《 编 译 原 理 》

实 验 指 导 书

李宏芒　编 写

适用专业：计算机科学与技术

合肥工业大学计算机与信息学院

2012 年 12 月

前　言

《编译原理》是计算机专业的一门核心课程，在计算机本科教学中占有十分重要的地位。由于《编译原理》课程兼有很强的理论性和实践性，并且编译程序构造的算法比较复杂，因而让学生在学习时普遍感到内容抽象、不易理解，难易掌握。但是掌握编译原理的基本理论和设计思想是非常重要的，尤其是将本课程的理论知识与计算机应用中的许多领域紧密联系与广泛应用结合，将有利于提高学生专业素质和适应社会多方面需要的能力，因此，通过理论授课和上机实践相结合，使学生对编译的基本概念、原理和方法有完整的和清楚的理解，并能正确地、熟练地加以运用；通过实验逐步提高学生的编程能力和调试程序的能力以及解决实际问题的能力，使学生培养出扎实的软件开发基本技能，并养成良好的编程风格，为进一步学习后续课程和将来从事应用软件开发奠定良好的基础。

实验课时具体内容安排如下：

序号	实验名称	课时	必（选）做
实验一	词法分析设计	4	必做
实验二	LL(1)预测分析	4	必作
实验三	LR 语法分析设计	4	必做
实验四	逆波兰表达式的产生及计算	3	选做
实验五	应用 DAG 进行局部优化	3	选做
实验六	C 语言子集编译程序	课外	选做

一、实验课的性质和目的

（1）深刻理解程序语言编译系统的结构及各部分的功能。

（2）熟练掌握设计和构造程序语言编译系统的基本原理和技术。

（3）能独立编写清晰、工整、结论正确的编译原理的源程序。

（4）能学会上机进行正确调试，并进行程序修改。即培养发现程序错误，排除错误的能力和经验。

二、实验课的基本要求：

（1）掌握编译程序的功能和结构。

（2）掌握词法分析器的设计方法与实现步骤加深对讲授内容的理解，尤其是一些语法给定，通过上机实验帮助掌握。

（3）掌握语法分析器的设计方法与实现步骤。

（4）掌握符号表和存储空间的组织。

（5）掌握代码优化的作用与实现方法

（6）掌握错误的诊断和校正方法。

三、主要实验教学方法

实验前，由任课教师落实实验任务，每个学生必须事先独立完成好程序的设计的源程序编写工作。实验课上对疑难点作集中辅导。实验过程中随时针对不同

的情况作个别启发式辅导。实验后，学生撰写并提交实验报告。最后，由实验教师根据每个学生的编程、上机调试能力、编程能力和实验结果及实验报告综合评定学生的实验成绩。

四、实验的重点与难点：

对词法分析设计、语法分析设计和中间代码的产生、代码优化等是本课程实践性环节的重点和难点。

五、实验教学手段

通过本课程的课内实验，使学生上机编程、调试来验证和巩固所学的编译原理理论及概念，逐步掌握词法分析的设计方法及实现技术。软件实验室为为每个学生提供了一台具有 WINDOWS 98/XP/NT/2000 操作系统的计算机和 VC++/VB/JAVA/TC 等软件环境。

六、实验考核成绩

《编译原理》是一门实践性很强的课程,要求在教学过程中必须十分重视实践性环节，包括平时练习作业、记分作业、上机实验等。尤其是要注重上机实验的重要性，必须通过上机实践才能真正掌握所学的知识和技能，所以要特别强调实验也将作为考核成绩的依据。实验成绩占平时成绩的 20%。每次必须完成规定的实验内容，并及时写出实验报告。

七、实验报告内容：

1．实验题目、班级、学号、姓名、完成日期。

2．写出数据结构及生成的算法描述。

3．画出算法流程图。

4．打印出源程序代码和给出测试的结果。

5．实验的评价、收获与体会。

写出在调试过程中出现的问题和解决的措施；分析讨论对策成功或失败的原因。

目　　录

实验一　词法分析设计

实验学时：4

实验类型：综合

实验要求：必修

一、实验目的

通过本实验的编程实践，使学生了解词法分析的任务，掌握词法分析程序设计的原理和构造方法，使学生对编译的基本概念、原理和方法有完整的和清楚的理解，并能正确地、熟练地运用。

二、实验内容

用 VC++/VB/JAVA 语言实现对 C 语言子集的源程序进行词法分析。通过输入源程序从左到右对字符串进行扫描和分解，依次输出各个单词的内部编码及单词符号自身值；若遇到错误则显示“Error”，然后跳过错误部分继续显示 ；同时进行标识符登记符号表的管理。

以下是实现词法分析设计的主要工作：

（1）从源程序文件中读入字符。

（2）统计行数和列数用于错误单词的定位。

（3）删除空格类字符，包括回车、制表符空格。

（4）按拼写单词，并用（内码，属性）二元式表示。(属性值——token 的机内表示)

（5）如果发现错误则报告出错

（6）根据需要是否填写标识符表供以后各阶段使用。

单词的基本分类：

◆ 关键字：由程序语言定义的具有固定意义的标识符。也称为保留字例如 if、 for、while、printf ； 单词种别码为 1。

◆ 标识符：用以表示各种名字，如变量名、数组名、函数名；

◆ 常数： 任何数值常数。如 125, 1,0.5,3.1416；

◆ 运算符：+、-、*、/；

◆ 关系运算符： <、<=、= 、>、>=、<>；

◆ 分界符： ；、,、、（、）、[、]；

三、**词法分析实验设计思想及算法**

1、主程序设计考虑：

◆ 程序的说明部分为各种表格和变量安排空间。

在具体实现时，将各类单词设计成结构和长度均相同的形式，较短的关键字后面补空。

k 数组------关键字表，每个数组元素存放一个关键字（事先构造好关键字表）。

s 数组------存放分界符表（可事先构造好分界符表）。为了简单起见，分界符、算术运算符和关系运算符都放在 s 表中（编程时，应建立算术运算符表和关系运算符表，并且各有类号），合并成一类。

id 和 ci 数组分别存放标识符和常数。

instring 数组为输入源程序的单词缓存。

outtoken 记录为输出内部表示缓存。

还有一些为造表填表设置的变量。

◆ 主程序开始后，先以人工方式输入关键字，造 k 表；再输入分界符等造 p 表。

◆ 主程序的工作部分设计成便于调试的循环结构。每个循环处理一个单词；接收键盘上送来的一个单词；调用词法分析过程；输出每个单词的内部码。

例如，把每一单词设计成如下形式： (type,pointer)

