21世纪 高等学校本科系列教材

总主编　吴中福

编译技术课程设计与上机指导

(40)

霍　林　　主编

重庆大学出版社

编译技术课程设计与上机指导

霍 林 主编

重庆大学出版社

内 容 简 介

《编译技术课程设计与上机指导》用简明、清晰、准确的语言描述编译技术实践环节所涉及的内容，在对各部分作简要说明后，给出了相应的程序流程图及可供参考的源程序，比较全面地涵盖了编译技术这门课的实践环节内容，其主要内容为：Lex 使用说明，YACC 使用说明，用 Lex 自动生成 Tiny 词法分析程序，用 YACC 自动生成 Tiny 语法分析程序，用 C 或 Pascal 语言为某一样本语言手工实现一个扫描程序，语法分析程序，语义生成程序，代码生成程序及为 Tiny 编译程序增加某一处理功能，最后附录了 Tiny 编译程序的源程序，Tiny 运行的模拟机器源代码。

本书系《编译技术》的配套实践教材，可供各类高等院校计算机专业作为实践教材，也可供从事计算机工作与研究的科技人员参考。

图书在版编目(CIP)数据

编译技术课程设计与上机指导/霍林主编. —重庆：重庆大学出版社，2001.9
计算机科学与技术专业本科系列教材
ISBN 7-5624-2333-4

Ⅰ.编… Ⅱ.霍… Ⅲ.编译程序—程序设计—高等学校—教学参考资料 Ⅳ.TP314

中国版本图书馆 CIP 数据核字(2001)第 061077 号

编译技术课程设计与上机指导

霍 林 主编

责任编辑 谭 敏

*

重庆大学出版社出版发行
新 华 书 店 经 销
重庆大学建大印刷厂印刷

*

开本：787×1092 1/16 印张：20.5 字数：512 千
2001 年 9 月第 1 版 2001 年 9 月第 1 次印刷
印数：1—6 000
ISBN 7-5624-2333-4/TP·293 定价：28.00 元

前言

编译技术是计算机专业的一门重要专业课程，它蕴涵着计算机学科中解决问题的思路、抽象问题和解决问题的方法。这门课在理论、技术、方法上对学生提供了系统而强有力的训练，是提高软件人员素质和能力的重要环节。本书正是从实践入手，加强学生对所学知识的深入理解及熟练掌握，为今后解决命令解释程序的编制、界面程序的开发等各种计算机实际应用系统中经常遇到的问题，打下良好基础。

本书共分为7大部分：第一部分用Lex自动生成Tiny词法分析程序，要求根据附录A介绍的Lex使用方法，生成Tiny语言的词法分析程序，从而掌握词法分析程序自动生成方法。第2部分用YACC自动生成Tiny语法分析程序，要求根据附录B介绍的YACC使用方法，生成Tiny语言的语法分析程序，从而掌握语法分析程序自动生成方法。第3部分为词法分析，要求用C语言为某样本语言手工实现一个扫描程序，此扫描程序的输入为某样本语言的源程序，输出为二元组序列构成的文件，同时能给出源程序中存在的词法错误。第4部分为语法分析，要求用C语言，采用自顶向下分析法中的递归子程序方法手工实现某样本语言的语法分析程序。此分析程序能对词法分析模块产生的二元组文件进行分析，找出语法错误，同时能调用语义分析模块的相应子程序生成代码生成模块需要的，由四元组序列构成的文件。第5部分为语义分析，要求用C语言为某样本语言手工实现一个语义分析程序，此程序能接收语法分析模块传来的各种参数，生成各语句的四元组序列，同时返回四元组序列头指针给语法分析模块。第6部分为代码生成，要求用C语言为某样本语言手工编写一个代码生成器，此生成器接收四元组文件，生成目标语言为汇编语言的程序，生成的目标代码通过汇编程序汇编后，生成一个可执行文件。第7部分为综合练习，要求阅读附录C的Tiny语言的小型编译程序后，对此编译程序增加一个功能。

将本书的第3至第6部分给出的源程序综合起来，可构成一个能实际运行的小型编译程序。在这几部分还分别给出了相应的数据结构，各数据类型的定义规则，动态符号表数据结构示意图，保留字和特殊符号表等信息，同时给出了主要模块的流程图。

