一、实验基础信息

> 个人信息

201700130011 —— 刘建东 —— 17级菁英班

> 实验信息

日期：2020.4.24
题目：PL/0 语言语法分析（BLOCK） —— 说明部分的处理

二、PL/0 语言文法 BNF

```
〈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>]〈语句〉
<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>
<无符号整数> → <数字>{<数字>}
<变量说明部分> → VAR<标识符>{ ,<标识符>};
<标识符> → <字母>{<字母>|<数字>}
<过程说明部分> → <过程首部><分程序>; {<过程说明部分>}
<过程首部> → procedure<标识符>;
<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>|<空>
<赋值语句> → <标识符>:=<表达式>
<复合语句> → begin<语句>{ ; <语句>}end
<条件> → <表达式><关系运算符><表达式>|odd<表达式>
<表达式> → [+|-]<项>{<加减运算符><项>}
<项> → <因子>{<乘除运算符><因子>}
<因子> → <标识符>|<无符号整数>|(<表达式>)
<加减运符> → +|-
<乘除运算符> → *|/
<关系运算符> → =|#|<|<=|>|>=
<条件语句> → if<条件>then<语句>
<过程调用语句> → call<标识符>
<当型循环语句> → while<条件>do<语句>
<读语句> → read(<标识符>{ , <标识符>})
<写语句> → write(<标识符>{, <标识符>})
<字母> → a|b|c…x|y|z
<数字> → 0|1|2…7|8|9
```

> BNF 规则

符号	意义

<>	必选项
[]	可选项
{}	可重复0至无数次的项
::=	被定义为

> 关键字编号

编号	关键字	编号	关键字	编号	关键字	编号	关键字
0	.	8	begin	16	<>	24	while
1	const	9	end	17	<	25	do
2	,	10	odd	18	<=	26	read
3	;	11	+	19	>	27	write
4	=	12	-	20	>=	28	(
5	var	13	*	21	if	29	)
6	procedure	14	/	22	then	30	标识符
7	:=	15	=	23	call	31	常量

三、语法分析过程

在语法分析的第一部分 —— “说明部分的处理” 中，我们需要完成如下两个任务：

1. 运用递归下降子程序法分析程序语法是否正确
2. 构造结构体数组 TABLE

TX0 →

NAME: a	KIND: CONSTANT	VAL: 35	
NAME: b	KIND: CONSTANT	VAL: 49	
NAME: c	KIND: VARIABLE	LEVEL: LEV	ADR: DX
NAME: d	KIND: VARIABLE	LEVEL: LEV	ADR: DX+1
NAME: e	KIND: VAEIABLE	LEVEL: LEV	ADR: DX+2
NAME: p	KIND: PROCEDURE	LEVEL: LEV	ADR:

TX1 →

NAME: g	KIND: VARIABLE	LEVEL: LEV+1	ADR: DX
∘	∘	∘	∘
∘	∘	∘	∘
∘	∘	∘	∘

3.1 递归下降子程序法

这部分内容比较常规，根据上述的 PL/0 语言文法的 BNF 进行程序编写即可，此处给出两个例子。

首先是 <程序> → <分程序>. 这个 BNF 表示，根据这个表达式我们可以得到下述 $Procedure()$ 函数，其中优先识别 $SubProcedure()$，再识别 '.' 这个关键字，并且需要判断当前是否是词法分析数组的末尾，三个条件均满足后才能返回 $true$，否则返回 $Error()$。其中 $p1$ 表示语法分析进行到的位置下标。

```
bool Error() {
    std::cout << "Error! p1: " << p1 << std::endl;
    return false;
}

bool Procedure() {
    if(!SubProcedure()) return Error();
    if(SYM[p1] == 0 && p1 == sz) return true;
    else return Error();
}
```