其中 type 指明单词的种类，例如：Pointer 指向本单词存放处的开始位置。

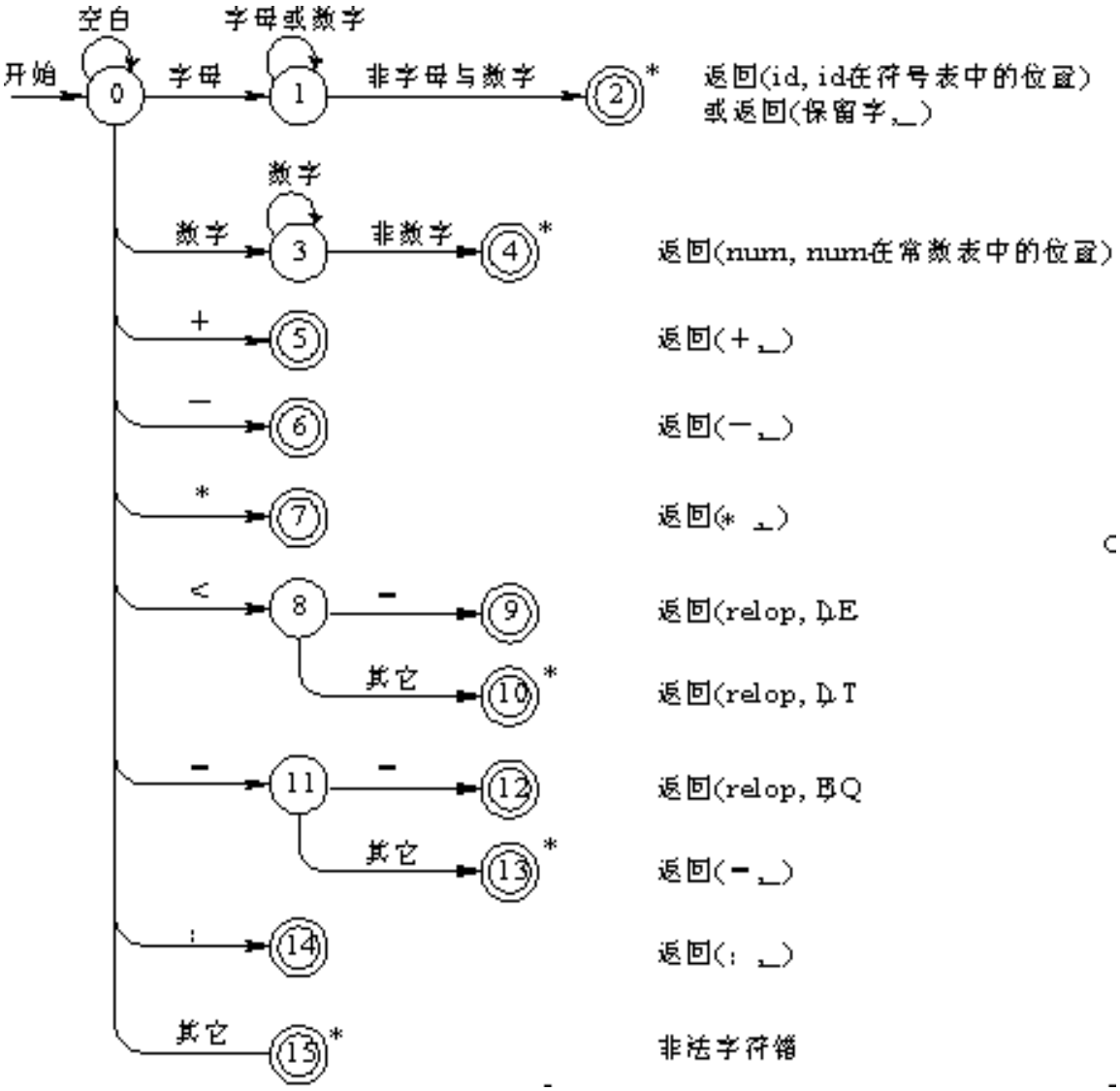
还有一些为造表填表设置的变量。

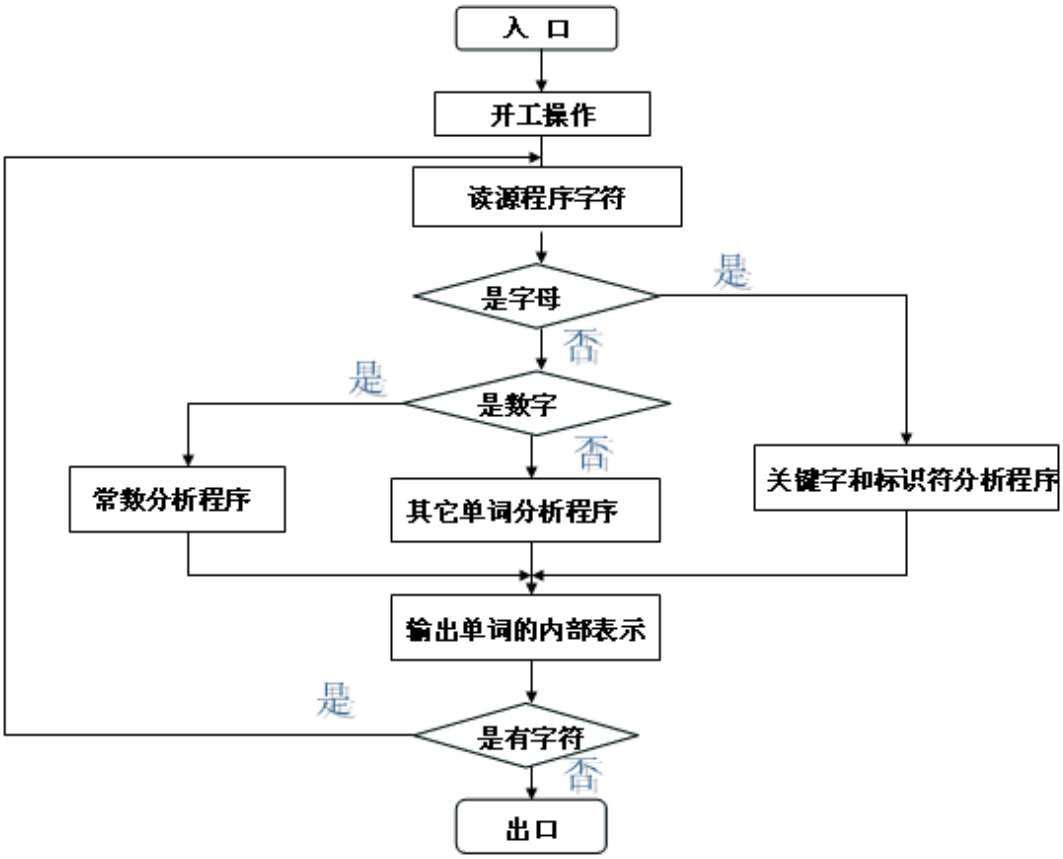
◆ 主程序开始后，先以人工方式输入关键字，造 k 表；再输入分界符等造 p 表。

◆ 主程序的工作部分设计成便于调试的循环结构。每个循环处理一个单词；接收键盘上送来的一个单词；调用词法分析过程；输出每个单词的内部码。

例如，把每一单词设计成如下形式: (type,pointer)

其中 type 指明单词的种类，例如：Pointer 指向本单词存放处的开始位置。





词法分析设计流程图

2、词法分析过程考虑

◆ 根据输入单词的第一个字符（有时还需读第二个字符），判断单词类，产生类号：以字符 k 表示关键字；id 表示标识符；ci 表示常数；s 表示分界符。

◆ 对于标识符和常数，需分别与标识符表和常数表中已登记的元素相比较，如表中已有该元素，则记录其在表中的位置，如未出现过，将标识符按顺序填入数组 id 中，将常数变为二进制形式存入数组中 ci 中，并记录其在表中的位置。**lexical** 过程中嵌有两个小过程：一个名为 **getchar**，其功能为从 **instring** 中按顺序取出一个字符，并将其指针 **pint** 加 1；另一个名为 **error**，当出现错误时，调用这个过程，输出错误编号。

◆ 要求：所有识别出的单词都用两个字节的等长表示，称为内部码。第一个字节为 t，第二个字节为 i。t 为单词的种类。关键字的 t= 1；分界符的 t= 2；算术运算符的 t= 3；关系运算符的 t= 4；无符号数的 t= 5；标识符的 t= 6。i 为该单词在各自表中的指针或内部码值。表 1 为关键字表；表 2 为

分界符表；表 3 为算术运算符的 i 值；表 4 为关系运算符的 i 值。

表1 关键字表

指针	关键字
0	do
1	end
2	for
3	if
4	printf
5	scanf
6	then
7	while

表2 分界符表

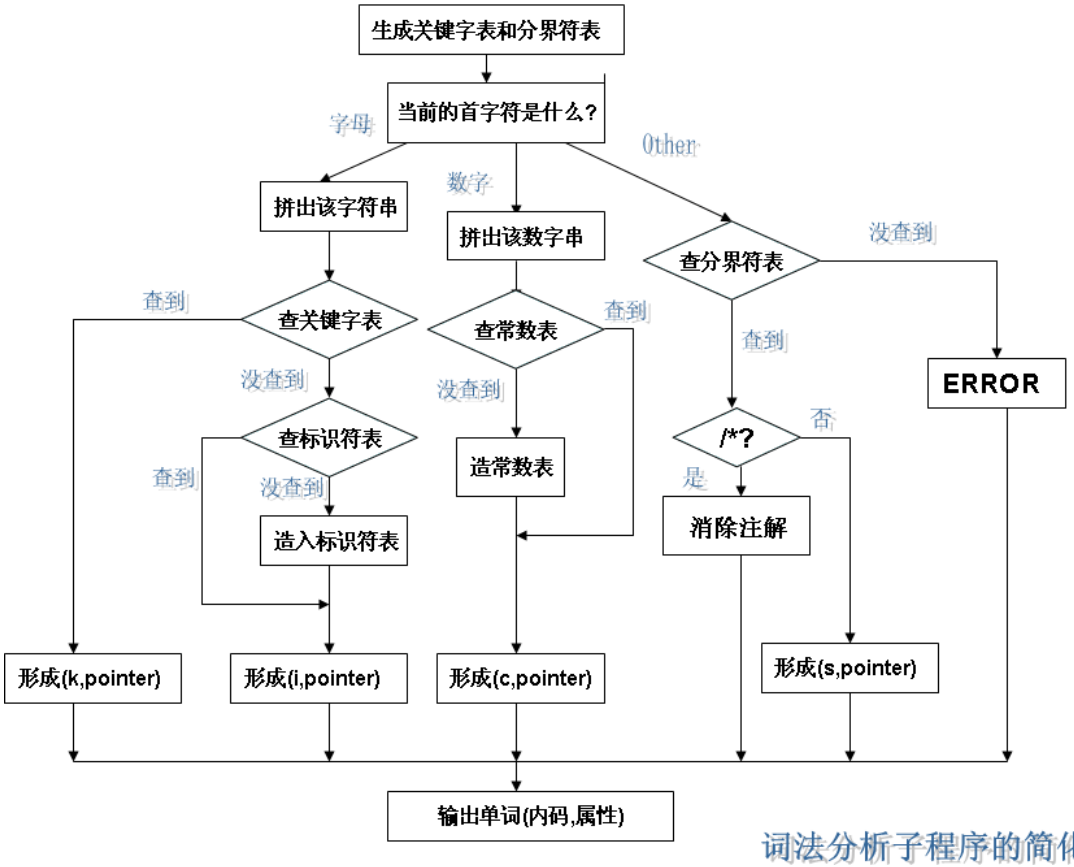
指针	分界符
0	,
1	;
2	(
3	)
4	[
5	]

表3 算术运算符

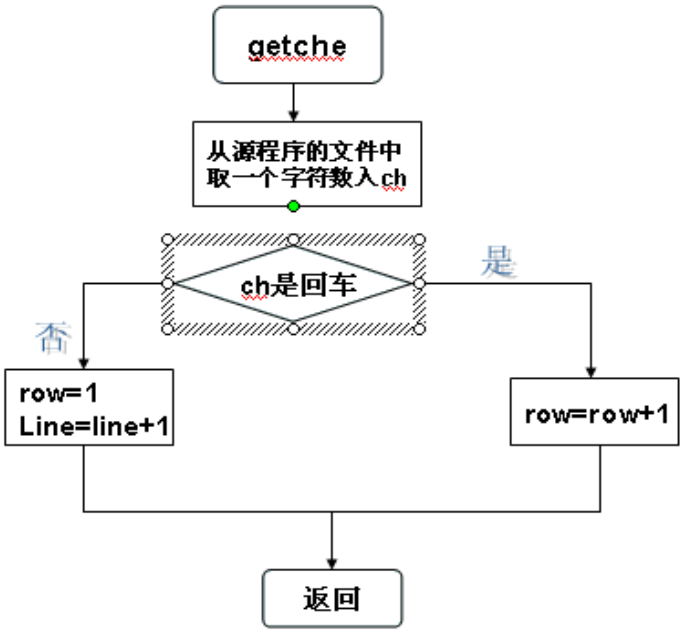
i值	算术运算符
10H	+
11H	-
20H	*
21H	/

表4 关系运算符

i值	关系运算符
00H	<
01H	<=
02H	=
03H	>
04H	>=
05H	<>



词法分析子程序的简化框图



取字符和统计字符行列位置子程序

四、实验要求

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、将标识符填写的相应符号表须提供给编译程序的以后各阶段使用。

3、根据测试数据进行测试。测试实例应包括以下三个部分:

- ◆ 全部合法的输入。
- ◆ 各种组合的非法输入。
- ◆ 由记号组成的句子。

4、词法分析程序设计要求输出形式:

例：输入 VC++语言的实例程序：

If i=0 then n++;

a < = 3b %);

输出形式为：

单词	二元序列 （单词种别，单词属性）	类 型	位置（行，列）
for	**(1,for)**	**关键字**	**（1，1）**
i	**(6,i)**	**标识符**	**（1，2）**
=	**(4，=)**	**关系运算符**	**（1，3）**

0	**(5，0)**	**常数**	**（1，4）**
then	**(1，then)**	**关键字**	**（1，5）**
n	**(6,n)**	**标识符**	**（1，6）**
++	**Error**	**Error**	**（1，7）**
;	**(2, ;)**	**分界符**	**（1，8）**
a	**(6,a)**	**标识符**	**（2，1）**
< =	**(4,<=)**	**关系运算符**	**（2，2）**
3b	**Error**	**Error**	**（2，4）**
%	**Error**	**Error**	**（2，4）**
)	**(2,))**	**分界符**	**（2，5）**
;	**(2, ;)**	**分界符**	**（2，6）**