关于编译技术的教材有许多，编者在参考这些教材的基础上，根据自己在教学中的体会，感到动手编制一个编译系统

对深入了解，掌握实现编译系统的各种方法及技术有极大帮助，因而在总结指导学生编制系统的基础上，再加上一些实例形成本书。霍林老师负责第 1 至第 7 章及附录 C，D，E 的编写，施霖老师负责附录 A，B 的编写。在这里，我要特别感谢我的学生：梁贵、卢东、曹莉、王海霞、苏颖翀、刘刚、刘宇等，他们对本书的编写给予了很大的支持。

由于水平有限，书中难免存在错误与不妥之处，敬请广大读者批评指正。

编　者

2001 年 5 月

目录

第 1 章

用 Lex 自动生成 Tiny 词法分析程序

词法分析程序自动生成器 Lex 的使用方法见附录 A，下面是一个 Lex 输入文件 Tiny.l，在用 Lex 生成 C 扫描程序 lex.yy.c 后，即可编译这个程序，得到 Tiny 语言的词法分析程序，将它与附录 C 中其他的 Tiny 源文件连接，就可得到一个基于 Lex 版本的编译系统。

```
////////////////////////////////////////////////
//File:tiny.l                                 //
//Lex specification for TINY                  //
//Compiler Construction:Principles and Practice //
//Kenneth C. Louden                           //
////////////////////////////////////////////////

%{
#include "globals.h"
#include "util.h"
#include "scan.h"
//lexeme of identifier or reserved word
char tokenString[MAXTOKENLEN+1];
%}

digit       [0-9]
number      {digit}+
letter      [a-zA-Z]
identifier  {letter}+
newline     \n
whitespace  [ \t]+

%%
```

```
"if"            {return IF;}
"then"          {return THEN;}
"else"          {return ELSE;}
"end"           {return END;}
"repeat"        {return REPEAT;}
"until"             {return UNTIL;}
"read"          {return READ;}
"write"             {return WRITE;}
":="            {return ASSIGN;}
"="             {return EQ;}
"<"             {return LT;}
"+"             {return PLUS;}
"-"             {return MINUS;}
"*"             {return TIMES;}
"/"             {return OVER;}
"("             {return LPAREN;}
")"             {return RPAREN;}
";"             {return SEMI;}
{number}        {return NUM;}
{identifier}        {return ID;}
{newline}       {lineno++;}
{whitespace}        {/* skip whitespace */}
"{"             {char c;
            do
            { c = input();
              if (c == '\n') lineno++;
            } while (c != '}');
          }
          {return ERROR;}
%%
TokenType getToken(void)
{ static int firstTime = TRUE;
  TokenType currentToken;
  if (firstTime)
  { firstTime = FALSE;
    lineno++
    yyin = source;
    yyout = listing;
  }
```

```
    currentToken = yylex( ) ;
    strncpy( tokenString , yytext , MAXTOKENLEN ) ;
    if ( TraceScan ) {
    fprintf( listing , " \t%d: " , lineno ) ;
    printToken( currentToken , tokenString ) ;
    }
    return currentToken;
}
```