接下来我们再来看看如何根据 <分程序> → [<常量说明部分>][<变量说明部分>][<过程说明部分>]<语句> 这条 BNF 表达式编写 $SubProcedure()$ 函数。编写方法依然比较简单，按顺序对非终结符进行识别即可。

```
bool SubProcedure() {
    if(SYM[p1] == 1){
        if(!ConstSpecification()) return Error();
    }
    if(SYM[p1] == 5){
        if(!VariableSpecification()) return Error();
    }
    if(SYM[p1] == 6){
        if(!ProcedureSpecification()) return Error();
    }
    return Statement();
}
```

最后给出本次实验中定义的所有非终结符子程序。

```
/*------------------- Procedure Function -------------------*/
bool Procedure();
bool SubProcedure();

/*------------------- Specification Function -------------------*/
bool ConstDefinition();
bool ConstSpecification();
```

```
bool VariableSpecification();
bool ProcedureSpecification();

/*------------------- Statement Function -------------------*/
bool Statement();
bool AssignStatement();
bool ConditionStatement();
bool LoopStatement();
bool CallStatement();
bool ReadStatement();
bool WriteStatement();
bool CompoundStatement();

/*------------------- Other Functions -------------------*/
bool Item();
bool Factor();
bool Condition();
bool Expression();
```

3.2 TABLE 表构造

首先我们需要明确 TABLE 表是什么，指导书上的要求：

- 说明部分的处理任务就是对每个过程（包括主程序）的说明对象造名字表。填写所在层次，标识符的属性和分配的相对地址等。标识符的属性不同则填写的信息不同。
- 所造的表放在全局变量 TABLE 中，TX 为指针，数组元素为结构体类型数据。LEV 给出层次，DX 给出每层的局部量的相对地址，没说明完一个变量后 DX 加 1。

因此不难发现，TABLE 表的构造主要就是识别关键字 "const"、"var"、"procedure"，因此我们只需要在识别到该位置时进行 TABLE 的构造即可。

所以接下来的问题变成了如何设置 TABLE 的结构，下述的结构体是我设置的 TABLE 表结构，其中 TABLE 表中每个表由若干的表项组成，每个表项都有一个连向另一个表的指针。

```
struct TableEntry {
    std::string NAME, KIND;
    int VAL, LEVEL, ADR, NXT;
    TableEntry() { ADR = NXT = -1; }
    TableEntry(std::string t1, int t2, int t3, int t4, int t5, int t6){
        NAME = t1; KIND = t2; VAL = t3; LEVEL = t4; ADR = t5; NXT = t6;
    }
};

struct Table {
    int PreTable, PreEntry;
    std::vector<TableEntry> table;
```

```
    Table() {
        PreTable = PreEntry = -1;
        table.clear();
    }
    Table(int t1, int t2) : PreTable(t1), PreEntry(t2) {}
};

extern std::vector<Table> TABLE;
```

定义完 TABLE 表结构后，我们再在识别 'const'、'var'、'procedure' 时进行表项添加即可。

```
bool ConstDefinition() {
    TableEntry entry;
    if(SYM[p1] == tagtype) {
        entry.NAME = ID[p2++];
        entry.KIND = CONSTANT;
        p1++;
    }
    else return Error();

    Judge(4);

    if(SYM[p1] == numtype) {
        entry.VAL = NUM[p3++];
        entry.LEVEL = LEV;
        TABLE[TX].table.push_back(entry);
        p1++;
        return true;
    }
    else return Error();
}

void RecognizeFlag(std::string flag) {
    TableEntry entry;
    entry.NAME = ID[p2++];
    entry.KIND = flag;
    entry.LEVEL = LEV;
    entry.ADR = ADR++;
    TABLE[TX].table.push_back(entry);
    p1++;
}

bool VariableSpecification() {
    Judge(5);
    if(SYM[p1] == tagtype) RecognizeFlag(VARIABLE);
    else return Error();

    while(SYM[p1] == 2) {
```

```
        p1++;
        if(SYM[p1] == tagtype) RecognizeFlag(VARIABLE);
        else return Error();
    }
    Judge(3);
    return true;
}

bool ProcedureSpecification() {
    Judge(6);
    if(SYM[p1] == tagtype){
        RecognizeFlag(PROCEDURE);
        TABLE.push_back(Table());
        TABLE[TABLE.size()-1].PreTable = TX;
        TABLE[TABLE.size()-1].PreEntry = TABLE[TX].table.size()-1;
        TABLE[TX].table[TABLE[TX].table.size()-1].NXT = TABLE.size()-1;
        TX = TABLE.size()-1, LEV++, ADR = DX;
    }
    else return Error();

    Judge(3);
    if(!SubProcedure()) return Error();
    Judge(3);
    LEV--;
    int PreEntry = TABLE[TX].PreEntry;
    TX = TABLE[TX].PreTable;
    ADR = ADR - 1 + TABLE[TX].table[PreEntry].ADR;
    TABLE[TX].table[PreEntry].ADR = ADR++;

    while(SYM[p1] == 6){
        if(!ProcedureSpecification()) return Error();
    }
    return true;
}
```

