《编译原理》

实验指导书

华东理工大学

信息科学与工程学院计算机系

一、PL/0 语言的扩展

1、课程实验要完成下列要求

（1）给 PL/0 语言增加像 C 语言那样的形式为/* …… */的注释。

（2）给 PL/0 语言增加带 else 子句的条件语句和 exit 语句。

（3）给 PL/0 语言增加输入输出语句。

（4）给 PL/0 语言增加布尔类型。

（5）给 PL/0 语言增加实数类型

（6）分离解释器和编译器为两个独立的程序。

2、扩展后的 PL/0 语言的语法

Program → Block .

Block →
[ConstDecl][TypeDecl][VarDecl][FuncDecl] **begin** Stmt {; Stmt } **end**

ConstDecl → **const** ConstDef ; {ConstDef ;}

ConstDef → ident = number ;

TypeDecl → **type** TypeDef {TypeDef }

TypeDef → ident = TypeExp ;

TypeExp → **integer** | **real** | **Boolean** | **array** '['number .. number']' **of** TypeExp

VarDecl → **var** VarDec {VarDec }

VarDec → ident {, ident} : Type ;

Type → **integer** | **real** | **Boolean** | ident

FuncDecl → FuncDec { FuncDec }

FuncDec → **procedure** ident [(ForParal)]; Block ; |
function ident [(ForParal)] : Type ; Block ;

ForParal → ident : Type {; ident : Type }

Stmt → IdentRef := Exp | **if** Exp **then** Stmt | **if** Exp **then** Stmt **else** Stmt |
begin Stmt {; Stmt } **end** | **while** Exp **do** Stmt | **exit** | ε |
call ident [(ActParal)] | **write** (Exp {, Exp }) |
read (IdentRef {, IdentRef })

IdentRef → ident ['['Exp']' { '['Exp']' }]

ActParal → Exp {, Exp }

Exp → SimpExp RelOp SimpExp | SimpExp

RelOp → = | <> | < | > | <= | >=

SimpExp → [+ | –] Term {+ Term | – Term | **or** Term}

Term → Factor {* Factor | / Factor | **div** Factor | **mod** Factor | **and** Factor}

Factor → IdentRef | number | (Exp) | **not** Factor | ident [(ActParal)] |
odd (SimpExp) | **true** | **false**

有关该扩展的说明如下：

（1）描述语言的部分符号的解释

符号→、|、[、]、{和}是描述语言的符号，其中方括号[…]中的部分可以出现 0 次或 1 次，花括号{…}中的部分可以出现任意次数，包括 0 次。

被描述语言若使用上述这些符号，则需要用单引号，例如在数组类型的定义和下标变量的引用中。

（2）词法部分

- 形式为/* …… */的注释也是词法单元之间的分隔符。
- 标识符 ident 是字母开头的字母数字串。

• number 是无符号数，若是实数，则采用小数点前后都有非空数字串这一种形式。

• 连续的两个点（出现在数组界的声明中）看成一个单词，不要把每个点看成一个单词。就像虽有<，但<=看成一个单词一样。

• 与上面两种情况有关的一个特殊现象：当整数后面紧跟两个点时，例如“10..”，看成整数 10 和两个点，不要把两个点分开，把“10.”看成实数缺少小数点后的非空数字串。

• 新增保留字：**type**、**array**、**of**、**integer**、**real**、**Boolean**、**function**、**else**、**write**、**read**、**exit**、**or**、**and**、**not**、**div**、**mod**、**true**、**false**。