五、实验步骤

1、根据流程图编写出各个模块的源程序代码上机调试。

2、 编制好源程序后，设计若干用例对系统进行全面的上机测试，并通过所设计的词法分析程序；直至能够得到完全满意的结果。

3、书写实验报告 ；实验报告正文的内容：

◆ 功能描述：该程序具有什么功能？

◆ 程序结构描述：函数调用格式、参数含义、返回值描述、函数功能；函数之间的调用关系图。

◆ 详细的算法描述（程序总体执行流程图）。

◆ 给出软件的测试方法和测试结果。

◆ 实验总结 （设计的特点、不足、收获与体会）。

实验二　LL(1)分析法

实验学时：4

实验类型：综合

实验要求：必修

一、实验目的

通过完成预测分析法的语法分析程序，了解预测分析法和递归子程序法的区别和联系。使学生了解语法分析的功能，掌握语法分析程序设计的原理和构造方法，训练学生掌握开发应用程序的基本方法。有利于提高学生的专业素质，为培养适应社会多方面需要的能力。

二、实验内容

- 根据某一文法编制调试 LL （ 1 ）分析程序，以便对任意输入的符号串进行分析。
- 构造预测分析表，并利用分析表和一个栈来实现对上述程序设计语言的分析程序。
- 分析法的功能是利用 LL（1）控制程序根据显示栈栈顶内容、向前看符号以及 LL（1）分析表，对输入符号串自上而下的分析过程。

三、 LL（1）分析法实验设计思想及算法

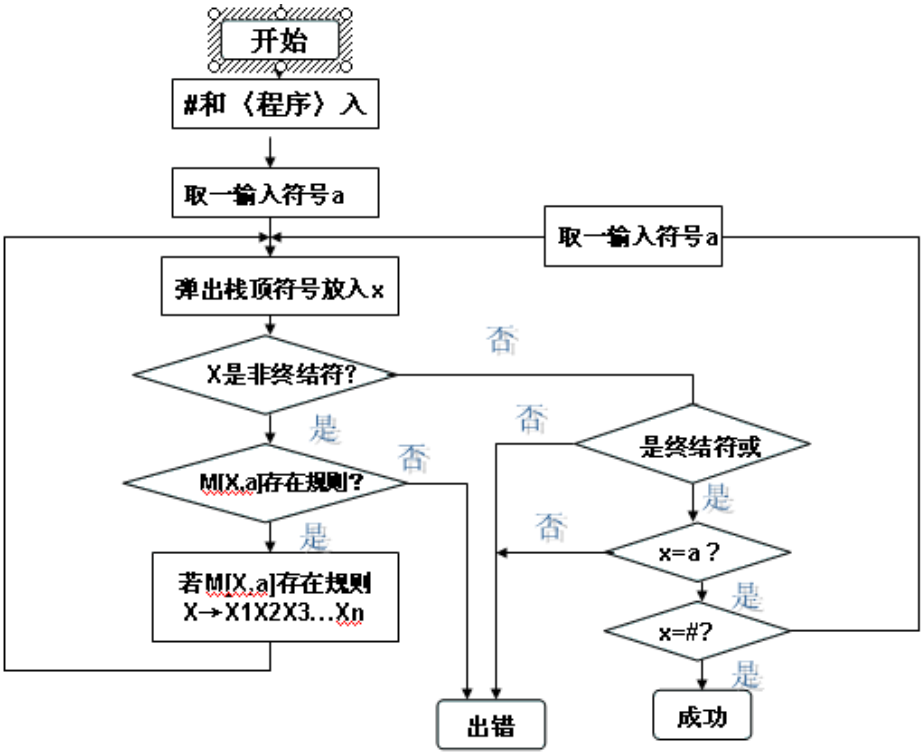
◆ **模块结构：**

（1）定义部分：定义常量、变量、数据结构。

（2）初始化：设立 LL(1)分析表、初始化变量空间（包括堆栈、结构体、数组、临时变量等）；

（3）控制部分：从键盘输入一个表达式符号串；

（4）利用 LL(1)分析算法进行表达式处理：根据 LL(1)分析表对表达式符号串进行堆栈（或其他）操作，输出分析结果，如果遇到错误则显示错误信息。



LL(1)预测分析程序流程

四、实验要求

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、如果遇到错误的表达式，应输出错误提示信息。

3、对下列文法，用 LL（1）分析法对任意输入的符号串进行分析：

（1）E->TG

（2）G->+TG|—TG

（3）G-> ε

（4）T->FS

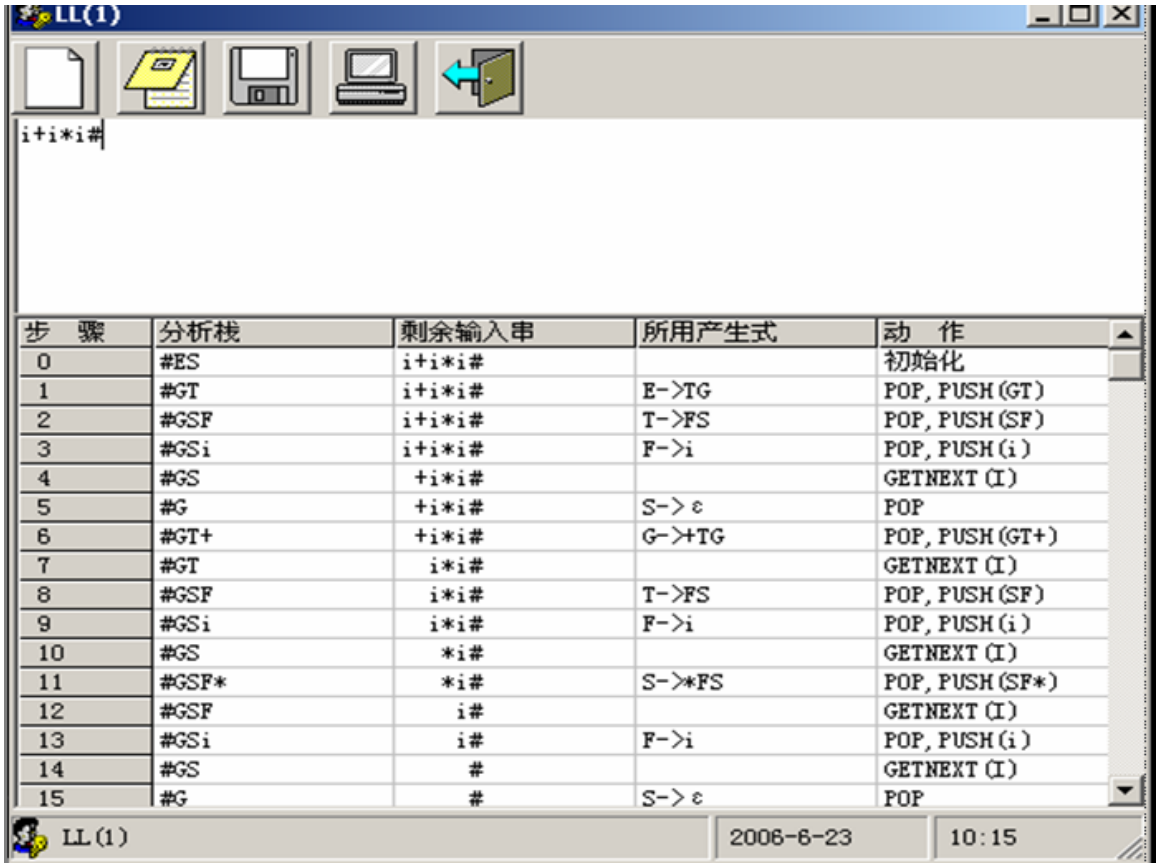
（5）S->*FS|/FS

（6）S-> ε

（7）F->(E)

（8）F->i

输出的格式如下：



五、实验步骤

1、根据流程图编写出各个模块的源程序代码上机调试。

2、 编制好源程序后，设计若干用例对系统进行全面的上机测试，并通过所设计的 LL(1)分析程序；直至能够得到完全满意的结果。

3、书写实验报告 ；实验报告正文的内容：

- ◆ 写出 LL（1）分析法的思想及写出符合 LL（1）分析法的文法。
- ◆ 程序结构描述：函数调用格式、参数含义、返回值描述、函数功能；函数之间的调用关系图。
- ◆ 详细的算法描述（程序执行流程图）。
- ◆ 给出软件的测试方法和测试结果。
- ◆ 实验总结 （设计的特点、不足、收获与体会）。

实验三　**LR(1)**分析法

实验学时：4

实验类型：验证

实验要求：必修

一、实验目的

构造 LR(1)分析程序，利用它进行语法分析，判断给出的符号串是否为该文法识别的句子，了解 LR（K）分析方法是严格的从左向右扫描，和自底向上的语法分析方法。

二、实验内容

对下列文法，用 LR（1）分析法对任意输入的符号串进行分析：

（1）E-> E+T

（2）E->T

（3）T-> T*F

（4）T->F

（5）F-> (E)

（6）F-> i

三、LR（1）分析法实验设计思想及算法

(1)总控程序，也可以称为驱动程序。对所有的 LR 分析器总控程序都是相同的。

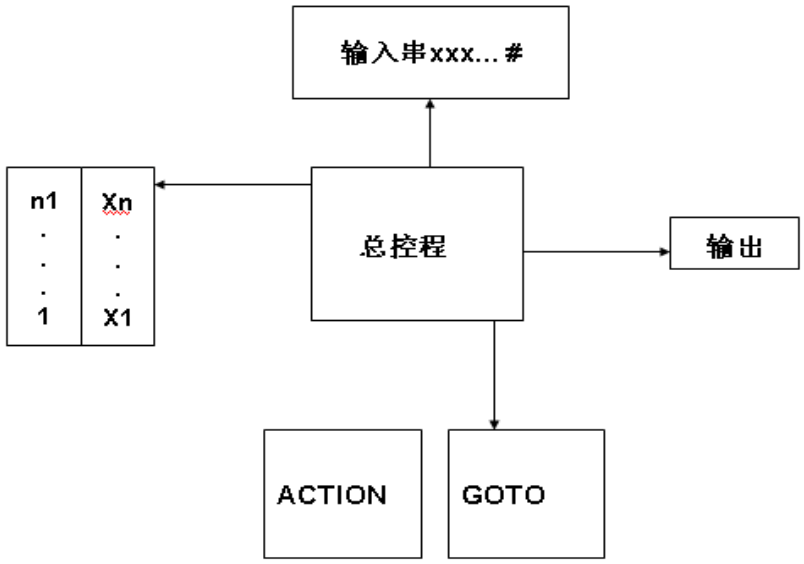
(2)分析表或分析函数，不同的文法分析表将不同，同一个文法采用的 LR 分析器不同时，分析表将不同，分析表又可以分为动作表（ACTION）和状态转换（GOTO）表两个部分，它们都可用二维数组表示。

