编 译 原 理

实验指导书

上海大学计算机学院

《编译原理》课程组

2016 年 3 月

目录

编译原理实验指导书

一、课程简介

1. **课程名称：**编译原理（Principle of Compiler）
2. **课程编码：**08305013
3. **课程总学时：** 60 学时[理论：30 学时；研讨：20 学时；实验：20 学时]
4. **课程总学分：** 5 学分

二、实验目的

编译原理是计算机类专业特别是计算机软件专业的一门重要专业课。设置该课程的目的在于系统地向学生讲述编译系统的结构、工作流程及编译程序各组成部分的设计原理和实现技术，使学生通过学习既掌握编译理论和方法方面的基本知识，也具有设计、实现、分析和维护编译程序等方面的初步能力。编译原理是一门理论性和实践性都比较强的课程。进行上机实验的目的是使学生通过完成上机实验题目加深对课堂教学内容的理解。同时培养学生实际动手能力。

三、实验环境

微机 CPU P4 以上，256M 以上内存，安装好 C 语言，或 C++，或 Visual C++ 开发环境。

四、实验任务

用 C/C++/Visual C++语言编写 PL/0 语言的词法分析程序、语法分析程序、语义分析程序、中间代码生成程序。

五、PL0 语言简介

PL/0 语言功能简单、结构清晰、可读性强，而又具备了一般高级程序设计语言的必须部分，因而 PL/0 语言的编译程序能充分体现一个高级语言编译程序实现的基本方法和技术。

1．PL/0 语言文法的 EBNF

<程序>::=<分程序>.

<分程序> ::=[<常量说明>][<变量说明>][<过程说明>]<语句>

<常量说明> ::=CONST<常量定义>{，<常量定义>};

<常量定义> ::=<标识符>=<无符号整数>

<无符号整数> ::= <数字>{<数字>}

<变量说明> ::=VAR <标识符>{, <标识符>};

<标识符> ::=<字母>{<字母>|<数字>}

<过程说明> ::=<过程首部><分程序>{; <过程说明> };

<过程首部> ::=PROCEDURE <标识符>;

<语句> ::=<赋值语句>|<条件语句>|<当循环语句>|<过程调用语句>

|<复合语句>|<读语句><写语句>|<空>

<赋值语句> ::=<标识符>:=<表达式>

<复合语句> ::=BEGIN <语句> {;<语句> }END

<条件表达式> ::= <表达式> <关系运算符> <表达式> |ODD<表达式>

<表达式> ::= [+|-]<项>{<加法运算符> <项>}

<项> ::= <因子>{<乘法运算符> <因子>}

<因子> ::= <标识符>|<无符号整数>| ‘(’<表达式>‘)’

<加法运算符> ::= +|-

<乘法运算符> ::= *|/

<关系运算符> ::= =|#|<|<=|>|>=

<条件语句> ::= IF <条件表达式> THEN <语句>

<过程调用语句> ::= CALL 标识符

<当循环语句> ::= WHILE <条件表达式> DO <语句>

<读语句> ::= READ‘(’<标识符>{,<标识符>}‘)’

<写语句> ::= WRITE‘(’<表达式>{,<表达式>}‘)’

<字母> ::= a|b|…|X|Y|Z

<数字> ::= 0|1|…|8|9

2．PL/0 语言的词汇表

序号	类别	单词	编码
1	基本字	begin、 call、 const、 do、end、 if、 odd、procedure、 read、 then、 var、 while、 write	beginsym, callsym, constsym dosym, endsym, ifsym, oddsym proceduresym, readsym, thensym varsym, whilesym, writesym
2	标识符		ident
3	常数		number
4	运算符	+、 -、 *、 /、 =、 #、 <、 <=、 >、 >=、 :=	plus, minus, times, slash, eql, neq, lss, leq, gtr, geq, becomes
5	界符	（、 ）、 ,、 ;、.	Lparen, rparen, comma, semicolon period

六、实验项目

实验一 识别标识符

1. 实验目的

- 根据 PL/0 语言的文法规范，编写 PL/0 语言的标识符识别程序。
- 通过设计调试标识符识别程序，实现从源程序中分出各个标识符的方法；加深对课堂教学的理解；为后序词法分析程序的实现打下基础。
- 掌握从源程序文件中读取有效字符的方法和产生源程序的内部表示文件的法。
- 掌握识别标识符的实现方法。
- 上机调试完成的识别标识符程序的实现。

2. 实验时间

2 学时。

3. 实验内容

输入 PL/0 语言源程序，输出源程序中所有标识符的出现次数。

4. 实验要求

- 识别程序读入 PL/0 语言源程序（文本文件），识别结果也以文本文件保存。
- 按标识符出现的顺序输出结果，每个标识符一行，采用二元式序列，即：(标识符值, 标识符出现次数)
- 源程序中字符不区分大小写，即：“a1”和“A1”是同一个标识符。
- 准备至少 5 组测试用例，每组测试用例包括：输入源程序文件和输出结果。
- 测试用例应该考虑各种合法标识符的组合情况。

5. 输入输出样例

输入：（文本文件）

```
Const num=100;
```

```
Var a1,b2;
Begin
  Read(A1);
  b2:=a1+num;
  write(A1,B2);
End.
```

输出：（文本文件）

（num: 2）

（a1: 4）

（b2: 3）

实验二 词法分析

1. 实验目的

- 根据 PL/0 语言的文法规范，编写 PL/0 语言的词法分析程序。
- 通过设计调试词法分析程序，实现从源程序中分出各种单词的方法；加深对课堂教学的理解；提高词法分析方法的实践能力。
- 掌握从源程序文件中读取有效字符的方法和产生源程序的内部表示文件的法。
- 掌握词法分析的实现方法。
- 上机调试编出的词法分析程序。

