

DIY Compiler and Linker

# 自己动手写 编译器、链接器

王博俊 张宇 编著

简化的C语言  
(源程序)

SCC编译器

Intel x86机器语言  
(目标程序)

链接器

可直接运行的  
EXE文件

库文件



清华大学出版社

---

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

**备用QQ:2404062482**

DIY Compiler and Linker

---

自己动手写  
编译器、链接器

---

王博俊 张宇 编著

---

清华大学出版社

## 内 容 简 介

本书讲述了一个真实编译器的开发过程,源语言是以 C 语言为蓝本,进行适当简化定义的一门新语言,称之为 SC 语言(简化的 C 语言),目标语言是大家熟悉的 Intel x86 机器语言。在本书中,读者将看到从 SC 语言定义,到 SCC 编译器开发的完整过程。本书介绍的 SCC 编译器,没有借助 Lex 与 Yacc 这些编译器自动生成工具,纯手工编写而成,更便于学习和理解。为了生成可以直接运行 EXE 文件,本书还实现了一个链接器。读完本书读者将知道一门全新的语言如何定义,一个真实的编译器、链接器如何编写。

本书适合各类程序员、程序开发爱好者阅读,也可作为高等院校编译原理课程的实践教材。

**郑重声明:**本书源代码作者已申请版权,仅供读者用于学习研究之目的。未经作者允许,严禁任何组织与个人将其在网络上传播或用于商业用途。对于侵权行为,作者保留提起法律诉讼的权利。源代码相关问题,请与作者联系,作者邮箱:1259809207@qq.com。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

### 图书在版编目(CIP)数据

自己动手写编译器、链接器/王博俊,张宇编著. --北京:清华大学出版社,2015

ISBN 978-7-302-38136-5

I. ①自… II. ①王… ②张… III. ①C 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字(2014)第 227835 号

责任编辑:袁勤勇 徐跃进

封面设计:傅瑞学

责任校对:李建庄

责任印制:李红英

出版发行:清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址:北京清华大学学研大厦 A 座

邮 编:100084

社总机:010-62770175

邮 购:010-62786544

投稿与读者服务:010-62776969, [c-service@tup.tsinghua.edu.cn](mailto:c-service@tup.tsinghua.edu.cn)

质量反馈:010-62772015, [zhiliang@tup.tsinghua.edu.cn](mailto:zhiliang@tup.tsinghua.edu.cn)

印刷者:北京富博印刷有限公司

装订者:北京市密云县京文制本装订厂

经 销:全国新华书店

开 本:185mm×260mm

印 张:22.25

字 数:557 千字

版 次:2015 年 2 月第 1 版

印 次:2015 年 2 月第 1 次印刷

印 数:1~2000

定 价:44.50 元

产品编号:059284-01



# 序 言

因为工作的关系,我经常和各企业的技术负责人交流。话题谈着谈着常常会转到他们目前共同的难题——技术人员招聘。这时不少人都会感慨,中国能做系统软件开发的技术人员太少,这方面的人太难找了。随着中国企业的发展,做系统和平台的需求不断增加,这种供需矛盾将越来越明显。

究其原因,很容易想到的是我们的高校教育、课程设置。美国顶尖大学计算机系基础课程教学里都非常重视项目实践,操作系统课往往要真的开发一个像模像样的操作系统原型,编译器课也真的要自己设计并实现一门有创新性的小语言……

在计算机科学的各门课程中,编译器的设计实践有着特殊的重要性。“龙书”的主要作者、哥伦比亚大学教授 Alfred V. Aho 曾经列举过编译器实践有诸多好处:

- 能让学生领悟到理论与实践的完美结合。比如编译原理所涵盖的正则表达式和自动机,在各种场合的应用是极其广泛的,对正则的掌握程度,从某种意义上讲甚至可以作为技术人员水平的一种尺度。
- 深入探索计算思维的多样性。与人类语言一样,不同类型的编程语言其实代表了不同的思维方式。只用过命令式语言的人可能没有想到,开启了大数据领域的 Map 与 Reduce,其实在函数式语言是一种非常常见的东西。

的确,深入了解编译器和编译原理,对于技术人员更好地理解 and 掌握自己最常用的语言和系统,从而提升自己的内力是有极大好处的。另一方面,随着 DSL(领域特定语言)的流行,需要技术人员开发自己语言的机会也越来越多。

然而,编译原理是计算机科学里公认比较难的一门课。虽然目前国外比较重要的编译理论教材(比如龙书的《编译原理》、虎书《现代编译原理》的 C 语言和 Java 版本、鲸书《高级编译器设计与实现》)基本上都有了中文版和英文影印版,但这些书往往更偏重理论,而且门槛较高,不太适合指导一线技术人员实践和自学。我认识的一位美籍华人技术专家 Ronald Mak 在 Wiley 出版过一本基于 Java 的“Writing Compilers and Interpreters”,比较贴近实践,但部头较大,而且没有看到中文版。

偶然的的机会,我得知王博俊在工作之余,写了一本以简化的 C 语言为例子讲述编译器和链接器实践的书。浏览了初稿之后,感觉全书内容简明,容易上手,又不失全面和系统,正好弥补了这方面的空白。特向大家推荐。

# 自序

纸上得来终觉浅，绝知此事要躬行。

——陆游

编译原理与技术的一整套理论在整个计算机科学领域占有相当重要的地位，学习它对程序设计人员有很大的帮助。我们考究历史会发现那些人人称颂的程序设计大师都是编译领域的高手，像写出 BASIC 语言的比尔·盖茨，Sun 公司的 Java 之父等，在编译领域都有很深的造诣。曾经在世界首富宝座上稳坐多年的比尔·盖茨也是从给微机编写 BASIC 语言编译器起家的，也正是这个 BASIC 编译器为比尔·盖茨和保罗·艾伦的微软帝国奠定了基础。这个编写 BASIC 语言编译器的经历，开启了比尔·盖茨的辉煌职业生涯。

编译器是一种相当复杂的程序，编写甚至读懂这样的一个程序都非易事，大多数的计算机科学家和专业人员也从来没有编写过一个完整的编译器。但是，几乎所有形式的计算都要用到编译器，而且任何一个与计算机打交道的专业人员都应掌握编译器的基本结构和操作。除此之外，计算机应用程序中经常遇到的一个任务就是命令解释程序和界面程序的开发，这比编译器要小，但使用的却是相同的技术。因此，掌握这一技术具有非常大的实际意义。

李国杰院士说：“随着微处理器技术的飞速发展，处理器性能在很大程度上取决于编译器的质量，编译技术成为计算机的核心技术，地位变得越来越重要。我国要发展自己的微处理器事业，必然要有自己的编译技术作为后盾。”

回过头来说一说是什么样的原因使我萌生了写这样一本书的想法。作者学习其他计算机课程感觉没有特别难懂的，唯独看编译原理的教材，看完了云里雾里的，感觉一知半解，我感觉可能是学的教材过于理论化，于是到书店把所有跟编译原理有关的书籍统统买回家，当然这也包括大家公认的编译原理三大经典书籍（龙书、虎书、鲸书）在内，每一本我都从头到尾翻一遍，好像什么都懂了，又感觉要真的自己动手写个编译器仍然是只有大师才能完成，对自己还是可望而不可即的事情。并且作者也了解到许多关于编译原理实践的悲观论调：“现有的编译器都是用 Lex 和 Yacc 构造的，从头开始手工编写一个完整的编译器几乎是不可能的。”可作者偏偏是那种“明知山有虎，偏向虎山行”的人，要知道早期的编译器可都是纯手工构造的，苦辣酸甜的征程就此开始，可是写个什么语言的编译器？这个编译器怎么定位？这一切都很茫然。

我开始研究编译原理书上的样例，希望能从中找到灵感，给上述问题找到答案。世界著名计算机科学家 N. Worth 编写的 PL/0 语言的编译程序是作者最先研究的编译器，它功能简单、结构清晰、可读性强，被认为是一个非常合适的小型编译程序的学习模型，可这个编译程序不支持数组、结构体、字符串，并且是以假想的栈式机器为例来编写的，而不是直接生成在某种 CPU，某种操作系统环境下直接可以运行的目标语言程序。“PL/0 语言的编译程序”作为编译器的学习模型，也只能算“娃子里面拔将军”，因为没有更好的，也只好将就着用了。至此，编译器定位问题算有了些眉目，作者希望构造一个更适合学习的编译器。

可是,另一个问题接踵而至,为什么那么多开源编译器不能直接用作编译器学习模型呢?我开始研究各个开源编译器的源代码,其中包括 GCC 的源代码,由于 GCC 支持多个前端语言和多种后端机器平台、AST (Abstract Syntax Tree) 和 RTL (Register Transfer Language) 又成了绕不过去的坎,还没学会怎么编写针对一种源语言、一种目标机器的编译器,就要去学习支持多种源语言多个机器平台的编译器,就好比一个婴儿还没学会走路就要学跑,这注定是要跌跟头的。

一面是过于简化的编译器学习模型,另一面是过于复杂的开源编译器,作为学习模型都不太合适。到这里,编译器定位问题算是彻底想清楚了,作者要构造一个教大家如何自己动手写编译器的学习模型。这个模型包括两大部分,第一部分是语言定义,第二部分是这个语言编译器的实现,这个编译器只支持一种源语言,目标语言也只支持一种。这个语言应该具备目前流行的高级语言的最主要特征。这个编译器要结构清晰,代码量要尽可能少,要能体现编写一个实用的编译器的完整过程与技术。这个编译器可以生成在操作系统中直接运行的 exe 文件,只要双击或在命令行执行就能看到结果的那种。

接下来作者开始思考另一个问题,编写个什么语言的编译器?作者研究了目前最流行的几种编程语言 C、C++、C#、Objective-C、Java,其中 C 语言是最简单的了,只有 32 个关键字,但是作者研究发现,C 语言还是有许多冗余的成分,作为学习模型还可以更简单一些。作者最终以 C 语言为蓝本,进行适当简化定义了一门新的语言,仅有 15 个关键字,称为 SC 语言。目标语言选择大家熟悉的 Intel x86 机器语言,编译器命名为 SCC 编译器。

在本书中,读者将看到从 SC 语言定义,到 SCC 编译器开发的完整过程。读完本书你将知道一门全新的语言如何定义,一个真实的编译器如何编写,这些对你来说将不再神秘,编译原理讲的理论与本书中讲述的 SC 语言定义及 SCC 编译器开发过程,是理论联系实际在编译领域的最好阐释。

如本书作为编译原理实践教材,作者建议安排 10 学时讲授。

本书投稿后,有幸请 CSDN 暨《程序员》杂志总编、刘江老师阅读了本书的初稿,并为本书做序,在此向刘老师表示最衷心的感谢。

本书临近出版之际,承蒙清华大学王生原老师阅读了本书终稿,并对书稿做了中肯评价:“本书特色鲜明,内容有深度,文笔也很不错,很值得出版。本书最大的特色是所选的目标平台,即 x86 处理器以及微软系统的 COFF 目标文件格式,这在教材中很少见到,可为国内的编译教学实践提供别具一格的素材。”同时,王老师还对本书提出了宝贵建议。在这里,向王老师表示由衷的敬意和最诚挚的感谢。

我还要感谢我的家人,他们的支持与鼓励是本书得以完成的保障。

要列出所有对本书出版有所帮助的人名是不可能的,因为有些困难是通过互联网解决的,我甚至不知道他们的名字。在此,谨向他们一并表示感谢!

