高等教育规划教材

编译技术

周尔强　周帆　韩蒙　陈文宇　编著

提供电子教案

下载网址 http://www.cmpedu.com

机械工业出版社
CHINA MACHINE PRESS

高等教育规划教材

编译技术

周尔强　周　帆　韩　蒙　陈文宇　编著

机械工业出版社

本书主要内容编排如下：第1章介绍编译器整体结构；第2章介绍一个简单的编译程序构造过程；第3章至第6章分别介绍词法分析、语法解析、语义分析、代码生成等过程中所面临的技术问题及解决方案；第7章介绍运行时存储空间的组织与分配；第8章介绍LCC（Learning Compiler with C）语言编译程序的C语言实现。本书在强调基础理论的同时，力求反映编译技术方面的最新成果，书中给出了大量代码，以帮助读者掌握编译器构造的相关技术。

本书文字简洁易懂，内容循序渐进、深入浅出，便于自学，适合作为高等学校计算机类专业的教材，也可作为软件工程技术人员的参考书。

本书配套授课电子课件，需要的教师可登录www.cmpedu.com免费注册，审核通过后下载，或联系编辑索取（QQ：2850823885，电话：010-88379739）。

图书在版编目（CIP）数据

编译技术/周尔强等编著. -北京：机械工业出版社，2015.9
高等教育规划教材
ISBN 978-7-111-50911-0

Ⅰ.①编… Ⅱ.①周… Ⅲ.①编译程序-程序设计-高等学校-教材
Ⅳ.①TP314

中国版本图书馆CIP数据核字（2015）第183750号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）
策划编辑：郝建伟 责任编辑：郝建伟
责任校对：张艳霞
责任印制：李 洋
北京振兴源印务有限公司印刷
2015年9月第1版·第1次印刷
184mm×260mm · 14.5印张 · 359千字
0001—3000册
标准书号：ISBN 978-7-111-50911-0
定价：39.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务
服务咨询热线：（010）88379833
读者购书热线：（010）88379649

网络服务
机 工 官 网：www.cmpbook.com
机 工 官 博：weibo.com/cmp1952
教育服务网：www.cmpedu.com
金 书 网：www.golden-book.com

出 版 说 明

当前，我国正处在加快转变经济发展方式、推动产业转型升级的关键时期。为经济转型升级提供高层次人才，是高等院校最重要的历史使命和战略任务之一。高等教育要培养基础性、学术型人才，但更重要的是加大力度培养多规格、多样化的应用型、复合型人才。

为顺应高等教育迅猛发展的趋势，配合高等院校的教学改革，满足高质量高校教材的迫切需求，机械工业出版社邀请了全国多所高等院校的专家、一线教师及教务部门，通过充分的调研和讨论，针对相关课程的特点，总结教学中的实践经验，组织出版了这套“高等教育规划教材”。

本套教材具有以下特点：

1）符合高等院校各专业人才的培养目标及课程体系的设置，注重培养学生的应用能力，加大案例篇幅或实训内容，强调知识、能力与素质的综合训练。

2）针对多数学生的学习特点，采用通俗易懂的方法讲解知识，逻辑性强、层次分明、叙述准确而精炼、图文并茂，使学生可以快速掌握，学以致用。

3）凝结一线骨干教师的课程改革和教学研究成果，融合先进的教学理念，在教学内容和方法上做出创新。

4）为了体现建设“立体化”精品教材的宗旨，本套教材为主干课程配备了电子教案、学习与上机指导、习题解答、源代码或源程序、教学大纲、课程设计和毕业设计指导等资源。

5）注重教材的实用性、通用性，适合各类高等院校、高等职业学校及相关院校的教学，也可作为各类培训班教材和自学用书。

欢迎教育界的专家和老师提出宝贵的意见和建议。衷心感谢广大教育工作者和读者的支持与帮助！

机械工业出版社

前　言

“编译原理”课程是计算机专业一门重要的专业基础课，也是计算机系统软件课程中非常重要的一个分支。在众多的原理性学习课程中，编译原理主要承担了语言实现原理、方法和技术的介绍。该课程内容有一定深度和难度，且综合性比较强，对学生专业知识掌握情况要求也比较高，学生在学习过程中会感到内容抽象、算法复杂，是一门公认的比较难学、比较难教的课程。很多学生也认为“编译原理”只能应用在实现程序语言的编译器上，而他们以后可能不会在编译器及其相关领域方面钻研，所以学习兴趣不高。