2. 实验时间

4 学时。

3. 实验内容

输入 PL/0 语言程序，输出程序中各个单词符号（关键字、专用符号以及其它标记）。

4. 实验要求

- 确定编译中单词种类、使用的表格、标识符与关键字的区分方法等。
- 词法分析器读入 PL/0 语言源程序（文本文件），识别结果也以文本文件保存。
- 词法分析器的输出形式采用二元式序列，即：

  (单词种类, 单词的值)
- 源程序中字符不区分大小写，即：“a1”和“A1”是同一个标识符。
- 准备至少 5 组测试用例，每组测试用例包括：输入源程序文件和输出结果。
- 测试用例必须包含所有的基本字、运算符、界符、以及各种标识符和常数。对不合法单词进行分类考虑测试用例，特别是对一些运算符要充分考虑各种组合。

5. 输入输出样例

输入：（文本文件）

```
const a=10;
var b,c;
begin
 read(b);
 c:=a+b;
 write(c)
end.
```

输出：（文本文件）

```
    (constsym,const)
    (ident  ,     a)
    (eql  ,       =)
    (number,     10)
    (semicolon,   ;)
    (varsym,   var )
    (ident,       b)
    (comma,      , )
    (ident,      c )
    (semicolon,   ;)
    (beginsym,begin)
    (readsym, read )
    (lparen,     ( )
    (ident,       b)
    (rparen,     ) )
    (semicolon,   ;)
    (ident,      c )
    (becomes,   := )
    (ident,      a )
    (plus,       + )
    (ident,     b )
    (semicolon,  ;)
```

```
(writesym,write)
(lparen,     ( )
(ident,      c )
(rparen,     ) )
(endsym,   end )
(period,     .)
```

实验三 语法分析

1. 实验目的

- 给出 PL/0 文法规范，要求编写 PL/0 语言的语法分析程序。
- 通过设计、编制、调试一个典型的语法分析程序，实现对词法分析程序所提供的单词序列进行语法检查和结构分析，进一步掌握常用的语法分析方法。
- 选择一种语法分析方法（递归子程序法、LL(1)分析法、算符优先分析法、SLR(1)分析法）；选择常见程序语言都具备的语法结构，如赋值语句，特别是表达式，作为分析对象。

2. 实验时间

4 学时。

3. 实验内容

- 已给 PL/0 语言文法，构造表达式部分的语法分析器。
- 分析对象〈算术表达式〉的 BNF 定义如下：

<表达式> ::= [+|-]<项>{<加法运算符> <项>}

<项> ::= <因子>{<乘法运算符> <因子>}

<因子> ::= <标识符>|<无符号整数>| ‘(’<表达式>‘)’

<加法运算符> ::= +|-

<乘法运算符> ::= *|/

4. 实验要求

- 确定语法分析的方法。
- 将实验二“词法分析”的输出结果，作为表达式语法分析器的输入，进行语法解析，对于语法正确的表达式，报告“语法正确”；对于语法错误的表达式，报告“语法错误”， 指出错误原因。
- 源程序中字符不区分大小写，即：“a1”和“A1”是同一个标识符。
- 准备至少 10 组测试用例，每组测试用例包括：输入文件和输出结果。

5. 输入输出样例

输入：（文本文件）

PL/0 表达式，用实验一的输出形式作为输入。例如：对于 PL/0 表达式，（a+15）*b 用如下二元组形式作为输入：

(lparen,( )

(ident, a)

(plus, +)

(number, 15)

(rparen,))

(times, *)

(ident, b)

输出：（直接用屏幕显示）

对于语法正确的表达式，报告“语法正确”；

对于语法错误的表达式，报告“语法错误”， 并指出错误原因。

6. 选做内容

学有余力的同学，可适当扩大分析对象。譬如：

① 除表达式（算术表达式）外，扩充对条件表达式的语法分析。

② 表达式中变量名可以是一般标识符，还可含一般常数、数组元素、函数调用等等。

③ 加强语法检查，尽量多和确切地指出各种错误。

④ 直接用 PL/0 表达式源语言作为输入，例如：（a+15）*b 作为输入。

实验四 语义分析

1. 实验目的

- 通过上机实习，加深对语法制导翻译原理的理解，掌握将语法分析所识别的语法范畴变换为某种中间代码的语义翻译方法。
- 掌握目前普遍采用的语义分析方法——语法制导翻译技术。
- 给出 PL/0 文法规范，要求在语法分析程序中添加语义处理，对于语法正确的算术表达式，输出其计算值。

2. 实验时间

4 学时

3. 实验内容

已给 PL/0 语言文法，在表达式的语法分析程序里，添加语义处理部分。

4. 实验要求

- 语义分析对象重点考虑经过语法分析后已是正确的语法范畴，实习重点是语义子程序。
- 在实验三“语法分析器”的里面添加 PL/0 语言“表达式”部分的语义处理。
- 计算表达式的语义值。
- 准备至少 10 组测试用例，每组测试用例包括：输入文件和输出结果。

5. 输入输出样例

输入：（文本文件）

PL/0 算术表达式，例如:

2 + 3 * 5

输出：（直接用屏幕显示）

17

实验五 中间代码生成

1. 实验目的

- 通过上机实习，加深对语法制导翻译原理的理解，掌握将语法分析所识别的语法范畴变换为某种中间代码的语义翻译方法。
- 掌握目前普遍采用的语义分析方法——语法制导翻译技术。
- 给出 PL/0 文法规范，要求在语法分析程序中添加语义处理，对于语法正确的表达式，输出其中间代码。