最后,回想本书 6 年的写作历程,愿以蒲松龄的一副对联与读者共勉:

有志者,事竟成,破釜沉舟,百二秦关终属楚;

苦心人,天不负,卧薪尝胆,三千越甲可吞吴。

王博俊

2015 月 1 月

# 目 录

|                        |    |
|------------------------|----|
| 第 1 章 引言               | 1  |
| 1.1 HelloWorld 编译过程分析  | 1  |
| 1.1.1 HelloWorld 程序源文件 | 1  |
| 1.1.2 词法分析             | 2  |
| 1.1.3 语法分析             | 3  |
| 1.1.4 语义分析             | 3  |
| 1.1.5 链接器              | 4  |
| 1.2 SCC 编译器简介          | 7  |
| 1.2.1 SCC 编译器架构        | 7  |
| 1.2.2 SCC 编译器开发环境      | 7  |
| 1.2.3 SCC 编译器运行环境      | 8  |
| 第 2 章 文法知识             | 10 |
| 2.1 语言概述               | 10 |
| 2.2 形式语言               | 11 |
| 2.2.1 字母表和符号串          | 11 |
| 2.2.2 文法与语言的形式定义       | 12 |
| 2.2.3 文法与语言的类型         | 13 |
| 2.2.4 程序设计语言描述工具       | 15 |
| 2.3 词法分析方法             | 16 |
| 2.3.1 词法定义例举           | 17 |
| 2.3.2 状态转换图            | 17 |
| 2.3.3 词法分析程序流程图        | 17 |
| 2.4 语法分析方法             | 18 |
| 2.4.1 LL 分析器           | 18 |
| 2.4.2 $LL(k)$ 文法       | 19 |
| 2.4.3 $LL(1)$ 文法       | 19 |
| 2.4.4 递归子程序法           | 21 |
| 2.4.5 文法的等价变换          | 24 |

|                        |    |
|------------------------|----|
| <b>第 3 章 SC 语言定义</b>   | 26 |
| 3.1 SC 语言的蓝本选择         | 26 |
| 3.1.1 K&R C            | 26 |
| 3.1.2 C89              | 26 |
| 3.1.3 C99              | 27 |
| 3.2 SC 语言对 C89 简化原则    | 27 |
| 3.3 SC 语言的字符集          | 27 |
| 3.3.1 基本字符集            | 28 |
| 3.3.2 扩展字符集            | 28 |
| 3.4 SC 语言词法定义          | 29 |
| 3.4.1 关键字              | 29 |
| 3.4.2 标识符              | 30 |
| 3.4.3 整数常量             | 31 |
| 3.4.4 字符常量             | 31 |
| 3.4.5 字符串常量            | 32 |
| 3.4.6 运算符及分隔符          | 32 |
| 3.4.7 注释               | 33 |
| 3.5 SC 语言语法定义          | 33 |
| 3.5.1 外部定义             | 33 |
| 3.5.2 语句               | 35 |
| 3.5.3 表达式              | 39 |
| 3.6 SC 语言与 C 语言功能对比    | 46 |
| 3.6.1 关键字              | 46 |
| 3.6.2 数据类型             | 46 |
| 3.6.3 存储类型             | 47 |
| 3.6.4 常量               | 47 |
| 3.6.5 变量               | 47 |
| 3.6.6 函数               | 48 |
| 3.6.7 语句               | 48 |
| 3.6.8 表达式              | 50 |
| <b>第 4 章 SC 语言词法分析</b> | 52 |
| 4.1 词法分析任务的官方说法        | 52 |
| 4.2 单词编码               | 53 |
| 4.3 词法分析用到的数据结构        | 55 |
| 4.3.1 动态字符串            | 56 |
| 4.3.2 动态数组             | 58 |
| 4.3.3 哈希表              | 61 |

|                            |           |
|----------------------------|-----------|
| 4.3.4 单词表 .....            | 62        |
| 4.4 错误处理,未雨绸缪 .....        | 67        |
| 4.5 词法分析过程 .....           | 72        |
| 4.5.1 词法分析主程序 .....        | 72        |
| 4.5.2 预处理 .....            | 76        |
| 4.5.3 解析标识符 .....          | 79        |
| 4.5.4 解析整数 .....           | 80        |
| 4.5.5 解析字符串 .....          | 80        |
| 4.5.6 词法分析流程图 .....        | 82        |
| 4.6 词法着色 .....             | 84        |
| 4.7 控制程序 .....             | 85        |
| 4.8 词法分析成果展示 .....         | 86        |
| <b>第5章 SC 语言语法分析 .....</b> | <b>87</b> |
| 5.1 外部定义 .....             | 87        |
| 5.1.1 翻译单元 .....           | 87        |
| 5.1.2 外部声明 .....           | 88        |
| 5.1.3 类型区分符 .....          | 90        |
| 5.1.4 结构区分符 .....          | 92        |
| 5.1.5 函数调用约定 .....         | 95        |
| 5.1.6 结构成员对齐 .....         | 95        |
| 5.1.7 声明符 .....            | 96        |
| 5.1.8 初值符 .....            | 100       |
| 5.2 语句 .....               | 101       |
| 5.2.1 复合语句 .....           | 102       |
| 5.2.2 表达式语句 .....          | 103       |
| 5.2.3 选择语句 .....           | 104       |
| 5.2.4 循环语句 .....           | 104       |
| 5.2.5 跳转语句 .....           | 105       |
| 5.3 表达式 .....              | 107       |
| 5.3.1 赋值表达式 .....          | 108       |
| 5.3.2 相等类表达式 .....         | 109       |
| 5.3.3 关系表达式 .....          | 109       |
| 5.3.4 加减类表达式 .....         | 110       |
| 5.3.5 乘除类表达式 .....         | 111       |
| 5.3.6 一元表达式 .....          | 112       |
| 5.3.7 后缀表达式 .....          | 113       |
| 5.3.8 初值表达式 .....          | 114       |
| 5.4 语法缩进 .....             | 116       |

|              |                     |            |
|--------------|---------------------|------------|
| 5.4.1        | 用到的全局变量及枚举          | 116        |
| 5.4.2        | 语法缩进程序              | 117        |
| 5.5          | 总控程序                | 118        |
| 5.6          | 成果展示                | 119        |
| <b>第 6 章</b> | <b>符号表</b>          | <b>120</b> |
| 6.1          | 符号表简介               | 121        |
| 6.1.1        | 收集符号属性              | 121        |
| 6.1.2        | 语义的合法性检查            | 122        |
| 6.2          | 符号表用到的主要数据结构        | 123        |
| 6.2.1        | 栈结构                 | 123        |
| 6.2.2        | 符号表结构               | 127        |
| 6.2.3        | 数据类型结构              | 132        |
| 6.2.4        | 存储类型                | 133        |
| 6.3          | 符号表的构造过程            | 134        |
| 6.3.1        | 外部声明                | 134        |
| 6.3.2        | 类型区分符               | 137        |
| 6.3.3        | 结构区分符               | 138        |
| 6.3.4        | 声明符                 | 144        |
| 6.3.5        | 变量初始化               | 149        |
| 6.3.6        | 复合语句                | 150        |
| 6.3.7        | sizeof 表达式          | 150        |
| 6.3.8        | 初等表达式               | 152        |
| 6.4          | 控制程序                | 153        |
| 6.5          | 成果展示                | 155        |
| <b>第 7 章</b> | <b>生成 COFF 目标文件</b> | <b>157</b> |
| 7.1          | COFF 文件结构           | 157        |
| 7.1.1        | 基本概念                | 157        |
| 7.1.2        | 总体结构                | 158        |
| 7.1.3        | COFF 文件头            | 158        |
| 7.1.4        | 节头表                 | 161        |
| 7.1.5        | 代码节内容               | 168        |
| 7.1.6        | 数据节与导入节内容           | 168        |
| 7.1.7        | COFF 符号表            | 169        |
| 7.1.8        | COFF 字符串表           | 173        |
| 7.1.9        | COFF 重定位信息          | 173        |
| 7.2          | 生成 COFF 目标文件        | 175        |
| 7.2.1        | 生成节表                | 176        |



|                     |            |
|---------------------|------------|
| 7.2.2 生成符号表         | 178        |
| 7.2.3 生成重定位信息       | 182        |
| 7.2.4 生成目标文件        | 183        |
| 7.3 成果展示            | 185        |
| <b>第8章 x86 机器语言</b> | <b>187</b> |
| 8.1 x86 机器语言简介      | 187        |
| 8.2 通用指令格式          | 188        |
| 8.2.1 指令前缀          | 188        |
| 8.2.2 操作码           | 190        |
| 8.2.3 ModR/M 字节     | 190        |
| 8.2.4 SIB 字节        | 191        |
| 8.2.5 偏移量与立即数       | 193        |
| 8.3 x86 寄存器         | 193        |
| 8.3.1 数据寄存器         | 193        |
| 8.3.2 变址寄存器         | 193        |
| 8.3.3 指针寄存器         | 194        |
| 8.3.4 段寄存器          | 194        |
| 8.3.5 指令指针寄存器       | 194        |
| 8.3.6 标志寄存器         | 195        |
| 8.4 指令参考            | 196        |
| 8.4.1 符号说明          | 196        |
| 8.4.2 数据传送指令        | 198        |
| 8.4.3 算术运算指令        | 200        |
| 8.4.4 逻辑运算指令        | 203        |
| 8.4.5 控制转移指令        | 205        |
| 8.4.6 串操作指令         | 208        |
| 8.4.7 处理器控制指令       | 208        |
| 8.5 生成 x86 机器语言     | 208        |
| 8.5.1 操作数栈          | 209        |
| 8.5.2 生成通用指令        | 210        |
| 8.5.3 生成数据传送指令      | 213        |
| 8.5.4 生成算术与逻辑运算指令   | 217        |
| 8.5.5 生成控制转移指令      | 221        |
| 8.5.6 寄存器使用         | 224        |
| 8.5.7 本章用到的全局变量     | 227        |
| 8.6 成果展示            | 227        |
| <b>第9章 SCC 语义分析</b> | <b>229</b> |
| 9.1 外部定义            | 229        |

|        |              |     |
|--------|--------------|-----|
| 9.1.1  | 声明与函数定义      | 229 |
| 9.1.2  | 初值符          | 232 |
| 9.1.3  | 函数体          | 234 |
| 9.2    | 语句           | 237 |
| 9.2.1  | 表达式语句        | 237 |
| 9.2.2  | 选择语句         | 238 |
| 9.2.3  | 循环语句         | 239 |
| 9.2.4  | 跳转语句         | 241 |
| 9.3    | 表达式          | 244 |
| 9.3.1  | 赋值表达式        | 244 |
| 9.3.2  | 相等类表达式       | 245 |
| 9.3.3  | 关系表达式        | 246 |
| 9.3.4  | 加减类表达        | 248 |
| 9.3.5  | 乘除类表达式       | 249 |
| 9.3.6  | 一元表达式        | 250 |
| 9.3.7  | 后缀表达式        | 253 |
| 9.3.8  | 初值表达式        | 257 |
| 9.4    | 成果展示         | 259 |
| 第 10 章 | 链接器          | 261 |
| 10.1   | 链接方式与库文件     | 261 |
| 10.2   | PE 文件格式      | 263 |
| 10.2.1 | 总体结构         | 263 |
| 10.2.2 | DOS 部分       | 264 |
| 10.2.3 | NT 头         | 265 |
| 10.2.4 | 节头表          | 272 |
| 10.2.5 | 代码节          | 272 |
| 10.2.6 | 数据节          | 274 |
| 10.2.7 | 导入节          | 274 |
| 10.3   | 链接器代码实现      | 278 |
| 10.3.1 | 生成 PE 文件头    | 278 |
| 10.3.2 | 加载目标文件       | 281 |
| 10.3.3 | 加载引入库文件      | 282 |
| 10.3.4 | 解析外部符号       | 285 |
| 10.3.5 | 计算节区的 RVA 地址 | 288 |
| 10.3.6 | 重定位符号地址      | 291 |
| 10.3.7 | 修正需要重定位的地址   | 292 |
| 10.3.8 | 写 PE 文件      | 293 |
| 10.3.9 | 生成 EXE 文件    | 295 |

|               |                              |            |
|---------------|------------------------------|------------|
| 10.4          | SCC 编译器、链接器总控程序 .....        | 297        |
| 10.5          | 成果展示 .....                   | 301        |
| 10.6          | 全书代码架构 .....                 | 302        |
| <b>第 11 章</b> | <b>SC 语言程序开发 .....</b>       | <b>304</b> |
| 11.1          | SC 语言程序开发流程 .....            | 304        |
| 11.2          | SCC 编译器测试程序 .....            | 304        |
| 11.2.1        | 表达式测试 .....                  | 304        |
| 11.2.2        | 语句测试 .....                   | 308        |
| 11.2.3        | 结构体测试 .....                  | 310        |
| 11.2.4        | 函数参数传递测试 .....               | 312        |
| 11.2.5        | 字符串测试 .....                  | 314        |
| 11.2.6        | 全局变量测试 .....                 | 315        |
| 11.3          | 语言举例 .....                   | 316        |
| 11.3.1        | 可接收命令行参数的控制台程序 .....         | 316        |
| 11.3.2        | 可接收命令行参数的 Win32 应用程序 .....   | 317        |
| 11.3.3        | HelloWindows 窗口程序 .....      | 318        |
| 11.3.4        | 文件复制程序 .....                 | 323        |
| 11.3.5        | 九九乘法表 .....                  | 325        |
| 11.3.6        | 打印菱形 .....                   | 326        |
| 11.3.7        | 屏幕捕捉程序 .....                 | 328        |
| <b>参考文献</b>   | .....                        | <b>336</b> |
| <b>附录 A</b>   | <b>SC 语言文法定义中英文对照表 .....</b> | <b>337</b> |

# 第 1 章

## 引言

世上无难事，只怕有心人

——民谚

编译器是将一种语言翻译为另一种语言的计算机程序。编译器将源语言编写的程序作为输入，而产生用目标语言编写的等价程序。通常，源语言为高级语言（面向人的语言），如 C、C++、FORTRAN 等，而目标语言为机器语言（面向目标机的语言），如 Intel x86、ARM、MIPS、SPARC 等。

本书将和读者一起编写一个完整的 SCC(Simplified C Compiler)编译器，源语言是新定义的一门语言，称为 SC(Simplified C)语言，也就是简化的 C 语言，目标语言是 Intel x86 机器语言。SCC 编译器编译过程如图 1.1 所示。



图 1.1 SCC 编译器编译过程

让我们一起踏上编写 SCC 编译器的美妙旅程，一路上可能鲜花烂漫，也可能荆棘丛生，作者作为本次旅行的导游，接下来要先给大家讲一下旅行指南，让大家对本次旅程有个大概了解。

### 1.1 HelloWorld 编译过程分析

马克思主义认识论认为，认识是一个在实践基础上，由感性认识上升到理性认识，又由理性认识回到实践的辩证发展过程。本书的开头首先分析 SC 语言编写的 HelloWorld 程序的编译过程，以便对本书将要实现的 SCC 编译器的各个阶段的功能有个感性认识，第 2 章学习编写 SCC 编译器用到的编译原理知识，从第 3 章开始 SCC 编译器的实践过程。

#### 1.1.1 HelloWorld 程序源文件

HelloWorld 作为所有编程语言的入门程序，占据着无法改变的地位，这个例程是从 Kernighan & Ritchie 合著的 *The C Programme Language* 开始有的，因为它的简洁、实用，可谓麻雀虽小，五脏俱全，一门语言的 HelloWorld 程序可以看作这门语言语法结构的一个缩影，因此后来几乎所有学习各种计算机语言的书籍都以 HelloWorld 程序作为学习这门语言的入门程序。下面就先看看用 SC 语言编写的 HelloWorld 程序。

```

1. /*****
2. * HelloWorld.c 源文件
3. *****/
4. int main()
5. {
6.     printf("Hello World!\n");
7.     return 0;
8. }
9.
10. void _entry()
11. {
12.     int ret;
13.     ret=main();
14.     exit(ret);
15. }
16.

