高等学校教材

GAODENGXUEXIAOJIAOCAI

编译原理

BIANYIYUANLI

主编 韩太鲁

石油大学出版社

编译原理

主　编　韩太鲁

副主编　姚栋义　董吉文

刘晓红　孙忠林

石油大学出版社

内容提要

编译原理是计算机本科专业的必修课，课程主要内容有：词法分析、语法分析、语义分析、中间代码生成及优化、目标代码生成等。

本书系统深入地介绍了编译系统的基本原理和方法，同时介绍了本学科的一些新的内容和知识，全书共分为八章。第一章介绍了算法语言的特点及发展过程，第二章介绍了形式语言的基本概念及简单的语法分析方法，这两章是后面几章的预备知识。第三章是词法分析部分，主要介绍了识别算法语言单词的词法分析器及其构造方法，同时介绍了正规式和有限自动机的基础理论。第四、五章在深入地介绍算法语言语法分析的基本理论及构造方法的同时给出了算法语言各类语句的文法定义，并给文法的产生式辅上了语法制导翻译的语义动作，为使语义翻译进一步形式化，第五章还专门讨论了翻译文法和属性翻译文法。为了提高目标代码的质量，第六章介绍了代码优化的基本理论和方法。第七章讲述了编译阶段如何在运行时对算法语言的目标代码进行存储管理，并给出了某些约束变量的一种分配方案。第八章讨论了目标代码的生成方法。作为编译系统原理和方法的应用，本书最后给出了编译原理课程设计的题目及具体要求。

本书可作为高等学校计算机专业本科生的教材，也可供从事计算机应用的工程技术人员或其他自学者学习参考。

本书是山东省多所高校共同组织编写的计算机专业面向21世纪系列教材之一。

图书在版编目（CIP）数据

编译原理/韩太鲁主编．—东营：石油大学出版社，2002．7
ISBN 7-5636-1575-X

Ⅰ．编…　Ⅱ．韩…　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(2002)第 093555 号

编 译 原 理

主　编　韩太鲁
副主编　姚栋义　董吉文　刘晓红　孙忠林

出版者：石油大学出版社（山东　东营　邮编　257061）
印刷者：临清市万方印务有限责任公司
发行者：石油大学出版社（电话 0546-8392062）
开　本：787×1092　1/16　印张：13.75　字数：348 千字
版　次：2002 年 7 月第 1 版　2002 年 7 月第 1 次印刷
定　价：19.00 元

前言

《编译原理》是计算机本科专业的必修课，课程主要内容有：词法分析、语法分析、语义分析、中间代码生成及优化、目标代码生成等。

本书系统深入地介绍了编译系统的基本原理和方法，同时介绍了本学科的一些新的内容和知识，全书共分为八章。第一章介绍了算法语言的特点及发展过程，第二章介绍了形式语言的基本概念及简单的语法分析方法，这两章是后面几章的预备知识。第三章是词法分析部分，主要介绍了识别算法语言单词的词法分析器及其构造方法，同时介绍了正规式和有限自动机的基础理论。第四、五章在深入地介绍算法语言语法分析的基本理论及构造方法的同时给出了算法语言各类语句的文法定义，并给文法的产生式辅上了语法制导翻译的语义动作，为使语义翻译进一步形式化，第五章还专门讨论了翻译文法和属性翻译文法。为了提高目标代码的质量，第六章介绍了代码优化的基本理论和方法。第七章讲述了编译阶段如何在运行时对算法语言的目标代码进行存储管理，并给出了某些约束变量的一种分配方案。第八章讨论了目标代码的生成方法。作为编译系统原理和方法的应用，本书最后给出了编译原理课程设计的题目及具体要求。本书教学参考时数为 68~76 学时。

本书的第一、三章由董吉文编写，第二、四章由韩太鲁编写，第五章由姚栋义编写，第六章和课程设计部分由孙忠林编写，第七、八章由刘晓红编写。书中的部分 C 语言程序由赵卫东进行了编写和审定，孙忠林对全书的编辑做了大量的工作，山东科技大学信息科学与工程学院研究生张艳琦、张瑞全同学及韩红同志录入了部分书稿。山东科技大学信息科学与工程学院的吴哲辉教授对本书作了全面审阅，提出了许多宝贵意见。

