

课程实践项目

一、PL/0 语言的扩展

1、课程实践要完成下列要求

- (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 语言增加带参数的过程。
- (5) 给 PL/0 语言增加布尔类型。
- (6) 给 PL/0 语言增加数组类型。
- (7) 给 PL/0 语言增加函数类型。
- (8) 给 PL/0 语言增加实数类型。
- (9) 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请参考教材 5.3.4 节。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、课程实践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。