新世纪高等院校精品教材

编译原理与技术

冯 雁 主编

浙江大學出版社

新世纪高等院校精品教材

编译原理与技术

冯　雁　主编

浙江大學出版社

图书在版编目(CIP)数据

编译原理与技术/冯雁主编. —杭州：浙江大学出版社，2004.8
新世纪高等院校精品教材
ISBN 7-308-03802-5

Ⅰ.编… Ⅱ.冯… Ⅲ.编译程序—程序设计—高等学校—教材 Ⅳ.TP314

中国版本图书馆 CIP 数据核字(2004)第 073331 号

责任编辑 张 明 杜希武 徐秀萍
出版发行 浙江大学出版社
(杭州浙大路 38 号 邮政编码 310027)
(E-mail：zupress@mail.hz.zj.cn)
(网址：http://www.zjupress.com)
排 版 杭州大漠照排印刷有限公司
印 刷 德清第二印刷厂
开 本 787mm×1092mm 1/16
印 张 21
字 数 551 千
版 印 次 2004 年 9 月第 1 版 2004 年 9 月第 1 次印刷
印 数 0001—3000
书 号 ISBN 7-308-03802-5/TP·262
定 价 29.00 元

序　言

编译原理与技术是计算机专业设置的一门重要的专业课程，本书力图以一种理论与实践紧密结合的方式，将编译程序的内容阐述清楚。围绕着如何构造一个编译器，介绍了现代编译器设计方法的理论，还包括了众多实例的讲解。本书既可以作为计算机专业的本科生和研究生的程序编译课程的教材，又可以用作专业人员的参考书或者用作编译器编写项目的入门参考书。虽然只有少数人真正从事编译方面的工作，但是通过对该课程的学习，使学生在理论、技术、方法上都得到了系统而有效的训练，有利于提高软件人员的素质和能力。

掌握编译程序的构造原理与方法的最好办法是自己设计与编写一个编译程序，但要自己设计与编写一个编译程序，首先必须基本掌握编译程序构造的一般原理。因此，本书非常强调基本原理的阐述，针对具体的原理，我们都有实例的讲解以配合对原理的理解。我们在编写过程中也尽量把编译原理的所有内容都包括在其中，这样，可以根据不同的教学要求对相关内容进行取舍，针对不同的教学对象进行合理安排。

本书介绍的概念和技术不仅适用于编译器的设计，也适用于一般的软件设计。例如，建立词法分析器的串匹配技术已用于文本编辑器、信息检索系统和模式识别器，上下文无关文法和语法制导翻译等概念已用于设计许多诸如排版、绘图系统这样的小语言，代码优化技术已用于程序验证器和从非结构化程序产生结构化程序的程序验证器之中。所以，对从事计算机专业的人们来说也具有重要的意义和价值。

我们在内容的安排上考虑到了以下两点：(1)不同学校的教学计划是不太一样的，如有的计算理论是作为必修的，有的作为选修，有的则不一定开设这门课，因此，我们还是对自动机理论作了详细的讲解，即使没有学过自动机理论，也可以从本书中得到这部分内容的补充。(2)不同的教师会使用不同的实践方法来加强对理论的理解和巩固。有些教师倾向于用一系列分开的小例子去研究技术，每个小例子都针对一个特定的概念；有些教师则偏向于做一个大的编译器项目，该项目或通过工具Lex和Yacc来实现，或通过手写代码(用递归下降分析)来实现。在本书中，我们也对工具Lex和Yacc进行了适当的讲解，可以根据需要来安排这部分的内容。

全书共9章，在一般情况下，本书每一章都独立于其他章节。

第1章对编译程序的一些基本概念及编译程序的构造作了一个概括性的介绍，还介绍了与编译器有关联的一些程序，并给出了一个简单编译程序的例子，使对编

译的全过程有一个初步的了解。

第2章专门讨论词法分析，这部分的理论基础是正规表达式与有限自动机，建立在这个理论基础之上的Lex，是一个建立词法分析器的自动化工具，本章还讨论了Lex实现及其使用的各种问题。