(3)分析栈，包括文法符号栈和相应的状态栈，它们均是先进后出栈。

分析器的动作就是由栈顶状态和当前输入符号所决定。

◆ LR 分析器由三个部分组成：

LR分析器结构：



- 其中:SP 为栈指针，S[i]为状态栈，X[i]为文法符号栈。状态转换表用 GOTO[i，X]=j 表示，规定当栈顶状态为 i，遇到当前文法符号为 X 时应转向状态 j，X 为终结符或非终结符。
- ACTION[i，a]规定了栈顶状态为 i 时遇到输入符号 a 应执行。动作有四种可能：

(1)移进：

action[i，a]= Sj：状态 j 移入到状态栈，把 a 移入到文法符号栈，其中 i,j 表示状态号。

(2)归约：

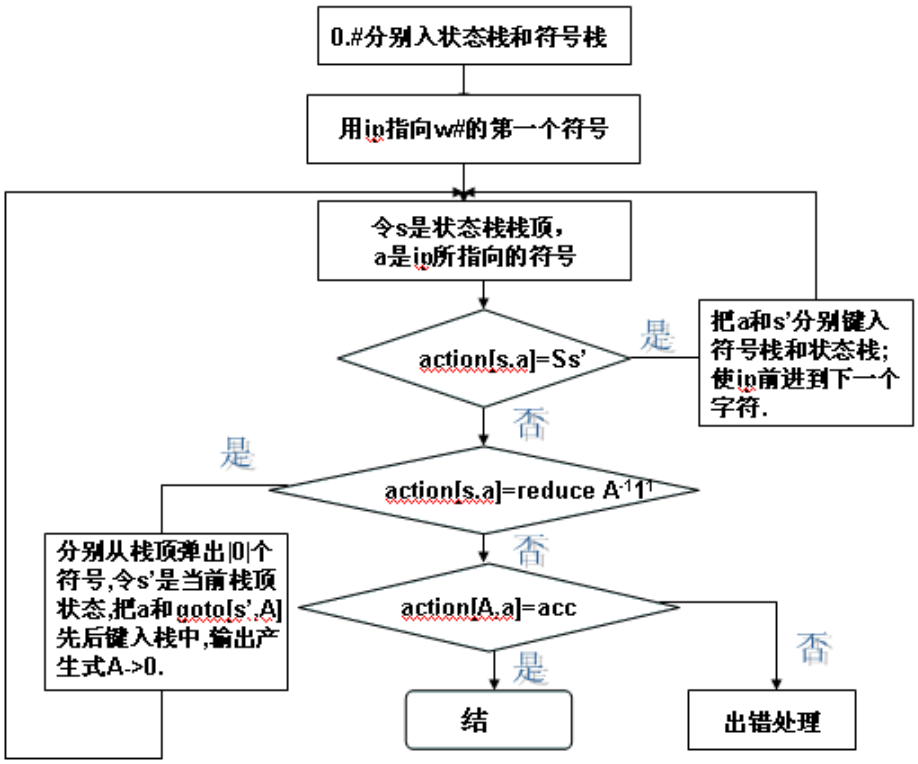
action[i，a]=rk：当在栈顶形成句柄时，则归约为相应的非终结符 A，即文法中有 A- B 的产生式，若 B 的长度为 R(即|B|=R)，则从状态栈和文法符号栈中自顶向下去掉 R 个符号，即栈指针 SP 减去 R，并把 A 移入文法符号栈内，j=GOTO[i,A]移进状态栈，其中 i 为修改指针后的栈顶状态。

(3)接受 acc:

当归约到文法符号栈中只剩文法的开始符号 S 时，并且输入符号串已结束即当前输入符是'#'，则为分析成功。

(4)报错:

当遇到状态栈顶为某一状态下出现不该遇到的文法符号时，则报错，说明输入端不是该文法能接受的符号串。



四、实验要求

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、如果遇到错误的表达式，应输出错误提示信息。

3、程序输入/输出实例：

输入一以#结束的符号串(包括+*（）i#)：在此位置输入符号串

输出过程如下：

步骤　　状态栈　　符号栈　　剩余输入串　　动 作

1　　0　　#　　i+i*i#　　移进

i+i*i 的 LR 分析过程				
步骤	**状态栈**	**符号栈**	**输入串**	**动作说明**
1	0	#	i+i*i#	ACTION[0,i]=S_5,状态 5 入栈
2	05	#i	+i*i#	r_6: F→i 归约,GOTO(0,F)=3 入栈
3	03	#F	+i*i#	r_4: T→F 归约,GOTO(0,T)=3 入栈
4	02	#T	+i*i#	r_2: E→T 归约,GOTO(0,E)=1 入栈

5	01	#E	+i*i#	ACTION[1,+]=S_6,状态 6 入栈
6	016	#E+	i*i#	ACTION[6,i]=S_5,状态 5 入栈
7	0165	#E+i	*i#	r_6: F→i 归约,GOTO(6,F)=3 入栈
8	0163	#E+F	*i#	r_4: T→F 归约,GOTO(6,T)=9 入栈
9	0169	#E+T	*i#	ACTION[9,*]=S_7,状态 7 入栈
10	01697	#E+T*	i#	ACTION[7,i]=S_5,状态 5 入栈
11	016975	#E+T*i	#	r_6:F→i 归约,GOTO(7,F)=10 入栈
12	0169710	#E+T*F	#	r_3: T→T*F 归约,GOTO(6,T)=9 入栈
13	0169	#E+T	#	r_1:E→E+T,GOTO(0,E)=1 入栈
14	01	#E	#	Acc：分析成功

4、输入符号串为非法符号串(或者为合法符号串)

算术表达式文法的 LR 分析表									
状态	**ACTION**						**GOTO**		
	i	**+**	*****	**(**	**)**	**#**	E	T	F
0	S_5			S_4			1	2	3
1		S_6				acc			
2		r_2	S_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	S_5			S_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	S_5			S_4				9	3
7	S_5			S_4					10
8		S_6			S_{11}				
9		r_1	S_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

五、实验步骤

1、根据流程图编写出各个模块的源程序代码上机调试。

2、编制好源程序后，设计若干用例对系统进行全面的上机测试，并通过所设计的 LR(1)语法分析程序；直至能够得到完全满意的结果。

3、书写实验报告 ；实验报告正文的内容：

- ◆ 描述 LR(1)语法分析程序的设计思想。
- ◆ 程序结构描述：函数调用格式、参数含义、返回值描述、函数功能；函数之间的调用关系图。
- ◆ 详细的算法描述（程序执行流程图）。
- ◆ 给出软件的测试方法和测试结果。
- ◆ 实验总结 （设计的特点、不足、收获与体会）。

实验四　逆波兰表达式的产生及计算

实验学时：3

实验类型：验证

实验要求：选修

一、实验目的

非后缀式用来表示的算术表达式转换为用逆波兰式来表示的算术表达式，并计算用逆波兰式来表示的算术表达式的值。

二、实验内容

将非后缀式用来表示的算术表达式转换为用逆波兰式来表示的算术表达式，并计算用逆波兰式来表示的算术表达式的值。

三、逆波兰表达式的产生及计算实验设计思想及算法

◆　逆波兰式定义

将运算对象写在前面，而把运算符号写在后面。用这种表示法表示的表达式也称做后缀式。逆波兰式的特点在于运算对象顺序不变，运算符号位置反映运算顺序。

◆　产生逆波兰式的前提

中缀算术表达式

◆　逆波兰式生成的设计思想及算法

(1)首先构造一个运算符栈，此运算符在栈内遵循越往栈顶优先级越高的原则。

(2)读入一个用中缀表示的简单算术表达式，为方便起见,设该简单算术表达式的右端多加上了优先级最低的特殊符号“#”。

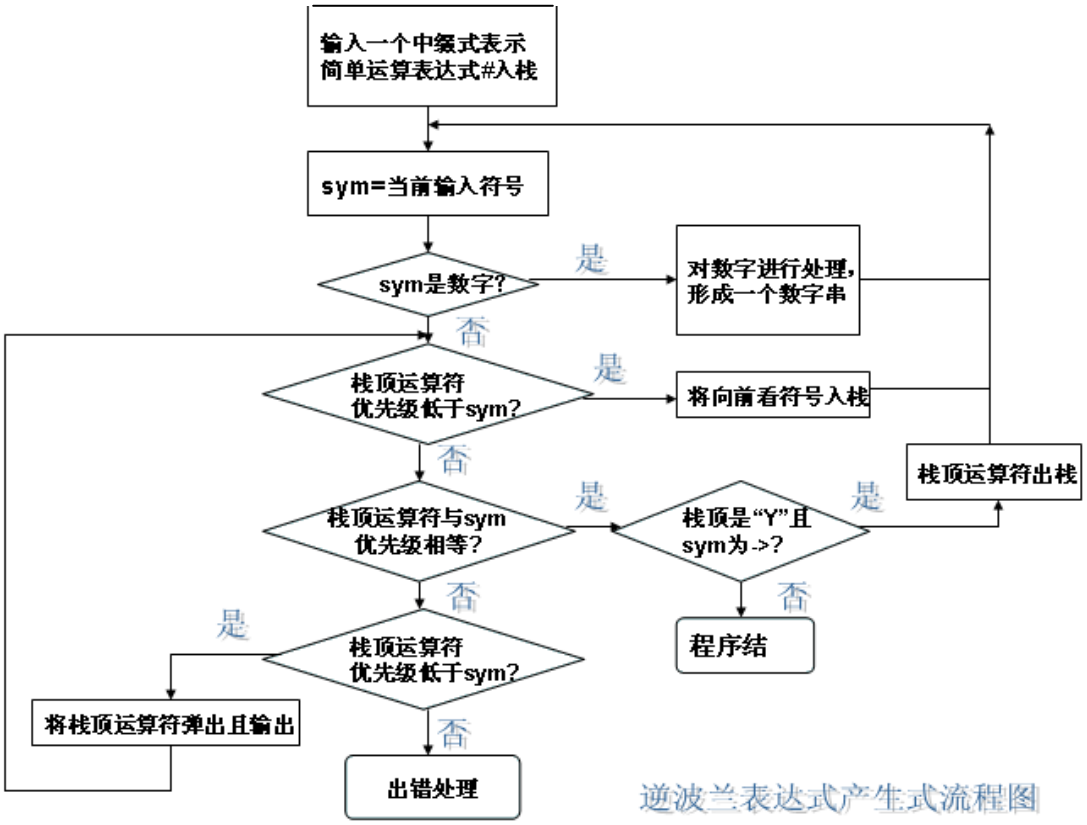
(3)从左至右扫描该算术表达式，从第一个字符开始判断，如果该字符是数字，则分析到该数字串的结束并将该数字串直接输出。