第 2 章

用 YACC 自动生成 Tiny 语法分析程序

语法分析程序自动生成器 YACC 的使用方法见附录 B，下面是一个 YACC 输入文件 Tiny.y，在用 YACC 生成 Tiny 的语法分析程序后，将它与附录 C 中其他的 Tiny 源文件连接，就可得到一个 Tiny 的编译系统。

```
////////////////////////////////////////////////////
//File:tiny.y                                     //
//The TINY Yacc/Bisson specification file         //
//Compiler Construction:Principles and Practice   //
//Kenneth C. Louden                               //
////////////////////////////////////////////////////
%{
#define YYPARSER //distinguishes Yacc output from other code files

#include "globals.h"
#include "util.h"
#include "scan.h"
#include "parse.h"

#define YYSTYPE TreeNode *
static char * savedName;//for use in assignments
static int savedLineNo;//ditto
static TreeNode * savedTree;//stores syntax tree for later return

%}

%token IF THEN ELSE END REPEAT UNTIL READ WRITE
%token ID NUM
%token ASSIGN EQ LT PLUS MI
```

```
%token ERROR

%%//Grammar for TINY

program          :atmt_seq
                 { savedTree = $1;}
                 ;
stmt_seq         :stmt_seq SEMI stmt
                 { YYSTYPE t = $1;
                   if (t! = NULL)
                   { while (t -> sibling! = NULL)
                       t = t -> sibling;
                     t -> sibling = $3;
                     $$ = $1;}
                   else $$ = $3;
                 }
                 |stmt { $$ = $1;}
                 ;
stmt             :if_stmt { $$ = $1;}
                 |repeat_stmt { $$ = $1;}
                 |assign_stmt { $$ = $1;}
                 |read_stmt { $$ = $1;}
                 |write_stmt { $$ = $1;}
                 |error { $$ = NULL;}
                 ;
if_stmt          :IF exp THEN stmt_seq END
                 { $$ = newStmtNode(IfK);
                   $$ -> child[0] = $2;
                   $$ -> child[1] = $4;
                 }
                 |IF exp THEN stmt_seq ELSE stmt_seq END
                 { $$ = newStmtNode(IfK);
                   $$ -> child[0] = $2;
                   $$ -> child[0] = $4;
                   $$ -> child[1] = $6;
                 }
                 ;
repeat_stmt      :REPEAT stmt_seq UNTIL exp
                 { $$ = newStmtNode(RepeatK);
```

```
                $$ -> child[0] = $2;
                $$ -> child[1] = $4;
              }
        ;
assign_stmt    :ID { savedName = copyString(tokenString);
              savedLineNo = lineno;}
        ASSIGN exp
              { $$ = newStmtNode(AssignK);
                $$ -> child[0] = $4;
                $$ -> attr.name = savedName;
                $$ -> lineno = savedLineNo;
              }
        ;
read_stmt      :READ ID
              { $$ = newStmtNode(ReadK);
                $$ - attr.name =
                 copyString(tokenString);
              }
        ;
write_stmt     :WRITE exp
              { $$ = newStmtNode(WriteK);
                $$ -> child[0] = $2;
              }
        ;
exp           :simple_exp LT simple_exp
              { $$ = newExpNode(OpK);
                $$ -> child[0] = $1;
                $$ -> child[1] = $3;
                $$ -> attr.op = LT;
              }
        |simple_exp EQ simple_exp
              { $$ = newExpNode(OpK);
                $$ -> child[0] = $1;
                $$ -> child[1] = $3;
                $$ -> attr.op = EQ;
              }
        |simple_exp { $$ = $1;}
        ;
simple_exp:simple_exp PLUS term
```

```
        { $$ = newExpNode(OpK);
          $$ -> child[0] = $1;
          $$ -> child[1] = $3;
          $$ -> attr.op = PLUS;
        }
    |simple_exp MINUS term
        { $$ = newExpNode(OpK);
          $$ -> child[0] = $1;
          $$ -> child[1] = $3;
          $$ -> attr.op = MINUS;
        }
    |term { $$ = $1;}
    ;
term    :term TIMES factor
        { $$ = newExpNode(OpK);
          $$ -> child[0] = $1;
          $$ -> child[1] = $3;
          $$ -> attr.op = TIMES;
        }
    |term OVER factor
        { $$ = newExpNode(OpK);
          $$ -> child[0] = $1;
          $$ -> child[1] = $3;
          $$ -> attr.op = OVER;
        }
    |factor { $$ = $1;}
    ;
factor  :LPAREN exp RPAREN
        { $$ = $2;}
    |NUM
        { $$ = newExpNode (ConstK);
          $$ -> attr.val = atoi(tokenString);
        }
    |ID { $$ = newExpNode(IdK);
        $$ -> attr.name =
          copyString(tokenString);
        }
    |error { $$ = NULL;}
    ;
```

```
%%

int yyerror(char * message)
{ fprintf(listing,"Syntax error at line %d:%s\n",lineno,message);

  printToken(yychar,tokenString);
  Error = TRUE;
  return 0;
}

//yylex calls getToken to make Yacc/Bison output
//cpmpatible with earlier versions of
//the TINY scanner
//
static int yylex(void)
{ return getToken();}

TreeNode * parse(void)
{ yyparse();
  return savedTree;
}
```

第3章 词法分析

在本词法分析模块中,主要分析自定义的类 C 语言的词法。自定义的语言主要包括数组定义,函数定义及附录 E 中描述的 Tiny 文法定义的语句,运算符号主要为本章第 2 部分特殊符号表里所定义的各类运算符。词法分析模块的主要功能有:

1)构造类 C 语言的静态符号表。

2)从源程序中逐一取出单词。

3)识别单词的属性并填入动态符号表中。

4)将单词转换成属性字,并输出二元组属性字流。

5)提供单词的出错处理。

3.1 系统设计框图及主要函数功能说明

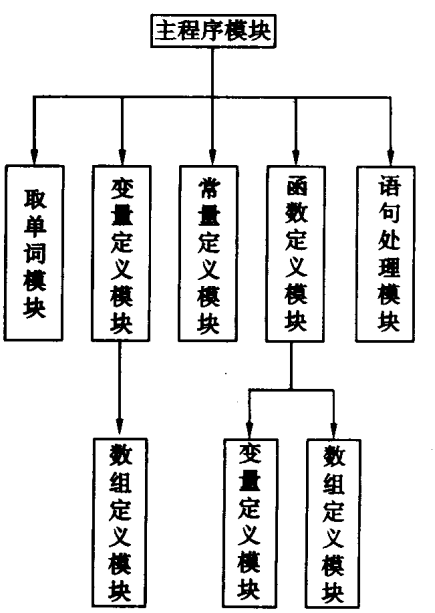

图 3.1

(1)主程序模块说明

函数名　Main()。

功能　1)调用词法分析函数 qffx(),扫描并处理源程序中的每一个单词。

2)测试词法分析函数运行的结果。

I/O 参数　1)输入参数为:要编译的源程序"aa.c"。

2)输出二元组,存放在文件"aaa.c"中。

与之相关的模块名:取单词模块、变量定义模块、常量定义模块、函数定义模块和语句处理模块。

(2)取单词模块说明

函数名　getchh():从文件"aa.c"中取一个字符。

getsy():将字符组合成单词,并存入单词栈中。

功能　从源文件"aa.c"中取出一个字符,并判断单词的类型,如:特殊符号、合法标识符、非法标识符、数字。

I/O 参数　1)从文件“aa.c”中取字符。

2)将单词存放于单词栈 getdq[3]中。

主要变量用途　1)FILE *in,*out:in 为指向文件“aa.c”的文件类型指针,out 为指向文件“aaa.c”的文件类型指针;

2)fl,hh,state:fl 表示单词的类型,hh 表示单词所处的行号,state 表示取单词中的状态。

3)c,id1[20],ta:c 用于存放从“aa.c”中取出的字符,id1[20]用于存放单词,ta=1 时,接收每一个字符做为单词,用在字符串常量时接收空格等。

与之有关的模块名:其他模块均与取单词模块有关,通过取单词模块取出单词,处理单词。

(3)变量定义处理模块说明

函数名　bldy(int type)。

功能　1)处理 int、char 和 float 三种类型变量的定义。

2)将变量的信息,如名字、类型、相对地址、在源代码中出现过的行号和所处层次等存入动态符号表。

3)如果定义出错或变量名非法,则给出出错信息。

4)输出变量的属性字。

I/O 参数　1)输入参数:在源代码中处于 int、char 或 float 到分号‘;’之间的单词。

2)将变量信息填入动态符号表,输出属性字。

程序的主要变量用途　shzh,blh,clh,cchh:分别指向数组结构、变量结构、常量结构、出错结构的指针,用于申请相应类型的结点,存放单词的信息。

shzn,bln,cln,cchn:用于将新结点连接到相应类型的链表中。

style,typee,record:用于记录类型的变量。

与之有关的模块名:取单词模块、数组定义模块。

(4)函数定义处理模块说明

函数名　hshdy(int type)。

功能　1)处理 int、char、float 和 void 四种类型函数的定义和函数的声明。

2)将函数的信息,如名字、类型、相对地址、在源代码中出现过的行号、所处层次和函数参数中定义的各种参数,以及参数的指针等填入动态符号表。

3)如果函数定义出错或变量名非法,则给出出错信息。

4)输出相应的属性字。

I/O 参数　1)输入参数:在源代码中处于 int、char、float 或 void 到分号‘;’(函数声明)或‘{’(函数定义)之间的单词。

2)将函数信息填入动态符号表,输出属性字。

主要变量用途

```
struct hsh  *p, *tt, *hshn;
struct shz  *shzh, **shzn;
struct bl   *blh, **bln;
struct cl   *clh, **cln;
struct cch  *cchh, **cchn;
```