第3章研究了上下文无关文法理论，以及有关分析和推导等概念，强调了二义性的消除等等，讨论了乔姆斯基(Chomsky)层次和上下文无关文法的限制。

第4章研究了自顶而下的分析算法，尤其是递归下降分析和LL(1)分析。并且介绍了First和Follow集的概念及构造。

第5章讨论自下而上的语法分析方法，分别介绍了算符优先分析法和LR语法分析方法，最后介绍了基于LALR(1)文法的分析器自动生成工具Yacc及其使用的问题，并提供了对样例语言的Yacc说明。

第6章着重研究静态语义分析的基本方法，包括属性文法和属性计算，并给出了符号表的构造和静态类型检查等方法，并具体探讨了符号表和Hash表的实现。

第7章讨论了运行时存储分配的问题，包括了Fortran的完全静态环境、各种基于栈的环境和类Lisp语言的全动态的堆式运行环境。

第8章讨论了中间代码生成，介绍了若干形式的中间代码，并结合具体语句，给出相应的代码生成样例，包括申明语句、赋值语句、布尔表达式、控制语句和过程调用的处理。

第9章研究了代码优化技术和代码生成技术，包括寄存器与临时单元的管理，寄存器分配方案，代码优化技术包括了分析器优化、线形优化和语法树上的优化。

本书可用于34学时至60学时左右的介绍性编译课程，可以用也可以不用有关Lex和Yacc编译器构造工具，自动机理论也可以根据具体情况考虑，如果是34学时的安排，则词法分析、语法分析、语义分析和代码生成都要讲的话，相对会比较紧张。有些内容或一些例子另安排为自学为好，如算符优先等等。语法分析也应择其重点讲一些，如自顶向下语法分析或自下而上语法分析，而不必安排全部讲。

如果是60个学时左右的话，则完成本书的教学是可能的。

本书由冯雁撰写第1章和第3章，徐海燕撰写第2章和第7章，鲁东明撰写第4章和第5章，王强撰写第6章，李莹撰写第8章和第9章。另外，在本书的写作过程中也得到了研究生刘洋、王翱宇、王国栋、李甜和陈岚的帮助，在此表示深切的谢意。由于水平所限，书中难免有一些不妥之处，敬请广大读者批评指正。

冯 雁

2004年7月

目　录

第1章

引　论

编译器的原理与技术具有十分普遍的意义，在计算机科学中也是比较成熟的学科，其发展历史虽然不长，但其内容却十分丰富，用途也十分广泛。它是计算机专业，特别是软件专业的主要基础课。编译器的编写涉及到程序设计语言、计算机体系结构、语言理论、算法和软件工程等学科。

本章主要内容包括：编译程序的组成、编译的各个阶段主要内容、与编译器相关的其他程序等，最后通过对一个简单编译器的介绍，让大众对编译器的全貌有一个初步的印象。

1.1　什么是编译程序

编译器是一个程序，是一个将一种语言翻译为另一种语言的计算机程序。人们要用计算机来解决问题，必须要让计算机明白要解决的问题以及如何解决这个问题。但是，人们所习惯的语言和计算机所能理解的机器指令有很大的差别，而用机器指令来描述人们想要解决的问题又十分的不便，于是计算机科学家设计了一种人们比较习惯的语言来描述要解决的问题。一般我们把这些称之为高级程序语言，而能被计算机直接理解与执行的语言即机器语言，我们称为低级机器语言。有时，也把高级程序语言称为源语言，把机器语言称为目标语言。

这一过程可以用图 1-1 表示：

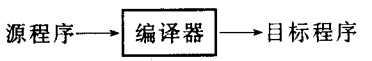

图 1-1　编译器

目前，世界上存在着数千种源语言，既有 Fortran 和 Pascal 这样的传统程序设计语言，也有各计算机应用领域中出现的专用语言。目标语言也同样广泛，可以是微处理机到超级计算机的任何计算机的机器语言。不同语言需要不同的编译器，根据编译器的构造方法或者它们要实现的功能，一般可分为一遍编译器、多遍编译器、调试编译器和优化编译器等等。虽然编译器的种类很多，但它们要完成的基本任务都是相同的，只不过有不同的侧重点。