(4)如果不是数字，该字符则是运算符，此时需比较优先关系。

做法如下：将该字符与运算符栈顶的运算符的优先关系相比较。如果，该字符优先关系高于此运算符栈顶的运算符，则将该运算符入栈。倘若不是的话，则将此

运算符栈顶的运算符从栈中弹出，将该字符入栈。

(5)重复上述操作(1)-(2)直至扫描完整个简单算术表达式，确定所有字符都得到正确处理，我们便可以将中缀式表示的简单算术表达式转化为逆波兰表示的简单算术表达式。



逆波兰表达式产生式流程图

运用以上算法分析表达式(a+b*c)*d 的过程如下：

当前符号	输入区	符号栈	输出区
(	a+b*c)*d		
a	+b*c)*d	(	
+	*c)*d	(	a
b	c)*d	(+	a
*	)*d	(+	ab
c	*d	(+*	ab
)	*d	(+*	abc
)	*d	(+	abc*
)	d	(	abc*

*			abc*+
d		*	abc*+
		*	abc*+d
			abc*+d

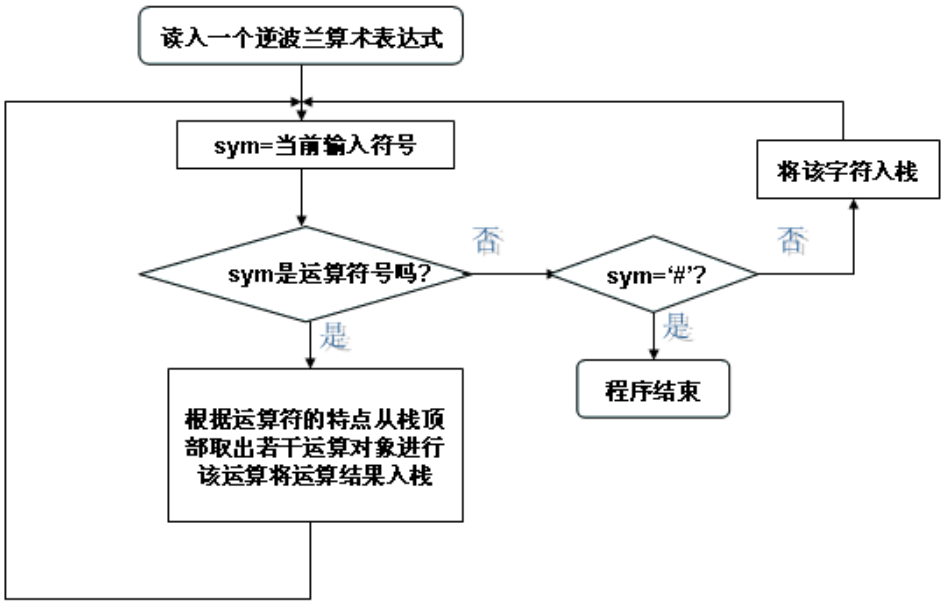
(1)构造一个栈，存放运算对象。

(2)读入一个用逆波兰式表示的简单算术表达式。

(3)自左至右扫描该简单算术表达式并判断该字符，如果该字符是运算对象，则将该字符入栈。若是运算符，如果此运算符是二目运算符，则将对栈顶部的两个运算对象进行该运算，将运算结果入栈，并且将执行该运算的两个运算对象从栈顶弹出。如果该字符是一目运算符，则对栈顶部的元素实施该运算，将该栈顶部的元素弹出，将运算结果入栈。

(4)重复上述操作直至扫描完整个简单算术表达式的逆波兰式，确定所有字符都得到正确处理，我们便可以求出该简单算术表达式的值。

◆ 逆波兰式计算的设计思想及算法



逆波兰式计算流程图

四、实验要求

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、如果遇到错误的表达式，应输出错误提示信息。

3、程序输入/输出实例：

◆ 输入以#结束的中缀表达式(包括+ - */（）数字#)。

例：(1)（a+b）　　(2)(a+b*c)　　(3) B+(-(A))*C

输出逆波兰表达式的格式如下：

(1) (a+b) ;→ab+)(

(2)(a+b*c)→abc*+)(

(3)B+(-A(A)) *C→BA)(-)(C*+

◆ 输入中缀表达式并计算结果：a* (b+c)+(-d)#;

输出逆波兰式:abc+*d@+

输入：a=3; b=1;c=2;d=5;

计算结果为:4

五、实验步骤

1、根据流程图编写出各个模块的源程序代码上机调试。

2、 编制好源程序后，设计若干用例对系统进行全面的上机测试，并通过所设计的逆波兰式的产生及计算程序；直至能够得到完全满意的结果。

3、书写实验报告 ；实验报告正文的内容：

◆ 描述逆波兰式的产生及计算程序的设计思想。

◆ 程序结构描述：函数调用格式、参数含义、返回值描述、函数功能；函数之间的调用关系图。

◆ 详细的算法描述（程序执行流程图）。

◆ 给出软件的测试方法和测试结果。

◆ 实验总结 （设计的特点、不足、收获与体会）。

实验五　应用 DGA 进行局部优化

实验学时：3

实验类型：设计

实验要求：选修

一、实验目的

使学生通过本次实验，能够对程序优化技术有一定的了解，掌握利于 DGA 进行局部优化的方法。

二、实验内容

对给定的四元式序列：

（1）：T1=A*B

（2）：T2=3/2

（3）：T3=T1-T2

（4）：X=T3

（5）：C=5

（6）：T4=A*B

（7）：C=2

（8）：T5=18+C

（9）：T6=T4*T6

（10）：Y=T6

要求：构造其相应的 DGA，并利于 DGA 进行了删除无用赋值、消除公共子表达式、合并已知量等到局部优化技术进行优化；再从所得到的 DGA 重建四元式序列。

三、LR（1）分析法实验设计思想及算法

由基本块构造 DGA 的算法描述如下：

```
for (i=0; i<QlistLength; i++)
```

```
{
取出第 i 四元式 Qi
if (NODE(B)==NULL)
   {
     建立一个以 B 为标记的叶结点，其编号为 NODE（B）
     switch(Qi)
     {
      case 0:   NODE(B)=n;   break;
      case 1:   if (NODE(B)是常数为标记的叶结点)
        {
          执行 P=op B;
          if (NODE(B)是处理 Qi 时新建立的结点)  删除 NODE（B）
          if (NODE(P)==NULL)  {建立 NODE（P）；NODE（P）=n}
        }
        else
        {
         if (DGA 中已有以 NODE（B）为惟一后继且标记为 op 的结点)
           令已有结点为 n;
         else
           建立新结点 n;
        }
      break;
     case 2: if (NODE(C)==NULL)
      {
         构造以 C 为标记的新结点；
         if (NODE(B)为常数结点  && NODE(C)为常数结点)
          {
           执行 P=B op C;
           if (NODE(B)或 NODE(C)是处理 Qi 时新建立的结点) 删除之；
```

if (NODE(P)==NULL) {建立 NODE（P）；NODE（P）=n}

break;

}

}

else

if (DGA 中已有以 NODE（B）和 NODE（C）分别为左、右后继，且标记为 op 的结点)

令已有结点为 n;

else

建立新结点 n;

break;

}

if (NODE(A)==NULL)

{

把 A 附加到结点 n;

NODE(A)=n;

}

else

{

从 NODE（A）的附加标识符集中将 A 删去；

把 A 附加到结点 n;

NODE(A)=n;

}

}

}

四、实验要求

1、对题目给定的的四元式序列能够实现 DGA 优化，使之不再出现题目所列之待优化情形；

2、能够输出优化后的四元式序列；

3、在部分同学完成实验题目的基础上，对实验程序进行改造，使之可以成为一个和般性的 DGA 优化程序。

五、实验步骤

1、根据流程图编写出各个模块的源程序代码上机调试。

2、编制好源程序后，用所给题目为例对系统进行全面的上机测试，并通过所设计的 DGA 优化分析实验程序；直至能够得到完全满意的结果。

3、书写实验报告 ；实验报告正文的内容：

- ◆ 描述 DGA 优化分析程序的设计思想。
- ◆ 程序结构描述：函数调用格式、参数含义、返回值描述、函数功能；函数之间的调用关系图。
- ◆ 详细的算法描述（程序执行流程图）。
- ◆ 给出软件的测试方法和测试结果。
- ◆ 实验总结 （设计的特点、不足、收获与体会）。

实验六　C 语言子集编译程序（可选）

实验学时：课外（40）

实验类型：大综合

实验要求：选修

一、实验目的

用 C 语言对一个 C 语言的子集编制一个一遍扫描的编译程序，以加深对编译原理的理解，掌握编译程序的实现方法和技术。

1．设计、编制并调试一个词法分析程序，加深对词法分析原理的理解。

2．编制一个递归下降分析程序，并对 C 语言的简单子集进行分析。

3．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成分变换中间代码的语义翻译方法。

二、实验要求、内容及学时

（一）待分析的 C 语言子集的词法：

1．关键字

main if else int char return void while

所有关键字都是小写。

2．专用符号

= + - * / < <= > >= == != ; : , { } [ ] ( )

3.其他标记 ID 和 NUM

通过以下正规式定义其他标记：

ID→letter(letter|digit)*　　NUM→digit(digit)*

letter→a|…|z|A|…|Z　　digit→0|…|9

4.空格由空白、制表符和换行符组成

空格一般用来分隔 ID、NUM、专用符号和关键字，词法分析阶段空格通常被忽略。各种单词符号对应的类别码：（采用一符一类别码,见下表）

单词符号	类别码	单词符号	类别码	单词符号	类别码
main	1	-	23	;	34
int	2	*	24	>	35
char	3	/	25	<	36
if	4	(	26	>=	37
else	5	)	27	<=	38
for	6	[	28	==	39
while	7	]	29	!=	40
ID	10	{	30	‘\0’	1000
NUM	20	}	31	ERROR	-1
=	21	,	32		
+	22	:	33		