上述代码需要注意，在识别 'procedure' 关键字之后，我们需要进入下一层的说明变量，因此需要设置每个表的前驱表项，用于识别完一个表之后返回。

四、运行结果

此部分我们将给出一份 PL/0 文法的程序及其对应的运行结果与 TABLE 表结果。

PL/0 文法程序

```
var x, y, z, q, r, n, f;
```

```
procedure multiply;
var a, b;
begin
  a := x;
  b := y;
  z := x*y
end;

procedure divide;
var w;
begin
  r := x;
  q := 0;
  w := y;
  while w <= r do w := 2 * w;
  while w > y do
  begin
    q := 2 * q;
    w := w / 2;
    if w <= r then
    begin
      r := r - w;
      q := q + 1
    end
  end
end;

procedure gcd;
var  g;
begin
  f := x;
  g := y;
  while f <> g do
  begin
    if f < g then g := g - f;
    if g < f then f := f - g
  end;
  z := f
end;

procedure fact;
begin
  if n > 1 then
  begin
    f := n * f;
    n := n - 1;
    call fact
```

```
  end
end;

begin
  read(x);read(y); call multiply;write(z);
  read(x);read(y); call divide; write(q); write(r);
  read(x);read(y); call gcd; write(z);
  read(n); f := 1; call fact; write(f)
end.
```

语法分析结果

```
Lexical Analyzer Succeed!
Syntax Analyzer Succeed!
------------------------------------------------------------------------------------------------------------
TX: 0         NAME: x                 KIND: VARIABLE              LEVEL: 0              ADR: 3
              NAME: y                 KIND: VARIABLE              LEVEL: 0              ADR: 4
              NAME: z                 KIND: VARIABLE              LEVEL: 0              ADR: 5
              NAME: q                 KIND: VARIABLE              LEVEL: 0              ADR: 6
              NAME: r                 KIND: VARIABLE              LEVEL: 0              ADR: 7
              NAME: n                 KIND: VARIABLE              LEVEL: 0              ADR: 8
              NAME: f                 KIND: VARIABLE              LEVEL: 0              ADR: 9
              NAME: multiply          KIND: PROCEDURE             LEVEL: 0              ADR: 14
------------------------------------------------------------------------------------------------------------
TX: 1         NAME: a                 KIND: VARIABLE              LEVEL: 1              ADR: 3
              NAME: b                 KIND: VARIABLE              LEVEL: 1              ADR: 4
------------------------------------------------------------------------------------------------------------
TX: 0         NAME: divide            KIND: PROCEDURE             LEVEL: 0              ADR: 18
------------------------------------------------------------------------------------------------------------
TX: 2         NAME: w                 KIND: VARIABLE              LEVEL: 1              ADR: 3
------------------------------------------------------------------------------------------------------------
TX: 0         NAME: gcd               KIND: PROCEDURE             LEVEL: 0              ADR: 22
------------------------------------------------------------------------------------------------------------
TX: 3         NAME: g                 KIND: VARIABLE              LEVEL: 1              ADR: 3
------------------------------------------------------------------------------------------------------------
TX: 0         NAME: fact              KIND: PROCEDURE             LEVEL: 0              ADR: 25
------------------------------------------------------------------------------------------------------------
TX: 4
------------------------------------------------------------------------------------------------------------
```