其实这是一种误解。通过学习编译程序的构造原理和技术，将有助于深刻理解和正确使用程序设计语言。如正规式和有穷自动机在文本编辑器中的广泛应用。有穷自动机在字符串查找中的运用、必经结点算法在网络中的运用，以及由文法来定义网络协议等。

此外，虽然编译原理基本内容已相对比较成熟，算法相对固定，但编译技术作为计算机语言发展的支柱，是计算机科学中发展最迅速、最成熟的一个分支，特别是近几年大量编译辅助工具应运而生，大大简化了编译器的实现过程。

基于以上因素，本书在编排时以提高学生的动手实践能力为重点，在选择性讲解必要的理论及算法的同时，鼓励学生尝试设计并实现一个新的语言编译器，在此基础上将编译程序中的各种算法和技术应用到各个领域，从而激发学生的创造性思维，培养学生的创新能力，为今后的学习、工作打下坚实的基础。

本书在总体上介绍现代编译系统构造过程中的基本实现技术和一些自动构造工具，旨在让学生掌握编译器构造技术的最新进展，并在此基础上能够根据实际需求快速而高效地实现特定语言的编译器。

全书章节安排如下：第 1 章是编译器的总体介绍，即现代编译器是如何组织的，其典型结构是什么。第 2 章给出了一个比较简单的例子，其目的是使学生对编译器各组成部分有一个感性的认识，使其理解不光停留在理论层面，而是付诸于实践。第 3 章至第 6 章则分别对典型编译器的各组成部分的实现细节及最新技术进行了介绍。第 7 章从理论的角度对运行时存储空间的组织与分配进行了介绍。作为总结，第 8 章给出了一个编译器设计与构造的完整实例，即以 C 语言为基础，设计并实现了 LCC（Learning Compiler with C）语言的编译程序，以帮助学生从整体上理解并掌握编译器构造相关技术，能够独立地完成编译器开发任务。作为教材，每章后均附有习题。书中除介绍了 LEX、YACC 使用方法外，还重点介绍了编译器基础软件框架 LLVM 及相关工具的使用实例。

本书作为高等院校编译技术的教材，体现了编译课程改革的方向。本课程建议授课学时为 64 学时，其中实验学时不少于 20 学时，并要求先修 C 语言、数据结构及汇编语言等课程。

本书由周尔强、周帆、韩蒙、陈文宇编写；全书由陈文宇组织与审阅。

对于书中存在的不足之处，恳请读者批评指正。

编　者

目　录

第1章 编译概述

程序设计语言是描述计算过程的符号，一般可分为两大类：第一类称为低级语言，包括机器语言和汇编语言等；第二类称为高级语言，如 C、C++、Java、Python、Perl 等。高级语言不论是在算法描述的能力上，还是在编写和调试程序的效率上，都远比低级语言优越。

然而，计算机硬件只能直接执行本机器对应的机器语言代码，而不能直接执行用高级语言或汇编语言编写的程序。因此，在设计并实现高级语言或汇编语言时，就需要使该语言能够被计算机所“理解”，解决这一问题的方法有两种：一种是对该语言所编写的程序进行翻译，另一种是对程序进行解释。

1.1 编译器与解释器

简单来说，一个编译器（Compiler）就是一个程序，它可以将某一种语言（源语言）编写的程序等价地翻译成为另一种语言（目标语言）编写的程序。如果一个翻译程序的源语言是某种高级语言，其目标语言是相当于某一计算机的汇编语言或机器语言，则称这种翻译程序为编译程序（或编译器）。编译器并不执行源代码，而是一次将其翻译成另一种语言，如汇编语言或机器码。如果目标程序是一个可执行的机器语言程序，那么它就可以被用户执行，处理输入并产生输出，整个执行过程不再需要编译器干预及支持，如图 1-1a 所示。编译器的主要任务之一是发现并报告它在翻译过程中发现的源程序中的错误。因为一些程序的错误只有在运行时才能显现出来，所以并不是所有的源程序错误都能被编译器诊断出来。