```

上面的程序似乎与 C 语言编写的没什么区别。但是,仔细一看会发觉用 SC 语言编写的 HelloWorld 程序多出一个 `_entry` 函数,它是干什么用的?从字面上理解应该是程序的入口点,没错, `_entry` 是上面 SC 语言 HelloWorld 程序的真正入口点。可能你认为,看来 SC 语言程序与 C 语言的入口点是有区别的,SC 语言程序的入口点是 `_entry`,C 语言程序的入口点是 `main` 函数。

讲述 C 语言的书上都说“`main` 是 C 语言程序的入口”,不知道大家是否对这句话的正确性怀疑过,是否深入探究过。在 Visual C++ 下,控制台程序的入口函数是 `mainCRTStartup`,由 `mainCRTStartup` 调用用户编写的 `main` 函数;图形用户界面(GUI)程序的入口函数是 `WinMainCRTStartup`,由 `WinMainCRTStartup` 调用用户编写的 `WinMain` 函数;`mainCRTStartup` 及 `WinMainCRTStartup` 函数的代码封装在已经编译好的 lib 库中,由链接器自动链接到生成的可执行文件中,所有这一切都是链接器悄悄干的,当然悄悄干的可不一定是坏事,也可能无名英雄做好事。`mainCRTStartup` 及 `WinMainCRTStartup` 函数就是这样的无名英雄,它们将该类程序开始都要做的一些必备且有技术含量的工作,悄悄地做了,并且背着广大程序员“悄悄”地做。SCC 编译器是一个学习模型,目的是让可执行文件中的每一函数都由用户自己的代码产生,不要有那么多“潜规则”,这样更有助于对编译过程的深入理解。

从 SC 语言编写的 HelloWorld 程序源代码中,大致可以了解以下内容:SC 语言还是保持了 C 语言的绝大多数功能,支持函数,函数以 `{` 开始,以 `}` 结束,支持变量声明,变量可以为 `int` 型,函数可以返回 `int` 型值,也可以返回 `void` 表示没有返回值,支持函数调用,支持字符串的处理。

### 1.1.2 词法分析

词法分析器读入 HelloWorld.c 源程序字符流,将它们组织为一系列具有词法含义的单词,词法分析器将给每个单词编上号,以便为语法分析提供便利。HelloWorld 程序的单词编码表如表 1.1 所示。

表 1.1 HelloWorld 单词编码表

| 单 词     | 编 码          | 单 词              | 编 码        |
|---------|--------------|------------------|------------|
| 关 键 字   |              | 常 量              |            |
| int     | KW_INT       | "Hello World!\n" | TK_CSTR    |
| return  | KW_RETURN    | 0                | TK_CINT    |
| void    | KW_VOID      | 标识符              |            |
| 运算符,分隔符 |              | main             | TK_IDENT+0 |
| (       | TK_OPENPA    | printf           | TK_IDENT+1 |
| )       | TK_CLOSEPA   | _entry           | TK_IDENT+2 |
| {       | TK_BEGIN     | ret              | TK_IDENT+3 |
| ;       | TK_SEMICOLON | exit             | TK_IDENT+4 |
| }       | TK_END       |                  |            |
| =       | TK_ASSIGN    |                  |            |

单词列比较清楚,就是程序中一个个独立单词,编码列中 KW\_INT、TK\_OPENPA、TK\_CSTR、TK\_IDENT 等,代表什么呢,它们都是单词枚举类型中的标识符,当成整型常量来理解就可以了。从表 1.1 中可以看出单词将被分为关键字、运算符、常量及标识符,每一个标识符都将有一个独立编码。

HelloWorld 单词编码看上去有些枯燥,好像也并不是很有意思,那么词法分析完成时有什么拿得出手的成果吗? 请看图 1.1,这里将关键字显示为绿色,运算符分隔符显示为红色,常量显示为黄色。

### 1.1.3 语法分析

语法分析是编译程序的核心部分。语法分析的作用是识别由词法分析给出单词序列是否是给定文法的正确句子(程序)。如果在语法分析阶段语法分析程序仅仅告诉我们“你的 HelloWorld.c 程序符合 SC 语言语法”,恐怕多少有点令人沮丧,花了多少天辛辛苦苦写出来的语法分析程序,就能干这点事,恐怕我们一点成就感都没有,因为 HelloWorld.c 符合 SC 语言语法,我们早就知道的。那么费不少工夫写出来的语法分析程序能不能干点有技术含量的活呢? 请看图 1.3,语法分析程序实现了对 HelloWorld 源程序语法缩进功能。图 1.3 与图 1.2 有什么区别呢? 请读者仔细对比一下。有了语法缩进程序,用 SC 语言写的代码再乱也不用担心了,哪怕程序完全写在一行也没关系,只要用语法自动缩进程序处理一下就美观了。

### 1.1.4 语义分析

编译器必须要忠实地将 SC 语言写的程序编译成机器指令,在语义分析阶段 SCC 编译器要当好“傀儡”,循规蹈矩,将“主子”(SC 语言)的意思忠实的传达给 Intel x86 处理器。在语义分析阶段将生成目标文件 HelloWorld.obj,它保存了语义分析的成果,来看一下目标文件生成过程,如图 1.4 所示。

```
C:\WINDOWS\system32\cmd.exe
E:\自己动手写编译器\代码样例\第四章>gcc.exe HelloWorld.c

int main()
{
    printf("Hello World!\n");
    return 0;
}

void _entry()
{
    int ret;
    ret = main();
    exit(ret);
}
End_Of_File
HelloWorld.c 词法分析成功!
```

图 1.2 HelloWorld 词法着色成果展示

```
C:\WINDOWS\system32\cmd.exe
E:\自己动手写编译器\代码样例\第五章>gcc.exe HelloWorld.c
int main()
{
    printf("Hello World!\n");
    return 0;
}
void _entry()
{
    int ret;
    ret = main();
    exit(ret);
}
End_Of_File
HelloWorld.c 语法分析通过!
```

图 1.3 HelloWorld 语法缩进成果展示

```
C:\WINDOWS\system32\cmd.exe
C:\sectest>gcc -o HelloWorld.obj -c HelloWorld.c
HelloWorld.c 编译成功
```

图 1.4 HelloWorld 编译生成目标文件

图 1.5 是 HelloWorld.obj 文件内容,是不是看上去跟天书一样,这是我们看到的第一封天书,第 7 章、第 8 章和第 9 章将对这封天书从不同侧面进行全方位解读。

### 1.1.5 链接器

严格地说,链接器不属于 SCC 编译器的工作范畴,但 SCC 编译器如果就停留在此处,恐怕有些扫兴。上面的 HelloWorld.obj 文件只能算是个半成品,还没法直接执行,就像一家汽车生产企业,将轮胎、发动机、车身等零部件生产出来了,但是还没有组装,这样的汽车当然没法跑起来,链接器充当着组装车间的角色,它将 HelloWorld.obj 与 C 运行时库链接生成 HelloWorld.exe 可执行文件。这个链接及执行过程如图 1.6 所示。

这可是令人激动的时刻,具有里程碑的意义。SCC 编译器最终可以生成可执行文件,执行 HelloWorld.exe 就会在命令行显示“Hello World!”,同样的 HelloWorld.exe 执行结果,但心情大不一样,因为这可是用自己亲手写的 SCC 编译器编译生成的。

HelloWorld.exe 既熟悉,又神秘,熟悉的是,运行 HelloWorld.exe 就会在命令行打印出“Hello World!”,神秘的是这个文件中到底存着什么,这个文件为什么能够“指挥”计算机



| T          | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | a  | b  | c  | d  | e  | f  |    |                  |           |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|-----------|
| 00000000h: | 4C | 01 | 08 | 00 | 00 | 00 | 00 | 00 | D0 | 01 | 00 | 00 | 08 | 00 | 00 | 00 | ;  | .....?.....      |           |
| 00000010h: | 00 | 00 | 00 | 00 | 2E | 74 | 65 | 78 | 74 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....text.....   |           |
| 00000020h: | 00 | 00 | 00 | 00 | 46 | 00 | 00 | 00 | 54 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....F...T.....  |           |
| 00000030h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 20 | 00 | 00 | 20 | 2E | 64 | 61 | 74 | ;                | ......dat |
| 00000040h: | 61 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | a.....           |           |
| 00000050h: | 9A | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | ?.....           |           |
| 00000060h: | 40 | 00 | 00 | C0 | 2E | 72 | 64 | 61 | 74 | 61 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | @..?rdata.....   |           |
| 00000070h: | 00 | 00 | 00 | 00 | 0E | 00 | 00 | 00 | 9A | 01 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....?           |           |
| 00000080h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 40 | 00 | 00 | 40 | 2E | 69 | 64 | 61 | ;  | .....@...ida     |           |
| 00000090h: | 74 | 61 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | ta.....          |           |
| 000000a0h: | A8 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | ?.....           |           |
| 000000b0h: | 40 | 00 | 00 | C0 | 2E | 62 | 73 | 73 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | @..?bss.....     |           |
| 000000c0h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | A8 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....?           |           |
| 000000d0h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 80 | 00 | 00 | C0 | 2E | 72 | 65 | 6C | ;  | .....e...?rel    |           |
| 000000e0h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 28 | 00 | 00 | 00 | ;  | .....{.....      |           |
| 000000f0h: | A8 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | ?.....           |           |
| 00000100h: | 00 | 08 | 00 | 40 | 2E | 73 | 79 | 6D | 74 | 61 | 62 | 00 | 00 | 00 | 00 | 00 | ;  | ...@.syntab..... |           |
| 00000110h: | 00 | 00 | 00 | 00 | 90 | 00 | 00 | 00 | D0 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....?..?.....   |           |
| 00000120h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 08 | 00 | 40 | 2E | 73 | 74 | 72 | ;  | .....@.str       |           |
| 00000130h: | 74 | 61 | 62 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 2B | 00 | 00 | 00 | ;  | tab.....+        |           |
| 00000140h: | 60 | 02 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....            |           |
| 00000150h: | 00 | 08 | 00 | 40 | 55 | 89 | E5 | 81 | EC | 00 | 00 | 00 | 00 | B8 | 00 | 00 | ;  | ...80?....?      |           |
| 00000160h: | 00 | 00 | 50 | E8 | FC | FF | FF | FF | 83 | C4 | 04 | B8 | 00 | 00 | 00 | 00 | ;  | ..P楼 題..?        |           |
| 00000170h: | E9 | 00 | 00 | 00 | 00 | 8B | E5 | 5D | C3 | 55 | 89 | E5 | 81 | EC | 04 | 00 | ;  | ?...?..?..?..?   |           |
| 00000180h: | 00 | 00 | E8 | FC | FF | FF | FF | 89 | 45 | FC | 8B | 45 | FC | 50 | E8 | FC | ;  | ..楼 题?..?..?     |           |
| 00000190h: | FF | FF | FF | 83 | C4 | 04 | 8B | E5 | 5D | C3 | 48 | 65 | 6C | 6C | 6F | 20 | ;  | 题..?..?..?       |           |
| 000001a0h: | 57 | 6F | 72 | 6C | 64 | 21 | 0A | 00 | 0A | 00 | 00 | 00 | 03 | 00 | 00 | 00 | ;  | World!.....      |           |
| 000001b0h: | 01 | 06 | 10 | 00 | 00 | 00 | 05 | 00 | 00 | 00 | 01 | 14 | 2F | 00 | 00 | 00 | ;  | ...../.....      |           |
| 000001c0h: | 04 | 00 | 00 | 00 | 01 | 14 | 3B | 00 | 00 | 00 | 07 | 00 | 00 | 00 | 01 | 14 | ;  | .....?           |           |
| 000001d0h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....            |           |
| 000001e0h: | 00 | 00 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 02 | 00 | ;  | .....            |           |
| 000001f0h: | 00 | 00 | 03 | 00 | 07 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....            |           |
| 00000200h: | 05 | 00 | 00 | 00 | 03 | 00 | 0C | 00 | 00 | 00 | 01 | 00 | 00 | 00 | 00 | 00 | ;  | .....            |           |
| 00000210h: | 00 | 00 | 03 | 00 | 00 | 00 | 03 | 00 | 13 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | .....            |           |
| 00000220h: | 00 | 00 | 00 | 00 | 01 | 00 | 20 | 00 | 02 | 00 | 18 | 00 | 00 | 00 | 00 | 00 | ;  | .....            |           |
| 00000230h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 20 | 00 | 02 | 00 | 1F | 00 | 00 | 00 | ;  | .....            |           |
| 00000240h: | 00 | 00 | 00 | 00 | 25 | 00 | 00 | 00 | 01 | 00 | 20 | 00 | 02 | 00 | 26 | 00 | ;  | .....?.....?     |           |
| 00000250h: | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 20 | 00 | 02 | 00 | ;  | .....            |           |
| 00000260h: | 00 | 2E | 64 | 61 | 74 | 61 | 00 | 2E | 62 | 73 | 73 | 00 | 2E | 72 | 64 | 61 | ;  | ..data..bss..rda |           |
| 00000270h: | 74 | 61 | 00 | 6D | 61 | 69 | 6E | 00 | 70 | 72 | 69 | 6E | 74 | 66 | 00 | 5F | ;  | ta.main.printf._ |           |
| 00000280h: | 65 | 6E | 74 | 72 | 79 | 00 | 65 | 78 | 69 | 74 | 00 | 00 | 00 | 00 | 00 | 00 | ;  | entry.exit.      |           |