五、源代码

main.cpp

```
#include "lexical_analyzer.h"
#include "syntax_analyzer.h"
#include <iostream>
#include <vector>
#include <string>

char indir[] = "PL0_code/input/PL0_code.in";
```

```
char outdir[] = "PL0_code/output/PL0_code.out";

int main() {
    // Lexical Analyzer
    if(!GETSYM(indir)) {std::cout << "Lexical Analyzer Error!" << std::endl; retu
rn 0;}
    else std::cout << "Lexical Analyzer Succeed!" << std::endl;
    SYM_OUTPUT(outdir);

    // Syntax Analyzer
    if(!BLOCK()) {std::cout << "Syntax Analyzer Error!" << std::endl; return 0;}
    else std::cout << "Syntax Analyzer Succeed!" << std::endl;
    TABLE_OUTPUT(outdir);
    return 0;
}
```

syntax_analyzer.h

```
#ifndef SYNTAX_ANALYZER_H
#define SYNTAX_ANALYZER_H

#include <string>
#include <vector>

#define Judge(i) if(SYM[p1] == i) {p1++;} \
                 else {return Error();}

struct TableEntry {
    std::string NAME, KIND;
    int VAL, LEVEL, ADR, NXT;
    TableEntry() { ADR = NXT = -1; }
    TableEntry(std::string t1, int t2, int t3, int t4, int t5, int t6){
        NAME = t1; KIND = t2; VAL = t3; LEVEL = t4; ADR = t5; NXT = t6;
    }
};

struct Table {
    int PreTable, PreEntry;
    std::vector<TableEntry> table;
    Table() {
        PreTable = PreEntry = -1;
        table.clear();
    }
    Table(int t1, int t2) : PreTable(t1), PreEntry(t2) {}
};
```

```
extern std::vector<Table> TABLE;

bool BLOCK();
void TABLE_OUTPUT(char *outdir);

/*------------------- Procedure Function -------------------*/
bool Procedure();
bool SubProcedure();

/*------------------- Specification Function -------------------*/
bool ConstDefinition();
bool ConstSpecification();
bool VariableSpecification();
bool ProcedureSpecification();

/*------------------- Statement Function -------------------*/
bool Statement();
bool AssignStatement();
bool ConditionStatement();
bool LoopStatement();
bool CallStatement();
bool ReadStatement();
bool WriteStatement();
bool CompoundStatement();

/*------------------- Other Functions -------------------*/
bool Item();
bool Factor();
bool Condition();
bool Expression();

#endif
```

syntax_analyzer.cpp

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include "lexical_analyzer.h"
#include "syntax_analyzer.h"

const std::string CONSTANT = "CONSTANT", VARIABLE = "VARIABLE", PROCEDURE = "PROC
EDURE";
const int DX = 3;
int TX = 0, LEV = 0, ADR = DX, p1, p2, p3, sz;
```

```
std::vector<Table> TABLE;

void DFS_OUTPUT(int TableNum, int EntryNum, int flag, std::ofstream &output) {
    // Table Header Output
    if(flag == 1){
        output << std::left << std::setfill('-') << std::setw(100) << "-" << std:
:endl;
        output << "TX: " << std::left << std::setfill(' ') << std::setw(10) << Ta
bleNum;
        if(EntryNum >= TABLE[TableNum].table.size()){
            output << std::endl;
            return;
        }
        flag = 0;
    }
    else output << "    " << std::left << std::setfill(' ') << std::setw(10) << "
 ";