> 📖 将汇编语言（源语言）翻译为机器语言（目标语言）的程序称为汇编程序。

解释器（Interpreter）是另一种常见的语言处理程序，它不产生源程序的目标代码，也不会一次把整个程序全部翻译出来，而是在每次运行程序时，每读取并翻译一行程序代码就立刻运行，然后再读取并翻译下一行，再运行，如此不断地进行下去。需要注意的是，源程序每次执行时都需要解释器对其进行翻译，而不像编译器那样只需要翻译一次，图 1-1b 所示展示了这种解释执行方法。解释执行的主要优点是便于对源程序进行调试和修改，解释器的错误诊断效果通常比编译器要好，但其运行效率相对编译方式来讲要低，即程序运行相对较慢。

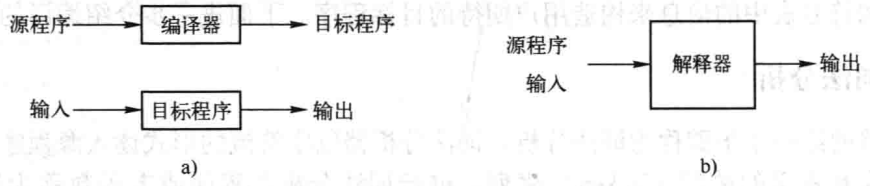

图 1-1 程序的编译执行与解释执行

a）编译执行 b）解释执行

一些高级语言的翻译程序结合了编译和解释两种方法。例如 Java 源程序首先被编译成一个称为字节码（Bytecode）的中间表示形式，然后由一个虚拟机对得到的字节码进行解释执行。这样安排的好处之一是在一台机器上编译得到的字节码可以在另一台机器上解释执行。

除了编译器之外，在创建一个可执行程序时还需要一些其他程序。例如一个较为复杂的程序可能被分割为多个模块，并存放于对应的源文件中。把源程序聚合在一起的任务有时会由一个被称为预处理器（Preprocessor）的程序完成。预处理器还负责把那些称为宏的缩写形式转化为源语言的语句，然后将经过预处理的源程序文件作为输入传递给编译器。

1.2 编译器的组织与结构

编译程序的内部结构及组织方式虽各有不同，但其主要工作可分为两大部分：分析（Analysis）与综合（Synthesis）。所谓分析，即对被编译的源程序进行词法、语法、语义等分析，得到与目标机器无关的中间代码；所谓综合，是在分析正确无误之后，综合所得到的中间代码及各种信息，最终得出可以执行的机器语言程序。

如果更加详细地研究编译过程，会发现它顺序执行了一组步骤。每个步骤把源程序的一种表示方式转换成另一种表示方式。一个典型的把编译程序分解成为多个步骤的方式如图 1-2 所示。在实践中，多个步骤可能被组合在一起，而这些组合在一起的步骤之间的中间表示不需要被明确地构造出来。

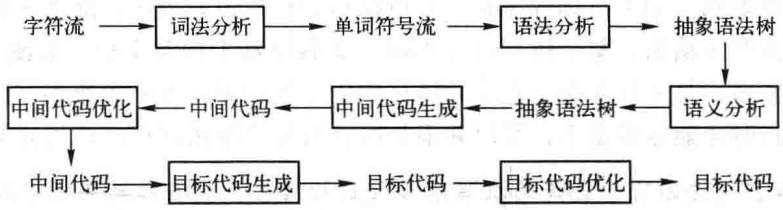

图 1-2 编译器的各个步骤

在这些步骤中，分析部分把源程序分解成为多个组成要素，并在这些要素之上加上语法结构。然后，它使用这个结构来创建该源程序的一个中间表示。如果分析部分检查出源程序没有按照正确的语法构成，或者语义上不一致，它就必须提供有用的信息，使得用户可以按此进行改正。分析部分还会收集有关源程序的信息，并把信息存放在一个称为符号表（Symbol Table）的数据结构中。符号表将和中间表示一起传送给综合部分。综合部分根据中间表示和符号表中的信息来构造用户期待的目标程序。下面进一步介绍编译的各个步骤。

