实验二 PL/0语言语法分析器

Fan Xinyan

2019.5.9

一、实验目的

用bison工具生成一个PL/0语言的语法分析程序，对PL/0源程序进行语法分析。

二、实验要求

（一）基本要求

对pl/0源程序，要求输出按归约顺序用到的语法规则，和语法单位的层次结构关系（语法树）。

（二）扩展要求

自定义for循环语句。并能够对for语句进行语法分析，输出语法层次结构。

三、实验环境

语法分析生成工具：bison、flex
编程语言：C
操作系统：macOS

四、实验内容

代码共分为三部分："header.h""lex.l""parse.y"。其中"header.h"为树节点数据结构和宏定义的声明，宏定义为符号类型与对应的整数，便于后续参数传递。"lex.l"为词法分析程序，识别出关键词、标识符、数字、界符、运算符等传入语法分析程序。"parse.y"为语法分析程序，根据PL/0的EBNF范式定义了语法规则，并构造了语法单位的层次结构（语法树）。

（一）终结符说明

程序中定义格式：

代码1 - 终结符定义

```
%token <str> IF        "if"
```

1、关键词

PL/0程序中涉及到的关键词共有15个。

keyword	procedure	call	const	var	odd
	while	do	if	then	begin
	end	read	write	for	to

2、标识符

PL/0程序中标识符的正规式为：

identifier	[a-zA-Z_][a-zA-Z0-9_]*

3、数字

PL/0程序中数字的正规式为：

number	[1-9][0-9]*|[0-9]

4、界符

PL/0程序中的界符有：

delimiter	;	,	:	(	)	.

5、运算符

<table>
<tr><td rowspan="2">operator</td><td>+</td><td>-</td><td>*</td><td>/</td><td>=</td><td>#</td></tr>
<tr><td>></td><td><</td><td>>=</td><td><=</td><td>:=</td><td></td></tr>
</table>

（二）文法设计

根据PL/0语言的EBNF范式设计文法。

<table>
<tr><th>EBNF范式</th><th>文法</th></tr>
<tr><td><程序>::=<分程序>.</td><td>program -> subprogram.</td></tr>
<tr><td><分程序> ::= [<常量说明部分>][<变量说明部分>][<过程说明部分>]<语句></td><td>subprogram -> constInstruction variableInstruction procedureInstruction sentence</td></tr>
<tr><td><常量说明部分> ::= CONST<常量定义>{,<常量定义>};</td><td>constInstruction -> CONST constDeclaration; | NULL</td></tr>
<tr><td><常量定义> ::= <标识符>=<无符号整数></td><td>constDeclaration -> ID = NUMBER | ID = NUMBER , constDeclaration</td></tr>
<tr><td><变量说明部分> ::= VAR<标识符>{,<标识符>}</td><td>variableInstruction -> VAR ID variableInstructionSuf ; | NULL
variableInstructionSuf -> , ID variableInstructionSuf | NULL</td></tr>
<tr><td><过程说明部分> ::= <过程首部><分程序>{;<过程说明部分>};</td><td>procedureInstruction -> procedureHeader subprogram ; procedureInstruction | NULL</td></tr>
<tr><td><过程首部> ::= PROCEDURE<标识符>;</td><td>procedureHeader -> PROCEDURE ID ;</td></tr>
<tr><td><语句> ::= <赋值语句>|<复合语句>|<条件语句>|<当型循环语句>|<for循环语句>|<过程调用语句>|<读语句>|<写语句>|<空></td><td>sentence -> assignment | compound | conditional | whileLoop | forLoop | procedureCall | readSent | writeSent | NULL</td></tr>
<tr><td><赋值语句> ::= <标识符>:=<表达式></td><td>assignment -> ID := expr</td></tr>
<tr><td><复合语句> ::= BEGIN<语句>{;<语句>}END</td><td>multsentence -> sentence ; mulsentence | sentence
compound -> BEGIN multsentence END</td></tr>
<tr><td><条件语句> ::= IF<条件>THEN<语句></td><td>conditional -> IF condition THEN sentence</td></tr>
</table>