本书是山东省多所高校共同组织编写的计算机专业面向 21 世纪系列教材之一，各参编学校的有关领导和专家都给予了大力支持，特别是山东科技大学信息科学与工程学院郑永果教授对本书的编写做了大量的组织工作，在此一并表示衷心的感谢。

由于编者水平有限，加之成书时间比较仓促，书中难免有不妥和错误之处，敬请读者批评指正。

编　者

2002 年 6 月于泰安

目 录

第一章 绪 论

1946年第一台电子计算机的问世，是人类科学发展史上的一个里程碑，它开创了人类脑力劳动自动化的先河，丰富了人类的精神财富，增强了人类的认识能力。仅仅经过半个多世纪的时间，计算机就经历了数代更新，由初期的单纯用于科学计算发展到广泛应用于社会的各个领域。在科学技术迅猛发展的今天，计算机的发展水平和普及程度已经成为衡量一个国家科学技术发展水平和现代化程度的重要标志。

众所周知，计算机系统包括硬件和软件两大部分，而软件的功能和质量很大程度上决定着整个计算机系统的功能。软件是整个计算机系统的灵魂，是计算机和使用者之间的桥梁。软件大体上可分为系统软件和应用软件，操作系统和编译程序等属于系统软件。

要使用计算机，就得编写程序。用来编写程序的语言称为程序设计语言。程序设计语言是开发软件的工具，是人机通信的媒介。程序员利用语言提供的各种机制把自己想做的工作告诉计算机就形成了程序。在我国，先后被广泛使用的高级程序设计语言有ALGOL，BASIC，FORTRAN，COBOL，LISP，PROLOG，PASCAL和C等。

计算机只能接受用机器语言编写的程序，为了使高级程序设计语言所编写的程序能够被计算机所接受，必须将高级语言编写的程序翻译成具体的机器语言，这一翻译工作是由预先存放在计算机中的翻译程序来实现的。这有些类似于我们阅读外文资料或听外国人讲话，如果不懂外文，就需要请一位翻译来将外文翻译成中文。

在计算机上预先装上一个编译程序，就相当于给计算机配上一个“翻译”，它可以根据高级语言的语法和语义将高级语言源程序翻译成计算机能够认识的机器语言程序，这样计算机就懂得了一种语言。当然，我们可以给计算机配上多个“翻译”，使它懂得多种语言，从而充分发挥计算机的作用。

将高级语言翻译成机器语言的翻译程序有两种，即编译程序和解释程序。若源程序是用高级语言所编写，经翻译加工后生成目标程序，然后目标程序可以在计算机上运行，这样的翻译程序就是编译程序。而解释程序虽然加工对象也是高级语言源程序，但在翻译过程中并不生成目标程序，而是边翻译边执行，即解释程序同时处理源程序和源程序要加工的数据。

实际上，高级语言源程序就是按一定规则排列的符号集合，而编译程序就是把这些符号集合变成机器指令的转换器，编译程序又称为编译器，它是计算机中的重要系统软件，是从事计算机科学学习和工作的所有人员都应熟悉和掌握的重要知识。

1.1 程序设计语言概述

在计算机系统中，程序设计语言可分为机器语言、汇编语言和高级语言，人们常常把机器语言和汇编语言统称为低级语言。

每台计算机都有自己的指令系统，包括若干条指令，每条指令让计算机执行一个特定的动作，每个指令代码都用数字代码来表示。不同型号的计算机，它们的指令系统一般是互不相同的。用这样的指令所编制的程序可以直接在计算机上运行，这样的指令系统称为机器语言。

所谓汇编语言，就是将计算机的指令系统用符号来表示。例如，用 ADD 表示加法指令、用 SUB 表示减法指令等。它与机器语言存在着一一对应的关系，即一条机器指令对应一条汇编指令，反之亦然。它不再是令人讨厌的数字代码，而是采用了容易辨认、易于记忆的符号。它的使用，将计算机工作者从浩繁的计算机代码编程中解放出来，同时也提高了程序的质量。