1.2.1 词法分析

编译器的第一个步骤称为词法分析。词法分析器以字符流的形式读入源程序，并且将它们组织成为有意义的单词（Token）序列。进行词法分析的程序或者函数称为词法分析器（Lexical Analyzer，简称 Lexer），也称扫描器（Scanner）。词法分析器一般以函数的形式存在，供语法分析器调用。词法分析器的工作任务如下：

1）识别出源程序中的单词符号，即语言所允许的语法符号。

2）删除无用的空白字符、回车符，以及其他与输入介质相关的非实质性字符。

3）删除注释。

4）进行词法检查，即源程序中的字符序列是否符合语言的词法规则，并报告所发现的错误。

对于每个单词，词法分析器产生如下形式的二元式作为输出：

```
<单词类别,单词属性>
```

例如，对于下面的赋值语句：

```
expr = expr * 5 + 3.14;                                    (1.1)
```

在进行词法分析时，其处理过程大致如下：

1）最左边的字符串“expr”是源程序中一个基本的语法单位（即变量），被词法分析器识别为“标识符”，其属性为该标识符的名称，即字符串“expr”，因此，字符串“expr”对应的词法输出为：<标识符,“expr”>。

2）赋值符号“=”是一个基本的语法单位，且每个符号对应一个类别，因此词法分析器将其映射为“赋值符号”，因符号类别能够完全表示其含义，因此符号不需要属性值，在输出时用“NULL”表示，最终“=”对应的词法输出为：<赋值符号,NULL>。

3）赋值符号右侧的字符串“expr”是一个基本的语法单位，应被识别为“标识符”，对应的词法输出为：<标识符,“expr”>。

4）赋值符号“*”是一个基本的语法单位，词法分析器将其映射为“乘号”，因此其对应的词法输出为：<乘号,NULL>。

5）整数“5”是一个基本的语法单位，对于常数，词法分析器将各类型的常数映射为相应的类别，属性值为常数的数值。这里将“5”映射为整数，其属性值为“5”，因此其对应的词法输出为：<整数,5>。

6）赋值符号“+”是一个基本的语法单位，词法分析器将其映射为“加号”，因此其对应的词法输出为：<加号,NULL>。

7）浮点数“3.14”是一个基本的语法单位，词法分析器将其映射为浮点数，其属性值为“3.14”，因此其对应的词法输出为：<浮点数，3.14>。

最终赋值语句 1.1 对应的词法分析输出结果为：

```
<标识符,"expr">
<赋值符号,NULL>
<标识符,"expr">
<乘号,NULL>
<整数,5>
<加号,NULL>
<浮点数,3.14>
```

通常，用枚举类型定义单词的类别，特别地，对于符号，用其对应的 ASCII 编码表示该

符号的类别，而其他符号的类别则从 258 开始编号。例如定义：

```
enum { IDENTIFIER = 258, INTEGER, FLOAT, ……};
```

则赋值语句 1.1 对应的词法分析输出结果为：

```
< 258,"expr" >
< '=',NULL >
< 258,'expr' >
<'*',NULL >
< 259,5 >
<'+',NULL >
< 260,3.14 >
```

词法分析程序经常作为一个函数被语法分析器调用，在这种情况下，词法分析器每次返回一个单词记号，包括单词类别及单词属性。

1.2.2 语法分析

作为编译的第 2 个步骤，语法分析需要根据所得到的单词记号序列构建源程序的结构。更具体地讲，语法分析器根据由词法分析器返回结果的第一个分量（即单词的类别）来创建树形的中间表示，该中间表示给出了词法分析产生的词法单元流的语法结构。一个常用的表示方法是抽象语法树（Abstract Syntax Tree，AST），树中的每个结点表示一个语法元素。