<table>
<tr><th>EBNF范式</th><th>文法</th></tr>
<tr><td><条件> ::= <表达式><关系运算符><表达式>|ODD<表达式></td><td>condition -> expr relationop expr | ODD expr</td></tr>
<tr><td><表达式> ::= [+|-]<项>{<加法运算符><项>}</td><td>expr -> item itemsuf | plusop item itemsuf
itemsuf -> plusop item itemsuf</td></tr>
<tr><td><项> ::= <因子>{<乘法运算符><因子>}</td><td>item -> factor factorsuf
factorsuf -> mulop factor factorsuf</td></tr>
<tr><td><因子> ::= <标识符>|<无符号整数>|'('<表达式>')'</td><td>factor -> ID | NUMBER | (expr)</td></tr>
<tr><td><加法运算符> ::= +|-</td><td>plusop -> + | -</td></tr>
<tr><td><乘法运算符> ::= *|/</td><td>mulop -> * | /</td></tr>
<tr><td><关系运算符> ::= =|#|<|<=|>|>=</td><td>relationop -> =|#|<|<=|>|>=</td></tr>
<tr><td><当型循环语句> ::= WHILE<条件>DO<语句></td><td>whileLoop -> WHILE condition DO sentence</td></tr>
<tr><td><for循环语句> ::= FOR<赋值语句>TO<条件>DO<语句></td><td>forLoop -> FOR assignment TO condition DO sentence</td></tr>
<tr><td><过程调用语句> ::= CALL<标识符></td><td>procedureCall -> CALL ID</td></tr>
<tr><td><读语句> ::= READ'('<标识符>{,<标识符>}')'</td><td>readSent -> READ (readpara)
readpara -> ID</td></tr>
<tr><td><写语句> ::= WRITE'('<表达式>{,<表达式>}')'</td><td>writeSent -> WRITE (writepara)
writepara -> expr , writepara</td></tr>
</table>

（三）数据结构

1、语法树每个节点

对树中每个节点（终结符或非终结符）储存节点的类型（属于终结符或非终结符中的具体类型）、节点的值（针对数值型）、节点的名称（在源程序中的名称）、节点的孩子数量、节点的孩子们（结构体指针、动态分配内存）。

代码2 - 语法树节点

```
typedef struct node{
    int type;
    int value;
    int childcnt;
    char* id;
    struct node** child;
}Node;
```

2、符号栈

维护一个符号栈结构主要用于语法自下而上分析时的“移进”和“归约”过程。移进：将一个终结符推进栈。归约：当栈顶形成某个产生式的候选式时，把这些符号从栈中弹出，把产生式的左部符号压入栈。

定义stop为指向栈顶的指针。

代码3 - 符号栈定义

```
Node* stack[STACK_MAXN+1];
int stop;
```

针对栈的核心操作：

(1) 压栈

代码4 - PUSH

```
void push(Node* s){
    stack[stop] = s;
    stop++;
}
```

(2) 出栈

代码5 - POP

```
void pop(){
    stop --;
    freenode(stack[stop]);
}
```

(3) 释放一个节点和它的孩子们

代码6 - FREENODE

```
void freenode(Node* p){
    if(p -> id != NULL){
        free(p -> id);
    }
    if(p -> childcnt != 0){
        int i;
        for(i = 0;i < p -> childcnt;++ i){
            freenode(p -> child[i]);
        }
    }
    free(p);
}
```

（4）返回栈顶元素

代码7 - TOP

```
Node* top(){
    return stack[stop-1];
}
```

（四）核心代码

1、移进

在文法中，当编译程序遇到终结符时，需要为该终结符申请新的Node并压入栈。具体函数如下，第一个参数表示该节点的类型，第二个参数表示该节点在源程序中的字符表示，第三个参数表示这个符号的数值大小。

代码8 - SHIFT

```
void shift(int type,char* id,int number){
    Node* a;
    a = (Node*)malloc(sizeof(Node));
    a->value = number;
    a->type = type;
    if(id == NULL) {
        a->id = NULL;
    } else {
            a->id = (char*)malloc(ID_MAX_SIZE);
            strcpy(a->id, id);
    }
    a->child = NULL;
    a -> childcnt = 0;
    push(a);
}
```

在代码语法规则定义中，具体表示如下，以文法“procedureHeader -> PROCEDURE ID ;”为例：

代码9 - 移进

```
procedureHeader : "procedure" "id" ";"     {
                                            shift(PROCEDURE_KEY,$1,0);
                                            shift(IDENTI,$2,0);
                                            shift(SEMI_OP,”;”,0);
                                            };
```