    // Table Information Output
    output << "NAME: " << std::left << std::setw(20) << TABLE[TableNum].table[Ent
ryNum].NAME;
    output << "KIND: " << std::left << std::setw(25) << TABLE[TableNum].table[Ent
ryNum].KIND;
    if(TABLE[TableNum].table[EntryNum].KIND == CONSTANT)
        output << "VAL: " << std::left << std::setw(15) << TABLE[TableNum].table[
EntryNum].VAL << std::endl;
    else
        output << "LEVEL: " << std::left << std::setw(15) << TABLE[TableNum].tabl
e[EntryNum].LEVEL
                << "ADR: " << TABLE[TableNum].table[EntryNum].ADR << std::endl;

    // Recursion & Backtrack Output
    if(TABLE[TableNum].table[EntryNum].KIND == PROCEDURE) DFS_OUTPUT(TABLE[TableN
um].table[EntryNum].NXT, 0, 1, output), flag = 1;
    if(EntryNum+1 < TABLE[TableNum].table.size()) DFS_OUTPUT(TableNum, EntryNum+1
, flag, output);
}

void TABLE_OUTPUT(char *outdir) {
    std::ofstream output(outdir);
    DFS_OUTPUT(0, 0, 1, output);
    output << std::left << std::setfill('-') << std::setw(100) << "-" << std::end
l;
    output.close();
}

bool Error() {
    std::cout << "Error! p1: " << p1 << std::endl;
```

```
    return false;
}

bool BLOCK() {
    TABLE.push_back(Table());
    p1 = p2 = p3 = 0; sz = SYM.size()-1;
    return Procedure();
}

/*------------------- Procedure Function -------------------*/
bool Procedure() {
    if(!SubProcedure()) return Error();
    if(SYM[p1] == 0 && p1 == sz) return true;
    else return Error();
}

bool SubProcedure() {
    if(SYM[p1] == 1){
        if(!ConstSpecification()) return Error();
    }
    if(SYM[p1] == 5){
        if(!VariableSpecification()) return Error();
    }
    if(SYM[p1] == 6){
        if(!ProcedureSpecification()) return Error();
    }
    return Statement();
}

/*------------------- Specification Function -------------------*/
bool ConstDefinition() {
    TableEntry entry;
    if(SYM[p1] == tagtype) {
        entry.NAME = ID[p2++];
        entry.KIND = CONSTANT;
        p1++;
    }
    else return Error();

    Judge(4);

    if(SYM[p1] == numtype) {
        entry.VAL = NUM[p3++];
        entry.LEVEL = LEV;
        TABLE[TX].table.push_back(entry);
        p1++;
        return true;
    }
```

```
    else return Error();
}

bool ConstSpecification() {
    Judge(1);

    if(!ConstDefinition()) return Error();

    while(SYM[p1] == 2){
        p1++;
        if(!ConstDefinition()) return Error();
    }
    Judge(3);
    return true;
}

void RecognizeFlag(std::string flag) {
    TableEntry entry;
    entry.NAME = ID[p2++];
    entry.KIND = flag;
    entry.LEVEL = LEV;
    entry.ADR = ADR++;
    TABLE[TX].table.push_back(entry);
    p1++;
}

bool VariableSpecification() {
    Judge(5);
    if(SYM[p1] == tagtype) RecognizeFlag(VARIABLE);
    else return Error();

    while(SYM[p1] == 2) {
        p1++;
        if(SYM[p1] == tagtype) RecognizeFlag(VARIABLE);
        else return Error();
    }
    Judge(3);
    return true;
}

bool ProcedureSpecification() {
    Judge(6);
    if(SYM[p1] == tagtype){
        RecognizeFlag(PROCEDURE);
        TABLE.push_back(Table());
        TABLE[TABLE.size()-1].PreTable = TX;
        TABLE[TABLE.size()-1].PreEntry = TABLE[TX].table.size()-1;
        TABLE[TX].table[TABLE[TX].table.size()-1].NXT = TABLE.size()-1;
```

