编译技术

课 程 设 计 指 导 书

题目：L 语言编译器的设计与实现

编写： 鲁 斌

院系： 计 算 机 系

2005 年 **11** 月 **1** 日

第一节 概 述

1 课程设计的目的和任务

编译技术是一门实践性很强的课程，只有通过实践，才能真正掌握。

实际的编译程序是十分复杂的，有时由多达十几万条指令组成。为此，编译原理的实践教学，采用简化高级程序设计语言文法的办法，重点针对词法分析、语法分析、语义分析和中间代码生成以及目标代码生成等几个关键环节进行编程和调试训练。

每个环节可作为一个独立的实践课题，分别编程调试，然后再连接在一起总调，从而实现一个完整的编译器。

2 实践方法

任何一个实用的高级语言，其语法都比较复杂，如选其作为源语言，很难实践全过程，故本实践将定义一个简化的类 PASCAL 语言——L 语言作为源语言，设计调试出它的编译程序。

因任务具有一定的难度和工作量，且时间有限，因此，分组进行实验，每组 2 人，分别承担词法分析/目标代码生成以及语法/语义分析两部分中的一个，具体分工由组内协商进行，并且整个代码的链接工作由承担词法分析任务的同学完成。

可使用 C、VC++等语言编程实现。

3 实践报告的规范和要求

每位学生完成承担任务后写出格式规范的实验报告，包括封面、任务说明、程序设计时考虑的算法和方法、流程图、主要代码说明、运行结果分析等。

4 L 语言定义

程序定义：

〈程序〉→ program〈标识符〉〈程序体〉.

〈程序体〉→〈变量说明〉〈复合句〉

变量定义：

〈变量说明〉→ var〈变量定义〉| ε

〈变量定义〉→〈标识符表〉:〈类型〉; | 〈标识符表〉:〈类型〉;〈变量定义〉

〈标识符表〉→〈标识符〉,〈标识符表〉 | 〈标识符〉

语句定义：

〈复合句〉→ begin〈语句表〉end

〈语句表〉→〈执行句〉;〈语句表〉 | 〈执行句〉

〈执行句〉→〈简单句〉 | 〈结构句〉

〈简单句〉→〈赋值句〉

〈赋值句〉→〈变量〉:＝〈表达式〉

〈变量〉→〈标识符〉

〈结构句〉→〈复合句〉 | 〈if 句〉 | 〈WHILE 句〉

〈if 句〉→ if〈布尔表达式〉then〈执行句〉| if〈布尔表达式〉then〈执行句〉else〈执

行句〉

〈while 句〉→ while〈布尔表达式〉do〈执行句〉

表达式定义：

〈表达式〉→〈算术表达式〉｜〈布尔表达式〉

〈算术表达式〉→〈算术表达式〉＋〈项〉｜〈算术表达式〉－〈项〉｜〈项〉

〈项〉→〈项〉＊〈因子〉｜〈项〉／〈因子〉｜〈因子〉

〈因子〉→〈算术量〉｜（〈算术表达式〉）

〈算术量〉→〈标识符〉｜〈整数〉｜〈实数〉

〈布尔表达式〉→〈布尔表达式〉or〈布尔项〉｜〈布尔项〉

〈布尔项〉→〈布尔项〉and〈布尔因子〉｜〈布尔因子〉

〈布尔因子〉→ not〈布尔因子〉｜〈布尔量〉

〈布尔量〉→〈布尔常数〉｜〈标识符〉｜（〈布尔表达式〉）｜〈关系表达式〉

〈关系表达式〉→〈标识符〉〈关系运算符〉〈标识符〉

〈关系运算符〉→〈|〈＝｜＝|〉＝|〉|〈〉

类型定义：

〈类型名〉→ integer｜bool | real

单词定义：

〈标识符〉→〈字母〉｜〈标识符〉〈字母〉｜〈标识符〉〈数字〉

〈整数〉→〈数字〉｜〈整数〉〈数字〉

〈实数〉→〈整数〉.｜〈实数〉〈数字〉

〈布尔常数〉→ true｜false

字符定义：

〈字母〉→ A｜B｜C｜D｜E｜F｜G｜H｜I｜J｜K｜L｜M｜N｜O｜P｜Q｜R｜S｜T｜
U｜V｜W｜X｜Y｜Z｜a｜b｜c｜d｜e｜f｜g｜h｜i｜j｜k｜l｜m｜n｜o｜
p｜q｜r｜s｜t｜u｜v｜w｜x｜y｜z