2、归约

在文法中，当编译程序遇到符合某一文法的候选式时，就需要将栈顶中满足文法右部类型的符号弹出栈，并将左部符号设为一个新的节点，即为这些被弹出符号的父节点，并压入栈中。具体函数如下。其中第一个参数表示左部符号的类型，第二个参数表示孩子的个数（右部符号的个数），第三个节点表示孩子的类型（右部符号的类型）。

代码10 - REDUCE

```
void reduce(int type,int child_type_cnt,int child_type[TYPE_MAXN]){
    Node* father;
    father = (Node*)malloc(sizeof(Node));
    father -> value = 0;
    father -> id = NULL;
    father -> type = type;

    int cnt = 0;
    int tmptop = stop;
    fflush(stdout);

    while(tmptop > 0 && inarray(stack[tmptop-1]->type, child_type_cnt, child_type) == 1){
        tmptop --;
        cnt ++;
    }
    father -> child = (Node**)malloc(cnt * sizeof(Node*));
    int idx = 0;

    while(!empty() && inarray(top() -> type, child_type_cnt, child_type) == 1 ){
        father -> child[idx] = (Node*)malloc(sizeof(Node));
        node_copy(father -> child[idx], stack[stop-1]);
        pop();
        idx ++;
    }
    father -> childcnt = idx;
    push(father);
}
```

代码11 - INARRAY判断右部符号是否在栈中

```
int inarray(int tp, int tpcnt, int tparray[TYPE_MAXN]){
    int i;
    for( i = 0;i < tpcnt;++ i){
        if(tp == tparray[i]){
            return 1;
        }
    }
    return 0;
}
```

代码12 - NODE_COPY

```
void node_copy(Node* copy,Node* copied){
    copy -> value = copied -> value;
    copy -> type = copied -> type;
    if(copied -> id == NULL) {
        copy-> id = NULL;
    }
    else {
        copy -> id = (char*)malloc(ID_MAX_SIZE);
        strcpy(copy -> id,copied -> id);
    }
    copy -> childcnt = copied -> childcnt;
    if(copy->childcnt != 0){
        copy -> child = (Node**)malloc(copy->childcnt * sizeof(Node*));
        int i;
        for( i = 0;i < copy-> childcnt;++ i){
                copy -> child[i] = (Node*)malloc(sizeof(Node));
                node_copy(copy->child[i],copied->child[i]);
        }
    }
}
```

在代码语法规则定义中，具体表示如下，以文法“procedureHeader -> PROCEDURE ID ;”为例：

代码13 - 归约

```
procedureHeader : "procedure" "id" ";"  {
                                        shift(PROCEDURE_KEY,$1,0);
                                        shift(IDENTI,$2,0);
                                        shift(SEMI_OP,";",0);
                                        child_tp_cnt = 3;
                                        child_tp[0] = PROCEDURE_KEY;
                                        child_tp[1] = IDENTI;
                                        child_tp[2] = SEMI_OP;
                                        reduce(PROCEDURE_HEADER,child_tp_cnt,child_tp);
                                        };
```

3、打印语法结构

在编译程序将PL/0源程序分析结束后，栈中留有一个节点，即整棵语法树的父节点，从该节点开始，用深度优先搜索的方式打印出整棵树。

代码14 - 打印语法树

```
void print_node(Node* p,int k){
    if(p == NULL) return;
    int type = p -> type;

    if(type == PROGRAM){for(int i = 0;i < k;++ i)fprintf(fout,"  ");fprintf(fout,"< program >\n"); }
    else if(type == SUBPROGRAM) {for(int i = 0;i < k;++ i)fprintf(fout,"  ");fprintf(fout,"< subprogram
>\n"); }
    …………//打印每个语法单元

    int childno = p -> childcnt;
    for(int i = childno - 1;i >= 0; -- i){
            print_node(p -> child[i],k+1);
    }
}
```

五、实验结果

将PL/0源程序存在test.txt文件中，完整PL/0源程序请看附录A。在MacOS系统下程序执行命令如下:

执行命令

```
bison -d parse.y
flex lex.l
gcc lex.yy.c parse.tab.c
./a.out
test.txt
```

（一）按归约顺序输出语法规则

在语法规则定义时，每个规则归约结束后输出该文法规则即可。具体输出结果请看附录B。

（二）语法树

源PL/0程序与输出结果详见附录C。

以for语句结构为例:

测试用到的FOR语句

```
for i := 1 to i <= 10 do
    begin
        b := a;
        i := i + 1;
    end;
```

打印出的语法树结构如下:

FOR语句的语法树

```
< for_loop_sentence >
    < FOR: for >
    < assignment_sentence >
        < ID: i >
        < ASSIGN_OP: := >
        < expression >
            < item >
                < factor >
                    < NUMBER: 1 >
    < TO: to >
    < condition >
        < expression >
            < item >
                < factor >
                    < ID: i >
        < relation_operator >
            < LEQ_OP: <= >
        < expression >
            < item >
                < factor >
                    < NUMBER: 10 >
    < DO: do >
    < sentence >
        < compound_sentence >
            < BEGIN: begin >
            < sentence >
                < assignment_sentence >
                    < ID: b >
                    < ASSIGN_OP: := >
                    < expression >
                        < item >
                            < factor >
                                < ID: a >
            < sentence >
                < assignment_sentence >
                    < ID: i >
                    < ASSIGN_OP: := >
                    < expression >
                        < item_suf >
                            < item >
                                < factor >
                                    < ID: i >
                            < plus_operator >
                                < PLUS_OP: + >
                            < item >
                                < factor >
                                    < NUMBER: 1 >
            < END: end >
```

从结果中可以看出，程序能够正确按照语法规则输出相应的语法层次结构。

附录A

源程序

源PL/0程序

```
procedure rec;
begin
    for i := 1 to i <= 10 do
    begin
        b := a;
        sum := 0;
        i := i + 1;
    end;
end;
procedure P;
        const x = 0, y = 1;
        procedure Q;
                var a;
                procedure  R;
                begin
                        write(x);
                end;
        begin
                read(a);
                read(n);
                if n > 0 then
                begin
                        if n < 10 then
                begin
                                b := a;
                                sum := 0;
                                while n>=0 do
                                begin
                                        sum := sum + b;
                                        b := b + 10 * a;
                                        n := n - 1;
                                end;
                        end;
                end;
                write(sum);
        end;
begin
        read(a);
        read(b);
        read(c);
        if a # b then max := a;
        if a <=b then max := b;
        if max > c then max := max;
        if max <=c then max := c;
        write(max, a);
        call Q;
end;
begin
    read(n, m);
    call rec;
end.
```

附录B

procedureHeader -> PROCEDURE ID ;
factor -> NUMBER
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
relationop -> <=
factor -> NUMBER
item -> factor factorsuf
expr -> item itemsuf
condition -> expr relationop expr
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
factor -> NUMBER
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
relationop -> +
factor -> NUMBER
item -> factor factorsuf
itemsuf -> plusop item itemsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
compound -> BEGIN multsentence END
unemptysentence -> compound
sentence -> unemptysentence
forLoop -> FOR assignment TO condition DO sentence
unemptysentence -> forLoop
sentence -> unemptysentence
compound -> BEGIN multsentence END
unemptysentence -> compound
sentence -> unemptysentence
subprogram -> constInstruction variableInstruction procedureInstruction sentence
procedureHeader -> PROCEDURE ID ;

constDeclaration -> ID = NUMBER
constDeclaration -> ID = NUMBER , constDeclaration
constInstruction -> CONST constDeclaration;
procedureHeader -> PROCEDURE ID ;
variableInstruction -> VAR ID variableInstructionSuf ;
procedureHeader -> PROCEDURE ID ;
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
writepara -> expr
writeSent -> WRITE (writepara)
unemptysentence -> writeSent
sentence -> unemptysentence
compound -> BEGIN multsentence END
unemptysentence -> compound
sentence -> unemptysentence
subprogram -> constInstruction variableInstruction procedureInstruction sentence
procedureInstruction -> procedureHeader subprogram ; procedureInstruction
readpara -> ID
readSent -> READ (readpara)
unemptysentence -> readSent
sentence -> unemptysentence
readpara -> ID
readSent -> READ (readpara)
unemptysentence -> readSent
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
relationop -> >
factor -> NUMBER
item -> factor factorsuf
expr -> item itemsuf
condition -> expr relationop expr
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
relationop -> <
factor -> NUMBER
item -> factor factorsuf
expr -> item itemsuf
condition -> expr relationop expr
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence

factor -> NUMBER
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
relationop -> >=
factor -> NUMBER
item -> factor factorsuf
expr -> item itemsuf
condition -> expr relationop expr
factor -> ID
item -> factor factorsuf
relationop -> +
factor -> ID
item -> factor factorsuf
itemsuf -> plusop item itemsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
relationop -> +
factor -> NUMBER
relationop -> *
factor -> ID
factorsuf -> mulop factor factorsuf
item -> factor factorsuf
itemsuf -> plusop item itemsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
relationop -> -
factor -> NUMBER
item -> factor factorsuf
itemsuf -> plusop item itemsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
compound -> BEGIN multsentence END

unemptysentence -> compound
sentence -> unemptysentence
whileLoop -> WHILE condition DO sentence
unemptysentence -> whileLoop
sentence -> unemptysentence
compound -> BEGIN multsentence END
unemptysentence -> compound
sentence -> unemptysentence
comditional -> IF condition THEN sentence
unemptysentence -> conditional
sentence -> unemptysentence
compound -> BEGIN multsentence END
unemptysentence -> compound
sentence -> unemptysentence
comditional -> IF condition THEN sentence
unemptysentence -> conditional
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
writepara -> expr
writeSent -> WRITE (writepara)
unemptysentence -> writeSent
sentence -> unemptysentence
compound -> BEGIN multsentence END
unemptysentence -> compound
sentence -> unemptysentence
subprogram -> constInstruction variableInstruction procedureInstruction sentence
procedureInstruction -> procedureHeader subprogram ; procedureInstruction
readpara -> ID
readSent -> READ (readpara)
unemptysentence -> readSent
sentence -> unemptysentence
readpara -> ID
readSent -> READ (readpara)
unemptysentence -> readSent
sentence -> unemptysentence
readpara -> ID
readSent -> READ (readpara)
unemptysentence -> readSent
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
relationop -> #
factor -> ID
item -> factor factorsuf
expr -> item itemsuf

condition -> expr relationop expr
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
comditional -> IF condition THEN sentence
unemptysentence -> conditional
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
relationop -> <=
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
condition -> expr relationop expr
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
comditional -> IF condition THEN sentence
unemptysentence -> conditional
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
relationop -> >
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
condition -> expr relationop expr
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
comditional -> IF condition THEN sentence
unemptysentence -> conditional
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
relationop -> <=

factor -> ID
item -> factor factorsuf
expr -> item itemsuf
condition -> expr relationop expr
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
assignment -> ID := expr
unemptysentence -> assignment
sentence -> unemptysentence
comditional -> IF condition THEN sentence
unemptysentence -> conditional
sentence -> unemptysentence
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
factor -> ID
item -> factor factorsuf
expr -> item itemsuf
writepara -> expr
writepara -> expr , writepara
writeSent -> WRITE (writepara)
unemptysentence -> writeSent
sentence -> unemptysentence
procedureCall -> CALL ID
unemptysentence -> procedureCall
sentence -> unemptysentence
compound -> BEGIN multsentence END
unemptysentence -> compound
sentence -> unemptysentence
subprogram -> constInstruction variableInstruction procedureInstruction sentence
procedureInstruction -> procedureHeader subprogram ; procedureInstruction
procedureInstruction -> procedureHeader subprogram ; procedureInstruction
readpara -> ID
readpara -> ID , readpara
readSent -> READ (readpara)
unemptysentence -> readSent
sentence -> unemptysentence
procedureCall -> CALL ID
unemptysentence -> procedureCall
sentence -> unemptysentence
compound -> BEGIN multsentence END
unemptysentence -> compound
sentence -> unemptysentence
subprogram -> constInstruction variableInstruction procedureInstruction sentence
program -> subprogram.