• **div**：整数除，**mod**：取模，/ ：实数除

（3）静态检查和动态检查

• 在类型定义 ident = TypeExp 中，TypeExp 若不是数组类型，则报告错误。即只允许给数组类型命名。

• 若 exit 语句没有处于任何 while 语句中，则是一个错误。

• 读写语句的变量引用和表达式只能是整型或实型。

• 过程和函数是有区别的。call 语句只能调用过程，表达式中只能出现函数调用。

• 表达式中整型和实型混合运算的类型检查。

•整型数据可以赋给实型变量（反之不行），实现时先将整型数据转换为实型数据，然后再赋值。

• 本语言有布尔类型，有布尔常量 false 和 true。C 语言虽有逻辑运算，但没有布尔类型。本语言不存在把 0 看成假，把非 0 看成真。

• 数组类型是按名字等价而不是结构等价。

• 数组类型不能作为过程和函数的参数类型，也不能作为函数的结果类型。

• 除上述几点之外的静态检查都是大家应该知道的，不在此叙述。

• 运行时检查下标表达式是否越界，越界则报告错误，停止程序的运行。

（4）语句的语义

只对特殊部分加以说明。

• exit 语句作为 while 语句的非正常出口语句。若处于多层 while 语句中，则它只作为包含该 exit 的最内层 while 语句的非正常出口。

• 读语句接受从键盘输入的数据，数据之间用空格分隔。

•读语句具有忽略当前输入行剩余字符，下一个读语句接受的数据另起一行的功能。

• 写语句输出的数据显示在屏幕上，数据之间用空格分隔。

•写语句具有结束当前输出行，下次输出另起一行的功能。

• 增加了写语句后，原来 sto 指令的输出功能取消，以免输出数据过多，不宜发现所关心的数据。

• 程序员定义的函数和过程的参数都是值调用参数，函数的结果类型不能是数组类型。

• 布尔类型的表达式按短路方式计算。

• 函数中没有 return 语句，在函数中，可以把函数名当作局部变量，通过对函数名赋值来把函数值返回。注意，无参函数名出现在表达式中时，是该无参函数的调用。

（5）分离解释器和编译器为两个独立的程序

• 编译器和解释器的接口是二进制的中间代码文件。

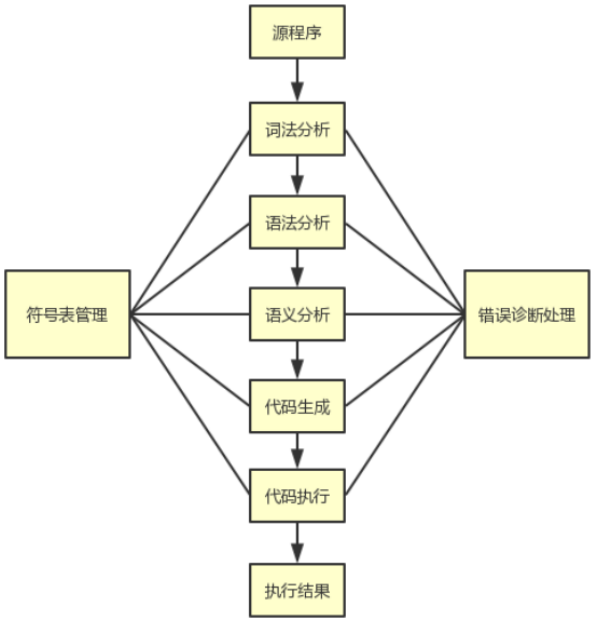
• 原编译器中 listcode 方式的中间代码列表输出功能略去，以便编译过程中的屏幕输出简洁（只有源程序和报错信息）；增加把完整的中间代码列表输出到文件的功能，以便实验评测时使用，该功能对于你测试和调试程序来说也是有用的。