〈数字〉→ 0｜1｜2｜3｜4｜5｜6｜7｜8｜9

5 源程序书写格式规定

(1) 单词必须在一行内写完，即：一个单词不能分两行写；

(2) 源程序语句的书写采用自由格式，即：一行可写多个语句，一个语句也可分多行写；

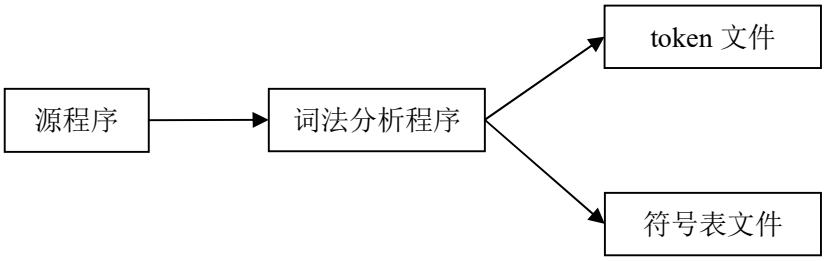
(3) 源程序不含注释；

(4) 语句以“；”结束，“end”前的一个语句的“；”不可以省略。

第二节 词法分析

1 目 的

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；掌握词法分析器作为子程序以及一遍的处理过程。



2 任 务

(1) 能对任何 L 语言源程序进行分析；
(2) 采用问答方式输入源程序文件名，然后进行词法分析；
(3) 分割单词并转换成机内表示形式，形成 token 文件（单词序列）、符号表文件；
(4) 删除空格等无用符号；
(5) 错误处理

- 给出的错误信息包括：总的出错个数，每个错误所在行号，错误编号及说明；
- 只处理以下两种错误，其它可不必考虑
  1. 非法字符：删除，即，不写入 token 文件
  2. 错误单词
     a) 包括三种形式：
        i. 数字开头的数字、字母串，如：3a56
        ii. 实数中出现两个小数点，如：3.14.15
        iii. 实数的小数部分出现字母，如：5.26B78
     b) 处理方式：截去后面出错部分，使其成为一个正确单词（即：常数）。如：3a56 转换为 3，3.14.15 转换为 3.14，5.26B78 转换为 5.26

3 数据结构

3.1 输入

L 源程序，为文本文件。

3.2 输出

一个单词序列文件（即：token 文件）和一个符号表文件，并输出错误信息。

(1) token 文件结构

```
typedef struct token
{
int label;   //单词序号
```

char name[30]; //单词本身

int code; //单词的机内码

int addr; //地址，单词为保留字时为-1，为标识符或常数时为大于 0 的数值，即在符号表中的入口地址。

} token;