低级语言都是依赖于具体计算机的语言，使用起来既烦琐又容易出错，程序不便于阅读和交流，实用性差，因此相继出现了很多高级语言。高级语言由表达各种不同意义的“关键词”和“表达式”按一定的语法语义规则所组成，它与人们习惯的数学语言和工程语言较为接近，因此比较直观、自然和容易理解。使用高级语言编写的程序比较易读、易写，易于交流、出版和存盘，一旦发现错误也易于修改，因而使用高级语言编写程序比用机器语言或汇编语言所费的劳动代价要低得多。

1.1.1 高级程序设计语言的特点和发展

高级程序设计语言随软件技术的发展而快速发展，20 世纪 60 年代初期，涌现出大量的高级程序设计语言，例如 FORTRAN，COBOL 和 ALGOL 60 等。它们的出现大大降低了编程工作的劳动强度，把编程人员从烦琐的低级语言中解放出来，由它们编制的软件奠定了最初的软件产业基础。但由于早期的高级语言类型单调，程序过多依赖程序员的技巧，滥用 goto 语句等原因使程序难读、难改、不易移植，很难编制出更大规模的程序。

20 世纪六七十年代兴起了结构化程序设计方法，希望以规范格式的软件结构消除复杂软件内部的混乱。于是结构化程序设计语言 PASCAL 和 C 得到广泛应用，一些早期的语言也纷纷改成结构化版本，例如 FORTRAN-77 和 COBOL-74 就是这个时代的产物，它们的共同特点是支持模块化程序设计思想，具有完备的数据结构、灵活通用的语句、清晰的书写格式、优美的设计风格。20 世纪 70 年代因受益于结构化程序设计，软件规模得到很大的提高，最大软件(美国导弹预警系统)已达 385 万句。

随着硬件的不断发展，人们就想编制出规模更大的软件。软件一大就无法管理和维护，即使采用结构化程序设计在更大规模的软件面前也是问题百出。所以，20 世纪 70 年代中期人们借助软件工程的方法使所编软件的规模进一步扩大，到 20 世纪 80 年代，最大的软件(美国航天飞机监控系统)已达 4000 万句。

但又过了将近 10 年，证实传统的软件工程对软件危机并没有多大的缓解，大型软件投资失败，不可靠性、不可维护性、不可移植性依然大量存在，寻求新的开发和表达技术已成为燃眉之急。20 世纪 90 年代初，在抽象数据类型和交互式环境思想基础上，以 Smalltalk-80 为代表的面向对象技术的出现，给危机重重的软件产业带来了希望之光。面向对象以其封装

的模块性、类和实例的抽象性、继承机制提供的软件重用性、重载多态和动态束定的灵活性所支持的软件的可扩充性，无疑成为当代最成功的软件技术。于是软件产业的各主导语言都纷纷向面向对象技术扩充，从C语言演变而成的C++程序设计语言就是它们的典型代表。它除了包括C语言的语法机制外还增加了类和实例、继承、重载运算符、虚函数、友员、内联等支持面向对象程序设计的机制，对程序的扩充、软件的重用带来了极大的方便，为提高编程效率和软件质量提供了有力的保证，成为应用程序开发者和系统程序员使用的最佳语言。

面向对象的程序设计(OOP)立意于创建软件重用代码，具备更好地模拟现实世界环境的能力，这使它被公认为是自上而下编程的优胜者。它通过给程序中加入扩展语句，把函数“封装”进编程所必需的“对象”中。面向对象的编程语言使得复杂的工作条理清晰，编写容易。然而一些早期的具有面向对象程序设计性能的程序语言，虽然具有面向对象的特征，但不能轻松地刻画出可视化对象，与用户交互能力较差，程序员仍然要编写大量的代码，特别是在编制Windows接口的软件上，这个问题显得尤其突出。20世纪90年代中期相继推出的基于可视化和面向对象的编程语言，以其迅捷的编译速度、强大的功能和易学灵活的特点，为越来越多的程序员所喜爱，它们的典型代表有VB、VC和Delphi等可视化编程环境。编程人员不必自己建立对象，只要在提供的程序框架中加入要完成功能的代码，就可以很容易地生成具有漂亮接口和良好结构的程序。这些可视化编程环境允许在一个具有真正OOP扩展的可视化编程环境中，使用面向对象的编程语言，如C++和Object Pascal等。这种革命性的组合，使得可视化编程与面向对象的开发框架紧密地结合起来，使它们成为触手可及的促进软件重用的开发工具，从而具有强大的吸引力。

