**电子科技大学计算机学院**

**标 准 实 验 报 告**

**（实验）课程名称 编译原理**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：叶文慧 学 号： 2017060101001指导教师：陈文宇**

**实验地点： HOME 实验时间：2020.5.1**

**一、实验室名称：无**

**二、实验项目名称：求n!的极小语言**

**三、实验学时：4**

**四、实验原理：（包括知识点，电路图，流程图）**

1. 左递归：当文法中出现如下的无限推导而不能匹配任何输入字符的情形成为左递归：A⇒Aα⇒Aαα⇒Aααα⇒…⇒ 左递归分为直接左递归和间接左递归。直接左递归是在文法中直接出现左递归，即有形如A->Aα的产生式;间接左递归则是通过推导产生的左递归。要通过递归下降分析法实现语言的语法分析需要先消除文法中所有的左递归。文法消除左递归后的文法如下：

**<程序>→<分程序>**

**<分程序>→begin <说明语句表>；<执行语句表> end**

**<说明语句表>→<说明语句><说明语句表>’**

**<说明语句表>’ →；<说明语句><说明语句表>’|ɛ**

**<说明语句>→<变量说明>│<函数说明>**

**<变量说明>→integer <变量>**

**<变量>→<标识符>**

**<标识符>→<字母><标识符>‘**

**<标识符>‘→<字母><标识符>‘|<数字><标识符>’|ɛ**

**<字母>→a│b│c│d│e│f│g│h│i│j│k│l│m│n│o│p│q │r│s│t│u│v│w│x│y│z**

**<数字>→0│1│2│3│4│5│6│7│8│9**

**<函数说明>→integer function <标识符>（<参数>）；<函数体>**

**<参数>→<变量>**

**<函数体>→begin <说明语句表>；<执行语句表> end**

**<执行语句表>→<执行语句><执行语句表>’**

**<执行语句表>’→ ；<执行语句><执行语句表>’|ɛ**

**<执行语句>→<读语句>│<写语句>│<赋值语句>│<条件语句>**

**<读语句>→read(<变量>)**

**<写语句>→write(<变量>)**

**<赋值语句>→<变量>:=<算术表达式>**

**<算术表达式>→<项><算术表达式>’**

**<算术表达式>’ →-<项><算术表达式>’|ɛ**

**<项>→<因子><项>’**

**<项>’ →\*<因子><项>’| ɛ**

**<因子>→<变量>│<常数>│<函数调用>**

**<常数>→<无符号整数>**

**<无符号整数>→<数字><无符号整数>’**

**<无符号整数>’ →<数字><无符号整数>’|ɛ**

**<函数调用>→<标识符>(算术表达式）**

**<条件语句>→if<条件表达式>then<执行语句>else <执行语句>**

**<条件表达式>→<算术表达式><关系运算符><算术表达式>**

**<关系运算符> →<│<=│>│>=│=│<>**

1. 当文法改造为无公共左因子,无左递归时,让每个非终结符对应一个过程，该过程对相应的非终结符产生式的右部短语进行语法分析，这种分析方法称为递归下降分析法。这样的分析程序称为递归下降分析器。

**五、实验目的：**

通过本次设计递归下降分析器的设计与实现实验，使同学们掌握自上而下的递归分析法的语法分析原理和程序设计方法，了解掌握递归下降分析方法。

**六、实验内容：（介绍自己所选的实验内容）**

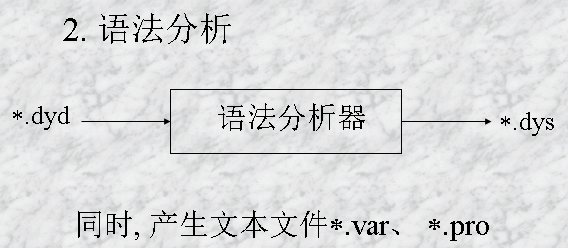
根据给定的方法，编写相应的递归下降的语法分析程序，实现对词法分析后的单词序列的语法检查和程序结构的分析，生成相应的变量名表和过程名表，并将编译中语法检查出来的错误写入相应的文件。

语法错分类:

(1)缺少符号错;

(2)符号匹配错

(3)符号无定义或重复定义。



**七、实验器材（设备、元器件）：**

操作系统：Windows 10

IDE： VS 2017

**八、实验步骤：（编辑调试的过程）**

1. 在VS 2017中创建工程；
2. 编写文件输入输出；
3. 对输入数据进行处理，按空格分成语句和种别码两部分；
4. 实现递归下降分析器，初步完成语法的分析；
5. 添加变量名表的相关功能；
6. 添加过程名表的相关功能；
7. 更改测试程序文件，编写出错处理函数；
8. 添加出错处理功能。