单词的机内码表示：

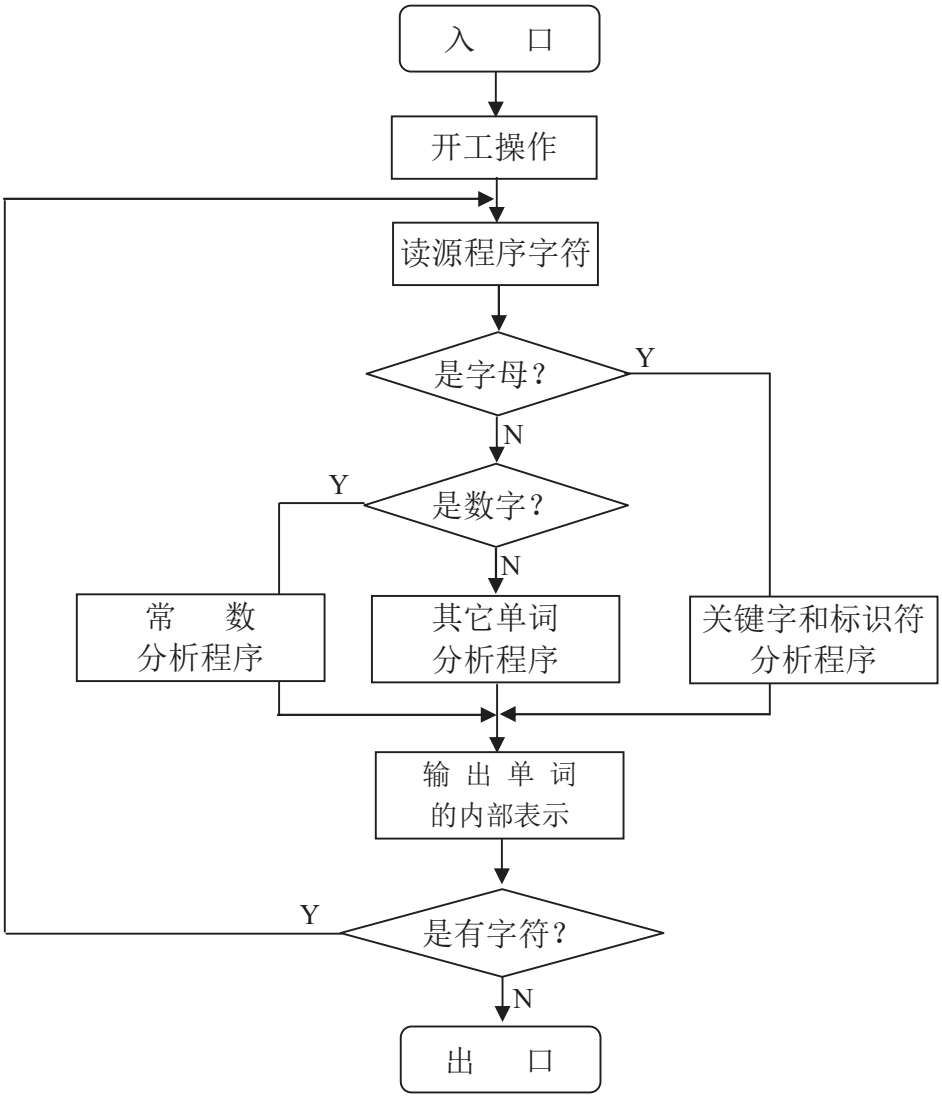
单词	编码	单词	编码	单词	编码	单词	编码
and	1	or	11	(	21	:=	31
begin	2	program	12	)	22	=	32
bool	3	real	13	+	23	<=	33
do	4	then	14	-	24	<	34
else	5	true	15	*	25	<>	35
end	6	var	16	/	26	>	36
false	7	while	17	.	27	>=	37
if	8	标识符	18	,	28		
integer	9	整数	19	:	29		
not	10	实数	20	;	30		

(2) 符号表文件结构

符号表用来存放 L 语言源程序中出现的标识符和常数，文件结构如下：

```
typedef struct symble
{
    int number;  //序号
    int type;  //类型
    char name[30];  //名字
} symble;
```

4 词法分析程序流程图


入 口
开工操作
读源程序字符
是字母？
Y
N
是数字？
Y
N
常 数
分析程序
其它单词
分析程序
关键字和标识符
分析程序
输 出 单 词
的内部表示
是有字符？
Y
N
出 口

第三节 语法/语义分析

1 目 的

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。

2 任 务

在词法分析程序产生的 token 文件、符号表文件基础上，完成语法和语义分析，产生相应的中间代码——四元式序列。在此，可把语法/语义分析作为独立的一遍进行处理。

采用如下四元式：

操作码助记符	四元式	意 义
:=	(51, a, 0, r)	r←a
+	(43, a, b, r)	r←a+b
-	(45, a, b, r)	r←a-b
*	(41, a, b, r)	r←a*b
/	(48, a, b, r)	r←b/a
j<	(53, a, b, n)	若 a<b 转至第 n 个四元式
j<=	(54, a, b, n)	若 a<=b 转至第 n 个四元式
j>	(57, a, b, n)	若 a>b 转至第 n 个四元式
j>=	(58, a, b, n)	若 a>=b 转至第 n 个四元式
j =	(56, a, b, n)	若 a=b 转至第 n 个四元式
j	(52, 0, 0, n)	转至第 n 个四元式
j <>	(55, a, b, n)	若 a<>b 转至第 n 个四元式

3 数据结构

3.1 输入

token 文件、符号表文件，其数据结构与词法分析产生的文件相同。

3.2 输出

- 四元式序列文件，其纪录结构如下：

```
typedef struct equ
{
    int op;   //四元式操作码
    int op1;   //操作数在符号表中的入口地址
    int op2;   //操作数在符号表中的入口地址
    int result;   //结果变量在符号表中的入口地址
} equ;
```

- 程序中可用数组 Equ 存放四元式序列，定义为：equ Equ[EQU_LEN]
- 可用变量 int LineOfEqu 做指针，指向下一个即将产生的四元式
- 符号表文件的结构与输入相同，语法分析中对于符号表不做操作，只是在文件头部增加一个记录变量多少的数据。