2. 实验时间

4 学时。

3. 实验内容

已给 PL/0 语言文法，在实验三的表达式语法分析程序里，添加语义处理部分输出表达式的中间代码，用四元式序列表示。

4. 实验要求

- 在实验三“语法分析器”的里面添加 PL/0 语言“表达式”部分的语义处理，输出表达式的中间代码。
- 中间代码用四元式序列表示。
- 准备至少 10 组测试用例，每组测试用例包括：输入文件和输出结果。

5. 输入输出样例

输入：（文本文件）

PL/0 表达式，例如:

a * （b + c）

输出：（文本文件）

(+, b, c, t1)

(*, a, t1, t2)

实验六 代码优化

1. 实验目的

- 通过上机实习，加深对代码优化的理解，掌握基本块优化、循环优化的方法。
- 掌握利用 DAG 进行基本块优化的技术。

2. 实验时间

4 学时。

3. 实验内容

对输入的一组四元式序列（基本块）构造其相应的 DAG，在构造 DAG 过程中利用删除无用赋值、消除公共子表达式、合并已知量等局部优化技术进行优化；再从所得到的 DAG 重建四元式序列。

4. 实验要求

- 设计恰当的 DAG 表示形式（数据结构）。
- 四元式的形式包括 0 型、1 型和 2 型三种。
- 用以文本文件形式输入、输出的四元式序列。
- 准备至少 5 组测试用例，每组测试用例包括：输入文件和输出结果。

5. 输入输出样例

输入：（文本文件）

（*，A，B，T1）

（/，6，2，T2）

（-，T1，T2，T3）

（=，T3，，X）

（=，5，，C）

（*，A，B，T4）

（=，2，，C）

（+，18，C，T5）

（*，T4，T5，T6）

（=，T6，，Y）

输出：（文本文件）

（*，A，B，T1）

（=，T1，，T4）

（=，3，，T2）

（-，T1，T2，T3）

（=，T3，，X）

（=，2，，C）

（+，18，C，T5）

（*，T4，T5，T6）

（=，T6，，Y）

七、参考文献

[1] 王生原等.《编译原理》（第 3 版）.清华大学出版社.2015 年。

[2] 陈火旺等.《编译原理》.国防工业出版社.2014 年。

[3] 杜书敏,王永宁.《编译程序设计原理》.北京大学出版社，1988 年。

[4] 李赣生等.《编译程序原理与技术》.清华大学出版社.1997 年。

八、附录——PL0 语言编译源程序清单（部分）

源代码

pl0c.h

```
/* 关键字个数 */
#define norw 13
/* 名字表容量 */
#define txmax 100
/* 所有的add1用于定义数组 */
#define txmaxadd1 101
/* number的最大位数 */
#define nmax 14
/* 符号的最大长度 */
#define al 10
/* 地址上界 */
#define amax 2047
/* 最大允许过程嵌套声明层数 */
#define levmax 3
/* 最多的虚拟机代码数 */
#define cxmax 200
#define cxmaxadd1 201
/* 当函数中会发生fatal error时，返回-1告知调用它的函数，最终退出程序 */
#define getsymdo if(-1==getsym())return -1
#define getchdo if(-1==getch())return -1
#define testdo(a,b,c) if(-1==test(a,b,c))return -1
#define gendo(a,b,c) if(-1==gen(a,b,c))return -1
#define expressiondo(a,b,c) if(-1==expression(a,b,c))return -1
#define factordo(a,b,c) if(-1==factor(a,b,c))return -1
#define termdo(a,b,c) if(-1==term(a,b,c))return -1
#define conditiondo(a,b,c) if(-1==condition(a,b,c))return -1
#define statementdo(a,b,c) if(-1==statement(a,b,c))return -1
#define constdeclarationdo(a,b,c) if(-1==constdeclaration(a,b,c))return -1
#define vardeclarationdo(a,b,c) if(-1==vardeclaration(a,b,c))return -1
typedef enum {false,true} bool;
/* 符号 */
enum symbol
{nul,ident,number,plus,minus,times,slash,oddsym,eql,neq,lss,leq,gtr,geq,lparen,rparen,comma,
semicolon,period,becomes,beginsym,endsym,ifsym,thensym,whilesym,writesym,readsym,dosym,calls
ym,constsym,varsym,procsym};
#define symnum 32

/* 名字表中的类型 */
enum object {constant,variable,procedur};
```

```
/* 虚拟机代码 */
enum fct {lit,opr,lod,sto,cal,inte,jmp,jpc};
#define fctnum 8

/* 虚拟机代码结构 */
struct instruction
{
    enum fct f; /* 虚拟机代码指令 */
    int l; /* 引用层与声明层的层次差 */
    int a; /* 根据f的不同而不同 */
};
FILE* fas;    /* 输出名字表 */
FILE* fa; /* 输出虚拟机代码 */
FILE* fa1; /* 输出源文件及其各行对应的首地址 */
FILE* fa2; /* 输出结果 */
bool listswitch; /* 显示虚拟机代码与否 */
bool tableswitch; /* 显示名字表与否 */
char ch; /* 获取字符的缓冲区，getch 使用 */
enum symbol sym; /* 当前的符号 */
char id[al]; /* 当前ident */
int num; /* 当前number */
int cc,ll,kk; /* getch使用的计数器，cc表示当前字符(ch)的位置 */
int cx; /* 虚拟机代码指针 */
char line[81]; /* 读取行缓冲区 */
char a[al]; /* 临时符号 */
struct instruction code[cxmaxadd1]; /* 存放虚拟机代码的数组 */
char word[norw][al]; /* 保留字 */
enum symbol wsym[norw]; /* 保留字对应的符号值 */
enum symbol ssym[256]; /* 单字符的符号值 */
char mnemonic[fctnum][5]; /* 虚拟机代码指令名称 */
bool declbegsys[symnum]; /* 表示声明开始的符号集合 */
bool statbegsys[symnum]; /* 表示语句开始的符号集合 */
bool facbegsys[symnum]; /* 表示因子开始的符号集合 */
/* 名字表结构 */
struct tablestruct
{
    char name[al];    /* 名字 */
    enum object kind;  /* 类型：const,var or procedure */
    int val; /* 数值，仅const使用 */
    int level; /* 所处层，仅const不使用 */
    int adr; /* 地址，仅const不使用 */
    int size; /* 需要分配的数据区空间，仅procedure使用 */
};
struct tablestruct table[txmaxadd1]; /* 名字表 */
```

```
FILE* fin;
FILE* fout;
char fname[al];
int err; /* 错误计数器 */
void error(int n);
int getsym();
int getch();
void init();
int gen(enum fct x,int y,int z);
int test(bool* s1,bool* s2,int n);
int inset(int e,bool* s);
int addset(bool* sr,bool* s1,bool* s2,int n);
int subset(bool* sr,bool* s1,bool* s2,int n);
int mulset(bool* sr,bool* s1,bool* s2,int n);
int block(int lev,int tx,bool* fsys);
void interpret();
int factor(bool* fsys,int* ptx,int lev);
int term(bool* fsys,int* ptx,int lev);
int condition(bool* fsys,int* ptx,int lev);
int expression(bool* fsys,int* ptx,int lev);
int statement(bool* fsys,int* ptx,int lev);
void listcode(int cx0);
int vardeclaration(int* ptx,int lev,int* pdx);
int constdeclaration(int* ptx,int lev,int* pdx);
int postion(char* idt,int tx);
void enter(enum object k,int* ptx,int lev,int* pdx);
int base(int l,int* s,int b);
```