二、实验报告内容

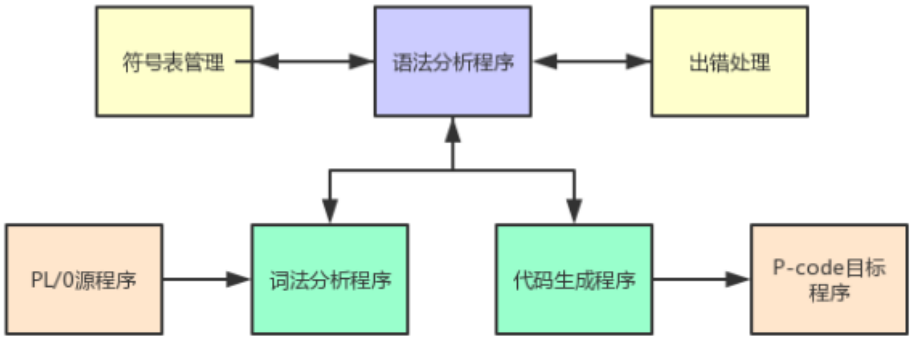
1.)编译系统介绍

一个经典的编译程序一般包括 7 个部分：词法分析，语法分析，语义分析及代码生成，代码优化（可省

略），代码执行，符号表管理，出错管理。这 7 个部分之间的关联关系如下图所示：



整个编译过程如下图所示：



扩展后的 PL/0 语言的语法：

扩展前	扩展后
Program	Block
Block	[ConstDecl][TypeDecl][VarDecl][FuncDecl] begin Stmt {; Stmt } end
ConstDecl	const ConstDef ; {ConstDef ;}
ConstDef	ident = number ;
TypeDecl	type TypeDef {TypeDef }
TypeDef	ident = TypeExp ;
TypeExp	integer | real | Boolean | array '['number .. number']' of TypeExp
VarDecl	var VarDec {VarDec }
VarDec	ident {, ident} : Type ;
Type	integer | real | Boolean | ident
FuncDecl	FuncDec { FuncDec }
FuncDec	procedure ident [(ForParal)]; Block ; | function ident [(ForParal)] : Type ; Block ;
ForParal	ident : Type {; ident : Type }
Stmt	IdentRef := Exp | if Exp then Stmt | if Exp then Stmt else Stmt | begin Stmt {; Stmt } end | while Exp do Stmt | exit | |

	call ident [(ActParal)] | write (Exp {, Exp }) | read (IdentRef {, IdentRef })
IdentRef	ident ['['Exp']' { '['Exp']' }]
ActParal	Exp {, Exp }
Exp	SimpExp RelOp SimpExp | SimpExp
RelOp	= | <> | < | > | <= | >=
SimpExp	[+ |] Term {+ Term | Term | or Term}
Term	Factor { Factor | / Factor | div Factor | mod Factor | and Factor}
Factor	IdentRef | number | (Exp) | not Factor | ident [(ActParal)] | odd (SimpExp) | true | false

2）词法分析

词法分析是编译的第一阶段。词法分析器的主要任务是读入源程序的输入字符，将它们组成词素，生成并输出一个词法单元序列，这个词法单元序列被输出到语法分析器进行语法分析。另外，由于词法分析器在编译器中负责读取源程序，因此除了识别词素之外，它还会完成一些其他任务，比如过滤掉源程序中的注释和空白，将编译器生成的错误消息与源程序的位置关联起来等。

词法分析子程序分析：词法分析子程序名为 getsym，功能是从源程序中读出一个单词符号（token），把它的信息放入全局变量 sym、id 和 num 中，语法分析器需要单词时，直接从这三个变量中获得。getsym 过程通过反复调用 getch 子过程从源程序过获取字符，并把它们拼成单词。getch 过程中使用了行缓冲区技术以提高程序运行效率。

词法分析器的分析过程：调用 getsym 时，它通过 getch 过程从源程序中获得一个字符。如果这个字符是字母，则继续获取字符或数字，最终可以拼成一个单词，查保留字表，如果查到为保留字，则把 sym 变量赋成相应的保留字类型值；如果没有查到，则这个单词应是一个用户自定义的标识符（可能是变量名、常量名或是过程的名字），把 sym 置为 ident，把这个单词存入 id 变量。查保留字表时使用了二分法查找以提高效率。如果 getch 获得的字符是数字，则继续用 getch 获取数字，并把它们拼成一个整数，然后把 sym 置为 number，并把拼成的数值放入 num 变量。如果识别出其它合法的符号（比如：赋值号、大于号、小于等于号等），则把 sym 则成相应的类型。如果遇到不合法的字符，把 sym 置成 nul。

词法分析程序 GETSYM 将完成下列任务：

1. 滤空格　2. 识别保留字　3. 识别标识符　4. 拼数　5. 拼复合词　6. 输出源程序

3）语法分析

语法分析是编译过程的一个逻辑阶段。语法分析的任务是在词法分析的基础上将单词序列组合成各类语法短语，如“程序”，“语句”，“表达式”等等。语法分析程序判断源程序在结构上是否正确.源程序的结构由上下文无关文法描述。