图 1.5 HelloWorld.obj 文件内容

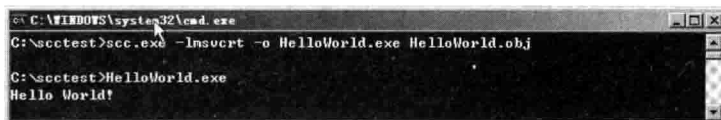


图 1.6 HelloWorld.obj 文件内容

打印出“Hello World!”。

表 1.2 就是 HelloWorld.exe 的文件内容,省略了部分全 0 的内容,这是我们见到的第二封天书,这封天书将在第 10 章进行全方位解读,请大家拭目以待。

表 1.2 HelloWorld.exe 文件内容

| 文件偏移      | 文件数据内容(十六进制表示)                                  | 数据代表的 ASCII 码字符  |
|-----------|---|------------------|
| 00000000h | 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 | MZ?.....         |
| 00000010h | B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 | ?.....@.....     |
| 00000020h | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....            |
| 00000030h | 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 | ..... €...       |
| 00000040h | 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 | ..?.???L?Th      |
| 00000050h | 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F | is program canno |
| 00000060h | 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 | t be run in DOS  |
| 00000070h | 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 | mode.... \$..... |

续表

[illegible]

## 1.2 SCC 编译器简介

### 1.2.1 SCC 编译器架构

通过上述 HelloWorld 编译过程分析,大家已经对 SCC 编译器有了大致了解,本节讨论 SCC 编译器整体架构,参见图 1.7。

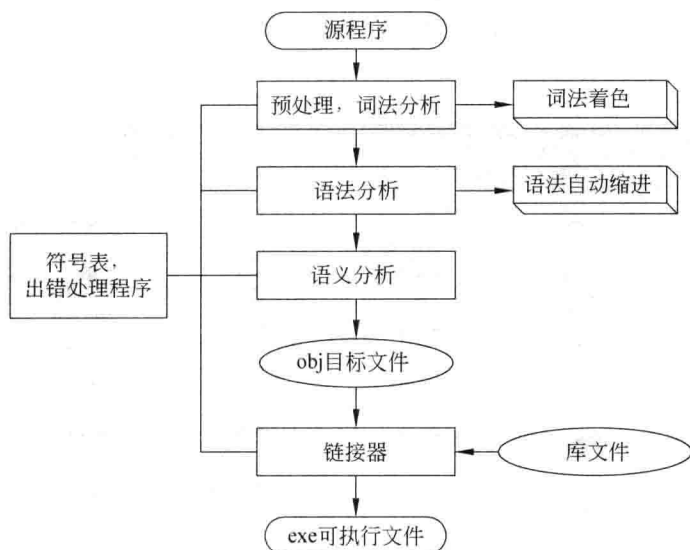


图 1.7 SCC 编译器架构

读者可能对这个图很熟悉,感觉没有什么新意,每本讲编译原理的书中都会有这个编译过程图。但大多是纸上谈兵,本书可是要实现图 1.1 中所有功能。有了前面对 HelloWorld 编译过程的分析,上面的 SCC 编译器架构也很容易理解,所以这里不多解释。给出这个图的目的,可用古人的话来表达为“不谋万世者,不足谋一时;不谋全局者,不足谋一域。”。SCC 编译器实现过程是一项复杂的、整体的过程,各个阶段既相对独立,又紧密相关,要求在每个阶段的程序设计上要考虑后续阶段能够方便使用。

### 1.2.2 SCC 编译器开发环境

SCC 编译器是在 Windows 操作系统中,使用 Visual Studio 6.0 中的 Visual C++ 6.0 开发的,读者可能会问为什么不用 Visual Studio .Net 呢? Visual Studio 6.0 虽然是 Microsoft 公司开发环境的老版本,但是鉴于其后继版本的主要功能变化都是为了支持 .Net 平台,并且安装后身躯庞大,体态臃肿,所以对于开发非 .Net 平台的程序,这个经典稳定的开发环境仍然是首选。

有必要对 Visual Studio 6.0 开发环境做一个简单介绍,以便读者对 Visual Studio 6.0 与 Visual C++ 6.0 的关系有个清晰的认识。Visual Studio 6.0 是微软公司在 1998 年前后推出的一个编程组件,Visual Studio 6.0 中含有 Visual Basic 6.0、Visual C++ 6.0、Visual J++ 6.0、Visual FoxPro 6.0、Visual SourceSafe 6.0 等,而 SCC 编译器开发只用到了其中

的 Visual C++ 6.0。

Visual C++ 6.0 IDE(Integrated Development Environment, 即集成开发环境)界面如图 1.6 所示。IDE 是用于程序开发环境的应用程序,一般包括代码编辑器、编译器、链接器、调试器和图形用户界面工具等,是集成了代码编写功能、分析功能、编译功能、调试功能等一体化的开发软件套件。人们习惯上把 IDE 称为编译器,这个称呼有些名不符实,并且会形成误导。其实在 Visual C++ 6.0 IDE 中,编译时 IDE 会安排 CL.EXE 来编译,当链接时 IDE 会指挥 LINK.EXE 来链接,这时 IDE 整个就是一甩手掌柜,被称作编译器完全是“浪得虚名”。CL.EXE 与 LINK.EXE 才是真正的幕后英雄,通过图 1.8 可看到在 IDE 背后一直默默奉献的两位“无名英雄”。



图 1.8 VC6 中的编译器和链接器

讲完 Visual C++ 6.0 IDE, 这里讲一下 SCC 编译器使用的开发语言,VC6 编译器支持 C 语言及 C++ 语言的编译,当源文件后缀为 .c 时按 C 语言来编译,当源文件后缀为 .cpp 时,按 C++ 来编译,SCC 编译器则完全使用 C 语言来开发。

### 1.2.3 SCC 编译器运行环境

这里说一下 SCC 编译器的运行环境,主要包括支持的处理器和操作系统两方面。Intel 80x86 平台和 Windows 是桌面计算机上最流行的配置仍是不争的事实,所以 SCC 编译器目前只支持 Windows 操作系统,处理器只支持兼容 Intel x86 指令集的处理器。

这里介绍一下 x86 指令集处理器,Intel 8086/8088/80186/80286 CPU 都为 16 位处理器,在市面上的 PC 中这些很多年前就已经销声匿迹,所以 SCC 编译器将不支持这些 16 位处理器的指令系统。1985 年,Intel 公司正式公布了 32 位处理器 80386,它采用 32 位指令系统,有 32 条地址线。80386 处理器在设计的时候考虑了多用户及多任务的需要,在芯片中增加了保护模式、优先级、任务切换和片内的存储单元管理等硬件单元。直到现在,运行于 80x86 处理器之上的多任务操作系统都是以 80386 的运行模式为基础的。本书中,x86 指兼容 80386 指令集的处理器。

从 80386 开始,在 Intel 公司向市场大量推出处理器芯片的同时,其他一些电脑公司和厂商如 AMD 和 Cyrix 等,也纷纷投入大量的人力财力进行处理器的开发和研制,并很快把研制出的产品推向市场。这些 CPU 芯片和 80386 芯片兼容,在编程上可以使用与 Intel 处理器相同的指令集。Intel 公司后来推出的 80486 及奔腾、赛扬、酷睿系列 CPU 都兼容 80386 指令集。

所谓 SCC 支持的操作系统有两层含义:一是 SCC 编译器所运行的操作系统;二是用 SCC 编译器编译生成的可执行文件所运行的操作系统。SCC 编译器所运行的操作系统及生成的可执行文件所运行的操作系统皆为 Win 32 操作系统或者可以兼容运行 Win 32 应用程序的操作系统,即 PC 上装的 Windows 2000、Windows XP、Windows Server 2003、Windows Vista、Windows 7、Windows 8 都支持。

## 第2章

# 文法知识

宜未雨而绸缪，毋临渴而掘井。

——朱柏庐

在正式开始编写编译器之前，需要学习一点编译原理的基础知识，这里不会像《编译原理》书籍那样长篇大论面面俱到地讲那些枯燥的理论。本书对理论知识讲授本着够用就行的原则，对于不好理解的知识，还会附以生动形象的例子帮助理解。

在正式介绍文法的知识之前，先来看一下西天取经团队成员的文法定义，以便对文法有个感性认识。

①  $\langle \text{西天取经团队成员} \rangle ::= \langle \text{师父} \rangle | \langle \text{徒弟成员} \rangle$

②  $\langle \text{师父} \rangle ::= \text{"唐僧"}$

③  $\langle \text{徒弟成员} \rangle ::= \text{"孙悟空"} | \text{"猪八戒"} | \text{"沙和尚"} | \text{"白龙马"}$

不用多解释，大家也知道上面文法的含义吧。西天取经团队成员是师父或徒弟，师父是“唐僧”，徒弟是“孙悟空”、“猪八戒”、“沙和尚”或“白龙马”。

问大家一个问题，西天取经团队成员中，有一位他的名字中第一字是“孙”，问这位成员是谁？读者可能会说，这么弱智的问题还好意思拿出来问，当然是“孙悟空”。

再举一个例子：

$\langle \text{陈述句} \rangle ::= \langle \text{陈述句内容} \rangle$ 。

再问大家一个问题，根据 $\langle \text{陈述句} \rangle$ 的方法定义， $\langle \text{陈述句内容} \rangle$ 以什么结尾？当然是以句号结尾。

如果上述两个问题你都答对了，那么恭喜你，你读这本书的词法分析和语法分析部分，将不会遇到太大的困难，因为本书的词法语法部分，用到的就是这个原理。后面文绉绉的对First集、Follow集的定义，其实描述的就是这点事。请大家带着如下两个问题来阅读本章内容，第一如何定义一门语言，第二如何对一门语言进行词法分析、语法分析。

## 2.1 语言概述

语言是由句子组成的集合，是由一组符号所构成的集合。汉语是所有符合汉语语法的句子的全体。英语是所有符合英语语法的句子的全体。程序设计语言是所有符合该语言语法定义的程序的全體。

语言有两方面来构成：语法和语义。语法表示构成语言句子的各个记号之间的组合规律，语义表示按照各种表示方法所表示的各个记号的特定含义，即各个记号和记号所表示的对象之间的关系。每种语言具有两个可识别的特性，即语言的形式和该形式相关联的意义。

下面以<句子>的定义为例来说明一下语法与语义的关系。

- ① <句子>::=<主语><谓语>{<宾语>}
- ② <主语>::=<名词>
- ③ <宾语>::=<名词>
- ④ <谓语>::=<动词>
- ⑤ <名词>::="人"|"狗"|"花"|"鸟"|"虫"|"鱼"|"水"
- ⑥ <谓语>::="吃"|"喝"|"打"|"咬"|"睡觉"|"游泳"|"开"

根据上面的语法定义,如果说“人打狗”,“狗喝水”,“花开”,“鸟睡觉”,“虫咬人”,“鱼游泳”这些句子符合语法,语义上也没问题。但是如果说“狗打人”,“水喝狗”,“花咬鱼”,“水睡觉”,“虫游泳鸟”,“鱼打狗”,这些句子语法确实没有问题,但是语义上非常荒谬。

下面给出语法与语义关系的官方描述:语法能够描述程序设计语言的大部分语法但不是全部,例如,标识符的先声明后使用无法用上下文无关文法描述。因此,语法分析器接受的语言是程序设计语言的超集。必须通过语义分析来剔除一些符合文法、但不合法的程序。语言是语义和语法的统一,语法结构是外表,语义结构是内在。

上述官方描述,现在有些不太理解没关系,等读完本章就完全理解了。

## 2.2 形式语言

如果不考虑语义,即只从语法这一侧面来看语言,这种意义下的语言称作形式语言。形式语言抽象地定义为一个数学系统。“形式”是指这样的事实:语言的所有规则只以什么符号串能出现的方式来陈述。形式语言理论是对符号串集合的表示法、结构及其特性的研究,是程序设计语言语法分析研究的基础。

### 2.2.1 字母表和符号串

正如英语是由句子组成的集合,而句子又是由单词和标点符号组成的序列那样。SC 程序设计语言,是由一切 SC 程序所组成的集合,而 SC 程序是由 if、else、for 等关键字符, +、-、\*、/ 等运算符, ;、,、{、} 等分隔符号,字母数字及下划线组成的标识符等基本符号所组成。从字面上看,每个程序都是一个“基本符号”串,设有一基本符号集,那么 SC 语言可看成在这个基本符号集上定义的,按一定规则构成的一切基本符号串组成的集合,因此有必要将有关符号串的一些概念做一下介绍,作为文法和语言的形式定义的预备知识。