pl0c.c

```
/*
Windows 下c语言PL/0编译程序
在Visual C++ 6.0和Visual C.NET上运行通过
使用方法：
运行后输入PL/0源程序文件名
回答是否输出虚拟机代码
回答是否输出名字表
fa.tmp输出虚拟机代码
fa1.tmp输出源文件及其各行对应的首地址
fa2.tmp输出结果
fas.tmp输出名字表
*/
#include <stdio.h>
#include "pl0c.h"
#include "string.h"
/* 解释执行时使用的栈 */
```

```
#define stacksize 500

int main()
{
    bool nxtlev[symnum];
    init();       /* 初始化 */
    fas=fopen("fas.tmp","w");
    fa1=fopen("fa1.tmp","w");
    printf("Input file?   ");
    fprintf(fa1,"Input file?   ");
    scanf("%s",fname);      /* 输入文件名 */
    fin=fopen(fname,"r");
    if(fin)
    {
        fprintf(fa1,"%s\n",fname);
        printf("List object code?(Y/N)");     /* 是否输出虚拟机代码 */
        scanf("%s",fname);
        listswitch=(fname[0]=='y'||fname[0]=='Y');
        printf("List symbol table?(Y/N)");    /* 是否输出名字表 */
        scanf("%s",fname);
        tableswitch=(fname[0]=='y'||fname[0]=='Y');
        err=0;
        cc=cx=ll=0;
        ch=' ';
        kk=al-1;
        if(-1!=getsym())
        {
            fa=fopen("fa.tmp","w");
            fa2=fopen("fa2.tmp","w");
            addset(nxtlev,declbegsys,statbegsys,symnum);
            nxtlev[period]=true;
            if(-1==block(0,0,nxtlev))   /* 调用编译程序 */
            {
                fclose(fa);
                fclose(fa1);
                fclose(fin);
                printf("\n");
                return 0;
            }
            fclose(fa);
            fclose(fa1);
            if(sym!=period)error(9);
            if(err==0)interpret(); /* 调用解释执行程序 */
            else
```

```
            {
                printf("Errors in pl/0 program");
            }
        }
        fclose(fin);
    }
    else
    {
        printf("Can't open file!\n");
        fprintf(fal,"Can't open file!\n");
        fclose(fal);
    }
    fclose(fas);
    printf("\n");
    return 0;
}
/* 在适当的位置显示错误 */
void error(int n)
{
    char space[81];
    memset(space,32,81);
    space[cc-1]=0; /* 出错时当前符号已经读完，所以cc-1 */
    printf("****%s!%d\n",space,n);
    fprintf(fal,"****%s!%d\n",space,n);
    err++;

}

/* 词法分析，获取一个符号 */
int getsym()
{
    int i,j,k;
    while(ch==' '||ch==10||ch==9)    /* 忽略空格、换行和TAB */
    {
        getchdo;
    }
    if(ch>='a'&&ch<='z')
    {            /* 名字或保留字以a..z开头 */
        k=0;
        do
        {
            if(k<al)
            {
                a[k]=ch;
```

```
            k++;
        }
        getchdo;
    }
    while(ch>='a'&&ch<='z'||ch>='0'&&ch<='9');
    a[k]=0;
    strcpy(id,a);
    i=0;
    j=norw-1;
    do /* 搜索当前符号是否为保留字 */
    {
        k=(i+j)/2;
        if(strcmp(id,word[k])<=0)j=k-1;
        if(strcmp(id,word[k])>=0)i=k+1;
    }
    while(i<=j);
    if(i-1>j)sym=wsym[k]; else sym=ident; /* 搜索失败则，是名字或数字 */
}
else
{
    if(ch>='0'&&ch<='9')
    {           /* 检测是否为数字：以0..9开头 */
        k=0;
        num=0;
        sym=number;
        do
        {
            num=10*num+ch-'0';
            k++;
            getchdo;
        }
        while(ch>='0'&&ch<='9'); /* 获取数字的值 */
        k--;
        if(k>nmax)error(30);
    }
    else
    {
        if(ch==':')        /* 检测赋值符号 */
        {
            getchdo;
            if(ch=='=')
            {
                sym=becomes;
                getchdo;
```

```
            }
            else
            {
                sym=nul; /* 不能识别的符号 */
            }
        }
        else
        {
            if(ch=='<')        /* 检测小于或小于等于符号 */
            {
                getchdo;
                if(ch=='=')
                {
                    sym=leq;
                    getchdo;
                }
                else
                {
                    sym=lss;
                }
            }
            else
            {
                if(ch=='>')        /* 检测大于或大于等于符号 */
                {
                    getchdo;
                    if(ch=='=')
                    {
                        sym=geq;
                        getchdo;
                    }
                    else
                    {
                        sym=gtr;
                    }
                }
                else
                {
                    sym=ssym[ch];      /* 当符号不满足上述条件时，全部按照单字符符号
处理 */
                    getchdo;
                }
            }
        }
```

```
        }
    }
    return 0;
}
/* 编译程序主体 */
int block(int lev,  /* 当前分程序所在层 */
          int tx, /* 名字表当前尾指针 */
          bool* fsys /* 当前模块后跟符号集合 */
          )
{
    int i;
    int dx; /* 名字分配到的相对地址 */
    int tx0; /* 保留初始tx */
    int cx0; /* 保留初始cx */
    bool nxtlev[symnum]; /* 在下级函数的参数中，符号集合均为值参，但由于使用数租实现，