编译器是一种相当复杂的程序，其代码的长度可从 1 万行到 100 万行不等。编写甚至读懂这样的一个程序决非易事，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均要用到编译器，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。

第一个编译器的问世可以追溯到 20 世纪 50 年代，IBM 的 John Backus 带领的一个研究小组对 FORTRAN 语言及其编译器进行了开发，由于当时处理中所涉及到的大多数程序设计语言的翻译并不为人所掌握，所以花费了 18 人年才得以实现。随后，Chomsky 对文法的研究，使得人们可以根据语言文法的难易程度以及识别它们所需的算法来为语言分类，其中的上下文无关文法被认为是程序设计语言中最有用的一种语言文法。语法分析的研究是在 20 世纪 60 年代和 70 年代，这一时期的研究比较完善地解决了对上下文无关文法所定义的语言的有效识别算法，现在这一部分已经是编译理论的一个标准部分。人们接着又深入研究了如何使目标代码更加有效的方法，通常称为优化技术。从第一个编译器出现至今，我们所掌握的有关编译器的知识已经得到了长足的发展。目前，我们已经系统地掌握了处理编译期间发生的许多重要技术。良好的实现语言、程序设计环境和软件工具也已经被成功开发出来。尽管近年来对此进行了大量的研究，但是基本的编译器设计在近 20 年中都没有太大的改变，它们正迅速成为计算机科学课程中的中心环节。

1.2 编译器的基本阶段

每个编译器虽然内部结构和组织方式各种各样，但主要由两部分组成：分析和综合。分析部分是对源程序进行分析，将源程序切分成一些基本块并形成源程序的中间语言表示；综合部分是在分析正确无误之后，把源程序的中间语言表示转换为所需的目标程序，综合出可以执行的机器语言程序，执行的结果应正确无误，同源程序应达到的目的完全一致。有时也把分析和综合称为前端和后端。为不同的机器编写相同源语言的编译器时，人们经常会采取这种方法：为所有的机器编写相同的编译器前端或者采用已有的编译器前端，然后为每种具体机器编写编译器的后端。同样，也可以将不同的源语言编译成同一种中间语言，对不同的前端使用相同的后端。这在很大程度上能提高编译器的代码重用和代码移植。但由于程序设计语言和机器构造的快速发展以及一些根本性的变化，这方面的工作是很难做到的，这方面的研究也只取得了有限的成果。

编译器内部包括了许多步骤或称为阶段，它们执行不同的逻辑操作。每个阶段将源程序从一种表示转换成另一种表示。编译器的一个典型阶段划分如图 1-2 所示。将这些阶段设想为编译器中一个个单独的片断是很有用的，尽管在应用中它们是经常组合在一起的，但它们确实是作为单独的代码操作过程来编写的。图 1-2 中的前三个阶段构成了编译器的分析部分，图中的符号表管理和错误处理是编译器的所有阶段都要涉及的两项活动。

编译器的若干阶段通常以一遍(pass)来实现，每遍读一次输入文件，产生一个输出文件，遍可以和阶段相对应，也可以和阶段无关，遍中通常含有若干个阶段。例如，词法分析、语法分析、语义分析以及中间代码的生成可以被组合成一遍，然后余下的部分组合成一遍。语法分析器根据它读到的标记识别语法结构，当它需要下一个标记时，它通过调用词法分析器获得所需的标记，当某一个语法结构识别后，语法分析器就调用中间代码生成器完成语义分析并生成中间代码的一部分。实际上，根据语言的不同，编译器可以是一遍通过，即所有的阶段一遍完成，但可能完成代码的质量不太高。大多数带有优化的编译器都需要超过一遍。