（二）词法分析程序的功能：

输入：所给文法的源程序字符串。

输出：二元组（syn,token 或 sum）构成的序列。其中，

syn　为单词类别码。

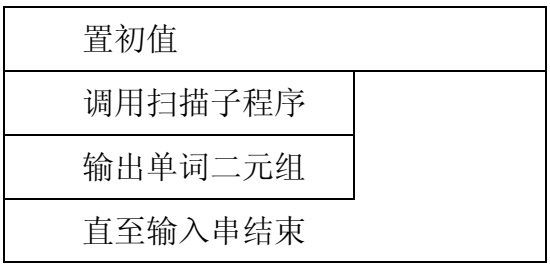
token 为存放的单词自身字符串。

sum　为整型常量。

（三）词法分析程序主要算法思想：

算法的基本任务是从字符串表示的源程序中识别出具有独立意义的单词符号，其基本思想是根据扫描到单词符号的第一个字符的种类，拼出相应的单词符号。

1．主程序示意结构图（如下）：

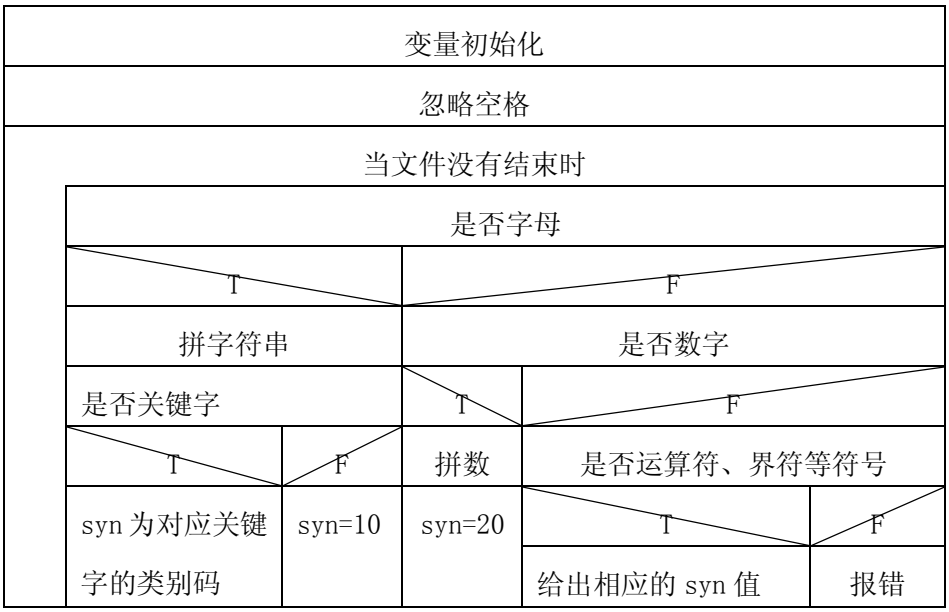


②程序中需要用到的主要变量：syn,token 和 sum。

2．扫描子程序（scaner）的算法思想

首先设置三个变量：token 用来存放构成单词符号的字符串；sum 用来存

放整型单词；syn 用来存放单词的类别码。扫描子程序主要部分 N—S 图如下：



（一）待分析的 C 语言子集的语法

用扩充的 BNF 表示如下：

1．<程序>→main()<语句块>

2. <语句块>→’{‘<语句串>’}’

3. <语句串>→<语句>{;<语句>};

4. <语句>→<赋值语句>|<条件语句>|<循环语句>

5. <赋值语句>→ID=<表达式>

6. <条件语句>→if(<条件表达式>)<语句块>

7. <循环语句>→while(<条件表达式>)<语句块>

8. <条件表达式>→<表达式><关系运算符><表达式>

9. <表达式>→<项>{+<项>|-<项>}

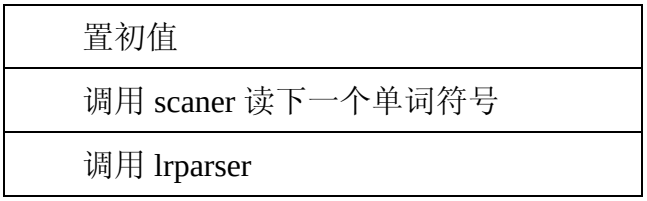
10.<项>→<项>{*<因子>|/<因子>}

11.<因子>→ID|NUM|(<表达式>)

12.<关系运算符>→<|<=|>|>=|==|!=

（二）语法分析程序的主要算法思想

1．主程序结构示意图如下：



结束

2．递归下降分析程序结构示意图如下：

<table>
<tr><td colspan="3">注：上接 lrparser</td></tr>
<tr><td colspan="3">是否单词串 main()</td></tr>
<tr><td colspan="2">T</td><td>F</td></tr>
<tr><td colspan="2">调用 scaner</td><td rowspan="5">出错处理</td></tr>
<tr><td colspan="2">调用语句块分析函数</td></tr>
<tr><td colspan="2">源程序是否结束</td></tr>
<tr><td>T</td><td>F</td></tr>
<tr><td>输出分析成功</td><td>出错处理</td></tr>
</table>

3．语句块分析结构示意图。

<table>
<tr><td colspan="3">是否{</td></tr>
<tr><td colspan="2">T</td><td>F</td></tr>
<tr><td colspan="2">调用 scaner</td><td rowspan="5">出错处理</td></tr>
<tr><td colspan="2">调用语句串分析函数（过程）</td></tr>
<tr><td colspan="2">是否}</td></tr>
<tr><td>T</td><td>F</td></tr>
<tr><td></td><td>出错处理</td></tr>
</table>

4．语句串分析结构示意图如下：

<table>
<tr><td colspan="2">调用 statement 函数</td></tr>
<tr><td colspan="2">当为 ；时</td></tr>
<tr><td></td><td>调用 scaner</td></tr>
<tr><td></td><td>调用 statement 函数</td></tr>
<tr><td colspan="2">出错处理</td></tr>
</table>

5．statement 函数 N—S 图如下：

<table>
<tr><td colspan="3">是否标识符</td></tr>
<tr><td>T</td><td colspan="2">F</td></tr>
<tr><td>调用 scaner</td><td colspan="2">是否 if</td></tr>
<tr><td>是否=</td><td>T</td><td>F</td></tr>
</table>

<table>
<tr><td>T</td><td>F</td><td>调用 scaner</td><td colspan="2">是否 while</td></tr>
<tr><td>调用 scaner</td><td rowspan="4">出错处理</td><td>调用 condition</td><td>T</td><td>F</td></tr>
<tr><td rowspan="3">调用 expression</td><td rowspan="3">调用语句块</td><td>调用 scaner</td><td rowspan="3">出错处理</td></tr>
<tr><td>调用 condition</td></tr>
<tr><td>调用语句块</td></tr>
</table>

6．expression 函数结构示意图如下：

<table>
<tr><td colspan="2">调用 term</td></tr>
<tr><td colspan="2">是否+、-</td></tr>
<tr><td>T</td><td>F</td></tr>
<tr><td>调用 scaner</td><td rowspan="2">出错处理</td></tr>
<tr><td>调用 term</td></tr>
</table>

7．term 函数结构示意图如下：

<table>
<tr><td colspan="2">调用 factor</td></tr>
<tr><td colspan="2">是否*、/</td></tr>
<tr><td>T</td><td>F</td></tr>
<tr><td>调用 scaner</td><td rowspan="2">出错处理</td></tr>
<tr><td>调用 factor</td></tr>
</table>

8.condition 函数结构示意图如下：

<table>
<tr><td colspan="2">调用 expression</td></tr>
<tr><td colspan="2">是否逻辑运算符</td></tr>
<tr><td>T</td><td>F</td></tr>
<tr><td>调用 scaner</td><td rowspan="2">出错处理</td></tr>
<tr><td>调用 expression</td></tr>
</table>

9．factor 函数结构示意图如下：

<table>
<tr><td colspan="4">是否标识符</td></tr>
<tr><td>T</td><td colspan="3">F</td></tr>
<tr><td rowspan="4">调用 scaner</td><td colspan="3">是否数字</td></tr>
<tr><td>T</td><td colspan="2">F</td></tr>
<tr><td rowspan="2">调用 scaner</td><td colspan="2">是否（</td></tr>
<tr><td>T</td><td>F</td></tr>
</table>

<table>
<tr><td rowspan="5"></td><td rowspan="5"></td><td colspan="2">调用 scaner</td><td rowspan="5">出错处理</td></tr>
<tr><td colspan="2">调用 expression</td></tr>
<tr><td colspan="2">是否）</td></tr>
<tr><td>T</td><td>F</td></tr>
<tr><td>调用 scaner</td><td>出错处理</td></tr>
</table>

语义分析部分：

（一）实验的输入和输出

输入是语法分析提供的正确的单词串，输出是四元式序列。

（二）算法思想

1．设置语义过程

①int gen(op,arg1,arg2,result)

该函数是将四元式(op,arg1,arg2,result)送到四元式表中。

②char *newtemp()

该函数回送一个新的临时变量名，临时变量名产生的顺序为：T1，T2，……

③int merg(p1,p2)

该函数将以 p1 和 p2 为头指针的两条链合并为一，合并后的链表首为返回值。

④int bp(p,t)

该函数的功能是把 p 所链接的每个四元式的第四区段都填为 t。

2．主程序示意图如下：

<table>
<tr><td>置初值</td></tr>
<tr><td>调用 scaner</td></tr>
<tr><td>……</td></tr>
<tr><td>调用 lrparser</td></tr>
<tr><td>打印四元式列表</td></tr>
<tr><td>结束</td></tr>
</table>

3．函数 lrparse 在原来语法分析的基础上插入相应的语义动作。

将输入串翻译成四元式序列。在实验中我们只对表达式、if 语句和 while 语句进行翻译，其具体翻译程序见实例。

算符优先分析法部分：（选作）

分析过程：先在算符栈置“$”，然后开始顺序扫描表达式。若读来的单词符

号是操作数，则直接进操作数栈，然后继续下一个单词符号。分析过程从头开始，并重复进行；若读来的单词符号是运算符θ_2，则将当前处于运算符栈顶的运算符θ_1的入栈优先函数 f 与θ_2的比较优先函数 g 进行比较。

1．若$f(\theta_1) \le g(\theta_2)$，则$\theta_2$进算符栈，并继续顺序往下扫描，分析过程重复进行。

2．若$f(\theta_1) > g(\theta_2)$，则产生对操作数栈顶的若干项进行$\theta_1$运算的中间代码，并从运算符栈顶移去$\theta_1$，且从操作数栈顶移去相应若干项，然后把执行$\theta_1$运算的结果压入操作数栈。接着以运算符栈新的项与$\theta_2$进行上述比较。