4 程序结构说明

为方便编程，将语言文法整理如下：

L→S | S ; L

S→id := E

S→if B then S

S→if B then S else S

S→while B do S

S→begin L end

变量说明语句的文法：

S→var D| ε

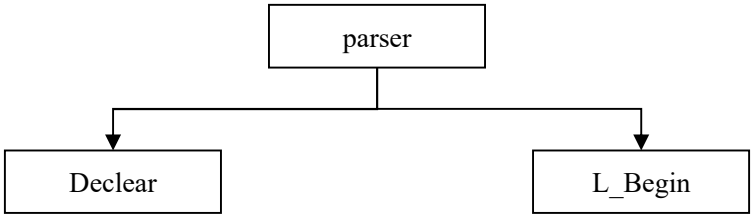
D→L : K ; | L : K ; D

L→i , L | i

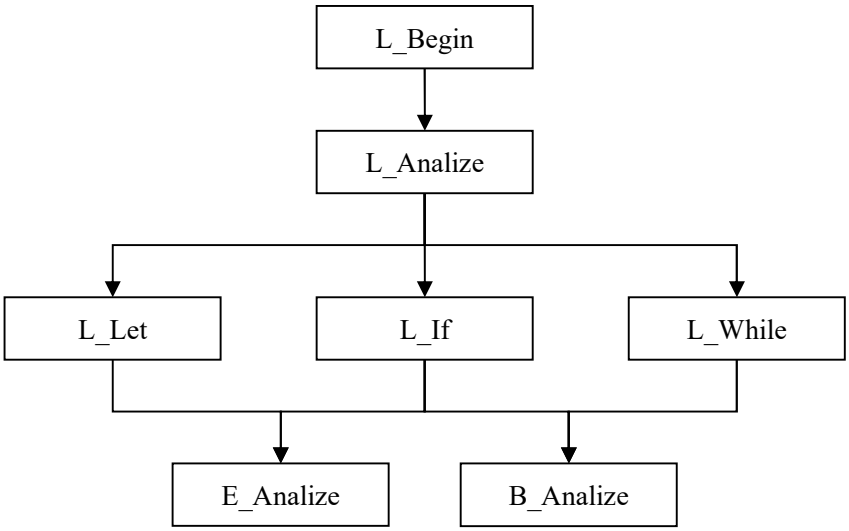
K→integer | bool | real

其中，B 表示布尔表达式，E 表示算术表达式

可采用递归下降分析法或其它方法进行语法分析。语法/语义分析程序可划分为三个模块，结构如下：



parser 为主程序模块，Declear 为说明语句分析模块，L_begin 为复合语句分析模块。其中 L_begin 结构图如下：



经常使用的过程和函数：

- gen(int op,int a,int b, int r)：生成一个四元式
- NewTemp()：产生一个临时变量，返回其在符号表中的入口地址
- BackPatch(int addr, int addr2)：回填函数，完成四元式转移目标的回填

第四节 目标代码生成

1 目 的

实践目标代码的生成方法。

2 任 务

编写一个目标代码生成程序，将 L 语言的中间代码程序翻译为目标代码程序（汇编语言程序），如下图：

目标机说明：

- 以 8086 微处理机为目标机，生成 8086 汇编指令
- 8086 是 16 位微处理器，数据总线为 16 位，地址总线为 20 位，可寻址 1MB 的空间
- 8086 有 8 个 16 位通用寄存器和一个标志寄存器。 这 8 个寄存器 AX-DI 都可以用作累加器。其中，BX 和 BP（基地址指针）寄存器通常用于指定数据区的基址，称为基址寄存器，SI 和 DI 大多用来表示相对基址的偏移量，称为变址寄存器
- 8086 的地址空间是分段的，每段 64KB
- 简单起见，本实验不涉及段间寻址，数据与代码都放在一个段内
- 实验中选用以下寻址功能，圆括号表示取其内容：
  - 寄存器寻址： MOV AX,BX;
    功能：AX←(BX)
  - 直接寻址：MOV AX,DATA;
    功能：AX←(DATA)
- 常用指令：
  - 传送指令：r 表示寄存器，m 表示内存单元
    MOV r, r/m r←(r/m), r/m 表示 r 或 m
    MOV r/m, r r/m←(r)
    MOV r/m, imm r/m←imm, imm 是立即数
  - 运算指令：包括 ADD，SUB，MUL，DIV，CMP 等。下面以 ADD 为例说明其用法：
    ADD r, r/m r←(r)+(r/m)
    ADD r/m, r/imm r/m←(r/m)+(r)或 imm
  - CMP 只影响标志位，不影响操作数的大小