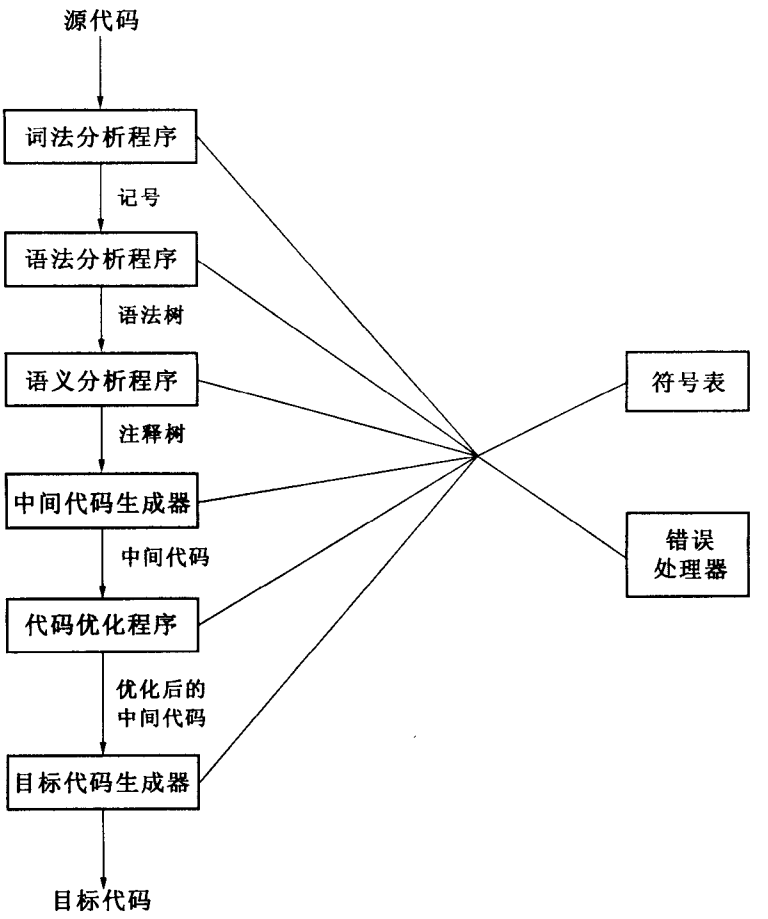

图 1-2　编译器的组成阶段

1.2.1　词法分析程序

在这个阶段编译器实际阅读源程序(通常以字符流的形式表示)。词法分析程序从左到右读入字符,按照源语言规定的词法规则,识别出一个个的单词(token),单词(token)又可以称为标记,标记同自然语言的英语单词相似。因此可以认为词法分析程序执行与拼写相似的任务。例如标识符、关键字、运算符、分隔符都是单词,编译程序把单词当作源语言的最小单位。每个标记表示一个逻辑上相关的字符序列,如关键字是一类标记,具体这类标记可能包括很多,如 if,while,for 等等,形成一个标记的字符序列称为该记号的词素(lexeme)或标记值,如 if 属于关键字这一类标记,它的词素或标记值是 if。

在 C 语言的表达式代码 `x[index] = initial + 2 * 4` 中包括了 20 个非空字符,但只有 10 个词素:

`x`	标识符
`[`	左括号
`index`	标识符
`]`	右括号

=	赋值
initial	标识符
+	加号
2	数字
*	乘号
4	数字

词法识别程序还可完成与识别标记一起执行的其他操作。例如，如果该标记值在符号表中不存在，还要在符号表中为它建立一个记录，会将该单词输入到符号表中。

1.2.2 语法分析程序

语法分析程序从词法识别程序中获取记号形式的源代码，并完成定义程序结构的语法分析(syntax analysis 或者 parsing)，这与自然语言中句子的语法分析类似。语法分析定义了程序的结构元素及其关系。通常将语法分析的结果表示为分析树(parse tree)或语法树(syntax tree)。

例如，还是那行 C 代码，它表示一个称为表达式的结构元素，该表达式是一个由左边为下标表达式、右边为算术表达式的赋值表达式组成。这个结构可按下面的形式表示为一个分析树，如图 1-3 所示。