以赋值语句 1.1 为例，其对应的抽象语法树如图 1-3 所示。其中，树中的内部结点表示一个运算，而该结点的子结点表示该运算的操作数。

该树显示了赋值语句中各个运算的执行顺序，树中有一个标号为 * 的内部结点，其左子结点为标识符 expr，右子结点为整数 5，标号为 * 的内部结点表明需要将 expr 与 5 相乘。而树中标号为 + 的内部结点则表示需要将相乘的结果与浮点数 3.14 相加。该树根结点的标号为 =，表示需要将相加的结果保存到标识符 expr 相应的存储位置中。这样的运算顺序与我们期望的运算顺序一致。

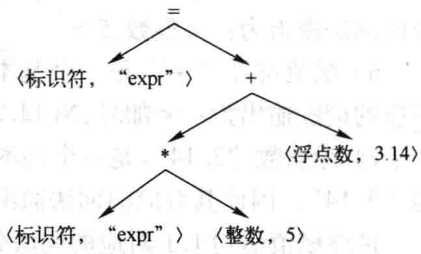

图 1-3 赋值语句（1.1）对应的抽象语法树

在语法分析过程中，语法分析程序试着为源程序构造一棵完整的语法树。如果这种尝试成功，表明该输入串在结构上的确是一个合乎语法的程序，否则，源程序中就必然存在语法错误。所得到的语法树实质上是一个有标记的有序树形结构。

1.2.3 语义分析

经过语法分析后，虽然明确了它的组成结构，但并不知道其中所出现的一些量的属性和

意义，更不知道各语法结构具有何种功能。要得知各语法成分的含义和用途，以及应进行的运算和操作，就需要进行编译的第3个步骤：语义分析。在该步骤中，分析器使用语法树和符号表中的信息来检查源程序是否与语言定义的含义和功能一致。它同时也收集类型信息，并把这些信息存放在语法树或符号表中，以便在随后的中间代码生成过程中使用。

语义分析的一个重要部分是类型检查，编译器检查每个运算符是否具有匹配的运算分量。比如，很多程序设计语言的定义中要求一个数组的下标必须是整数。如果用一个浮点数作为数组下标，编译器就必须报告错误。

程序设计语言可能允许某些类型转换，这被称为自动类型转换。比如，一个二元算术运算符可以应用于一对整数或者一对浮点数。如果这个运算符应用于一个浮点数和一个整数，那么编译器可以把该整数转换成为一个浮点数。

通常，语义分析过程所需进行检查的项目十分繁杂。对一些常见的语言来说，需要检查：在说明语句中是否有矛盾的类型说明；在表达式中，对某些运算符而言，是否有类型不匹配的运算对象；在过程调用中，实际参数与形式参数是否在个数、次序、种属等方面按相应语言的规定进行对应等。由此可见，在语义分析过程中，语义分析程序也需要进行频繁的查表和填表工作。

1.2.4 代码生成与优化

在源程序的语法分析和语义分析完成之后，很多编译器生成一个明确的、低级的或类机器语言的中间表示。这样就可以把这个表示看作是某个抽象机器的程序。该中间表示应该具有两个重要的性质：它应该易于生成，且能够被轻松地翻译为目标机器上的语言。

中间代码的生成是与语义分析紧密相连的。但由于迄今对于程序语言的语义描述还没有一个公认的形式化系统，因此，对编译程序中间代码生成部分的设计，在一定程度上仍凭借经验来完成。对于采用语法制导翻译的编译程序，通常的做法是将产生中间代码的工作交给语义过程来完成。即每当一个语义过程被调用而对相应的语法结构进行语义分析时，它就根据此语法结构的语义，结合在分析时所获得的语义信息，产生相应的中间代码，再把后者放到中间代码的序列中去。