**字母表:**字母表是元素的非空有穷集合,把字母表中的元素称为符号,因此字母表也称为符号集。不同语言可以有不同的字母表,例如汉语的字母表中包括汉字、数字及标点符号等。这里读者可要注意了,“字母表”可不要只机械地理解成英文字母表的 26 个英文字母。对于 SC 语言定义来说,语法定义时的字母表是指经词法分析识别出的一个个单词符号,如 if、else、for、+、-、\*、/、;、,、{、} 等。词法分析的字母表则是 a~z、A~Z、\_、+、-、\*、/ 等源码字符。

**符号:**字母表中的元素,语法分析时指一个个单词,词法分析指一个个源码字符。

**符号串:**由字母表中的符号组成的任何有穷序列称为符号串。语法分析时 int abc=1 代表一个符号串。词法分析时 abc 就代表一个符号串。在符号串中,符号的顺序是很重要的,例如符号串 abc 就不同于 cba。可以使用字母表示符号串,如 x=abc 表示“x 是由符号



a、b 和 c,并按此顺序组成的符号串”。

**空符号串:** 无任何符号的符号串或长度为零的符号串,记为  $\epsilon$ 。

**符号串集合:** 若集合  $A$  中的一切元素都是某字母表上的符号串,则称  $A$  为该字母表上的符号串集合。

**符号串相等:** 若  $x, y$  是集合上的两个符号串,则  $x=y$  当且仅当组成  $x$  的每一个符号和组成  $y$  的每一个符号依次相等。

**符号串的长度:**  $x$  为符号串,其长度  $|x|$  等于组成该符号串的符号个数。

例:  $x=abc, |x|=3$

**符号串的连接:** 若  $x, y$  是定义在  $\Sigma$  上的符号串,且  $x=XY, y=YX$ ,则  $x$  和  $y$  的连接  $xy=XYXX$  也是  $\Sigma$  上的符号串。注意,一般  $xy \neq yx$ ,而  $\epsilon x = x\epsilon$ 。

**符号串集合的乘积运算:** 令  $A, B$  为符号串集合,定义

$$AB = \{xy \mid x \in A, y \in B\}$$

**符号串集合的幂运算:** 有符号串集合  $A$ ,定义

$$A_0 = \{\epsilon\}, A_1 = A, A_2 = AA, A_3 = AAA, \dots, A_n = A_{n-1}A = AA_{n-1}, n > 0$$

**符号串集合的闭包运算:** 设  $A$  是符号串集合,定义

$A^+ = A_1 \cup A_2 \cup A_3 \cup \dots \cup A_n \cup \dots$  称为集合  $A$  的正闭包。

$A^* = A_0 \cup A^+$  称为集合  $A$  的闭包。

例:  $A = \{x, y\}$

$$A^+ = \{ \underset{A_1}{x, y}, \underset{A_2}{xx, xy, yx, yy}, \underset{A_3}{xxx, xxy, xyx, xyy}, \dots \}$$

$$A^* = \{ \epsilon, \underset{A_0}{x, y}, \underset{A_1}{xx, xy, yx, yy}, \underset{A_2}{xxx, xxy, xyx, xyy}, \dots \}$$

$$\underset{A_0}{A_0} \quad \underset{A_1}{A_1} \quad \underset{A_2}{A_2} \quad \underset{A_3}{A_3}$$

若  $A$  为 SC 语言的基本字符集

$$A = \{a, b, \dots, z, A, B, \dots, Z, 0, 1, \dots, 9, +, -, *, /, (, ), = \dots\}$$

$B$  为 SC 语言单词集

$$B = \{\text{char, short, if, for, break}, \dots, \text{abc, xl, yl}, \dots\}$$

则  $B \subset A^*$ 。

SC 语言程序是定义在  $B$  上的符号串。若令  $C$  为 SC 语言程序集合,则  $C \subset B^*$ , 程序  $\subset C$ 。

## 2.2.2 文法与语言的形式定义

文法是对语言结构的定义与描述,即从形式上用于描述和规定语言结构的称为文法(或称为“语法”)。

当表述一种语言时,就是要说明这种语言的句子。如果语言只含有有穷多个句子,则只需列出句子的有穷集。如果语言含有无穷多个句子,则存在如何给出它的有穷表示的问题。这需要一种规则,用这些规则来描述语言的结构,可以把这些规则看成一种元语言,这些规则就称为文法。下面给出文法的定义。进而在文法定义的基础上,给出推导的概念,句型、句子和语言的定义。

**定义** 文法  $G$  定义为四元组  $(V_N, V_T, P, S)$ 。其中  $V_N$  为非终结符(是可以被取代的符

号)集; $V_T$  为终结符(语言中用到的基本元素,不能再被分解成更小的单位)集; $P$  为产生式(也称规则)的集合; $V_N$ 、 $V_T$  和  $P$  是非空有穷集。 $S$  称作识别符号或开始符号,它是一个非终结符,至少要在一条产生式中作为左部出现。

$V_N$  和  $V_T$  不含公共的元素,即  $V_N \cap V_T = \emptyset$ 。通常用  $V$  表示  $V_N \cup V_T$ ,  $V$  称为文法  $G$  的字母表或字汇表。其中产生式,是形如  $\alpha \rightarrow \beta$  或  $\alpha ::= \beta$  的  $(\alpha, \beta)$  有序对,其中  $\alpha$  是字母表  $V$  的正闭包  $V^+$  中的一个符号,  $\beta$  是  $V^*$  中的一个符号。 $\alpha$  称为规则的左部,  $\beta$  称为规则的右部。

为定义文法所产生的语言,还需要引入推导的概念,即定义  $V^*$  中的符号之间的关系:直接推导  $\Rightarrow$ 、长度为  $n(n \geq 1)$  的推导  $\Rightarrow^+$  和长度为  $n(n \geq 0)$  的推导  $\Rightarrow^*$ 。

**定义** 如  $\alpha \rightarrow \beta$  是文法  $G = (V_N, V_T, P, S)$  的规则(或说是  $P$  中的一产生式),  $\gamma$  和  $\delta$  是  $V^*$  中的任意符号,若有符号串  $v, w$  满足:

$$v = \gamma\alpha\delta, \quad w = \gamma\beta\delta$$

则说  $v$  (应用规则  $\alpha \rightarrow \beta$ ) 直接产生  $w$ , 或者说,  $w$  是  $v$  的直接推导, 也可以说,  $w$  直接归约到  $v$ , 记作  $v \Rightarrow w$ 。

**定义** 如果存在直接推导的序列:

$$v = w_0 \Rightarrow w_1 \Rightarrow w_2 \cdots \Rightarrow w_n = w, (n > 0)$$

则称  $v$  推导出(产生)  $w$  (推导长度为  $n$ ), 或称  $w$  归约到  $v$ 。记作  $v \Rightarrow^+ w$ 。

**定义** 若有  $v \Rightarrow^+ w$ , 或  $v = w$ , 则记作  $v \Rightarrow^* w$ 。

**定义** 设  $G[S]$  是一文法, 如果符号串  $x$  是从识别符号推导出来的, 即有  $S \Rightarrow^* x$ , 则称  $x$  是文法  $G[S]$  的句型。若  $x$  仅由终结符号组成, 即  $S \Rightarrow^* x, x \in V_T^+$ , 则称  $x$  为  $G[S]$  的句子。

**定义** 文法  $G$  所产生的语言定义为集合  $\{x | S \Rightarrow^* x, \text{其中 } S \text{ 为文法识别符号, 且 } x \in V_T^+\}$ 。可用  $L(G)$  表示该集合。

从这个定义看出两点: 第一, 符号串  $x$  可从识别符号推出, 也即  $x$  是句型。第二,  $x$  仅由终结符号组成, 即  $x$  是文法  $G$  的句子。也就是说, 文法描述的语言是该文法一切句子的集合。

**定义** 若  $L(G_1) = L(G_2)$ , 则称文法  $G_1$  和  $G_2$  是等价的。也就是说, 如果两个文法定义的语言一样, 则称这两个文法是等价的。

**定义** 如果在推导的任何一步  $v \Rightarrow w$ , 其中  $v, w$  是句型, 都是对  $v$  中的最左(最右)非终结符进行替换, 则称这种推导为最左(最右)推导。

### 2.2.3 文法与语言的类型

1956 年, 乔姆斯基(Chomsky)建立了形式语言理论, 这种理论对计算机科学有着深刻的影响, 特别是对程序设计语言的设计、编译方法和计算复杂性等方面更有重大的作用。

#### 2.2.3.1 文法分类

乔姆斯基把文法分成 4 种类型, 即 0 型、1 型、2 型和 3 型。这几类文法的差别在于对产生式施加不同的限制。下面看一下 4 种类型文法的定义。

**0 型文法:** 设  $G = (V_N, V_T, P, S)$ , 如果它的每个产生式  $\alpha \rightarrow \beta$  是这样一种结构:  $\alpha \in (V_N \cup V_T)^*$  且至少含有一个非终结符, 而  $\beta \in (V_N \cup V_T)^*$ , 则  $G$  是一个 0 型文法。一个非常重要的理论结果是: 0 型文法的能力相当于图灵机(Turing)。或者说, 任何 0 型文法语言都

是递归可枚举的；反之，递归可枚举集必定是一个 0 型语言。0 型文法对文法规则的表示形式不作任何限制，从而能使定义的语言提供充分的描述功能。但 0 型文法不保证语言的递归性，即不能确保语句合法的可判性，所以很少用于定义自然语言。

**1 型文法：**1 型文法也称上下文有关文法，其可描述的语言为上下文有关语言，其对应的识别器为线性有界自动机。它是在 0 型文法的基础上，规定对每一个  $\alpha \rightarrow \beta$ ，都有  $|\beta| \geq |\alpha|$ 。这里的  $|\beta|$  表示的是  $\beta$  的长度。自然语言是上下文有关的语言，文法规则允许其左部有多个符号（至少包括一个非终结符），以指示上下文相关性；但要求规则右部符号的个数不少于左部，以确保语言的递归性（即语句合法的可判性）。

**2 型文法：**2 型文法也称上下文无关文法，其可描述的语言为上下文无关语言，其对应的识别器为下推自动机。下推自动机比下面讲到的识别 3 型方法的有限状态自动机复杂：除了有限状态组成部分外，还包括一个长度不受限制的栈。2 型文法是在 1 型文法的基础上，再满足：每一个  $\alpha \rightarrow \beta$  都有  $\alpha$  是非终结符。如  $A \rightarrow Ba$ ，符合 2 型文法要求。如  $Ab \rightarrow Bab$  虽然符合 1 型文法要求，但不符合 2 型文法要求，因为其  $\alpha = Ab$ ，而  $Ab$  不是一个非终结符。上下文无关文法有足够的描述能力描述现今程序设计语言的语法结构，例如描述各种表达式、描述各种语句等。上下文无关文法及其语言已广泛应用于定义程序设计语言，这种文法的规则限定其左部只能是单一的非终结符，即非终结符通过文法规则的扩展性重写是相互独立的，不受其他符号的影响，所以称为上下文无关。

**3 型文法：**3 型文法也称正规文法，其可描述的语言为正则语言，其对应的识别器为有限状态自动机。它是在 2 型文法的基础上满足： $A \rightarrow \alpha | \alpha B$ （右线性）或  $A \rightarrow \alpha | B\alpha$ （左线性）。可以看出，3 型文法规则表示形式高度受限，使得正则语言可以用有限状态自动机程序作高效的句法分析。有限状态自动机有若干状态，其中必有一个为起始状态，并至少有一个结束状态；自动机的输入会导致状态变化，并在到达目标状态时停机。面向正则语言句法分析的有限状态自动机就以文法规则左部的非终结符指示当前状态，文法规则右部的终结符作为输入，终结符后的非终结符就是自动机将到达的下一状态；若终结符后无非终结符，则自动机在当前状态下停机。若输入结束且此时自动机处于结束状态，则输入就作为一个合法语句而接受；否则输入的是非法语句。尽管正规文法简单并易于分析，但文法规则的表示太受限制，使其无法用于描述哪怕是人工制定的语言。正规文法主要用于西文（如英语）的词法分析阶段，如切分单词和识别非法字符。

**注意：**上面例子中的大写字母表示的是非终结符，而小写字母表示的是终结符。

4 种类型文法的规则受限情况、相应的语言类型、相对应的识别器之间的关系如表 2.1 所示。

表 2.1 文法、语言及相应识别器之间的关系

| 乔姆斯基文法层次 | 文法别名    | 语言类型    | 规则限制                                 | 识别器     |
|----------|---------|---------|--------------------------------------|---------|
| 3 型文法    | 正规文法    | 正则语言    | 左部必须是单一非终结符，右部必须是单一终结符或单一终结符后跟单一非终结符 | 有穷状态自动机 |
| 2 型文法    | 上下文无关文法 | 上下文无关语言 | 左部必须是单一非终结符                          | 下推自动机   |
| 1 型文法    | 上下文有关文法 | 上下文有关语言 | 左部至少包括一个非终结符，右部符号的个数不少于左部            | 线性有界自动机 |
| 0 型文法    | 无限制文法   | 递归可枚举语言 | 无                                    | 图灵机     |