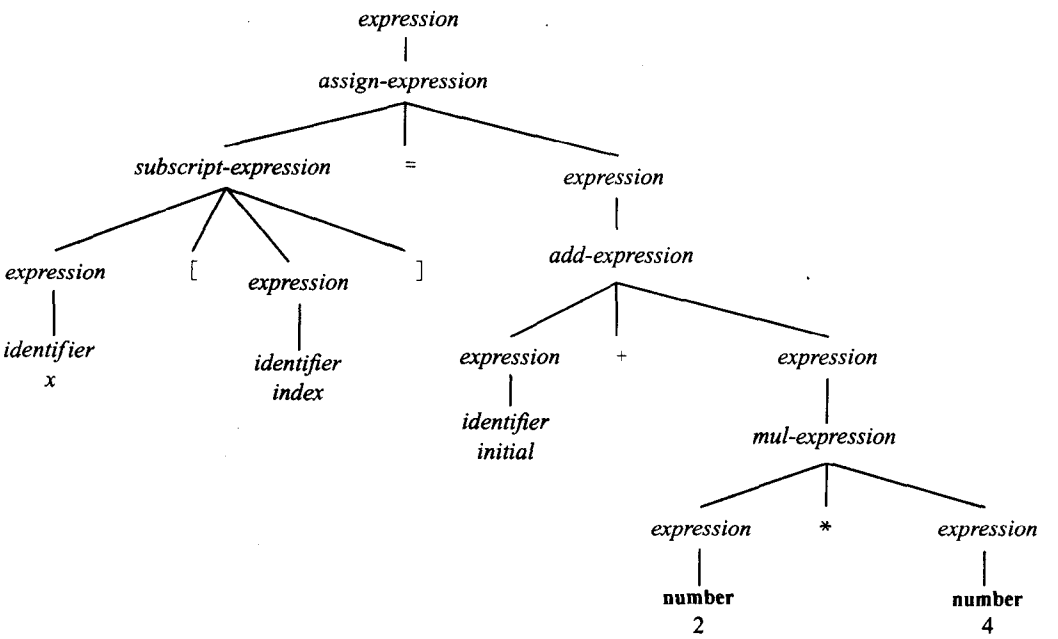

图 1-3 语法分析后生成的分析树

分析树的内部节点均由其表示的结构名标示出，而分析树的叶子则表示输入中的记号序列(结构名以不同字体表示以示与记号的区别)。图 1-4 给出了这种与该分析树相对应的更一般的内部表示语法树。语法树是分析树的压缩表示形式。在语法树中，许多节点(包括记号节点在内)已经消失。例如，如果知道表达式是一个下标运算，则不再需要用括号“[”和“]”来表示该操作是在原始输入中。

在语法分析程序中，如果源程序没有语法错误，就能正确地给出其分析树或语法树，否则，

要指出语法错误，并给出相应的诊断信息。

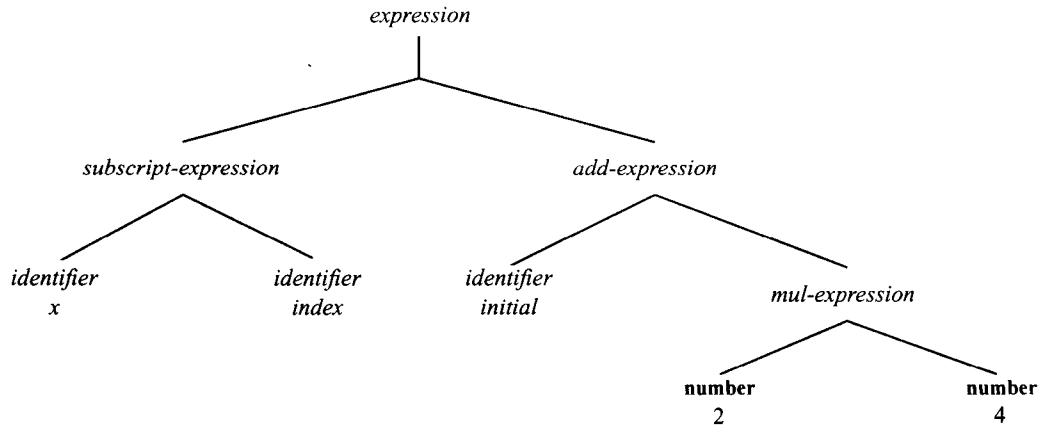

图 1-4　语法树

1.2.3　语义分析程序

程序的语义就是它的“意思”，它与语法或结构不同。程序的语义确定程序的运行，但是大多数的程序设计语言都具有在执行之前被确定而不由语法表示和由分析程序分析的特征。只有语法、语义正确的源程序才能被翻译成正确的目标代码。