                            传递进来的是指针，为防止下级函数改变上级函数的集合，开辟新的空间
                            传递给下级函数，之后所有的nxtlev都是这样 */

    dx=3;
    tx0=tx;       /* 记录本层名字的初始位置 */
    table[tx].adr=cx;
    gendo(jmp,0,0);
    if(lev>levmax)error(32);
    do
    {
        if(sym==constsym)  /* 收到常量声明符号，开始处理常量声明 */
        {
            getsymdo;
            do
        {
                constdeclarationdo(&tx,lev,&dx); /* dx的值会被constdeclaration改变，使用指
针 */
                while(sym==comma)
                {
                    getsymdo;
                    constdeclarationdo(&tx,lev,&dx);
                }
                if(sym==semicolon)
                {
                    getsymdo;
                }
                else error(5);
            }
            while(sym==ident);
```

```
}
if(sym==varsym)        /* 收到变量声明符号，开始处理变量声明 */
{
    getsymdo;
    do
    {
        vardeclarationdo(&tx,lev,&dx);
        while(sym==comma)
        {
            getsymdo;
            vardeclarationdo(&tx,lev,&dx);
        }
        if(sym==semicolon)
        {
            getsymdo;
        }
        else error(5);
    }
    while(sym==ident);
}
while(sym==procsym) /* 收到过程声明符号，开始处理过程声明 */
{
    getsymdo;
    if(sym==ident)
    {
        enter(procedur,&tx,lev,&dx);     /* 记录过程名字 */
        getsymdo;
    }
    else error(4);     /* procedure后应为标识符 */
    if(sym==semicolon)
    {
        getsymdo;
    }
    else error(5);     /* 漏掉了分号 */
    memcpy(nxtlev,fsys,sizeof(bool)*symnum);
    nxtlev[semicolon]=true;
    if(-1==block(lev+1,tx,nxtlev))return -1;  /* 递归调用 */
    if(sym==semicolon)
    {
        getsymdo;
        memcpy(nxtlev,statbegsys,sizeof(bool)*symnum);
        nxtlev[ident]=true;
        nxtlev[procsym]=true;
        testdo(nxtlev,fsys,6);
```

```
            }
            else error(5);      /* 漏掉了分号 */
        }
        memcpy(nxtlev,statbegsys,sizeof(bool)*symnum);
        nxtlev[ident]=true;
        testdo(nxtlev,declbegsys,7);
    }
    while(inset(sym,declbegsys));    /* 直到没有声明符号 */
    code[table[tx0].adr].a=cx;  /* 开始生成当前过程代码 */
    table[tx0].adr=cx; /* 当前过程代码地址 */
    table[tx0].size=dx;     /* 声明部分中每增加一条声明都会给dx增加1，声明部分已经结束，dx就是当前过程数据的size */
    cx0=cx;
    gendo(inte,0,dx);  /* 生成分配内存代码 */
        /* 语句后跟符号为分号或end */
    memcpy(nxtlev,fsys,sizeof(bool)*symnum);  /* 每个后跟符号集和都包含上层后跟符号集和，以便补救 */
    nxtlev[semicolon]=true;
    nxtlev[endsym]=true;
    statementdo(nxtlev,&tx,lev);
    gendo(opr,0,0);     /* 每个过程出口都要使用的释放数据段指令 */
    memset(nxtlev,0,sizeof(bool)*symnum);/*分程序没有补救集合 */
    testdo(fsys,nxtlev,8); /* 检测后跟符号正确性 */
    listcode(cx0);      /* 输出代码 */
    return 0;
}
/* 初始化 */
void init()
{
    int i;
    /* 设置单字符符号 */
    for(i=0;i<=255;i++)ssym[i]=nul;
    ssym['+']=plus;
    ssym['-']=minus;
    ssym['*']=times;
    ssym['/']=slash;
    ssym['(']=lparen;
    ssym[')']=rparen;
    ssym['=']=eql;
    ssym[',']=comma;
    ssym['.']=period;
    ssym['#']=neq;
    ssym[';']=semicolon;
    /* 设置保留字名字 */
```

```
    strcpy(&(word[0][0]),"begin");
    strcpy(&(word[1][0]),"call");
    strcpy(&(word[2][0]),"const");
    strcpy(&(word[3][0]),"do");
    strcpy(&(word[4][0]),"end");
    strcpy(&(word[5][0]),"if");
    strcpy(&(word[6][0]),"odd");
    strcpy(&(word[7][0]),"procedure");
    strcpy(&(word[8][0]),"read");
    strcpy(&(word[9][0]),"then");
    strcpy(&(word[10][0]),"var");
    strcpy(&(word[11][0]),"while");
    strcpy(&(word[12][0]),"write");
    /* 设置保留字符号 */
    wsym[0]=beginsym;
    wsym[1]=callsym;
    wsym[2]=constsym;
    wsym[3]=dosym;
    wsym[4]=endsym;
    wsym[5]=ifsym;
    wsym[6]=oddsym;
    wsym[7]=procsym;
    wsym[8]=readsym;
    wsym[9]=thensym;
    wsym[10]=varsym;
    wsym[11]=whilesym;
    wsym[12]=writesym;
    /* 设置指令名称 */
    strcpy(&(mnemonic[lit][0]),"lit");
    strcpy(&(mnemonic[opr][0]),"opr");
    strcpy(&(mnemonic[lod][0]),"lod");
    strcpy(&(mnemonic[sto][0]),"sto");
    strcpy(&(mnemonic[cal][0]),"cal");
    strcpy(&(mnemonic[inte][0]),"int");
    strcpy(&(mnemonic[jmp][0]),"jmp");
    strcpy(&(mnemonic[jpc][0]),"jpc");
/* 供getsym取一个字符，每次读一行，存入line缓冲区，line被getsym取空时 再读一行*/
int getch()
{
    if(cc==ll)
    {
        if(feof(fin))
        {
            printf("program incomplete");
```

```
            return -1;
        }
        ll=0;
        cc=0;
        printf("%d ",cx);
        fprintf(fa1,"%d ",cx);
        ch=' ';
        while(ch!=10)
        {
            fscanf(fin,"%c",&ch);
            printf("%c",ch);
            fprintf(fa1,"%c",ch);
            line[ll]=ch;
            ll++;
        }
        printf("\n");
        fprintf(fa1,"\n");
    }
    ch=line[cc];
