编 译 原 理

一、要求

基本拓展要求

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

1

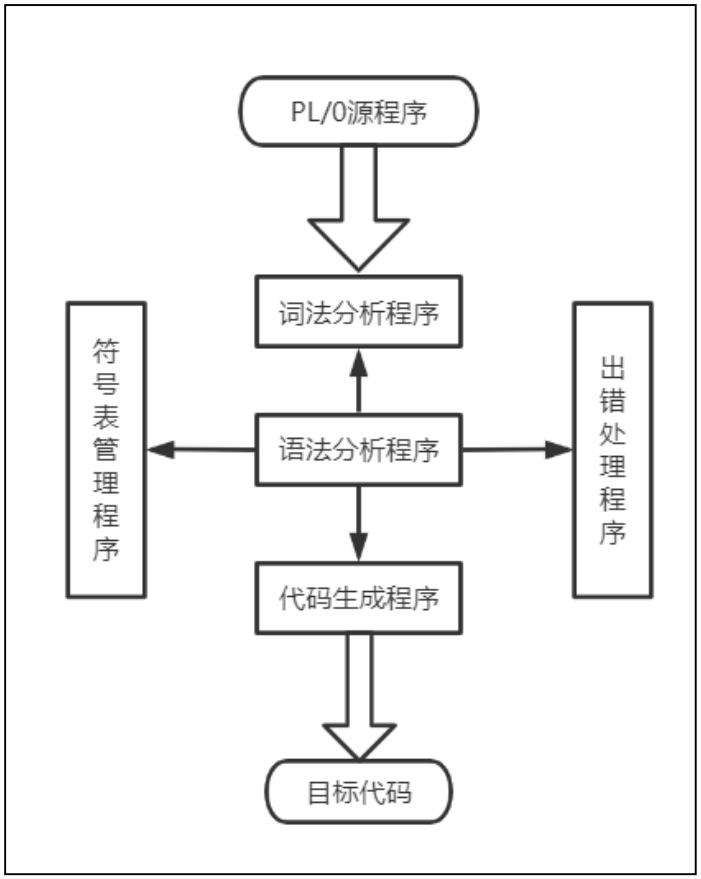
二、拓展PL/0文法

拓展PL/0语言文法的EBNF表示

PL/0语法单位	EBNF描述
<程序>	::= <分程序>.
<分程序>	::= [<常量说明部分>][变量说明部分>]{<过程说明部分>}begin<语句>end
<常量说明部分>	::= const<常量定义>{,<常量定义>};
<常量定义>	::= <标识符>=<无符号整数>| <标识符>=<实数>
<变量说明部分>	::= var<标识符>{,<标识符>}:<变量类型>;{<标识符>{,<标识符>}:<变量类型>}
<变量类型>	::=integer|real|Boolean
<过程说明部分>	::= <过程首部><分程序>
<过程首部>	::= procedure<标识符>{<标识符>{,<标识符>}:<变量类型>;{<标识符>{,<标识符>}:<变量类型>}};
<语句>	::= <赋值语句>|<条件语句>|<当型循环语句>|<过程调用语句>|<读语句>|<写语句>|<复合语句>|<退出循环语句>|<空>
<关系表达式>	::= <表达式><关系运算符><表达式>|<表达式>
<关系运算符>	::= =|<>|<|<=|>|>=
<表达式>	::= [+|-]<项>{+<项>|-<项>|or <项>}
<项>	::= <因子>{*<因子>|/<因子>|div <因子>|mod <因子>|and <因子>}
<因子>	::= <标识符>|<数>|(<表达式>)|not <因子>|odd<表达式>|true|false
<退出循环语句>	::=exit()
<赋值语句>	::= <标识符>:=<关系表达式>
<条件语句>	::= if<关系表达式>then<语句>[else<语句>]
<当型循环语句>	::= while<关系表达式>do<语句>
<过程调用语句>	::= call <标识符>{(<关系表达式>{,<关系表达式>})}
<复合语句>	::= begin<语句>{;<语句>}end
<读语句>	::= read(<标识符>{,<标识符>})
<写语句>	::= write(<标识符>{,<标识符>})