语义分析的一个重要组成部分是类型检查。类型检查负责检查每个操作符的操作数是否满足源语言的说明。例如，很多程序设计语言都要求每当一个实数用于数组的索引时都要报错，也有一些程序设计语言允许一些操作数的强制类型转换。例如，一个二元算术操作数可以分别是一个整数和实数，那么，编译器就要把整数强制转换成实数。

在上面的 C 表达式 `x[index] = initial + 2 * 4` 中，该行分析之前收集的典型类型信息可能是：x 是一个整型值的数组，index 则是一个整型变量。接着，语义分析程序将用所有的子表达式类型来标注语法树，并检查赋值是否使这些类型有意义了，如若没有，则声明一个类型匹配错误。在上例中，所有的类型均有意义，有关语法树的语义分析结果可用以下注释了的树（图 1-5）来表示：

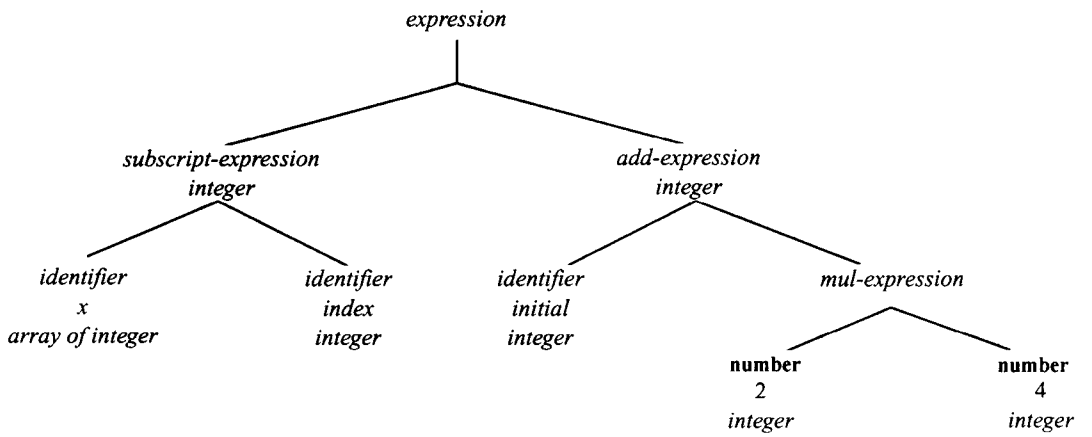

图 1-5　具有语义信息的语法树

1.2.4 中间代码生成器

编译器在完成语法分析和语义分析以后，把源程序转换为一个中间语言表示。一般来说，这种中间语言表示应有两个重要的性质：易于产生和易于译成目标程序。中间代码生成器可以看成是一个独立的编译阶段，也可以是语义分析阶段的一部分工作。

源程序的中间表示形式有很多。例如，像“三地址码”，是一种类似于某种机器的汇编语言，这种机器的每个存储单元的作用类似寄存器。三地址码由指令序列组成，每个指令最多有三个操作数。上面的C代码表达式可转换成下面的三地址代码：

```
temp1 := 8  (经过了源代码层次的优化)
temp2 := initial + temp1
x[index] = temp2
```

后面的有关章节中将具体讨论编译程序中使用的几种中间语言表示及有关中间语言代码的生成算法。

1.2.5 代码优化程序

代码优化阶段主要为了提高中间代码的质量，以便提高目标程序的运行速度。代码优化可以在语义分析之后完成，也可以在中间代码生成后完成，语义分析之后的代码优化可能只依赖于源代码。在1.2.4节中就包括了一个源代码层次的优化。每个编译器不论在已完成的优化种类方面还是在优化阶段的定位中都有很大的差异。进行了大量优化工作的编译程序通常称作“优化编译程序”，这种编译程序的大部分编译时间都花在这个阶段。在实用上仍有许多简单有效的优化算法，能够显著地提高目标程序的运行速度而不明显地降低编译速度。

