	衣服:0.1
	相似:0.05		...	穿著:0.05	...	穿衣:0.08
	...			戴著:0.04		梳理:0.07
				...		填料:0.03
						敷藥:0.01
						...

透過適當的選擇，我們可以選出每個詞彙的最佳翻譯詞彙，如下圖所示。這個句子就是該語句的中文翻譯結果。

I	like	the	girl	in	red	dressing
我	喜歡	這個	女孩	穿著	紅色的	衣服

插圖 2: 選出的最佳翻譯詞彙

慢著，仔細看過這個翻譯的人應該會發現，翻譯的結果與原文的意義不同。正確的翻譯應該是『我喜歡這個穿著紅色衣服的女孩』，而非『我喜歡這個女孩穿著紅色的衣服』。這是怎麼回事。

這個問題的原因是，中文與英文有著不同的語法結構，因此，直接將詞彙翻譯後的結果，往往會形成語法不通，或者是語意錯誤的情形。

因此，我們可以在統計的過程當中加入規則的對應方法，然後同樣利用優化的方式選取較好的對應配對，以便完成整個統計翻譯的過程。

但是，即便如此，統計式翻譯到目前為止，大約就只能作到像 Google 翻譯那樣的水準，還沒辦法翻譯得很好！

關於統計式翻譯的方法，筆者還沒有學得很透徹，就不再進一步勉強述說，有興趣的讀者可以看看相關的參考文獻，像是大陸的「宗成慶」教授就有寫比較完整的投影片，很值得參考^{21 22}。

9.4. 結語

機器翻譯技術可初略的分為統計式與規則式兩類，早期的研究者較注重規則式的翻譯方法，但是由於網際網路上的語料越來越多，目前統計式的方法比較受到重視。但是，兩者都有相當大的困難，未來兩者也可能出現融合的情況，但是如何能有效的融合仍然等待新一代的研究者之投入。

許多大型軟體公司 — 像是 Google 與微軟，都設立了研發部門，特別針對機器翻譯進行研究，因為這個市場具有龐大的商業潛力。Google 挖角優秀的機器翻譯研究人員 Franz Josef Och 的原因也正是為了這個龐大的潛在市場，但是即便 Och 在機器翻譯領域的表現是如此傑出，但是 Google 恐怕還是太過輕視這個問題的困難度，以至於曾經發有新聞報導說 Google 認為機器翻譯技術可以在 5 年內 (2014 年時) 達到人類翻譯的水準，這是筆者認為幾乎不可能的。

21 《自然語言理解》— <http://www.nlpr.ia.ac.cn/cip/ZongReportandLecture/ReportandLectureIndex.htm>

22 第二讲 - 统计机器翻译 — http://www.nlpr.ia.ac.cn/cip/ZongReportandLecture/Lecture_on_NLP/Chp-11.2.pdf

以往對機器翻譯的研究較著重於理論的描述，近來則在統計式翻譯的領導之下，許多程式能力優秀的研究者開始嶄露頭角，Och 就是一個程式能力很強的研究者，因此才會創作出像 Giza++、mkcls、YASMET 等機器翻譯的相關子系統。

機器翻譯領域已經受到開放原始碼運動的影響，開始在程式上進行交流分享的動作。因此像 Giza++²³ 等系統後來也成為完整的統計式翻譯軟體 Moses²⁴ 的建構基礎，筆者相信這對機器翻譯的學術研究會有相當正面的影響。

本書採用了一個與一般「自然語言處理」的書及稍有不同的角度切入，並且取材偏向於簡單易於實作的方式，以便讓程式人可以輕易的看懂這些程式，這種方式可以比較快速的帶領具有程式能力的讀者快速的進入「語言處理」的領域，而非只是在理論上打轉。但是這種方式也有一些缺陷，例如本書就很少闡述較為研究性的主題，這讓本書的實作性較強，理論性較弱。這樣的安排比較適合程式人入門，但對於想要進一步研究「自然語言處理」的朋友而言，可能就必須再閱讀其他書籍，以便能學習更進一步的主題。

23 Giza++ : <http://code.google.com/p/giza-pp/>

24 Moses : <http://www.statmt.org/moses/>

附錄 A. 中文詞彙學習程式 – C#

執行結果

```
G:\csharp\WordLearner>csc WordLearner.cs  
Microsoft (R) Visual C# 2010 編譯器版本 4.0.30319.1  
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
G:\csharp\WordLearner>WordLearner Story.txt
```

4 (來的) 3

5 (的鈔票) 5

11 (鈔票) 16

2 (是一個) 2

4 (一個) 5

3 (信仰) 2

3 (的國) 3

4 (國家) 5
4 (人們) 7
2 (打從心) 2
2 (心裡) 2
2 (相信) 2
...
...

C# 原始程式: WordLearner.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;

class CountMap : Dictionary<String, int>
{
    public void add(String str)
    {
        if (ContainsKey(str))
            this[str]++;
        else
```

```

        this.Add(str, 1);
    }
}

class WordLearner
{
    public static String ChineseSymbols = "。、\n: ? ! 「 」 『 』 —— ( ) [ ] ……~~~~~____ • ·";

    public CountMap countMap = new CountMap();

    public static void Main(String[] args)
    {
        String corpus = fileToText(args[0]);
        WordLearner learner = new WordLearner();
        learner.run(corpus);
        List<String> words = learner.getWordList();
        print(words);
    }

    public static void print(List<String> words)
    {
        foreach (String word in words)
            Console.WriteLine(word);
    }

    public static String fileToText(String filePath)
    {

```

```

    StreamReader file = new StreamReader(filePath);
    String text = file.ReadToEnd();
    file.Close();
    return text;
}

public WordLearner() {}

public void run(String corpus)
{
    count(corpus, countMap);
}

public void count(String text, CountMap countMap)
{
    for (int i = 0; i < text.Length; i++)
    {
        String substr = text.Substring(i, Math.Min(text.Length - i, 6));
        for (int len = 2; len < substr.Length; len++)
        {
            char ch = substr[len-1];
            String head = substr.Substring(0, len);
            countMap.add(head);
            if (ch >= 0 && ch <= 127 || ChineseSymbols.IndexOf(ch) >= 0)
                break;
        }
    }
}

```



```

}

public List<String> getWordList()
{
    List<String> words = new List<String>();
    CountMap rfreeMap = new CountMap();
    CountMap lfreeMap = new CountMap();
    foreach (String str in countMap.Keys)
    {
        String head = str.Substring(0, str.Length-1);
        String tail = str.Substring(1, str.Length-1);
        rfreeMap.add(head);
        lfreeMap.add(tail);
    }
    foreach (String str in countMap.Keys)
    {
        if (rfreeMap.ContainsKey(str) && lfreeMap.ContainsKey(str))
            if (rfreeMap[str] >= 2 && lfreeMap[str] >= 2)
                words.Add(lfreeMap[str] + "(" + str + ")" + rfreeMap[str]);
    }
    return words;
}
}

```