从软件产业的角度，当然希望得到表达能力更强、生产效率更高、易读易改的好语言。特别是计算机硬件突飞猛进的发展，使软件经常处于跟不上、赶不上、不好用的危机之中，寻求好的语言始终是软件产业的头等大事。可以肯定地说，随着计算机的发展，将会有更多更好的编程语言呈现在我们面前。

1.1.2 高级程序设计语言的参数传递

过程(或函数)的定义和调用是程序设计语言的主要特征之一，也是模块化程序设计的主要手段和节省程序代码、扩充语言能力的主要途径。

过程(或函数)在定义之后就可以在程序的其他地方调用它。调用与被调用者之间的信息交流可以通过全局量或由参数传递来实现。下面我们讨论程序设计语言中参数传递的实现。

1. 参数

首先请看下面的C语言程序：

```
#include "stdio.h"
int max(int x,int y)
{ int z;
   if (x>y) z=x;
   else z=y;
   return(z);      }
void main( )
{ int a,b,c;
```

```
    scanf("%d%d",&a,&b);
    c=max(a,b);
    printf("max=%d\n",c);    }
```

程序中定义了一个整型函数 max，函数中的 x 和 y 称为形式参数，简称形参。在主函数 main 中通过调用语句：

```
c=max(a,b);
```

对所定义的函数 max 进行调用，并将函数的返回值赋给变量 c，其中的 a 和 b 称为实在参数，简称实参。在此，实在参数不仅仅可以是变量，也可以是常数或者是一个比较复杂的表达式，但实参的类型、个数和顺序应该与形参一致。在程序执行到调用语句时，实参与形参进行参数传递，并执行被调函数 max。现在的问题是如何把实在参数传递给相应的形式参数。实现参数传递的常用方式有传地址、传值和传名字三种，下面我们分别介绍。

2. 传地址

所谓传地址是指把实在参数的地址传递给相应的形式参数。过程(或函数)的每个形式参数都对应一个相应的内存空间，称为形式单元。形式单元用来存放相应的实在参数的地址。当调用一个过程(或函数)时，调用段首先把实在参数的地址传递到一个临时设置的工作单元中。当程序转入被调用段之后，被调用段再把临时设置的工作单元中的实参地址取进自己相应的形式单元中，过程(或函数)体对形式参数的任何引用或赋值都被处理成为对形式单元的间接访问，形参的值发生变化，等价于与之对应的实参发生变化。当过程(或函数)执行完毕返回时，形式单元(它们都为指针)所指的实参单元就具有了变化以后的值。

例如，对下面的 PASCAL 程序：

```
program example(output);
var
    x,y:integer;
procedure add(var s: integer; var r: integer);
begin
    s:=s+r
end;
begin
    x:=0;
    y:=2;
    add(x, y);
    writeln('x=',x)
end.
```

程序中定义了过程 add，它包含两个形参 s 和 r，它们都是变量参数。在程序调用语句 add(x, y)中实参 x 与形参 s 对应，实参 y 与形参 r 对应，它们之间都采用传地址的方式进行参数传递。在调用语句执行过程中完成下列操作：

① 把实参 x 和 y 的地址分别传递到已准备好(由编译分配)的两个单元(t_1 和 t_2)中；

② s:=t_1 且 r:= t_2，使形参得到对应实参的地址；

③ 将 s 所指向的单元的内容(x)与 r 所指向的单元的内容(y)相加，完成表达式 s+r 的计算；

④ 将表达式 s+r 的运算结果送到 s 所指的内存单元(x)中；

⑤ 返回主程序。

由此可见，在程序调用过程中若形参发生变化，那么等价于对应的实参发生变化。以上程序中，实参 x 因形参 s 被赋值而由调用前的 0 变为 2。

3. 传值

这是一种最简单的参数传递方式。调用段把实在参数的值计算出来并放在一个能使被调用段拿到的地方。当被调用段开始工作时，首先把这些值抄进自己的形式单元，然后跟使用局部量一样使用这些形式单元。如果实在参数不是指针变量，那么在这种情况下被调用段无法改变实参的值。