    cc++;
    return 0;
}
/* 生成一项名字表 */
void enter(enum object k,    /* 名字种类const,var or procedure */
           int* ptx,         /* 名字表尾指针的指针,为了可以改变名字表尾指针的值,以后所有的ptx
都是这样 */
           int lev,          /* 名字所在的层次,, 以后所有的lev都是这样 */
           int* pdx          /* dx为当前应分配的变量的相对地址, 分配后要增加1, 所以使用指
针, 以后所有的pdx都是这样 */
           )
{
    (*ptx)++;
    strcpy(table[(*ptx)].name,id); /* 全局变量id中已存有当前名字的名字 */
    table[(*ptx)].kind=k;
    switch(k)
    {
        case constant:     /* 常量名字 */
            if(num>amax)
            {
                error(31);     /* 数值越界 */
                num=0;
            }
            table[(*ptx)].val=num;
            break;
```

```
            case variable:     /* 变量名字 */
                 table[(*ptx)].level=lev;
                 table[(*ptx)].adr=(*pdx);
                 (*pdx)++;
                 break;
            case procedur:     /*  过程名字  */
                 table[(*ptx)].level=lev;
                 break;
       }
}
/* 查找名字的位置 */
/* 找到则返回在名字表中的位置，否则返回0 */
int postion(char* idt,  /* 要查找的名字 */
                 int tx    /* 当前名字表尾指针 */
                 )
{
     int i;
     strcpy(table[0].name,idt);
     i=tx;
     while(strcmp(table[i].name,idt)!=0)i--;
     return i;
}

/* 常量声明处理 */
int constdeclaration(int* ptx,
                          int lev,
                          int* pdx)
{
     if(sym==ident)
     {
          getsymdo;
          if(sym==eql||sym==becomes)
          {
               if(sym==becomes)error(1);    /* 把=写成了:= */
               getsymdo;
               if(sym==number)
               {
                    enter(constant,ptx,lev,pdx);
                    getsymdo;
               }
               else error(2);     /* 常量说明=后应是数字 */
          }
          else error(3);     /* 常量说明标识后应是= */
     }
```

```
    else error(4);     /* const后应是标识 */
    return 0;
}
/* 变量声明处理 */
int vardeclaration(int* ptx,int lev,int* pdx)
{
    if(sym==ident)
    {
        enter(variable,ptx,lev,pdx);     /* 填写名字表 */
        getsymdo;
    }
    else error(4);     /* var后应是标识 */
    return 0;
}
/* 语句处理 */
int statement(bool* fsys,int* ptx,int lev) /* 参数意义见block和enter函数 */
{
    int i,cx1,cx2;
    bool nxtlev[symnum];    /*  意义见block函数 */
    if(sym==ident)     /* 准备按照赋值语句处理 */
    {
        i=postion(id,*ptx);
        if(i==0)error(11); /* 变量未找到 */
        else
        {
            if(table[i].kind!=variable)
            {
                error(12);     /* 赋值语句格式错误 */
                i=0;
            }
        }
        getsymdo;
        if(sym==becomes)
        {
            getsymdo;
        }
        else error(13);     /* 检测赋值符号 */
        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
        expressiondo(nxtlev,ptx,lev);     /* 处理赋值符号右侧表达式 */
        if(i!=0)
        {
            gendo(sto,lev-table[i].level,table[i].adr);     /* expression将执行一系列指令，
但最终结果将会保存在栈顶，执行sto命令完成赋值 */
        }
```

```
}
else
{
    if(sym==readsym)   /* 准备按照read语句处理 */
    {
        getsymdo;
        if(sym!=lparen)error(34);   /* 格式错误，应是左括号 */
        else
        {
            do
            {
                getsymdo;
                if(sym==ident)i=postion(id,*ptx);    /* 查找要读的变量 */
                else i=0;
                if(i==0)error(35); /* read()中应是声明过的变量名 */
                else
                {
                    gendo(opr,0,16);   /* 生成输入指令，读取值到栈顶 */
                    gendo(sto,lev-table[i].level,table[i].adr);    /* 储存到变量 */
                }
                getsymdo;
            }
            while(sym==comma); /* 一条read语句可读多个变量 */
        }
        if(sym!=rparen)
        {
            error(33);    /* 格式错误，应是右括号 */
            while(!inset(sym,fsys))/* 出错补救，直到收到上层函数的后跟符号 */
                getsymdo;
        }
        else
        {
            getsymdo;
        }
    }
    else
    {
        if(sym==writesym)  /* 准备按照write语句处理，与read类似 */
        {
            getsymdo;
            if(sym==lparen)
            {
                do
                {
```

```
                    getsymdo;
                    memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                    nxtlev[rparen]=true;
                    nxtlev[comma]=true;          /* write的后跟符号为) or , */
                    expressiondo(nxtlev,ptx,lev);    /* 调用表达式处理，此处与read不
同，read为给变量赋值 */
                    gendo(opr,0,14);   /* 生成输出指令，输出栈顶的值 */
                }
                while(sym==comma);
                if(sym!=rparen)error(33);   /* write()中应为完整表达式 */
                else
                {
                    getsymdo;
                }
            }