上面的中间代码经过优化可以改为：

```
x[index] := initial + 8
```

1.2.6 目标代码生成器

目标代码生成器由得到的中间代码生成目标机器的代码。尽管大多数编译器直接生成目标代码，但是为了便于理解，本书用汇编语言来编写目标代码。正是在编译的这个阶段中，目标机器的特性成了主要因素。例如，整型数据类型的变量和浮点数据类型的变量在存储器中所占的字节数或字数也很重要。

代码生成器要对源程序中使用的变量分配寄存器，然后为每一条中间语言指令选择合适的机器指令，包括确定机器指令的操作码或编址模式。在上面的例子中，必须决定怎样存储整型数来为数组索引生成代码。例如，下面是所给表达式的一个可能的样本代码序列(在假设的汇编语言中)：

```
MOV R0, index     ; value of index → R0
MUL R0, 2         ; double value in R0
```

```
MOV R1, &x          ; address of x → R1
ADD R1, R0          ; add R0 to R1
MOV R2, initial     ; value of initial → R2
ADD R2, 8           ; add 8 to R2
MOV * R1,R2         ; value of R2 → address in R1
```

在以上代码中，为编址模式使用了一个类似 C 的协定，因此 &a 是 a 的地址（也就是数组的基地址），* R1 则意味着间接寄存器地址（因此最后一条指令将 R2 的值存放在 R1 包含的地址中）。这个代码还假设机器执行字节编址，并且整型数占据存储器的两个字节（所以在第二条指令中用 2 作为乘数）。

在这个阶段中，编译器还会有很多代码优化的工作可做，以改进由代码生成器生成的目标代码。这种改进包括选择编址模式以提高性能，将速度慢的指令更换成速度快的，以及删除多余的操作。在上面给出的目标代码中，还可以做许多更改。在第二条指令中，利用移位指令替代乘法（通常地，乘法很费时间），还可以使用更有效的编址模式（例如用索引地址来执行数组存储）。使用了这两种优化后，目标代码就变成：

```
MOV R0, index       ; value of index → R0
SHL R0              ; double value in R0
MOV R2, initial     ; value of initial → R2
ADD R2, 8           ; add 8 to R2
MOV &x[R0],R2       ; value of R2 → address x +  R0
```

编译过程的前面几个阶段，包括词法分析、语法分析、语义分析直至中间代码生成，依赖于被编译的源语言，因此，将这几个阶段归在一起，称为编译程序的前端。前端还包括与这几个部分有关的符号表操作，代码优化中与目标机无关的部分也属于前端。前端中各个阶段中的错误处理部分也属于前端。编译过程中同目标机有关的部分属于编译程序的后端，后端与源语言无关，只同中间语言有关。后端包括代码优化中涉及目标机的部分、目标代码生成以及相关的错误处理及符号表操作。

1.2.7 符号表管理

在编译过程中，源程序中的标识符及其各种属性都要记录在符号表中，这些属性提供标识符的存储分配信息、类型信息、作用域信息等等。对于过程标识符，还有参数信息，包括参数的个数及类型，参数结合的方式等等。

符号表是一个数据结构。每个标识符在符号表中都有一条记录，记录的每个域对应于该标识符的一个属性。这种数据结构允许我们快速地找到每个标识符的记录，并在该记录中快速地存储和检索信息。

当源程序中的一个标识符被词法分析器识别出来时，词法分析器将在符号表中为该标识符建立一条记录，而它的属性信息将由后面的各个阶段写入符号表，并在需要时使用。例如，当进行语义分析和中间代码生成时，需要知道标识符是哪种类型，以便知道源程序是否正确地使用了这些标识符。因此，会涉及到很多查询和填入属性的问题。