### 2.2.3.2 几类语言之间的关系

现在已经明白了上述4类文法相应代表的4种语言类型,那么这4种语言之间是什么关系呢?参见图2.1。

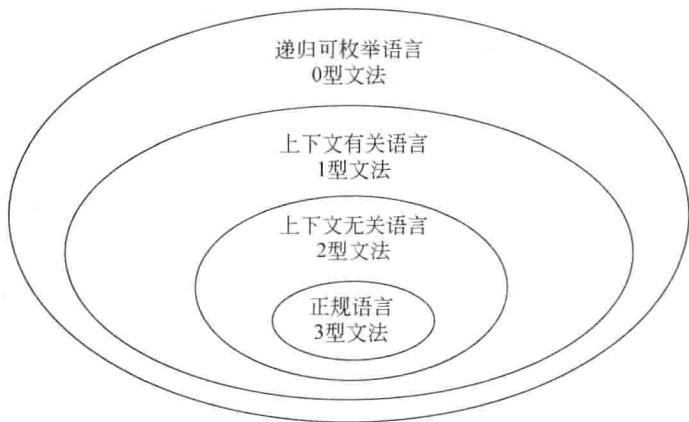


图 2.1 几类语言之间的关系

从图2.1可以看出,正规语言类包含于上下文无关语言类,上下文无关语言类包含于上下文有关语言类,上下文有关语言类包含于递归可枚举语言类。这里的包含都是集合的真包含关系,也就是说,存在递归可枚举语言不属于上下文有关语言类,存在上下文有关语言不属于上下文无关语言类,存在上下文无关语言不属于正规语言类。

### 2.2.4 程序设计语言描述工具

程序设计语言可以使用上下文无关文法或者巴科斯范式来描述,巴科斯范式的表达能力等价于上下文无关文法。上下文无关方法前面已经介绍过,下面来认识一下巴科斯范式。

巴科斯范式(Backus-Naur Form, BNF)是由 John Backus 和 Peter Naur 首次引入一种形式化符号来描述给定语言的语法(最早用于描述 ALGOL 60 编程语言)。BNF 类似一种数学游戏:从一个符号开始(称为起始标志,实例中常用 S 表示),然后给出替换前面符号的规则。BNF 语法定义的语言只不过是一个字符串集合,可以按照下述规则书写,这些规则称为书写规范(产生式规则),形式如下:

`symbol ::= alternative1 | alternative2 ...`

每条规则声明 `::=` 左侧的符号必须被右侧的某一个可选项代替。BNF 只包括 3 种元符号,如表 2.2 所示。

表 2.2 BNF 元符号表

| 元 符 号                 | 表 示 含 义     | 备 注   |
|-----------------------|-------------|---|
| <code>::=</code>      | 定义为         | 有的书上用 <code>-&gt;</code> , 或 <code>: =</code> |
| <code> </code>        | 或者          |   |
| <code>&lt;&gt;</code> | 尖括号用于括起非终结符 |   |

明白了 BNF 表示语法的书写规则,下面来看一下用 BNF 表示的有符号整数,

```

<有符号整数>::=<无符号整数>|<正负号><无符号整数>
<无符号整数>::=<数字>|<无符号数><数字>
<数字>::=0|1|2|3|4|5|6|7|8|9
<正负号>::=+|-

```

不知道你对上面的<有符号整数>文法定义是否满意,感觉是否够简洁,是否容易理解,是否想过还有没有更好的定义方法?可能你还来不及想这些,我们先来看一下前人对BNF的看法:BNF有着可选项和重复不能直接表达的问题。作为替代,它们需要利用中介规则或两选一规则,对于可选项,定义要么是空的,要么是可选的产生式的规则;对于重复,递归的定义要么是被重复的产生式,要么是自身的规则。既然前人认识到了这些问题,那么这些问题有没有用更好的方法来解决呢?下面就看一下那些不安于现状的前辈为了解决这些问题又研究出了什么新鲜玩意儿。

扩展巴科斯范式(EBNF)是表达作为描述计算机编程语言和形式语言的正规方式的上下文无关文法的元语法符号表示法。它是基本巴科斯范式(BNF)元语法符号表示法的一种扩展。它最初由尼克劳斯·维尔特开发,最常用的EBNF变体由ISO-14977标准所定义。下面通过表2.3看一下EBNF的元符号及其含义。

表 2.3 EBNF 元符号表

| EBNF 元符号 | 含 义           | EBNF 元符号 | 含 义       |
|----------|---------------|----------|-----------|
| ::=      | 定义为,推导为       | ()       | 括号内看作一项   |
|          | 或             | .        | 一条生成规则的结束 |
| { }      | 含 0 次在内任意多次重复 | <>       | 非终结符      |
| [ ]      | 含 0 次和 1 次    | " "      | 终结符       |

BNF原来只有3个元符号,EBNF成了8个元符号,想必EBNF比BNF功能强大多了。这里要特别提示一下,EBNF在定义语言方面并不比BNF更强大,只是更方便。凡是用EBNF写的东西都可以转换成BNF的形式。下面再看一下<有符号整数>的EBNF定义:

```

<有符号整数>::=[<正负号>]<有符号整数>
<无符号整数>::=<数字>{<数字>}
<数字>::="0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
<正负号>::="+"|"-"

```

用EBNF定义文法是不是比BNF更方便,也更容易理解。引入EBNF除了上述优点之外,另一个更重要原因是为了更方便地把文法映射到递归下降分析程序的真实代码。当需要手动构造递归下降分析程序的时候,通常把上下文无关文法改写为EBNF是必需的。请大家务必记住上面这句话,如果现在不理解没关系,可以在后面的章节中逐步理解。

多数编程语言标准都使用EBNF来定义语言的语法,这样做的好处是在语言的语法上没有争议,而且有助于编译器的编写。

## 2.3 词法分析方法

为了设计词法分析器,可以根据正规文法定义建立状态转换图。状态转换图是个有向图,其中的结点表示状态,通常用整数或字母命名;每条有向边其标记表示从该边起点的状

态转换到终点的状态需要的条件。对应词法分析器规格说明中的每个正规定义,分别建立一个状态转换图,其中每条有向边的标记分别表示识别词汇时从输入缓冲区接收一个输入字符。开始识别一个词形时对应的状态,称为**开始状态**。识别完一个词形,词法分析器应该返回一个词汇时对应的状态,称为**接收状态**。从开始状态到接收状态形成的路径上,每条有向边的标记形成该状态转换图识别的词形。为了区分,对接收状态用带双圈的结点表示,其他状态都用带单圈的结点表示。下面以一个简单的例子来说明词法分析器是如何构造的。

### 2.3.1 词法定义例举

支持整数四则运算计算器的词法定义:

`<IDENTIFIER> ::= <Alphabet> {<Alphabet>}`

`<INTEGER> ::= <Digit> {<Digit>}`

`<TK_PLUS> ::= "+"`

`<TK_MINUS> ::= "-"`

`<TK_STAR> ::= "*"`

`<TK_DIVIDE> ::= "/"`

`<TK_ASSIGN> ::= "="`

### 2.3.2 状态转换图

图 2.2 为词法分析器的状态转换图。

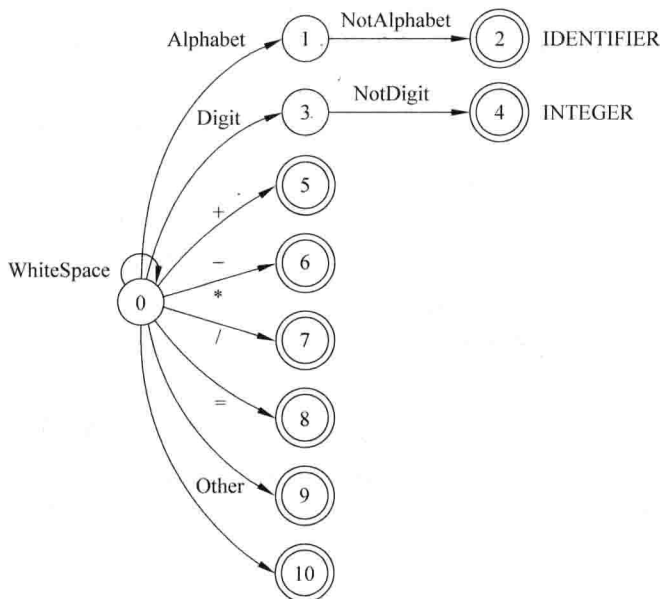


图 2.2 词法分析器的状态转换图

### 2.3.3 词法分析程序流程图

有了上述状态转换图,很容易就可以得到词法分析器的算法框图,请大家自己观察一下

状态转换图 2.2 与程序流程图 2.3 之间的关系。

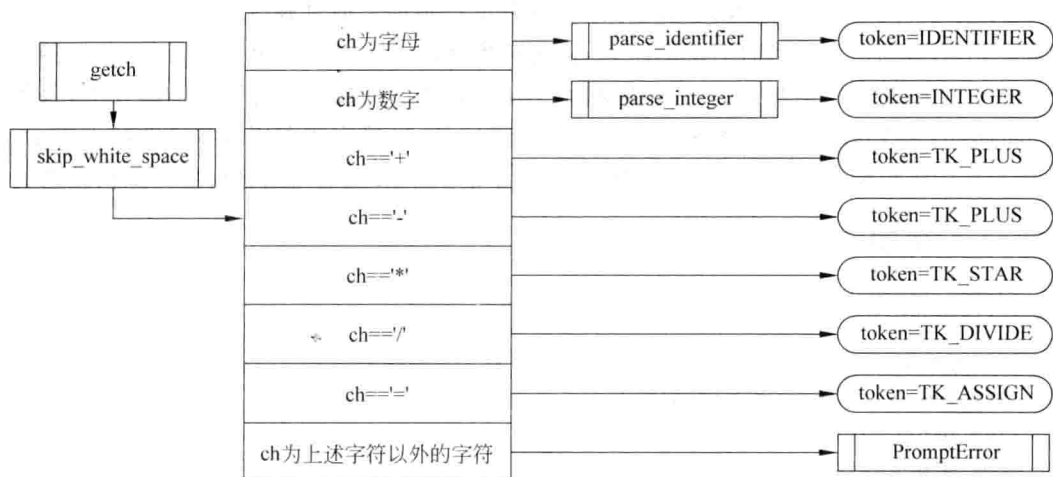


图 2.3 词法分析程序流程图

## 2.4 语法分析方法

语法分析的作用是识别由词法分析给出的单词符号序列是否是给定文法的正确句子（程序），目前语法分析常用的方法有自顶向下分析和自底向上分析两大类。所谓**自顶向下分析法**，是从文法的开始符号出发，反复使用各种产生式，寻找“匹配”于输入符号串的推导。**自底向上分析法**，则是从输入符号串开始，逐步进行“归约”到文法的开始符号。由于本书的语法分析方法采用的是自顶向下分析法，所以这里只介绍自顶向下分析法。

自顶向下分析法也称面向目标的分析方法，也就是从文法的开始符号出发企图推导出与输入的单词串完全相匹配的句子。若输入串是给定文法的句子，则必能推出，反之必然出错。自顶向下分析法对文法有一定的限制，但由于实现方法简单、直观，便于手工构造，因而仍是目前最常用的方法之一。自顶向下分析法又分为两类，不确定的自顶向下分析方法和确定的自顶向下分析方法。**不确定的自顶向下分析方法**是带回溯的分析，“不确定”的意思：当某个非终结符的产生式有多个候选，而面临当前的输入符号无法确定选用哪个产生式，从而引起回溯。这种方法实际上是一种穷举的试探方法，因此效率低，代价高，因而极少使用。**确定的自顶向下分析方法**，是从文法的开始符号出发，考虑如何根据当前的输入符号（单词符号）唯一地确定选用哪个产生式替换相应非终结符以往下推导。确定的自顶向下分析方法：对文法有一定的限制，但实现方法简单、直观，便于手工构造或自动生成语法分析器，目前仍是常用的方法之一。

### 2.4.1 LL 分析器

LL 分析器是一种自顶向下的上下文无关语法分析器。第一个 L 是指 Left to right，即从左至右扫描输入串，第二个 L 是指 Leftmost derivation，即分析过程中将使用最左推导，能以此方法分析的语法称为 LL 语法。



一个 LL 分析器若被称为  $LL(k)$  分析器,表示它向前看  $k$  个符号才可决定如何推导,即选择哪个产生式(规则)进行推导。若对某个语法而言,存在一个分析器可以在不用回溯的情况下处理这个语法,则这个语法称为  $LL(k)$  语法。在这些语法中较严格的  $LL(1)$  语法相当受欢迎,由于它的分析器只需要多看一个符号就可以产生分析结果。那些需要很大的  $k$  才能产生分析结果的编程语言,在分析时的需求也较高。

下面首先举例说明  $LL(k)$  文法,然后研究一下适合用确定的自顶向下分析方法分析的文法需要具备的条件。

### 2.4.2 $LL(k)$ 文法

我们先来看一下  $LL(0)$  文法,它根本不需要检查终结符,总能够选择适当的产生式。这种情况只发生在所有符号只有一个替换符的情形,而如果只有一个替换符就意味着语言只有一个字符串,也就是说  $LL(0)$  没有意义。下面介绍一个  $LL(0)$  文法的例子:

```
<光杆司令> ::= <军长>
<军长> ::= <师长>
<师长> ::= <旅长>
<旅长> ::= <团长>
<团长> ::= <营长>
<营长> ::= <连长>
<连长> ::= <排长>
<排长> ::= <班长>
<班长> ::= <士兵>
<士兵> ::= "王老五"
```