附录C

完整语法树

```
******************** tree ********************
< program >
 < subprogram >
  < procedure_instruction >
   < procedure_header >
    < PROCEDURE: procedure >
    < ID: rec >
    < SEMI_OP: ; >
   < subprogram >
    < sentence >
     < compound_sentence >
      < BEGIN: begin >
      < sentence >
       < for_loop_sentence >
        < FOR: for >
        < assignment_sentence >
         < ID: i >
         < ASSIGN_OP: := >
         < expression >
          < item >
           < factor >
            < NUMBER: 1 >
        < TO: to >
        < condition >
         < expression >
          < item >
           < factor >
            < ID: i >
         < relation_operator >
          < LEQ_OP: <= >
         < expression >
          < item >
           < factor >
            < NUMBER: 10 >
        < DO: do >
        < sentence >
         < compound_sentence >
          < BEGIN: begin >
          < sentence >
           < assignment_sentence >
            < ID: b >
            < ASSIGN_OP: := >
            < expression >
             < item >
```

```
                        < factor >
                          < ID: a >
                < sentence >
                  < assignment_sentence >
                    < ID: sum >
                    < ASSIGN_OP: := >
                    < expression >
                      < item >
                        < factor >
                          < NUMBER: 0 >
                < sentence >
                  < assignment_sentence >
                    < ID: i >
                    < ASSIGN_OP: := >
                    < expression >
                      < item_suf >
                        < item >
                          < factor >
                            < ID: i >
                        < plus_operator >
                          < PLUS_OP: + >
                        < item >
                          < factor >
                            < NUMBER: 1 >
                < END: end >
        < END: end >
< procedure_instruction >
  < procedure_header >
    < PROCEDURE: procedure >
    < ID: P >
    < SEMI_OP: ; >
  < subprogram >
    < const_instruction >
      < CONST: const >
      < const_declaration >
        < ID: x >
        < EQ_OP: = >
        < NUMBER: 0 >
        < COMMA_OP: , >
        < const_declaration >
          < ID: y >
          < EQ_OP: = >
          < NUMBER: 1 >
      < SEMI_OP: ; >
    < procedure_instruction >
      < procedure_header >
        < PROCEDURE: procedure >
        < ID: Q >
```

```
 < SEMI_OP: ; >
< subprogram >
 < variable_instruction >
  < VAR: var >
  < ID: a >
  < SEMI_OP: ; >
 < procedure_instruction >
  < procedure_header >
   < PROCEDURE: procedure >
   < ID: R >
   < SEMI_OP: ; >
  < subprogram >
   < sentence >
    < compound_sentence >
     < BEGIN: begin >
     < sentence >
      < write_sentence >
       < WRITE: write >
       < LEFTPARAN: ( >
       < write_parameter >
        < expression >
         < item >
          < factor >
           < ID: x >
       < RIGHTPARAN: ) >
     < END: end >
 < sentence >
  < compound_sentence >
   < BEGIN: begin >
   < sentence >
    < read_sentence >
     < READ: read >
     < LEFTPARAN: ( >
     < read_parameter >
      < ID: a >
     < RIGHTPARAN: ) >
   < sentence >
    < read_sentence >
     < READ: read >
     < LEFTPARAN: ( >
     < read_parameter >
      < ID: n >
     < RIGHTPARAN: ) >
   < sentence >
    < condition_sentence >
     < IF: if >
     < condition >
      < expression >
```

```
    < item >
      < factor >
        < ID: n >
  < relation_operator >
    < GE_OP: > >
  < expression >
    < item >
      < factor >
        < NUMBER: 0 >
< THEN: then >
< sentence >
  < compound_sentence >
    < BEGIN: begin >
    < sentence >
      < condition_sentence >
        < IF: if >
        < condition >
          < expression >
            < item >
              < factor >
                < ID: n >
          < relation_operator >
            < LE_OP: < >
          < expression >
            < item >
              < factor >
                < NUMBER: 10 >
        < THEN: then >
        < sentence >
          < compound_sentence >
            < BEGIN: begin >
            < sentence >
              < assignment_sentence >
                < ID: b >
                < ASSIGN_OP: := >
                < expression >
                  < item >
                    < factor >
                      < ID: a >
            < sentence >
              < assignment_sentence >
                < ID: sum >
                < ASSIGN_OP: := >
                < expression >
                  < item >
                    < factor >
                      < NUMBER: 0 >
            < sentence >
```

```
< while_loop_sentence >
  < WHILE: while >
  < condition >
    < expression >
      < item >
        < factor >
          < ID: n >
    < relation_operator >
      < GEQ_OP: >= >
    < expression >