■ 转移指令：Z 是标志位，S 是符号位，O 是溢出位

指令码	意义	条件
JZ, JE	结果为 0 或相等则转	Z=1，(A) = (B)
JNZ, JNE	结果不为 0 或不相等则转	Z=0，(A)≠(B)
JNL, JGE	大于等于转	(S∨O)=0，(A)≥(B)
JL, JNGE	小于转	(S∨O)=1，(A)<(B)
JG, JNLE	大于转	(S∨O∨Z)=0，(A)>(B)
JMP	无条件转移	

3 数据结构

3.1 输入

四元式序列文件和符号表文件，其结构与语法/语义分析程序的输出一致。

3.2 输出

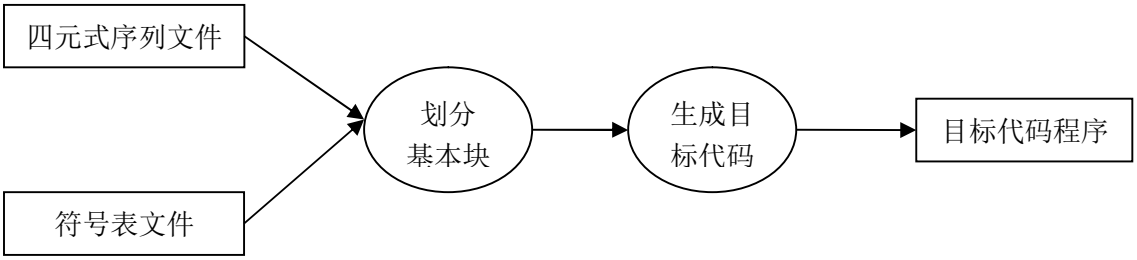
一个汇编代码文件，并无特殊数据结构。

4 程序参考结构：

将中间代码程序（四元式序列）翻译成汇编程序可按以下步骤进行：

(1) 划分基本块

(2) 对每个基本块生成基本块的目标代码



为了划分和记录基本块，对四元式结构作以下修改：

```
typedef struct GenStruct
{
    int label;
    char op[4];
    int code;
    int addr1;
    int addr2;
    int result;
    int out_port;   //记录该四元式是否为一个基本块的入口，是则为 1，否则为 0。
} GenStruct;
```

5 寄存器分配策略

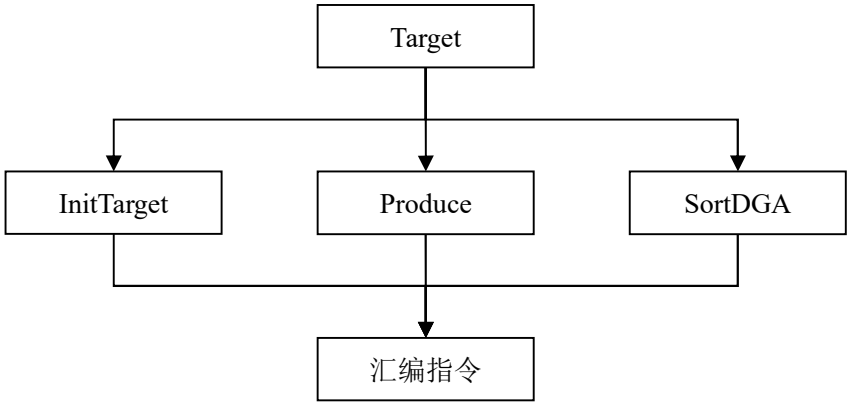
主要采用四个通用寄存器： ax, bx, cx, dx, 其中，ax, cx 的作用固定，ax 用作累加器，cx 用作循环计数器，在四元式翻译时直接应用不再分配。所以分配策略只用于 bx 与 dx，具

体算法如下：

```
if (bx 未被使用或已分配给了变量 a) {
    bx 分配给变量 a;
}
else {
    if (dx 未被使用或已分配给了变量 a) {
        dx 分配给变量 a;
    }
    else {
        其它策略;
    }
}
```

注：a 为变量在符号表的入口地址。

6 代码生成器的模块结构及说明



实现时，整个程序的四元式表和目标代码文件说明为全局数据。调用划分基本块模块之后，返回新的四元式序列（带入口标记）。先根据基本块的入口，再查找下一入口，两个入口之间就是该基本块。