上面是个  $\langle$ 光杆司令 $\rangle$  的文法定义,由于每个产生式右边只有一个产生式,闭着眼睛都不会选错,所以是个  $LL(0)$  文法,但是这个  $\langle$ 光杆司令 $\rangle$  的定义实在有些无聊,一点实际意义都没有。

再来看一个  $LL(1)$  文法的例子:

```
<三国桃园结义成员> ::= "刘备"|"关羽"|"张飞"
```

只要向前看一个字符,是“刘”、“关”或是“张”,就可以从右侧 3 个可选项中做出正确选择。

再来看一个  $LL(2)$  文法的例子:

```
<某班级成员> ::= "张三"|"李四"|"王五"|"王一"
```

如果向前看一个字符,不能保证在右侧 4 个选项中选择绝对正确,例如已经知道某一位班级成员姓“王”,那么这个成员可能是“王五”,也可能是“王一”,也就是说,最不利的情况下必须向前看两个字符才能保证在上述 4 个选项中做出正确选择。

其他  $LL(k)$  文法的例子,依此类推。

### 2.4.3 $LL(1)$ 文法

$LL(1)$  文法在实际当中应用最多,所以这里重点研究一下  $LL(1)$  文法。 $LL(1)$  的含义

是：第一个 L 表明自顶向下分析是从左向右扫描输入串，第 2 个 L 表明分析过程中将使用最左推导，1 表明只须向右看一个符号便可决定如何推导，即选择哪个产生式（规则）进行推导。

### 2.4.3.1 LL(1)文法的判别

LL(1)文法的判别需要依次计算 FIRST 集、FOLLOW 集和 SELECT 集，然后判断是否为 LL(1)文法，最后再进行句子分析。

**定义** 设  $G = (V_N, V_T, P, S)$  是上下文无关文法  $\text{FIRST}(\alpha) = \{a \mid \alpha \xRightarrow{*} a\beta, a \in V_T, \alpha, \beta \in V^*\}$ ，若  $\alpha \xRightarrow{*} \epsilon$ ，则规定  $\epsilon \in \text{FIRST}(\alpha)$ ，称  $\text{FIRST}(\alpha)$  为  $\alpha$  的开始符号集或首字符集。

**定义** 设  $G = (V_N, V_T, P, S)$  是上下文无关文法， $A \in V_N$ ， $S$  是开始符号， $\text{FOLLOW}(A) = \{a \mid S \xRightarrow{*} \mu A \beta, \text{且 } a \in V_T, a \in \text{FIRST}(\beta), \mu \in V_T^*, \beta \in V^+\}$ ，若  $S \xRightarrow{*} \mu A \beta$ ，且  $\beta \xRightarrow{*} \epsilon$ ，则  $\# \in \text{FOLLOW}(A)$ 。这里用 '#' 作为输入串的结束符。

**定义** 给定上下文无关文法的产生式  $A \rightarrow \alpha, A \in V_N, \alpha \in V^*$ ，若  $\alpha \not\xRightarrow{*} \epsilon$ ，则  $\text{SELECT}(A \rightarrow \alpha) = \text{FIRST}(\alpha)$ 。如果  $\alpha \xRightarrow{*} \epsilon$ ，则  $\text{SELECT}(A \rightarrow \alpha) = (\text{FIRST}(\alpha) - \{\epsilon\}) \cup \text{FOLLOW}(A)$ 。

**定义** 一个上下文无关文法是 LL(1)文法的充分必要条件是，对每个非终结符  $A$  的两个不同产生式， $A \rightarrow \alpha, A \rightarrow \beta$ ，满足  $\text{SELECT}(A \rightarrow \alpha) \cap \text{SELECT}(A \rightarrow \beta) = \emptyset$ 。其中  $\alpha, \beta$  不同时能  $\xRightarrow{*} \epsilon$ 。

### 2.4.3.2 计算 FIRST 集

根据 FIRST 集定义对每一文法符号  $X \in V$  计算  $\text{FIRST}(X)$ ：

(1) 若  $X \in V_T$ ，则  $\text{FIRST}(X) = \{X\}$ 。

(2) 若  $X \in V_N$ ，且有产生式  $X \rightarrow a \cdots, a \in V_T$ ，则  $a \in \text{FIRST}(X)$ 。

(3) 若  $X \in V_N, X \rightarrow \epsilon$ ，则  $\epsilon \in \text{FIRST}(X)$ 。

(4) 若  $X \in V_N; Y_1, Y_2, \dots, Y_i \in V_N$ ，且有产生式  $X \rightarrow Y_1 Y_2 \cdots Y_n$ ；当  $Y_1 Y_2 \cdots Y_{i-1}$  都  $\xRightarrow{*} \epsilon$  时，(其中  $1 \leq i \leq n$ )，则  $\text{FIRST}(Y_1), \text{FIRST}(Y_2), \dots, \text{FIRST}(Y_{i-1})$  的所有非  $\{\epsilon\}$  元素和  $\text{FIRST}(Y_i)$  都包含在  $\text{FIRST}(X)$  中。

(5) 当(4)中所有  $Y_i \xRightarrow{*} \epsilon, (i=1, 2, \dots, n)$ ，则

$$\text{FIRST}(X) = \text{FIRST}(Y_1) \cup \text{FIRST}(Y_2) \cup \cdots \cup \text{FIRST}(Y_n) \cup \{\epsilon\}$$

反复使用上述(4)~(5)步直到每个符号的 FIRST 集合不再增大为止。

### 2.4.3.3 计算 FOLLOW 集

根据 FOLLOW 集定义，对文法中每一个文法符号  $A \in V_N$ ，计算  $\text{FOLLOW}(A)$ ：

(1) 设  $S$  为文法中开始符号，把  $\{\#\}$  加入  $\text{FOLLOW}(S)$  中。

(2) 若  $A \rightarrow \alpha B \beta$  是一个产生式，则把  $\text{FIRST}(\beta)$  的非空元素加入  $\text{FOLLOW}(B)$  中。

如果  $\beta \xRightarrow{*} \epsilon$  则把  $\text{FOLLOW}(A)$  也加入  $\text{FOLLOW}(B)$  中。

(3) 反复使用(2)直到每个非终结符的 FOLLOW 集不再增大为止。

### 2.4.3.4 计算 SELECT 集

根据 SELECT 集定义,对文法中每一个形如  $A \rightarrow \alpha$  产生式,计算  $\text{SELECT}(A \rightarrow \alpha)$ :

(1) 求  $\text{FIRST}(\alpha)$ ;

(2) 若  $\epsilon \notin \text{FIRST}(\alpha)$ ,则令  $\text{SELECT}(A \rightarrow \alpha) = \text{FIRST}(\alpha)$ ,否则求  $\text{FOLLOW}(A)$ ,并令  $\text{SELECT}(A \rightarrow \alpha) = \text{FIRST}(\alpha) \cup \text{FOLLOW}(A)$ 。

下面给出一个计算 FIRST 集、FOLLOW 集、SELECT 集的实例,设有文法:

```
E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → i | (E)
```

求其 3 种集合。

解:

```
FIRST(E) = FIRST(T) = FIRST(F) = {i, (}
FIRST(E') = {+, ε}
FIRST(T') = {*, ε}
FOLLOW(E) = FOLLOW(E') = {), #}
FOLLOW(T) = FOLLOW(T') = {+, ), #}
FOLLOW(F) = {*, +, ), #}
SELECT(E → TE') = FIRST(T) = {i, (}
SELECT(E' → +TE') = FIRST(+TE') = {+}
SELECT(E' → ε) = FIRST(ε) ∪ FOLLOW(E') = {ε, ), #}
SELECT(T → FT') = FIRST(F) = {i, (}
SELECT(T' → *FT') = FIRST(*FT') = {*}
SELECT(T' → ε) = FIRST(ε) ∪ FOLLOW(T') = {ε, +, ), #}
SELECT(F → i) = FIRST(i) = {i}
SELECT(F → (E)) = FIRST((E)) = {(}
```

### 2.4.4 递归子程序法

确定的自顶向下分析方法分为两种,一种是递归子程序法;另一种是预测分析法。递归子程序法是比较简单直观且易于手工构造的一种语法分析方法,本书 SCC 编译器语法分析采用的就是递归子程序法。它要求文法满足 LL(1)文法,它的实现思想是对文法中每个非终结符编写一个递归过程,每个过程的功能是识别由该非终结符推出的串,当某非终结符的产生式有多个候选时能够按 LL(1)形式可唯一地确定选择某个候选进行推导。由于递归子程序法对每个过程可能存在直接或间接递归调用,所以对某个过程在退出之前可能又被调用,因此有些信息需要保留,通常在入口时要保留某些信息,出口时需恢复。由于递归过程是遵循先进后出规律,所以通常开辟先进后出栈来处理。下面以一个简单的例子来说明如何利用递归子程序法来构造语法分析程序,这个例子包括 3 部分内容:语法定义、语法描述图和语法分析程序。

### 2.4.4.1 语法定义举例

```

<program>::=<statement>{ statement }"#"
<statement>::=<expression>"\n"
<expression>::=<multiplicative_expression>{
    "+" <multiplicative_expression>
  | "-" <multiplicative_expression>}
<multiplicative_expression>::=<primary_expression>{
    "*" <primary_expression>
  | "/" <primary_expression>}
<primary_expression>::=<INTEGER>|"(" <expression> ")"

```

### 2.4.4.2 语法描述图

本例的语法描述图如图 2.4 所示。

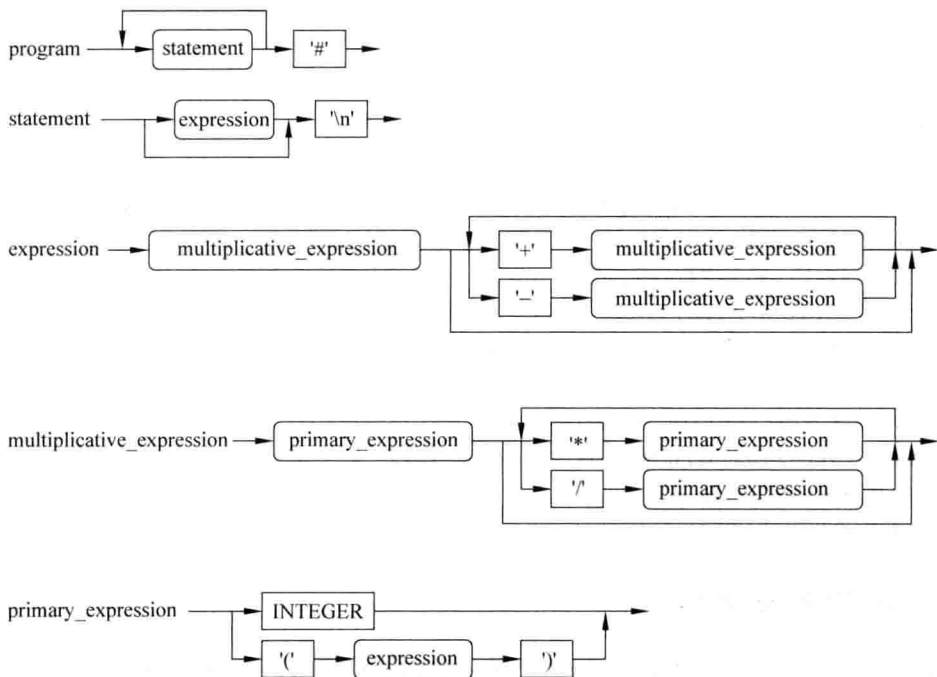


图 2.4 语法描述图

### 2.4.4.3 构造语法分析程序

```

void main()
{
    get_token();
    program();
}

void program()

```

```
{  
    while(token != '#')  
    {  
        statement();  
    }  
}
```

```
void statement()
```

```
{  
    if (token != '\n')  
    {  
        expression();  
    }  
    else  
    {  
        get_token();  
    }  
}
```

```
void expression()
```

```
{  
    multiplicative_expression();  
    if (token == '+' || token == '-')  
    {  
        get_token();  
        multiplicative_expression();  
    }  
}
```

```
void multiplicative_expression()
```

```
{  
    primary_expression();  
    if (token == '*' || token == '/')  
    {  
        get_token();  
        primary_expression();  
    }  
}
```

```
void primary_expression()
```

```
{  
    primary_expression();  
    if (token == INTEGER)  
    {  
        get_token();  
    }  
}
```

```

    }
    else if (token=='(')
    {
        get_token();
        expression();
        get_token();
        if (token==')')
            error("缺少右括号");
    }
    else
    {
        error("报错");
    }
}

```

请大家自己观察思考一下语言的文法定义、文法描述图和递归子程序法构造的语法分析程序之间的关系,第5章语法分析用到的就是这部分知识。

## 2.4.5 文法的等价变换

确定的自顶向下分析法要求给定语言的文法必须是 LL(1)形式,然而,不一定每个语言都是 LL(1)文法,对一个语言的非 LL(1)文法是否能变换为等价的 LL(1)形式以及如何变换是这里讨论的主要问题。由 LL(1)文法的定义可知若文法中含有左递归或含有左公共因子,则该文法肯定不是 LL(1)文法,因而,设法消除文法中的左递归,提取左公共因子对文法进行等价变换。

### 2.4.5.1 提取左公共因子

若文法中含有形如:  $A \rightarrow \alpha\beta | \alpha\gamma$  的产生式,这导致了对相同的产生式右部的 FIRST 集相交。即有  $SELECT(A \rightarrow \alpha\beta)SELECT(A \rightarrow \alpha\gamma) \neq \Phi$  不满足 LL(1)文法的充要条件。