在上面关于传地址的例子中，如果把过程 add 的首部说明改为：

```
procedure  add( s: integer; r: integer);
```

那么，形参 s，r 与它们对应的实参之间就采用传值方式进行参数传递。在这种情况下，与形参 s，r 对应的实参可以是表达式形式(传地址方式实参只能是变量)，如调用语句可以为：

```
add(x, sqr(y))
```

在执行到该调用语句时，首先计算实参表达式的值，然后将表达式的值传到已经准备好的工作单元中，在过程 add 开始工作后，从工作单元中得到实参的值，跟使用局部量一样参加过程中的运算。这样，虽然形参 s 的值在过程的运行过程中被改变，但并不影响实参的值。

4. 传名字

这是 ALGOL 60 所使用的一种参数传递方式。ALGOL 语言用替换规则解释传名字参数的意义：过程调用的作用相当于把被调用段的过程体抄到调用出现的地方，但把其中任一出现的形参都替换成相应的实参(文字替换)。如果在替换时发现过程体中的局部名和实参中的名字使用相同的标识符(重名)，则必须用不同的标识符来分别表示这些局部名。而且，为了表现实参的整体性，必要时在替换前先把它用括号括起来。

例如，对下面的 ALGOL 过程：

```
procedure swap(M,N);
integer M,N;
begin
integer   L;
      L:=M;
      M:=N;
      N:=L;
end;
```

若有过程调用语句 swap(K(I),K(J))采用传名方式进行参数传递，那么过程调用的作用相当于执行了下面的语句：

```
L:= K(I);
K(I):= K(J);
K(J):=L;
```

在这种参数传递方式的实现过程中，进入被调用段之前不对实在参数预先进行计值，而是让过程体中每当使用到相应的形参时才对它实行计值，所以它跟传地址和传值方式均不相

同。

除以上三种参数传递方法之外，还有传结果等方式，这里我们不予讨论。

1.2 编译过程概述

编译程序完成从源程序到目标程序的翻译工作，是一个复杂的过程。从概念上来讲，一个编译程序的整个工作过程是划分成几个阶段进行的，每个阶段将源程序的一种表示形式转换成另一种表示形式，各个阶段进行的操作在逻辑上是紧密连接在一起的，图 1-1 给出了一个编译过程的各个阶段。图中将编译过程划分成词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成六个阶段，这是一种典型的划分方法。事实上，某些阶段可以组合在一起。另外还有两个重要的工作：表格管理和错误检测，它们与上述六个阶段都有联系。编译过程中源程序的各种信息被保留在各种不同的表格里，编译各阶段的工作都涉及到构造、查找或更新有关的表格，因此需要有表格管理的工作。如果编译过程中发现源程序有错误，编译程序应报告错误的性质和错误发生的地点，并且将错误所造成的影响限定在尽可能小的范围内，使得源程序的其余部分能继续被编译下去，有些编译程序还能自动校正错误，这些工作称之为错误检测。错误检测能力的强弱也是评价一个编译程序优劣的标准之一。下面我们介绍各阶段的任务。

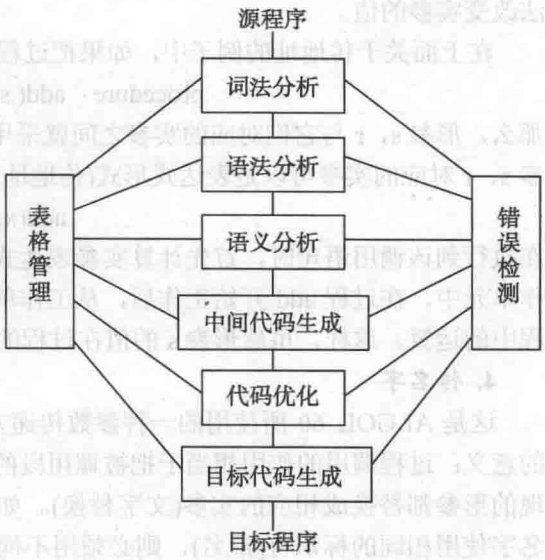

图 1-1 编译的各个阶段

1.2.1 词法分析

词法分析阶段是编译过程的第一个阶段。这个阶段的任务是从左到右一个字符一个字符地读入源程序，对构成源程序的字符串进行扫描和分解，从而识别出一个个单词(也称单词符号)。这里所谓的单词是指逻辑上紧密相连的一组字符，这些字符具有集体含义，比如标识符是由字母字符开头，后跟字母、数字字符的字符序列组成的一种单词，保留字(又称关键词或基本字)也是一种单词，此外还有算符、常数和界符(标点符号、左右括号等)等等。例如某源程序片段如下：

```
var
  sum, first, count :real ;
begin
  sum:= first + count * 10
```

end.