            gendo(opr,0,15);   /* 输出换行 */
        }
        else
        {
            if(sym==callsym)   /* 准备按照call语句处理 */
            {
                getsymdo;
                if(sym!=ident)error(14);    /* call后应为标识符 */
                else
                {
                    i=postion(id,*ptx);
                    if(i==0)error(11); /* 过程未找到 */
                    else
                    {
                        if(table[i].kind==procedur)
                        {
                            gendo(cal,lev-table[i].level,table[i].adr);    /* 生成
call指令 */
                        }
                        else error(15);    /* call后标识符应为过程 */
                    }
                    getsymdo;
                }
            }
            else
            {
                if(sym==ifsym)     /* 准备按照if语句处理 */
                {
                    getsymdo;
```

```
                    memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                    nxtlev[thensym]=true;
                    nxtlev[dosym]=true;     /* 后跟符号为then或do */
                    conditiondo(nxtlev,ptx,lev); /* 调用条件处理（逻辑运算）函数 */
                    if(sym==thensym)
                    {
                        getsymdo;
                    }
                    else error(16);     /* 缺少then */
                    cx1=cx;   /* 保存当前指令地址 */
                    gendo(jpc,0,0);     /* 生成条件跳转指令，跳转地址未知，暂时写0 */
                    statementdo(fsys,ptx,lev);  /* 处理then后的语句 */
                    code[cx1].a=cx;     /* 经statement处理后，cx为then后语句执行完的位置，它正是前面未定的跳转地址 */
                }
                else
                {
                    if(sym==beginsym)  /* 准备按照复合语句处理 */
                    {
                        getsymdo;
                        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                        nxtlev[semicolon]=true;
                        nxtlev[endsym]=true;    /* 后跟符号为分号或end */

                        /* 循环调用语句处理函数，直到下一个符号不是语句开始符号或收到end */
                        statementdo(nxtlev,ptx,lev);
                        while(inset(sym,statbegsys)||sym==semicolon)
                        {
                            if(sym==semicolon)
                            {
                                getsymdo;
                            }
                            else error(10);     /* 缺少； */
                            statementdo(nxtlev,ptx,lev);
                        }
                        if(sym==endsym)
                        {
                            getsymdo;
                        }
                        else error(17);     /* 缺少end或； */
                    }
                    else
                    {
```

```
                    if(sym==whilesym)  /* 准备按照while语句处理 */
                    {
                        cx1=cx;  /* 保存判断条件操作的位置 */
                        getsymdo;
                        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                        nxtlev[dosym]=true;     /* 后跟符号为do */
                        conditiondo(nxtlev,ptx,lev);     /* 调用条件处理 */
                        cx2=cx;  /* 保存循环体的结束的下一个位置 */
                        gendo(jpc,0,0);     /* 生成条件跳转，但跳出循环的地址未知
*/
                        if(sym==dosym)
                        {
                            getsymdo;
                        }
                        else error(18);     /* 缺少do */
                        statementdo(fsys,ptx,lev);  /* 循环体 */
                        gendo(jmp,0,cx1);  /* 回头重新判断条件 */
                        code[cx2].a=cx;     /* 反填跳出循环的地址，与if类似 */
                    }
                    memset(nxtlev,0,sizeof(bool)*symnum);/* 语句结束无补救集合
*/
                    testdo(fsys,nxtlev,19);/* 检测语句结束的正确性 */
                }
            }
        }
    }
    return 0;
}

/* 表达式处理 */
int expression(bool* fsys,int* ptx,int lev)    /* 参数意义见block和enter函数 */
{
    enum symbol addop; /* 用于保存正负号 */
    bool nxtlev[symnum];

    if(sym==plus||sym==minus)   /* 开头的正负号，此时当前表达式被看作一个正的或负的项 */
    {
        addop=sym;     /* 保存开头的正负号 */
        getsymdo;
        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
        nxtlev[plus]=true;
        nxtlev[minus]=true;
```

```
            termdo(nxtlev,ptx,lev);/* 处理项 */
            if(addop==minus)gendo(opr,0,1);  /* 如果开头为负号生成取负指令 */
        }
        else /* 此时表达式被看作项的加减 */
        {
            memcpy(nxtlev,fsys,sizeof(bool)*symnum);
            nxtlev[plus]=true;
            nxtlev[minus]=true;
            termdo(nxtlev,ptx,lev);/* 处理项 */
        }
        while(sym==plus||sym==minus)