现将产生式  $A \rightarrow \alpha\beta | \alpha\gamma$  等价交换为  $A \rightarrow \alpha(\beta | \gamma)$ ,可进一步引入非终结符  $A'$ ,使产生式变换为  $A \rightarrow \alpha A'$ ,  $A' \rightarrow \beta | \gamma$ 。写成一般形式为  $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n$  提取左公因子后变为  $A \rightarrow \alpha(\beta_1 | \beta_2 | \dots | \beta_n)$ 再引进非终结符  $A'$ ,变为  $A \rightarrow \alpha A'$ ,  $A' \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$ 。若在  $\beta_1, \beta_2, \beta_3 \dots$ 中仍含有左公共因子,可再进行提取,这样反复进行提取直到所引进的新非终结符的有关产生式均无左公共因子为止。

### 2.4.5.2 消除左递归

一个文法含有下列形式的产生式之一时:

- (1)  $A \rightarrow A\beta, A \in V_N, \beta \in V^*$ 。
- (2)  $A \rightarrow B\beta, B \rightarrow A\alpha, A, B \in V_N, \alpha, \beta \in V^*$ 。

则称该文法是左递归的。一个文法是左递归时,不能采取自顶向下分析法。为了使某些含有左递归的文法经过等价变换消除左递归后可能变为 LL(1)文法,可采取如下方法消除左递归。

① 消除直接左递归,把直接左递归改写为右递归,如对文法  $G: S \rightarrow Sa, S \rightarrow b$  可改写为

$$S \rightarrow bS', \quad S' \rightarrow aS' \mid \epsilon$$

一般情况下,假定关于  $A$  的全部产生式是:  $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$ , 其中,  $\alpha_i (1 \leq i \leq m)$  不等于  $\epsilon$ ,  $\beta_j (1 \leq j \leq n)$  不以  $A$  开头,消除直接左递归后改写为

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A', A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon$$

② 消除间接左递归。

对于间接左递归的消除需先将间接左递归变为直接左递归,然后再按①消除直接左递归。

## 第 3 章

# SC语言定义

合抱之木，生于毫末；九层之台，起于累土；千里之行，始于足下。

——老子

盖一栋大楼之前，要先绘出这幢大楼的蓝图来指导施工。搞社会主义现代化建设，要由社会主义现代化的宏伟蓝图来指引前进的方向。本章就要勾勒出 SC 语言的蓝图，来指导 SCC 编译器的开发。有了第 2 章关于文法知识的铺垫，本章要定义一下 SC 语言。以前用 C 语言写程序，只有老老实实在地按 C 语言的语法规则写。现在我们由言听计从的规则执行者，变成了规则制定者，这种感觉很是不错，可是肩上也多了一份沉甸甸的责任，因为要对 SC 语言负责，这个语言有没有生命力，语言功能定位非常重要。

### 3.1 SC 语言的蓝本选择

SC 语言以 C 语言为蓝本进行简化，但 C 语言也有多个不同的版本，那么 SC 语言以 C 语言哪个版本为蓝本呢。下面简单介绍一下 C 语言自诞生到现在，出现的几个主要版本。

#### 3.1.1 K&R C

早期的 C 语言还没有标准化，源自 Kernighan & Ritchie 合著的 *The C Programme Language* 的 C 描述可算作“正式”的标准，所以此时的 C 也称为 K&R C。之后 C 语言一直不断地发生细微的变化，各编译器厂商也有自己的扩展，这个过程一直持续到 20 世纪 80 年代末。

#### 3.1.2 C89

考虑到标准化的重要，ANSI(American National Standards Institute)制定了第一个 C 标准，在 1989 年被正式采用(American National Standard X3.159—1989)，故称为 C89，也称为 ANSI C。该标准随后被 ISO 采纳，成为国际标准(ISO/IEC 9899:1990)。

C89 的主要改动：

- 定义了 C 标准库；
- 新的预处理命令和特性；
- 函数原型(prototype)；
- 新关键字 `const`、`volatile`、`signed`；
- 宽字符、宽字符串和多字节字符；
- 转化规则、声明(declaration)、类型检查的改变。



### 3.1.3 C99

这是目前最新的标准,由 ISO 于 1999 年制定(ISO/IEC 9899:1999),故称为 C99。

C99 的主要改动反映在以下几个方面:

- 复数(complex);
- 整数(integer)类型扩展;
- 变长数组;
- Boolean 类型;
- 非英语字符集的更好支持;
- 浮点类型的更好支持;
- 提供全部类型的数学函数;
- C++ 风格注释(//)。

C99 是当前的标准,但它仍未得到广泛支持,虽然标准发布已经多年。C99 对 C89 的改动非常大,如果编写 C99 的代码,那么可移植性必然受到限制。此外,个人认为 C99 的一些新特性在大多数程序设计中并不是必须的。C89 目前仍然使用最广泛,并得到所有主流编译器的支持。K&R C 现在只会有一些非常旧的代码中才能见到了,除非要维护旧代码,否则不应该再使用它。

所以,我们最终决定 SC 语言以 C89 为蓝本进行简化。

## 3.2 SC 语言对 C89 简化原则

下面首先说一下 SC 语言对 C 语言进行简化的原则:

- (1) 支持单字节、双字节、四字节的的基本数据类型;
- (2) 支持数组、结构体;
- (3) 支持字符串;
- (4) 支持函数、局部变量、全局变量;
- (5) 支持条件语句、循环语句;
- (6) 支持基本的算术运算、关系运算;
- (7) 能用多种方式实现的功能,只保留一种,例如 C 语言的循环语句有 for 循环、do 循环、while 循环,只保留一种;
- (8) 原来 C 语言中绝大多数人用不到的一些功能去掉,例如 auto 关键字恐怕没有人用到过。

## 3.3 SC 语言的字符集

字符是组成语言的最基本的元素。SC 语言字符集由字母、数字、空格、标点和特殊字符组成。在字符常量、字符串常量和注释中还可以使用汉字或其他可表示的图形符号。

SC 语言包括两个字符集,一个是用于书写 SC 语言源文件的字符集,称为源码字符集;另一个是 SC 语言编译后在执行环境中解释的字符集,称为执行字符集。除在字符常量及

字符串常量中出现的转义字符外,两个字符集完全相同。

两个字符集中成员又分为两类,一类为基本字符集,这些字符可以出现在代码的任何位置;另一类为扩展字符集,扩展字符集包含基本字符集,扩展字符集中除基本字符集外的其他字符只能出现在源码的注释和字符串中。在字符常量或字符串常量中,执行字符集中的成员应当用对应的源字符集成员或者用由一个反斜线字符\后紧跟一个字符所组成的转义序列来表示。在基本执行字符集中应有一个其字节的所有位均置为0的,称为空字符的字符,空字符用来终止一个字符串。

### 3.3.1 基本字符集

基本源字符集和基本执行字符集均应至少具有下列成员。

#### 1. 字母与下划线

##### (1) 字母。

- 英文字母表中的 26 个小写字母。

```
a b c d e f g h i j k l m
n o p q r s t u v w x y z
```

- 英文字母表中的 26 个大写字母。

```
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
```

##### (2) 下划线\_。

#### 2. 10 个数字

```
0 1 2 3 4 5 6 7 8 9
```

#### 3. 标点和特殊字符

```
+ - * / % = ! < > . &
( ) [ ] { } ; , \ " ' ,
```

#### 4. 空白符

空格符、制表符、换行符等统称为空白符。空白符只在字符常量和字符串常量中起作用。在其他地方出现时,只起间隔作用,编译程序对它们忽略不计。因此在程序中使用空白符与否,对程序的编译不发生影响,但在程序中适当的地方使用空白符将增加程序的清晰性和可读性。

#### 5. 空字符

字节的所有位均置为0的,称为空字符,空字符用来终止一个字符串。

### 3.3.2 扩展字符集

源字符集中可包含多字节字符,用于表示扩展字符集的成员。执行字符集中也可包含扩展字符集。下列条件对两个字符集均应成立:

- 应包含基本字符集所定义的单字节字符;
- 任何附加成员的存在、含义和表示均是地域特定的,是跟编译器的实现相关的,编译器有权决定支持哪些附加成员。

### 3.4 SC 语言词法定义

SC 语言的单词可以分为以下几类：关键字、标识符、整数常量、字符常量、字符串常量、运算符及分隔符、注释。

#### 3.4.1 关键字

##### 1. 定义

```
<char 关键字>::="char"
<short 关键字>::="short"
<int 关键字>::="int"
<void 关键字>::="void"
<struct 关键字>::="struct"
<if 关键字>::="if"
<else 关键字>::="else"
<for 关键字>::="for"
<continue 关键字>::="continue"
<break 关键字>::="break"
<return 关键字>::="return"
<sizeof 关键字>::="sizeof"
<__cdecl 关键字>::="__cdecl"
<__stdcall 关键字>::="__stdcall"
<__align 关键字>::="__align"
```

##### 2. 语义

关键字是由语言规定的具有特定意义的字符串,通常也称为保留字,用户定义的标识符不应与关键字相同。SC 语言这些关键字的功能如表 3.1 所示。

表 3.1 SC 语言关键字表

| 分 类     | 名 称      | 含 义                   |
|---------|----------|-----------------------|
| 数据类型关键字 | char     | 声明字符型变量或函数            |
|         | short    | 声明短整型变量或函数            |
|         | int      | 声明整型变量或函数             |
|         | void     | 声明函数无返回值,声明无类型指针      |
|         | struct   | 声明结构体变量               |
| 控制语句关键字 | if       | 条件语句                  |
|         | else     | 条件语句否定分支(与 if 连用)     |
|         | for      | 循环语句                  |
|         | continue | 结束当前循环,开始下一轮循环        |
|         | break    | 跳出当前循环                |
|         | return   | 子程序返回语句(可以带参数,也可不带参数) |

续表

| 分 类       | 名 称       | 含 义                     |
|-----------|-----------|-------------------------|
| 类型长度计算关键字 | sizeof    | 计算类型长度                  |
| 函数调用约定关键字 | __cdecl   | __cdecl 调用约定            |
|           | __stdcall | __stdcall 调用约定          |
| 结构成员对齐关键字 | __align   | __align(n)强制结构成员对齐到 $n$ |

### 3.4.2 标识符

#### 1. 定义

```

<标识符> ::= <非数字> {<数字> | <非数字>}
<非数字> ::= "_" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n"
              | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
              | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M"
              | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
<数字> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

```

#### 2. 解释

标识符只能是字母(A~Z, a~z)、数字(0~9)、下划线(\_)组成的字符串,并且其第一个字符必须是字母或下划线。

#### 3. 语义

标识符可以代表很多实体:函数、变量、结构体名称、结构体成员、数组名。变量是一个存储区域,它的解释依赖于两个主要属性:它的存储类型和它的数据类型。存储类型决定了与该标识符相关联的存储区域的生命周期,数据类型决定了该对象值的含义。一个名字还有一个作用域,作用域即程序中可见此名字的区域。

#### 4. 举例

以下标识符是合法的:

a、x、x3、BOOK\_1、sum5。

以下标识符是非法的:

3s 以数字开头。

s \* T 出现非法字符 \*。

bowy-1 出现非法字符-(减号)。

#### 5. 使用标识符需要注意的问题

(1) 在标识符中,大小写是有区别的,例如 BOOK 和 book 是两个不同的标识符。

(2) 标识符虽然可由程序员随意定义,但标识符是用于标识某个量的符号。因此,命名应尽量有相应的意义,以便于阅读理解,做到“顾名思义”。

(3) SC 语言的关键字不能用作标识符名称。



表 3.2 中列出的字符称为“转义字符”，意思是将反斜杠\后面的字符转换成另外的意义。如\n 中的 n 不代表字母 n 而作为“换行”符。

### 3. 语义

整型字符常量的类型是 int，如字符常量包含单个字符或转义序列，则它的值是将该单字符或转义序列值的 char 类型的对象转换为 int 类型时所得到的值。包含多于一个字符，或包含一个不在基本执行字符集中表示的字符或转义序列的整型字符常量的值是实现定义的。

## 3.4.5 字符串常量

### 1. 定义

<字符串常量>::="(<串字符>)"

<串字符>::=<转义字符>|源字符集中除双引号字符"、反斜线字符\或换行字符外的任何字符

### 2. 解释

字符串常量是由双引号括起来的零个或多个多字节字符组成的序列，例如“盗版严重阻碍了中国的经济发展和自主创新”。

### 3. 语义

字符串常量实际上是字节数组，该数组的元素类型为 char，用该多字节字符序列中的各个字节初始化数组的各个元素。该数组长度为表示该字符串的多字节字符序列的长度加 1，多出的这最后一个字节的值为 0，标识字符串的结束。

## 3.4.6 运算符及分隔符

### 1. 定义

<加号>::="+"

<减号>::="-"

<星号>::="\*"

<除号>::="/"

<取余号>::="%"

<等于号>::="=="

<不等于号>::="!="

<小于号>::="<"

<小于等于号>::="<="

<大于号>::=">"

<大于等于号>::=">="

<赋值等号>::="="

<箭头>::="->"

<点号>::=","

<与号>::="&"

<左小括号>::="("

<右小括号>::=")"

<左中括号>::="["

---

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

**备用QQ:2404062482**