3．重复步骤 1，2，直到“$”和“$”配对为止。

三、实验环境

Windows 操作系统

C 或 Visual C++

四、实验参考（参考代码）

```
#ifndef _GLOBALS_H
#define _GLOBALS_H

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define _SYN_MAIN          1
#define _SYN_INT           2
#define _SYN_CHAR          3
#define _SYN_IF            4
#define _SYN_ELSE          5
#define _SYN_FOR           6
#define _SYN_WHILE         7

#define _SYN_ID            10
```

```
#define _SYN_NUM            20

#define _SYN_ASSIGN         21
#define _SYN_PLUS           22
#define _SYN_MINUS          23
#define _SYN_TIMES          24
#define _SYN_DIVIDE         25
#define _SYN_LPAREN         26
#define _SYN_RPAREN         27
#define _SYN_LEFTBRACKET1   28
#define _SYN_RIGHTBRACKET1  29
#define _SYN_LEFTBRACKET2   30
#define _SYN_RIGHTBRACKET2  31
#define _SYN_COMMA          32
#define _SYN_COLON          33
#define _SYN_SEMICOLON      34

#define _SYN_LG             35
#define _SYN_LT             36
#define _SYN_ME             37
#define _SYN_LE             38
#define _SYN_EQ             39
#define _SYN_NE             40
#define _SYN_END            1000

#define _SYN_ERROR          -1
#define MAXLENGTH           255

#ifndef _SEMANTEM_H
#define _SEMANTEM_H
```

```
    /*四元组的结构*/
    typedef struct QUAD{
        char op[MAXLENGTH];           /*操作符*/
        char argv1[MAXLENGTH];       /*第一个操作数*/
        char argv2[MAXLENGTH];       /*第二个操作数*/
        char result[MAXLENGTH];      /*运算结果*/
    }QUATERNION;

void lrparse(void);          /*语法语义分析主函数*/

#endif

union WORDCONTENT{
    char T1[MAXLENGTH];
    int  T2;
    char T3;
    };

typedef struct WORD{
    int syn;
    union WORDCONTENT value;
}WORD;

#ifndef _SCAN_H
#define _SCAN_H

#define _TAB_LEGNTH    4

#define _KEY_WORD_END  "waiting for you expanding"

void Scaner(void);
```

```
#endif

QUATERNION *pQuad;
int nSuffix,nNXQ,ntc,nfc;
extern WORD uWord;
extern int gnColumn,gnRow;
FILE *fw;
char *strFileName;  //用于存放输出的文件名，由于通常默认的输出是屏幕，也称标准输出，//可将 fw=stdout,stdout 是 C 语言标准对屏幕输出给的默认名
char *strSource;   //用于存储源程序，把源程序的代码放到一个数组中存储

char *Expression(void);
char *Term(void);
char *Factor(void);
void Statement_Block(int *nChain);

/*FILE *Source;*/
FILE *fw;
char *strSource;

void Do_Tag(char *strSource);
void Do_Digit(char *strSource);
void Do_EndOfTag(char *strSource);
void Do_EndOfDigit(char *strSource);
void Do_EndOfEqual(char *strSource);
void Do_EndOfPlus(char *strSource);
void Do_EndOfSubtraction(char *strSource);
void Do_EndOfMultiply(char *strSource);
void Do_EndOfDivide(char *strSource);
void Do_EndOfLParen(char *strSource);
```

```
void Do_EndOfRParen(char *strSource);
void Do_EndOfLeftBracket1(char *strSource);
void Do_EndOfRightBracket1(char *strSource);
void Do_EndOfLeftBracket2(char *strSource);
void Do_EndOfRightBracket2(char *strSource);
void Do_EndOfColon(char *strSource);
void Do_EndOfComma(char *strSource);
void Do_EndOfSemicolon(char *strSource);
void Do_EndOfMore(char *strSource);
void Do_EndOfLess(char *strSource);
void Do_EndOfEnd(char *strSource);
void PrintWord(WORD uWord);
void ApartWord(char *strSource);
void PrintError(int nColumn,int nRow,char chInput);
void Scaner(void);

int gnColumn,gnRow,gnLocate,gnLocateStart;

WORD uWord;

char *KEY_WORDS[20]={"main","int","char","if","else","for",
                     "while","void",_KEY_WORD_END};

int IsDigit(char chInput)
//判断扫描的字符是否数字
{
    if(chInput<='9'&&chInput>='0') return 1;
    else return 0;
}

int IsChar(char chInput)
```

```
//判断扫描的字符是否字母
{
    if((chInput<='z'&&chInput>='a')||(chInput<='Z'&&chInput>='A'))
        return 1;
    else return 0;
}

void Do_Start(char *strSource)
//开始识别一个单词
{
    gnLocateStart=gnLocate;
    switch(strSource[gnLocate]){
    case '+': Do_EndOfPlus(strSource);          break;
    case '-': Do_EndOfSubtraction(strSource);   break;
    case '*': Do_EndOfMultiply(strSource);      break;
    case '/': Do_EndOfDivide(strSource);        break;
    case '(': Do_EndOfLParen(strSource);        break;
    case ')': Do_EndOfRParen(strSource);        break;
    case '[': Do_EndOfLeftBracket1(strSource);  break;
    case ']': Do_EndOfRightBracket1(strSource); break;
    case '{': Do_EndOfLeftBracket2(strSource);  break;
    case '}': Do_EndOfRightBracket2(strSource); break;
    case ':': Do_EndOfColon(strSource);         break;
    case ',': Do_EndOfComma(strSource);         break;
    case ';': Do_EndOfSemicolon(strSource);     break;
    case '>': Do_EndOfMore(strSource);          break;
    case '<': Do_EndOfLess(strSource);          break;
    case '=': Do_EndOfEqual(strSource);         break;
    case '\0': Do_EndOfEnd(strSource);          break;
    default:
        if(IsChar(strSource[gnLocate]))
```

```
{
    Do_Tag(strSource);
}
else
    if(IsDigit(strSource[gnLocate]))
    {
        uWord.value.T2=strSource[gnLocate]-'0';
        Do_Digit(strSource);
    }
    else
    {
        if(strSource[gnLocate]!=' '
            &&strSource[gnLocate]!='\t'
            &&strSource[gnLocate]!='\n'
            &&strSource[gnLocate]!='\r')
        {
            PrintError(gnColumn,gnRow,strSource[gnLocate]);
        }
        if(strSource[gnLocate]=='\n'
            ||strSource[gnLocate]=='\r')
        {
            gnColumn++;
            gnRow=1;
        }
        else
            if(strSource[gnLocate]=='\t')
            {
                gnColumn+=_TAB_LEGNTH;
            }
            else
                gnRow++;
```

```
                    gnLocate++;
                    Do_Start(strSource);
            }
            break;
    }
    return;
}

void Do_Tag(char *strSource)
//识别标识符的中间状态
{
    gnLocate++;
    gnRow++;
    if(IsChar(strSource[gnLocate])||IsDigit(strSource[gnLocate]))
    {
        Do_Tag(strSource);
    }
    else
        Do_EndOfTag(strSource);

    return;
}

void Do_Digit(char *strSource)
//识别整数的中间状态
{
    gnLocate++;
    gnRow++;
    if(IsDigit(strSource[gnLocate]))
    {
```

```
        uWord.value.T2=uWord.value.T2*10+strSource[gnLocate]-'0';
        Do_Digit(strSource);
    }
    else Do_EndOfDigit(strSource);

    return;
}


void Do_EndOfTag(char *strSource)
//识别标识符的最后状态
{
    int nLoop;

    uWord.syn=_SYN_ID;

    strncpy(uWord.value.T1,strSource+gnLocateStart,gnLocate-gnLocateStart);
    uWord.value.T1[gnLocate-gnLocateStart]='\0';

    nLoop=0;
    while(strcmp(KEY_WORDS[nLoop],_KEY_WORD_END))
    {
        if(!strcmp(KEY_WORDS[nLoop],uWord.value.T1))
        {
            uWord.syn=nLoop+1;
        }
        nLoop++;
        }
    return;
}
```

```
void Do_EndOfDigit(char *strSource)
//识别数的最后状态
{
    uWord.syn=_SYN_NUM;
    return;
}

void Do_EndOfEqual(char *strSource)
//识别==的最后状态，它的开始状态在 Do_Start 中已处理，
//运算符没有中间状态，因为最多由两个符号组成，
//而数和标识符可以由多个终结符组成。
//以下类似的函数命名，其功能类似，不再加注。
//以下如：+，-，*，/，(，0，[，]，{，},:，逗号，……
{
    if(strSource[gnLocate+1]!='=')
    {
        uWord.syn=_SYN_ASSIGN;
        uWord.value.T3=strSource[gnLocate];
    }
    else
    {
        gnLocate++;
        gnRow++;
        uWord.syn=_SYN_EQ;
        strcpy(uWord.value.T1,"==");
    }
    gnLocate++;
    gnRow++;
    return;
}
```

```
void Do_EndOfPlus(char *strSource)
{
    uWord.syn=_SYN_PLUS;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;
    return;
}
void Do_EndOfSubtraction(char *strSource)
{
    uWord.syn=_SYN_MINUS;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;

    return;
}

void Do_EndOfMultiply(char *strSource)
{
    uWord.syn=_SYN_TIMES;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;
    return;
}

void Do_EndOfDivide(char *strSource)
```

```
{
    uWord.syn=_SYN_DIVIDE;
    uWord.value.T3=strSource[gnLocate];
    gnLocate++;
    gnRow++;
    return;
}

void Do_EndOfLParen(char *strSource)
{
    uWord.syn=_SYN_LPAREN;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;

    return;
}

void Do_EndOfRParen(char *strSource)
{
    uWord.syn=_SYN_RPAREN;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;
    return;
}

void Do_EndOfLeftBracket1(char *strSource)
{
```

```
    uWord.syn=_SYN_LEFTBRACKET1;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;
    return;
}

void Do_EndOfRightBracket1(char *strSource)
{
    uWord.syn=_SYN_RIGHTBRACKET1;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;
    return;
}

void Do_EndOfLeftBracket2(char *strSource)
{
    uWord.syn=_SYN_LEFTBRACKET2;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;

    return;
}

void Do_EndOfRightBracket2(char *strSource)
{
```

```
    uWord.syn=_SYN_RIGHTBRACKET2;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;
    return;
}

void Do_EndOfColon(char *strSource)
{