        {
            addop=sym;
            getsymdo;
            memcpy(nxtlev,fsys,sizeof(bool)*symnum);
            nxtlev[plus]=true;
            nxtlev[minus]=true;
            termdo(nxtlev,ptx,lev);/* 处理项 */
            if(addop==plus)
            {
                gendo(opr,0,2);    /* 生成加法指令 */
            }
            else gendo(opr,0,3);    /* 生成减法指令 */
        }
        return 0;
}
/* 条件处理 */
int condition(bool* fsys,int* ptx,int lev)     /* 参数意义见block和enter函数 */
{
        enum symbol relop;
        bool nxtlev[symnum];
        if(sym==oddsym)    /* 准备按照odd运算处理 */
        {
            getsymdo;
            expressiondo(fsys,ptx,lev);
            gendo(opr,0,6);    /* 生成odd指令 */
        }
        else
        {
            /* 逻辑表达式处理 */
            memcpy(nxtlev,fsys,sizeof(bool)*symnum);
            nxtlev[eql]=true;nxtlev[neq]=true;
            nxtlev[lss]=true;nxtlev[leq]=true;
            nxtlev[gtr]=true;nxtlev[geq]=true;
```

```
            expressiondo(nxtlev,ptx,lev);
            if(sym!=eql&&sym!=neq&&sym!=lss&&sym!=leq&&sym!=gtr&&sym!=geq)error(20);
            else
            {
                relop=sym;
                getsymdo;
                expressiondo(fsys,ptx,lev);
                switch(relop)
                {
                    case eql:
                        gendo(opr,0,8);
                        break;
                    case neq:
                        gendo(opr,0,9);
                        break;
                    case lss:
                        gendo(opr,0,10);
                        break;
                    case geq:
                        gendo(opr,0,11);
                        break;
                    case gtr:
                        gendo(opr,0,12);
                        break;
                    case leq:
                        gendo(opr,0,13);
                        break;
                }
            }
        }
        return 0;
}
/* 项处理 */
int term(bool* fsys,int* ptx,int lev)/* 参数意义见block和enter函数 */
{
        enum symbol mulop; /* 用于保存乘除法符号 */
        bool nxtlev[symnum];

        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
        nxtlev[times]=true;
        nxtlev[slash]=true;
        factordo(nxtlev,ptx,lev);   /* 处理因子 */
        while(sym==times||sym==slash)
        {
```

```
            mulop=sym;
            getsymdo;
            factordo(nxtlev,ptx,lev);
            if(mulop==times)
            {
                gendo(opr,0,4);    /* 生成乘法指令 */
            }
            else
            {
                gendo(opr,0,5);    /* 生成除法指令 */
            }
        }
        return 0;
}
/* 因子处理 */
int factor(bool* fsys,int* ptx,int lev)   /* 参数意义见block和enter函数 */
{
        int i;
        bool nxtlev[symnum];
        testdo(facbegsys,fsys,24);  /* 检测因子的开始符号 */
        while(inset(sym,facbegsys)) /* 循环直到不是因子开始符号 */
        {
            if(sym==ident)     /* 因子为常量或变量 */
            {
                i=postion(id,*ptx);     /* 查找名字 */
                if(i==0)error(11); /* 名字未声明 */
                else
                {
                    switch(table[i].kind)
                    {
                        case constant:     /* 名字为常量 */
                            gendo(lit,0,table[i].val);  /* 直接把常量的值入栈 */
                            break;
                        case variable:     /* 名字为变量 */
                            gendo(lod,lev-table[i].level,table[i].adr);    /* 找到变量地址并
将其值入栈 */
                            break;
                        case procedur:     /* 名字为过程 */
                            error(21);     /* 不能为过程 */
                            break;
                    }
                }
                getsymdo;
            }
```

```
        else
        {
            if(sym==number)    /* 因子为数 */
            {
                if(num>amax)
                {
                    error(31);
                    num=0;
                }
                gendo(lit,0,num);
                getsymdo;
            }
            else
            {
                if(sym==lparen)    /* 因子为表达式 */
                {
                    getsymdo;
                    memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                    nxtlev[rparen]=true;
                    expressiondo(nxtlev,ptx,lev);
                    if(sym==rparen)
                    {
                        getsymdo;
                    }
                    else error(22);    /* 缺少右括号 */
                }
                test(fsys,facbegsys,23);    /* 因子后有非法符号 */
            }
        }
    }
    return 0;
}
```