      < item >
        < factor >
          < NUMBER: 0 >
  < DO: do >
  < sentence >
    < compound_sentence >
      < BEGIN: begin >
      < sentence >
        < assignment_sentence >
          < ID: sum >
          < ASSIGN_OP: := >
          < expression >
            < item_suf >
              < item >
                < factor >
                  < ID: sum >
              < plus_operator >
                < PLUS_OP: + >
              < item >
                < factor >
                  < ID: b >
      < sentence >
        < assignment_sentence >
          < ID: b >
          < ASSIGN_OP: := >
          < expression >
            < item_suf >
              < item >
                < factor >
                  < ID: b >
              < plus_operator >
                < PLUS_OP: + >
              < item >
                < factor >
                  < NUMBER: 10 >
                < multi_operator >
                  < MUL_OP: * >
                < factor >
```

```
                      < ID: a >
                < sentence >
                 < assignment_sentence >
                  < ID: n >
                  < ASSIGN_OP: := >
                  < expression >
                   < item_suf >
                    < item >
                     < factor >
                      < ID: n >
                    < plus_operator >
                     < MINUS_OP: - >
                    < item >
                     < factor >
                      < NUMBER: 1 >
                < END: end >
            < END: end >
        < END: end >
    < sentence >
     < write_sentence >
      < WRITE: write >
      < LEFTPARAN: ( >
      < write_parameter >
       < expression >
        < item >
         < factor >
          < ID: sum >
      < RIGHTPARAN: ) >
    < END: end >
< sentence >
 < compound_sentence >
  < BEGIN: begin >
  < sentence >
   < read_sentence >
    < READ: read >
    < LEFTPARAN: ( >
    < read_parameter >
     < ID: a >
    < RIGHTPARAN: ) >
  < sentence >
   < read_sentence >
    < READ: read >
    < LEFTPARAN: ( >
    < read_parameter >
     < ID: b >
    < RIGHTPARAN: ) >
  < sentence >
   < read_sentence >
```

```
  < READ: read >
  < LEFTPARAN: ( >
  < read_parameter >
   < ID: c >
  < RIGHTPARAN: ) >
< sentence >
 < condition_sentence >
  < IF: if >
  < condition >
   < expression >
    < item >
     < factor >
      < ID: a >
   < relation_operator >
    < UEQ_OP: # >
   < expression >
    < item >
     < factor >
      < ID: b >
  < THEN: then >
  < sentence >
   < assignment_sentence >
    < ID: max >
    < ASSIGN_OP: := >
    < expression >
     < item >
      < factor >
       < ID: a >
< sentence >
 < condition_sentence >
  < IF: if >
  < condition >
   < expression >
    < item >
     < factor >
      < ID: a >
   < relation_operator >
    < LEQ_OP: <= >
   < expression >
    < item >
     < factor >
      < ID: b >
  < THEN: then >
  < sentence >
   < assignment_sentence >
    < ID: max >
    < ASSIGN_OP: := >
    < expression >
```

```
          < item >
            < factor >
              < ID: b >
< sentence >
  < condition_sentence >
    < IF: if >
    < condition >
      < expression >
        < item >
          < factor >
            < ID: max >
      < relation_operator >
        < GE_OP: > >
      < expression >
        < item >
          < factor >
            < ID: c >
    < THEN: then >
    < sentence >
      < assignment_sentence >
        < ID: max >
        < ASSIGN_OP: := >
        < expression >
          < item >
            < factor >
              < ID: max >
< sentence >
  < condition_sentence >
    < IF: if >
    < condition >
      < expression >
        < item >
          < factor >
            < ID: max >
      < relation_operator >
        < LEQ_OP: <= >
      < expression >
        < item >
          < factor >
            < ID: c >
    < THEN: then >
    < sentence >
      < assignment_sentence >
        < ID: max >
        < ASSIGN_OP: := >
        < expression >
          < item >
            < factor >
```

```
            < ID: c >
     < sentence >
      < write_sentence >
       < WRITE: write >
       < LEFTPARAN: ( >
       < write_parameter >
        < expression >
         < item >
          < factor >
           < ID: max >
        < COMMA_OP: , >
        < write_parameter >
         < expression >
          < item >
           < factor >
            < ID: a >
       < RIGHTPARAN: ) >
     < sentence >
      < call_procedure >
       < CALL: call >
       < ID: Q >
     < END: end >
 < sentence >
  < compound_sentence >
   < BEGIN: begin >
   < sentence >
    < read_sentence >
     < READ: read >
     < LEFTPARAN: ( >
     < read_parameter >
      < read_parameter >
       < ID: m >
      < ID: n >
      < COMMA_OP: , >
     < RIGHTPARAN: ) >
   < sentence >
    < call_procedure >
     < CALL: call >
     < ID: rec >
   < END: end >
< EOP_OP: . >
```