语法分析子程序分析：语法分析子程序采用了自顶向下的递归子程序法，语法分析同时也根据程序的语意生成相应的代码，并提供了出错处理的机制。语法分析主要由分程序分析过程（block）、常量定义分析过程（constdeclaration）、变量定义分析过程（vardeclaration）、语句分析过程（statement）、表达式处理过程（expression）、项处理过程（term）、因子处理过程（factor）和条件处理过程（condition）构成。这些过程在结构上构成一个嵌套的层次结构。除此之外，还有出错报告过程（error）、代码生成过程（gen）、测试单词合法性及出错恢复过程（test）、登录名字表过程（enter）、查询名字表函数（position）以及列出类 PCODE 代码过程（listcode）作过语法分析的辅助过程。

由 PL/0 的语法图可知：一个完整的 PL/0 程序是由分程序和句号构成的。因此，本编译程序在运行的时候，通过主程序中调用分程序处理过程 block 来分析分程序部分（分程序分析过程中还可能会递归调用 block 过程），然后，判断最后读入的符号是否为句号。如果是句号且分程序分析中未出错，则是一个合法的 PL/0 程序，可以运行生成的代码，否则就说明源 PL/0 程序是不合法的，输出出错提示即可。

4） 语义分析

语义分析是编译过程的一个逻辑阶段， 语义分析的任务是对结构上正确的源程序进行上下文有关性质的审查，进行类型审查。语义分析是审查源程序有无语义错误，为代码生成阶段收集类型信息。

PL/0 的语义分析主要进行以下检查：

1） 是否存在标识符先引用未声明的情况；

2） 是否存在已声明的标识符的错误引用；

3） 是否存在一般标识符的多重声明。

5） 中间代码生成

中间代码生成是产生中间代码的过程。所谓“中间代码”是一种结构简单、含义明确的记号系统，这种记号系统复杂性介于源程序语言和机器语言之间，容易将它翻译成目标代码。另外，还可以在中间代码一级进行与机器无关的优化。

6） 语法错误处理

PL/0 编译程序对语法错误的处理采用两种办法：

①对于一些易于校正的错误，如丢了逗号、分号等，指出出错的位置，加以校正，继续进行分析。

②对于难于校正的错误，给出错误的位置与性质，跳过后面一些单词，直到下一个可以进行正常语法分析的语法单位。

错误类型：

(1) 应为=而不是:=

(2) =后应为数

(3) 标识符后应为=

(4) const，var，procedure type 后应为标识符

(5) 遗漏逗号或分号

(6) 过程声明后的记号不正确

(7) 应为语句

(8) 分程序内的语句部分后的记号不正确

(9) 应为句号

(10) 语句之间漏分号

(11) 标识符未声明

(12) 不可向常量或过程或布尔型赋值

(13) 应为赋值运算符:=

(14) call 后应为标识符

(15) 不可调用常量或变量或函数

(16) 应为 then

(17) 应为分号或 end

(18) 应为 do

(19) 语句后的记号不正确

(20) 应为关系运算符

(21) 表达式内不可有过程标识符

(22) 遗漏右括号

(23) 因子后不可为此记号

(24) 表 b 达式不能以此记号开始

(25) 这个数太大

(26) 这个常数或地址偏移太大

(27) 程序嵌套层次太多

(28) type 的类型不是合法的 interger\real 等类型

(29) 定义 array 格式不合法
(30) read\write 后面遗漏左右括号
(31) read write 后面的括号里应该是声明过的变量名
(32) 缺少变量名
(33) 缺少类型声明 v
(34) 遗漏左右括号
(35) 形参和实参个数不匹配
(36) 数组溢出
(37) 应为数组类型
(38) 没有声明函数返回值的类型
(39) 函数返回值的类型不正确
(40) 数组引用不正确，比如没有给出各个维数的下标值或者下标值不是整型
(41) exit 没有在 while 循环里面
(42) 变量重复声明
(43) 函数参数类型不匹配
(44) 运算符两边的类型不正确 运算符包括加减乘除运算符、关系运算符以及 odd 等
(45) 不同类型之间变量不能赋值
(46) if，while 里面的条件不是布尔型
(47) 语句结束后不能再有其他东西
(48) block 声明完了之后必须为 begin
(49) 没有声明变量的类型
(50) 对类型名赋值
(51) 小数点后没有数字

三、课程实验成果的提交和测试环境

1、提交编译器和解释器的源程序、用于评测时介绍自己的实现方法和技术细节；提交编译器和解释器的目标程序，用于评测时展示自己成果的正确性。

2、测试环境：Windows XP 平台，不提供任何 C 或 C++的编译工具。