目前常见的中间代码形式有逆波兰表示、三元式、四元式及树形结构等。在第6章中，将介绍LLVM（Low Level Virtual Machine）编译器基础架构框架下的虚拟指令系统，该指令系统与程序语言无关。在赋值语句（1.1）中，假如变量expr为整型，则该语句对应的LLVM指令序列如下。

```
1) %1 = load i32 * %expr, align 4
2) %2 = mul nsw i32 %1,5
3) %3 = sitofp i32 %2 to double
4) %4 = fadd double %3,3.140000e+00
5) %5 = fptosi double %4 to i32
6) store i32 %5,i32 * %expr, align 4
```

其中第1条指令是从内存中读取变量expr的值并保存到临时变量%1中；第2条指令将取到的值（用临时变量%1表示）与5相乘，并将结果保存到临时变量%2中，此时，临时

变量%1 和%2 都是整型；第 3 条指令将乘法计算所得的值先转换为系统默认的浮点类型，即在 32 位机器上将转换为 32 位的浮点类型（类似于 C 语言的 float 类型），在 64 位机器上将转换为 64 位的浮点类型（类似于 C 语言的 double 类型）；在类型转换之后，第 4 条指令将两值相加，所得到的和（浮点类型）保存到临时变量%4 中；因该值最终要保存于整型变量 expr 中，因此第 5 条指令再次进行类型转化，即将上一步所得到的和%4 转换为整型；最后，第 6 条指令将计算结果保存回变量 expr 中，完成赋值操作。鉴于 LLVM 基础架构框架已成为编译器开发，特别是指令生成与优化中强大的基础工具，本书在中间代码生成部分将使用该指令系统。

一般来说，一个不进行优化处理的中间代码和目标代码的质量是比较低的。这里的质量通常有两个衡量的标准，即程序所占用存储空间的大小（空间指标）和程序运行时所需的时间（时间指标）。因此需要对中间代码及目标代码分别进行优化，如果从与具体计算机的关系上看，可分为与机器无关的优化和与机器相关的优化。如果从与源程序的关系看，又可分为局部优化和全局优化。应当指出，对于一个进行优化处理工作的编译程序而言，虽然它在工作时可得到质量较高的目标程序，然而却是以增加编译程序本身的时空复杂度和可靠性作为代价的。另外，也有这样一些优化项目，它们在时间效率和空间效率上是相互矛盾的。因此在设计一个编译程序时，究竟应考虑哪些优化项目，以及各种优化项目进行到何种程度，应权衡利弊，根据具体情况而定。

1.2.5 符号表管理及错误处理

在编译过程中，需要经常收集、记录或查询源程序中所出现的各种量的有关属性（信息）。这些属性可以提供标识符的存储分配信息、类型信息、作用域信息等。对于函数标识符，还有参数信息，包括参数的个数及类型，以及实参形参结合的方式等。为此，编译程序需要建立或持有一批不同用途的表格（如常数表、各种名字表、循环层次表等），通常将它们统称为符号表。

从实现的角度看，符号表是一种含记录的数据结构，通常一个标识符在符号表中占一个记录，记录中除了标识符的名字域之外，还有记录该标识符的属性的域。符号表在编译过程中使用频繁，是影响编译速度的主要因素。因此，对符号表的组织的要求是存储与查找的效率。

程序人员在编写程序时难免会出现错误。例如，在词法分析中，可能发现字符拼写错误。在语法分析中，需要检查单词序列是否违反语言的结构规则。在语义分析中，编译程序需要进一步查出是否存在语法上虽正确但含有无意义的操作。例如，当两个标识符相加时，如果一个是数组名，另一个是函数名，虽然语法上合法，但语义有错。诸如此类的各种错误，都在相应的阶段进行处理。一个比较完善的编译程序应具有广泛的程序查错能力，并能准确地报告源程序中错误的种类及错误出现的位置。同时，编译程序还应具有一定的“校错”能力。