```
        TX = TABLE.size()-1, LEV++, ADR = DX;
    }
    else return Error();

    Judge(3);
    if(!SubProcedure()) return Error();
    Judge(3);
    LEV--;
    int PreEntry = TABLE[TX].PreEntry;
    TX = TABLE[TX].PreTable;
    ADR = ADR - 1 + TABLE[TX].table[PreEntry].ADR;
    TABLE[TX].table[PreEntry].ADR = ADR++;

    while(SYM[p1] == 6){
        if(!ProcedureSpecification()) return Error();
    }
    return true;
}

/*------------------- Statement Function -------------------*/
bool Statement() {
    if(SYM[p1] == tagtype) return AssignStatement();
    if(SYM[p1] == 21) return ConditionStatement();
    if(SYM[p1] == 24) return LoopStatement();
    if(SYM[p1] == 23) return CallStatement();
    if(SYM[p1] == 26) return ReadStatement();
    if(SYM[p1] == 27) return WriteStatement();
    if(SYM[p1] == 8) return CompoundStatement();
    return true;
}

bool AssignStatement() {
    if(SYM[p1] == tagtype) p1++, p2++;
    else return Error();

    Judge(7);
    return Expression();
}

bool ConditionStatement() {
    Judge(21);
    if(!Condition()) return Error();
    Judge(22);
    return Statement();
}

bool LoopStatement() {
    Judge(24);
```

```
    if(!Condition()) return Error();
    Judge(25);
    return Statement();
}

bool CallStatement() {
    Judge(23);
    if(SYM[p1] == tagtype) p1++, p2++;
    else return Error();
    return true;
}

bool ReadStatement() {
    Judge(26);
    Judge(28);
    if(SYM[p1] == tagtype) p1++, p2++;
    else return Error();

    while(SYM[p1] == 2){
        p1++;
        if(SYM[p1] == tagtype) p1++, p2++;
        else return Error();
    }

    Judge(29);
    return true;
}

bool WriteStatement() {
    Judge(27);
    Judge(28);
    if(SYM[p1] == tagtype) p1++, p2++;
    else return Error();

    while(SYM[p1] == 2){
        p1++;
        if(SYM[p1] == tagtype) p1++, p2++;
        else return Error();
    }

    Judge(29);
    return true;
}

bool CompoundStatement() {
    Judge(8);
    if(!Statement()) return Error();
    while(SYM[p1] == 3){
```

```
        p1++;
        if(!Statement()) return Error();
    }
    Judge(9);
    return true;
}

/*------------------- Other Functions -------------------*/
bool Item() {
    if(!Factor()) return Error();
    while(SYM[p1] == 13 || SYM[p1] == 14){
        p1++;
        if(!Factor()) return Error();
    }
    return true;
}

bool Factor() {
    if(SYM[p1] == tagtype) {
        p1++, p2++;
        return true;
    }
    if(SYM[p1] == numtype) {
        p1++, p3++;
        return true;
    }
    if(SYM[p1] == 28) {
        p1++;
        if(!Expression()) return Error();
        Judge(29);
        return true;
    }
    return Error();
}

bool Condition() {
    if(SYM[p1] == 10){
        p1++;
        return Expression();
    }
    else{
        if(!Expression()) return Error();
        if(SYM[p1] >= 15 && SYM[p1] <= 20) { p1++; return Expression();}
        else return Error();
    }
}

bool Expression() {
```

```
    if(SYM[p1] == 11 || SYM[p1] == 12) p1++;
    if(!Item()) return Error();
    while(SYM[p1] == 11 || SYM[p1] == 12) {
        p1++;
        if(!Item()) return Error();
    }
    return true;
}
```