    uWord.syn=_SYN_COLON;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;

    return;
}

void Do_EndOfComma(char *strSource)
{
    uWord.syn=_SYN_COMMA;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;

    return;
}

void Do_EndOfSemicolon(char *strSource)
```

```
{
    uWord.syn=_SYN_SEMICOLON;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;

    return;
}

void Do_EndOfMore(char *strSource)
{
    if(strSource[gnLocate+1]!='=')
    {
        uWord.syn=_SYN_LG;
        uWord.value.T3=strSource[gnLocate];
    }
    else
    {
        gnLocate++;
        gnRow++;
        uWord.syn=_SYN_ME;
        strcpy(uWord.value.T1,">=");
    }

    gnLocate++;
    gnRow++;
    return;
}

void Do_EndOfLess(char *strSource)
```

```
{
    if(strSource[gnLocate+1]!='=')
    {
        uWord.syn=_SYN_LT;
        uWord.value.T3=strSource[gnLocate];
    }
    else
    {
        gnLocate++;
        gnRow++;
        uWord.syn=_SYN_LE;
        strcpy(uWord.value.T1,"<=");
    }

    gnLocate++;
    gnRow++;

    return;
}

void Do_EndOfEnd(char *strSource)
{
    uWord.syn=_SYN_END;
    uWord.value.T3=strSource[gnLocate];

    gnLocate++;
    gnRow++;

    return;
}
void PrintWord(WORD uWord)
```

```
//输出单词二元组（种别码，单词）
{
    if(uWord.syn<=_SYN_ID
        ||uWord.syn==_SYN_ME
        ||uWord.syn==_SYN_LE
        ||uWord.syn==_SYN_EQ)
    {
        fprintf(fw,"\n(%d,\t%s)",uWord.syn,uWord.value.T1);
    }
    else if(uWord.syn==_SYN_NUM)
        {
            fprintf(fw,"\n(%d,\t%d)",uWord.syn,uWord.value.T2);
        }
        else
        {
            fprintf(fw,"\n(%d,\t%c)",uWord.syn,uWord.value.T3);
        }

    return;
}

void ApartWord(char *strSource)
//识别出当前合法文件中所有单词，所以是一个调用 scaner()函数的循环。
{
    gnColumn=gnRow=1;
    gnLocate=gnLocateStart=0;

    while(strSource[gnLocate])
    {
        Scaner();
    }
```

```
    return;
}

void PrintError(int nColumn,int nRow,char chInput)
{
    fprintf(fw,"\n 无法识别的单词-->Col:%d\tRow:%d\tChar:%c",
        nColumn,nRow,chInput);

    return;
}

void Scaner(void)
//词法分析框架函数
{
    Do_Start(strSource);
    PrintWord(uWord);

    return;
}

//以下注释中为词法分析部分的测试主函数
/*void main()
{
    strSource="main(){int i=10;while(i) i=i-1;}";

    fw=stdout;
    ApartWord(strSource);
}
*/
```

```
/ / 只要做词法分析，则下面代码不要.

void LocateError(int nColumn,int nRow)
//输出错误所在的行、列提示信息
{
    fprintf(fw,"\nCol:%d\tRow:%d--->",nColumn+1,nRow);
}

void error(char *strError)
//输出具体错误信息（传给 fw 指向的文件，即输出文件），
//本程序错误信息种类设置较简单，可以自己补充。
{
    LocateError(gnColumn,gnRow);
    fprintf(fw,"%s",strError);
    return;
}

void Match(int syn,char *strError)
//判断当前识别出的单词是否与语法中约定的相符，
//若相符，调用 scaner()识别下一个单词备用；否则报语法分析错
{
    if(syn==uWord.syn) Scaner();
    else error(strError);

    return;
}

void gen(char *op,char *argv1,char *argv2,char *result)
//随着进程生成四元组，将其保存在 pQuad 结构体数组中，为输出作准备
{
    sprintf(pQuad[nNXQ].op,op);
```

```
    sprintf(pQuad[nNXQ].argv1,argv1);
    sprintf(pQuad[nNXQ].argv2,argv2);
    sprintf(pQuad[nNXQ].result,result);
    nNXQ++;
    return;
}

void PrintQuaternion(void)
//输出四元组
{
    int nLoop;
    for(nLoop=1;nLoop<nNXQ;nLoop++)
    {
        fprintf(fw,"\n%d:%s,\t%s,\t%s,\t%s",
            nLoop,pQuad[nLoop].op,pQuad[nLoop].argv1,
            pQuad[nLoop].argv2,pQuad[nLoop].result);
    }
}

char *Newtemp(void)
//自动产生中间结果标识“T1”，“T2”等字符串，以备输出四元组用
{
    char *strTempID=(char *)malloc(MAXLENGTH);
    sprintf(strTempID,"T%d",++nSuffix);

    return strTempID;
}

int merg(int p1,int p2)
//合并 p1 和 p2
{
```

```
    int p,nResult;
    if(p2==0) nResult=p1;
    else
    {
        nResult=p=p2;
        while(atoi(pQuad[p].result))
        {
            p=atoi(pQuad[p].result);
            sprintf(pQuad[p].result,"%s",p1);
        }
    }
    return nResult;
}

void bp(int p,int t)
//将中间结果（如：t1,t2 等）填到四元组中
{
    int w,q=p;
    while(q)
    {
        w=atoi(pQuad[q].result);
        sprintf(pQuad[q].result,"%d",t);
        q=w;
    }

    return;
}

char *Expression(void)
//处理表达式
{
```

```
    char opp[MAXLENGTH],*eplace,
    eplace1[MAXLENGTH],eplace2[MAXLENGTH];
    eplace=(char *)malloc(MAXLENGTH);
    strcpy(eplace1,Term());
    strcpy(eplace,eplace1);
    while(uWord.syn==_SYN_PLUS||uWord.syn==_SYN_MINUS)
    {
        sprintf(opp,"%c",uWord.value.T3);
        Scaner();
        strcpy(eplace2,Term());
        strcpy(eplace,Newtemp());
        gen(opp,eplace1,eplace2,eplace);
        strcpy(eplace1,eplace);
    }
    return eplace;
}

char *Term(void)
//处理相乘或相除（可以连乘连除,所以下面的分析是一个循环）的项
{
    char opp[2],*eplace,*eplace1,*eplace2;
    eplace=(char *)malloc(MAXLENGTH);
    eplace=eplace1=Factor();
    strcpy(eplace,eplace1);
    while(uWord.syn==_SYN_TIMES||uWord.syn==_SYN_DIVIDE)
    {
        sprintf(opp,"%c",uWord.value.T3);
        Scaner();
        eplace2=Factor();
        eplace=Newtemp();
        gen(opp,eplace1,eplace2,eplace);
```

```
        eplace1=eplace;
    }

    return eplace;
}

char *Factor(void)
//处理表达式中的因子，如 a1*12 中的 a1 和 12,即可以为数或标识符.
{
    char *eplace=(char *)malloc(MAXLENGTH);
    if(uWord.syn==_SYN_ID||uWord.syn==_SYN_NUM)
    {
        if(uWord.syn==_SYN_ID)
        {
            sprintf(eplace,"%s",uWord.value.T1);
        }
        else sprintf(eplace,"%d",uWord.value.T2);
        Scaner();
    }
    else
    {
        Match(_SYN_LPAREN,"(");
        eplace=Expression();
        Match(_SYN_RPAREN,")");
    }

    return eplace;
}

void Condition(int *etc,int *efc)
//处理条件表达式
```

```
{
    char opp[3],*eplace1,*eplace2;
    char strTemp[4];

    eplace1=Expression();
    if(uWord.syn<=_SYN_NE||uWord.syn>=_SYN_LG)
    {
        switch(uWord.syn)
        {
            case _SYN_LT:
            case _SYN_LG:
                sprintf(opp,"%c",uWord.value.T3);
                break;
            default:
                sprintf(opp,"%s",uWord.value.T1);
                break;
        }
        Scaner();
        eplace2=Expression();
        *etc=nNXQ;
        *efc=nNXQ+1;
        sprintf(strTemp,"j%s",opp);
        gen(strTemp,eplace1,eplace2,"0");
        gen("j","","","0");//条件表达式对应的四元组第一项加标志 j
    }
    else error("关系运算符");
}

void Statement(int *nChain)
//分析赋值、if、while 语句
{
```

```
char strTemp[MAXLENGTH],eplace[MAXLENGTH];
int nChainTemp,nWQUAD;
switch(uWord.syn)
{
    //处理赋值语句
    case _SYN_ID:
        strcpy(strTemp,uWord.value.T1);
        Scaner();
        Match(_SYN_ASSIGN,"=");
        strcpy(eplace,Expression());
        Match(_SYN_SEMICOLON,";");
        gen("=",eplace,"",strTemp);
        *nChain=0;
        break;

    //处理 if 语句
    case _SYN_IF:
        Match(_SYN_IF,"if");
        Match(_SYN_LPAREN,"(");
        Condition(&ntc,&nfc);
        bp(ntc,nNXQ);
        Match(_SYN_RPAREN,")");
        Statement_Block(&nChainTemp);
        *nChain=merg(nChainTemp,nfc);
        break;

    //处理 while 语句
    case _SYN_WHILE:
        Match(_SYN_WHILE,"while");
        nWQUAD=nNXQ;
        Match(_SYN_LPAREN,"(");
```

```
            Condition(&ntc,&nfc);
            bp(ntc,nNXQ);
            Match(_SYN_RPAREN,")");
            Statement_Block(&nChainTemp);
            bp(nChainTemp,nWQUAD);
            sprintf(strTemp,"%d",nWQUAD);
            gen("j","","",strTemp);
            *nChain=nfc;
            break;
    }
    return;
}

void Statement_Sequence(int *nChain)
//语句序列分析函数
{
    Statement(nChain);
    while(uWord.syn==_SYN_ID
        ||uWord.syn==_SYN_IF
        ||uWord.syn==_SYN_WHILE)
    {
        bp(*nChain,nNXQ);
        Statement(nChain);
    }
        bp(*nChain,nNXQ);

        return;
}
void Statement_Block(int *nChain)
//分析语句块函数，语名块是{……}语句
{
```

```
    Match(_SYN_LEFTBRACKET2,"{");
    Statement_Sequence(nChain);
    //上行分析语句块中语句序列，即花括号中的部分
    Match(_SYN_RIGHTBRACKET2,"}");
}

void Parse(void)
//语法语义分析函数
{
    int nChain;

    Scaner();
    Match(_SYN_MAIN,"main");
    Match(_SYN_LPAREN,"(");
    Match(_SYN_RPAREN,")");
    Statement_Block(&nChain);
    if(uWord.syn!=_SYN_END) fprintf(fw,"源程序非正常结束");
    PrintQuaternion();
}

void lrparse(void)
//语法语义分析函数，主要是先处理准备和收尾工作，
//中间调用 parse()进行语法语义工作
{
    pQuad=(QUATERNION *)malloc(strlen(strSource)*sizeof(QUATERNION));
    nSuffix=0;
    nfc=ntc=nNXQ=1;
    fw=fopen(strFileName,"w");
    Parse();
    fclose(fw);
}
```

```
void main()
{
    char str[]="main(){i=2*3+4;if(i>10) {j=3;}while (j>0) {k=1;}}";
    //注:上行为待编译源程序

    char filename[200];

    //以下为全局变量赋初值
    gnColumn=gnRow=1;
    gnLocate=gnLocateStart=0;
    nNXQ=0;
    strSource=str;

    //打开输出即单词、四元组存放的文件
    strcpy(filename,"d:\\c\\compiler\\test1.txt");
    strFileName=filename;
    //注意只要将文件名传给 strFileName，文件实际在
    //函数 lrparse 中打开。
    lrparse();
}
```