词法分析阶段通过对该程序片段的扫描、分解，将会把组成这段程序的字符识别为如下单词序列：

1. 保留字	var	2. 标识符	sum
3. 逗号	,	4. 标识符	first
5. 逗号	,	6. 标识符	count
7. 冒号	:	8. 保留字	real
9. 分号	;	10. 保留字	begin
11. 标识符	sum	12. 赋值号	:=
13. 标识符	first	14. 加号	+
15. 标识符	count	16. 乘号	*
17. 整数	10	18. 保留字	end
19. 界符	.		

可以看出，五个字符即“b”，“e”，“g”，“i”和“n”构成了一个分类为保留字的单词“begin”，两个字符即“:”和“=”构成了表示赋值运算的符号“:=”。这些单词间的空格在词法分析阶段都被滤掉了。

我们使用 id1，id2 和 id3 分别表示 sum，first 和 count 三个标识符的内部形式，那么经过词法分析后，上述程序片断中的赋值语句 sum:= first + count * 10 可表示为：

id1:= id2 + id3 * 10 (1.1)

1.2.2 语法分析和语义分析

在编译程序的实际实现中语法分析和语义处理常常是组合在一起的。语法分析的任务是在词法分析的基础上将单词序列分解成各类语法单位，如“程序”、“语句”、“表达式”等等。一般这种语法单位，可表示成语法树，比如式(1.1)经语法分析后可以确定它是 Pascal 语言的“赋值语句”，语法分析程序将其表示成如图 1-2 所示的语法树(或称分析树)，它描绘了输入的语法结构。这种语法结构更常见的内部表示由图 1-3 给出。