三、系统结构

PL/0编译层次结构



四、词法分析

运算符与保留字

操作符	含义	全局变量
+	加法	ADD_SYM
-	减法	SUB_SYM
*	乘法	MULT_SYM
/	实数除法	DIV_SYM
<	小于	LESS_SYM
>	大于	GRE_SYM
=	等于	EQUAL_SYM
<=	小于等于	ELESS_SYM
>=	大于等于	EGRE_SYM
<>	不等于	NEQUAL_SYM
:=	赋值	ASSI_SYM
:	定义变量类型	COLON_SYM

操作符	含义	全局变量
or	或操作	OR_SYM
and	与操作	AND_SYM
not	非操作	NOT_SYM
div	整数除法操作	INT_DIV_SYM
mod	取余操作	MOD_SYM

保留字	含义	全局变量
const	定义常量	CONST_SYM
var	定义变量	VAR_SYM
procedure	定义子程序	PROCEDURE_SYM
begin	程序开始	BEGIN_SYM
end	程序结束	END_SYM
odd	对2取余	ODD_SYM
if	条件语句	IF_SYM
then	条件传递	THEN_SYM
else	条件转折	ELSE_SYM
call	调用子程序	CALL_SYM
while	循环判断条件	WHILE_SYM
do	循环递进	DO_SYM
read	输入语句	READ_SYM
write	输出语句	WRITE_SYM
integer	整树	INTEGER_SYM
real	实数	REAL_SYM
Boolean	布尔型变量	BOOLEAN_SYM
exit	退出while循环	EXIT_SYM
true	布尔值 真	TRUE_SYM
false	布尔值 假	FALSE_SYM

五、语法分析

系统流程设计图

语法分析是编译程序的核心部分。
作用是识别由词法分析给出的单词符号穿是否是给定文法的正确句子。
在本次实验中采用自顶向下的分析方法。

语法分析函数表

函数名	功能	函数原型
error_exc	语法分析错误处理	void error_exc(int error_grammar_type)
BLOCK	语法分析程序入口	void BLOCK(treeNode *tn)
subBlock	分程序	subblock(treeNode *tn)
const_BLOCK	常量说明分析程序	void const_BLOCK(treeNode *tn)
const_def	常量定义	void const_def(treeNode *tn)
var_BLOCK	变量说明	void var_BLOCK(treeNode *tn)
procedure_BLOCK	过程说明	void procedure_BLOCK(treeNode *tn)
procedure_head	过程首部	void procedure(treeNode *tn)
sen_BLOCK	语句	void sen_BLOCK(treeNode *tn)
assi_sen	赋值语句	void assi_sen(treeNode *tn)
condition_sen	条件判断语句	void condition_sen(treeNode *tn)
do_while_sen	当型循环语句	void do_while_sen(treeNode *tn)
exit_sen	退出循环语句	void exit_sen(treeNode *tn)
proc_call_sen	过程调用语句	void proc_call_sen(treeNode *tn)
read_sen	读语句	void read_sen(treeNode *tn)
write_sen	写语句	void write_sen(treeNode *tn)
complex_sen	复合语句	void complex_sen(treeNode *tn)
rel_expression	关系表达式	void real_expression(treeNode *tn)
expression	表达式	void expression(treeNode *tn)
item	项	void item(treeNode *tn)
factor	因子	void factor(treeNode *tn)