1.3 总结与展望

上面简要介绍了编译程序的功能、结构、工作流程及组成编译程序的一些相关问题。对

于图 1-2 所示的编译器工作流程，不难看出，从对源程序的词法分析开始到中间代码生成，编译程序所完成的处理工作只依赖于源语言，而与运行目标程序的计算机（或目标语言）无关，通常将上述各个环节统称为编译程序的前端（Front End），它相当于 1.2 节中所说的编译程序的分析部分。属于综合部分的代码优化和目标代码生成，一般只依赖于目标语言，通常将它称为编译程序的后端（Back End）。不过，代码优化程序中的一些优化项目也可与目标语言无关，但为一致起见，也将它们列入后端。如果能将编译程序严格划分为前端和后端两个相对独立的部分，并以中间代码作为其间信息交流的载体，以这样的模式来构造编译程序，将会给编译程序的开发和维护带来许多好处。例如，当需要改动源语言时（如源语言版本升级），只需对编译程序的前端进行相应的修改即可。又如，当需要变更一种语言的目标计算机时（即移植编译程序），则只需对新的目标计算机重新开发相应的后端即可。

目前比较流行的 GCC（GNU Compiler Collection）就是一种能处理 C、C ++ 、Objective C、Fortran、Java 和 Ada 等多种语言的编译程序的集合。这些编译器都生成同一种 AST 形式的中间代码，其后端对 AST 形式的中间代码进行综合处理，最终产生相应的目标代码。

为了更进一步地了解编译程序各步骤的功能，使读者对编译器有一个较为全面的认识，在下一章中将首先讨论并实现一个简单的编译器，然后再按图 1-2 所示的结构，对一个编译程序进行剖析，并在后续的各章中分别介绍编译程序各个部分的构造原理和实现方法。

1.4 习题

1. 何谓源程序、目标程序、翻译程序、编译程序和解释程序？它们之间有何种关系？
2. 一个典型的编译系统通常由哪些部分组成？各部分的主要功能是什么？
3. 列出'C 语言中的全部关键字，并编写程序从 C 语言源程序中识别这些关键字。
4. 编译器和解释器之间的区别是什么？编译器相对于解释器的优点是什么？解释器相对于编译器的优点是什么？

第2章　实现一个简单编译器

本章通过开发一个简单的编译器来演示第1章所介绍的各编译步骤，进而引入相关理论及技术。这个编译器可以将具有代表性的程序设计语言语句翻译为三地址代码（一种中间表示形式）。本章的重点是编译器的前端，特别是词法分析、语法分析和中间代码生成。

首先建立一个能够将算术表达式转换为三地址代码的语法制导翻译器。然后扩展这个翻译器，使它能将某些程序片段（如图2-1a所示）转换为如图2-1b所示的三地址代码。

```
a = 1;
b = 10;

while (a < b)
    a = a + 1;

b = a + b;
```

a)

```
100:  a = 1
101:  b = 10
102:  if a< b goto 104
103:  goto 107
104:  t1 = a+1
105:  a = t1
106:  goto 102
107:  t2 = a+b
108:  b = t2
```

b)

图2-1　需处理的程序片段及对应的三地址代码

a）程序片段　b）三地址代码

在讨论如何翻译之前，先来了解下这个程序片段的语言组成与功能，即如何描述这样一段程序代码。

2.1　语言定义

为程序设计语言提供一种简明扼要又易于理解的描述是很困难的，但这对语言的成功又是必不可少的。显然，程序设计语言的实现者必须能够确定语言的表达式、语句和程序单元如何构成，以及执行的预期效果。实现者的任务难度，部分取决于语言描述的完整性和准确性。最后，语言的使用者必须能够查阅语言参考手册，确定如何编写软件解决方案。

学习程序设计语言与学习自然语言一样，可以分为语法的学习和语义的学习。程序设计语言的语法是它的表达式、语句和程序单元的组织形式，即以什么方式来表现语言各组成部分；它的语义是这些表达式、程序单元的意义。例如，图2-1程序片段中的while语句的语法是：

```
while（布尔表达式）　语句
```

其语法含义是：该语句必须以英文单词“while”开始，之后是布尔表达式，且布尔表达式必须用小括号括起来，该语句最后一部分是语句，即只要是语句，就可以跟在布尔表达