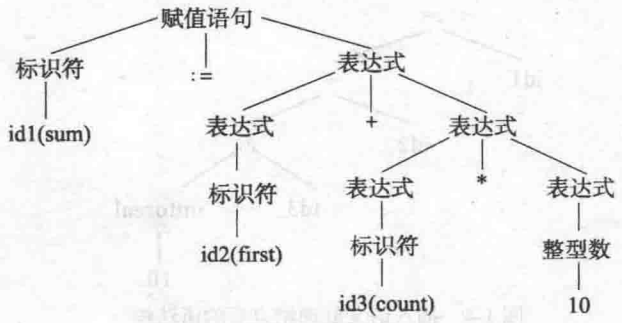

图 1-2 语句 id1: = id2 + id3* 10 的语法树

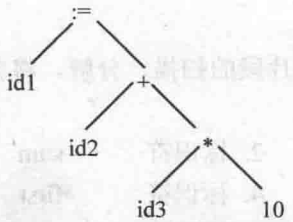

图 1-3　语句 id1: = id2 + id3* 10 的语法树的内部表示

语法分析所依据的是语言的语法规则，即描述程序结构的规则。通过语法分析确定整个输入串是否构成一个语法上正确的程序。

程序的结构通常是用递归规则表示的，例如我们可以用下面的规则来定义表达式：

(1) 任何标识符是表达式；

(2) 任何常数(整常数、实常数)是表达式；

(3) 若 e1 和 e2 都是表达式，那么 e1+e2，e1*e2，(e1)也都是表达式。

类似地，语句也可以递归定义，如：

(1) “<标识符>:=<表达式>”是语句；

(2) “while　<表达式> do <语句> ”和“if　<表达式>　then　<语句> else <语句>”都是语句。

上述赋值语句 id1:= id2 + id3 * 10 之所以能表示成图 1-2 形式的语法树，依据的就是赋值语句和表达式的定义规则。

语义分析是在语法分析程序确定出语法单位后，审查各语法单位有无语义错误，并为代码生成阶段收集类型信息的。比如进行类型检查，检查每个算符是否具有语言规范允许的运算对象，当不符合语言规范时，编译程序应报告错误，例如，有的编译程序当发现实型数作为数组下标时报告出错。再如，某些语言规定运算对象可被进行强制类型转换，那么当二目运算施于一整型数和一实型数时，编译程序应该将整型数转换成实型数而不能认为源程序发生类型错误，就像语句 sum := first + count * 10 中，算符*有两个运算对象：count 是实型、10 是整型，那么在语义分析进行类型检查后，应在语法分析所得语法树上增加一个语义处理结点，表示整型数 10 经过一个一目算符 inttoreal 后变成实型数 10.0，这样图 1-3 变成图 1-4 所示。

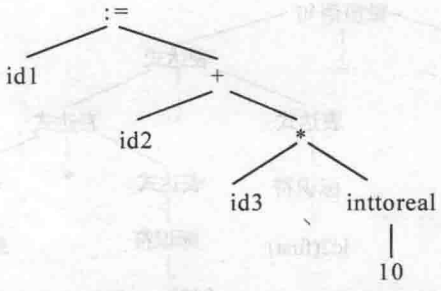

图 1-4　插入语义处理结点后的语法树

1.2.3　中间代码生成和优化

在完成上述语法分析和语义处理工作后，编译程序将源程序变成一种内部表示形式，这

种内部表示形式叫做中间语言或中间代码，它是一种结构简单、含义明确的记号系统。中间代码有多种形式，常见的有四元式、三元式和逆波兰式等，它们的共同特点是容易生成且容易翻译成目标代码。例如，四元式的形式为：(算符，运算对象 1，运算对象 2，结果)，对于源程序 sum:= first + count * 10 可以生成如下的四元式：

(1) (inttoreal , 10 , — , t1)
(2) (* , id3 , t1 , t2)
(3) (+ , id2 , t2 , t3)
(4) (:= , t3 , — , id1) (1.2)

其中：id1、id2、id3 分别表示 sum、first、count 的机器内部表示，t1、t2、t3 是临时生成的名字，用于表示中间运算结果。

为了使生成的目标代码更为高效，可以对前面产生的中间代码进行变换或进行改造，这就是中间代码的优化。在(1.2)式中，对于四元式(1)所对应的工作，可以在编译阶段完成，即不用生成相应的目标代码，所以可以简化掉；另外，中间变量 t3 仅用来将其值传递给 id1，也可以简化掉。简化后，上面的四个四元式可以由以下两个四元式来替代：

(1) (* , id3 , 10.0 , t1)
(2) (+ , id2 , t1 , id1) (1.3)

经过这样处理后，由中间代码生成的目标代码更加节省空间和时间。当然，要生成高质量的目标代码，还要做其他的优化工作，例如公共子表达式的删除、强度削弱、循环优化等。

1.2.4 目标代码

目标代码即指机器所能认识的机器指令代码或汇编指令代码，所以，编译程序中目标代码生成阶段的任务就是把中间代码变换成特定机器上的绝对指令代码或可重定位的指令代码或汇编指令代码。这是编译的最后阶段，它的工作与硬件系统结构和指令含义有关，这个阶段的工作很复杂，涉及到硬件系统功能部件的运用、机器指令的选择、各种数据类型变量的存储空间分配以及寄存器的调度等。

例如，对于上面所述源程序 sum := first + count * 10 对应的中间代码(1.3)可能生成下面的某汇编代码：

```
(1) MOV   id3,   R2
(2) MUL   #10.0, R2
(3) MOV   id2,   R1
(4) ADD   R1,    R2
(5) MOV   R1,    id1          (1.4)
```

在这段目标代码中，使用了两个寄存器(R1 和 R2)。第一条指令将 id3 内容送到寄存器 R2 中；第二条指令实现实型数 10.0 与 R2 的内容相乘；第三条指令将 id2 内容送到寄存器 R1 中；第四条指令将寄存器 R1 和 R2 的内容相加并把结果保存在 R1 中；第五条指令将寄存器 R1 的内容送到 id1 所对应的内存中，从而完成赋值操作。