目的代码结构

P-code 语言格式

F	L	A

指令格式

F ： 操作码

L ： 层次差 （标识符引用层减去定义层）

A ： 不同的指令含义不同

运行栈的存储分配

①SL:静态链，指向定义该过程的直接外过程(或主程序)运行时最新数据段的基地址。

②DL:动态链，指向调用该过程前正在运行过程的数据段基地址。

③RA:返回地址，记录调用该过程时目标程序的断点，即调用过程指令的下一条指令的地址。

P-code指令含义

指令分类	指令格式	指令功能
过程调用相关指令	int 0 A	在栈顶开辟A个存储单元
	opr 0 0	结束被调用过程，返回调用点并退栈
	cal L A	调用地址为A的过程，调用过程与被调用过程的层差为L
	pas 0 A	将过程的A个实参向上移动3个单位
存取指令	lit 0 A	立即数A存入t所指单元，t加1
	lod L A	将层差为L、偏移量为A的存储单元的值取到栈顶，t加1
	sto L A	将栈顶的值存入层差为L、偏移量为A的单元，t减1
转移指令	jmp 0 A	无条件转移至地址A
	jpc 0 A	若栈顶为0，则转移至地址A，t减1
一元运算和比较指令	opr 0 1	求栈顶元素的相反数，结果值留在栈顶
	opr 0 6	栈顶内容若为奇数则变为1，若为偶数则变为0
二元运算指令	opr 0 2	次栈顶与栈顶的值相加，结果存入次栈顶，t减1
	opr 0 3	次栈顶的值减去栈顶的值，结果存放次栈顶，t减1
	opr 0 4	次栈顶的值乘以栈顶的值，结果存放次栈顶，t减1
	opr 0 5	次栈顶的值除以以栈顶的值，取整结果存放次栈顶，t减1
	opr 0 7	次栈顶的值对栈顶的值取模，结果存放次栈顶，t减1
	opr 0 20	次栈顶的值除以以栈顶的值，结果存放次栈顶，t减1
二元比较指令	opr 0 8	次栈顶与栈顶内容若相等，则将1存于次栈顶，t减1
	opr 0 9	次栈顶与栈顶内容若不相等，则将1存于次栈顶，t减1
	opr 0 10	次栈顶内容若小于栈顶，则将1存于次栈顶，t减1
	opr 0 11	次栈顶内容若大小于等于栈顶，则将1存于次栈顶，t减1
	opr 0 12	次栈顶内容若大于栈顶，则将1存于次栈顶，t减1
	opr 0 13	次栈顶内容若小于等于栈顶，则将1存于次栈顶，t减1
输入输出指令	opr 0 14	栈顶的值输出至控制台屏幕，t减1
	opr 0 15	控制台屏幕输出一个换行
	opr 0 16	从控制台读入一行输入，置入栈顶，t加1
逻辑指令	opr 0 17	次栈顶和栈顶内容做且运算，结果存放次栈顶，t减1
	opr 0 18	次栈顶和栈顶内容做或运算，结果存放次栈顶，t减1
	opr 0 19	次栈顶和栈顶内容做非运算，结果存放次栈顶，t减1

错误处理

错误类型采用枚举值设置，如此设计的原因是采用枚举值更具有可读性和可扩展性。错误类型共有34种。

```
enum ERROR_GRAMMAR {
    const_state_end_wrong,//常数语句结尾错误
    number_wrong,//数错误
    const_assign_wrong,//常数定义错误
    identifier_wrong, //不是标识符
    var_state_end_wrong,//变量说明的结束错误
    var_no_type_wrong,//变量无类型
    var_type_wrong,//变量类型错误
    procedure_state_end_wrong,//过程语句结束错误
    procedure_name_redefined,//过程名与其他名字重复
    procedure_head_wrong,//过程首部结束错
    not_procedure_name,//不是过程名
    var_assign_wrong,//不是变量赋值
    Boolean_assign_wrong,//Bollean变量赋值错误
    integer_assign_wrong,//integer变量赋值错误
    expression_in_factor_wrong,//因子中包含的表达式的错误结束
    factor_wrong,//增加因子错误
    if_then_wrong,//if语句错误
    condition_wrong, //不是正确的比较运算符
    while_do_wrong,//while语句错误
    exit_wrong,//exit语句错误
    exit_IsNotIn_while,//exit语句不在循环中
    read_end_wrong,//read结束错误
    read_wrong,//read错误
    read_var_wrong,//被读入的不是变量
    read_Boolean_wrong,//Boolean类型变量无法被读入
    write_end_wrong,//write语句结束错误
    write_wrong,//write语句错误
    complex_end_wrong,//不是正确符合语句结束，或者中间语句缺少分号
    wrong_program_end,//错误的过程结束
    var_undefined,//未定义
    var_wrong,//不是变量
    var_redefined,//变量重定义
    const_redefined,//常量重定义,
    call_end_wrong,//过程调用结束错误
    real_para_wrong,//形参与实参个数不等
};
```