**九、实验数据及结果分析：（实验运行结果介绍或者截图，对不同的结果进行分析）**

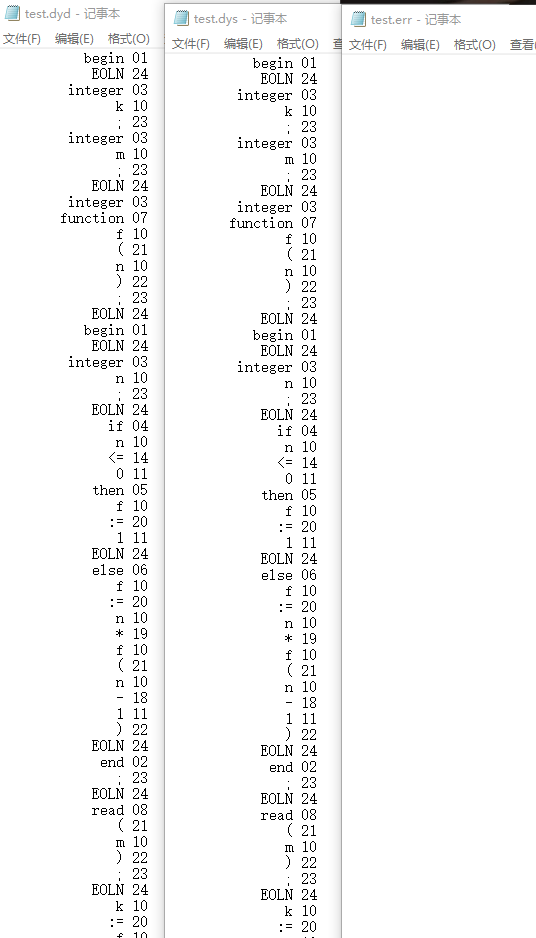


图1. 正确的语法分析输入文件及语法输出

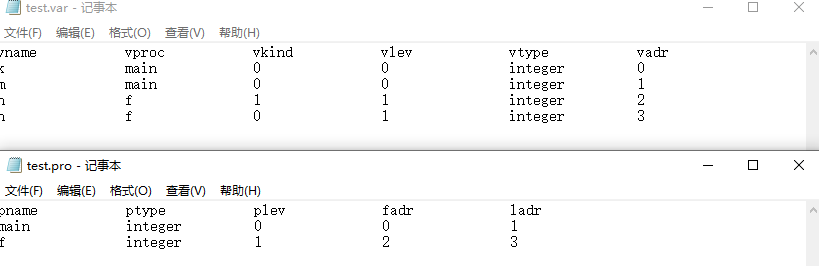


图2. 语法分析变量名表与过程名表输出

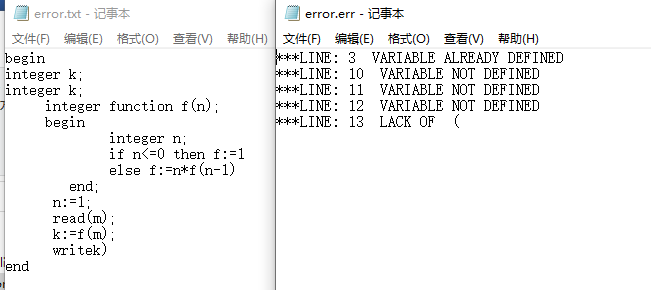


图3. 错误源文件及错误输出

**十、实验结论：（联系理论知识进行说明）**

本次实验实现了语法分析器，实现了语法分析器的功能，更加深刻地理解了递归下降分析法。

**十一、总结及心得体会：**

通过本次实验，我对语法分析器的理解更加深刻，锻炼了自己的代码能力。

**十二、对本实验过程及方法、手段的改进建议：**

无

**报告评分：**

**指导教师签字：**

代码详情请前往<https://github.com/pulupulupu/Compiler_Labs>查看

#pragma once

#include<vector>

#include<string>

#include<sstream>

#include<fstream>

#include<iostream>

#include<iomanip>

using namespace std;

#define BEGIN 1

#define END 2

#define INTEGER 3

#define IF 4

#define THEN 5

#define ELSE 6

#define FUNCTION 7

#define READ 8

#define WRITE 9

#define IDENTIFIER 10

#define CONSTANT 11

#define EOLN 24

#define EOF 25

//错误类型

#define LACK\_OF " LACK OF "

#define VARIABLE\_ALREADY\_DEFINED " VARIABLE ALREADY DEFINED "

#define VARIABLE\_NOT\_DEFINED " VARIABLE NOT DEFINED "

#define DOES\_NOT\_MATCH " DOES NOT MATCH "

class Statements

{

public:

string statement; //语句

int code; //语句代码

static void ReadFile(); //读文件

};

class Processes

{

string pname; //过程名

string ptype; //过程类型，默认integer

int plev; //过程层次

int fadr; //第一个变量在变量名表中的位置

int ladr; //最后一个变量在变量名表中的位置

public:

Processes(string, int, string,int,int);

static void WriteFile(); //写文件

bool IfInTheTable(); //过程是否已经在表里了

};

class Variables

{

string vname; //变量名

string vproc; //所属过程

bool vkind; //分类

string vtype; //变量类型

int vlev;//变量层次

int vadr;//变量位置

public:

Variables(string, string, bool, int, int,string);//初始化类

bool IfNotDefined(); //判断是否已定义

bool IfParameter(); //判断是否是参数

static void WriteFile(); //写文件

friend void Processes::WriteFile();

};

class Error

{

string ErrorType; //错误的类型

int ErrorLine; //错误所在行数

string ErrorDisCription; //错误描述//

public:

Error(int, string,string);

static void WriteFile(); //写文件

};

void Advance();

void Program(); //程序

void SubProgram(); //分程序

void StatementList(); //说明语句表

void StatementListA(); //说明语句表'

void ExecutionStatementList(); //执行语句表

void ExecutionStatementListA(); //执行语句表'

void ExplanatoryStatement(); //说明语句

void VariableDescription(); //变量说明

void FunctionDescription(); //函数说明

void Variable(); //变量

void Identifier(); //标识符

//void IdentifierA(); //标识符'

//void Letter(); //字母

//void Number(); //数字

void Parameter(); //参数

void FunctionBody(); //函数体

void ExecuteStatement();//执行语句

void ReadSentence(); //读语句

void WriteSentence(); //写语句

void AssignmentStatement(); //赋值语句

void ConditionalStatement(); //条件语句

void ArithmeticExpression(); //算术表达式

void ArithmeticExpressionA(); //算术表达式'

void Item();//项

void ItemA();//项'

void Factor();//因子

void Constant();//常数

void FunctionCall();//函数调用

//void UnsignedInteger(); //无符号整数

//void UnsignedIntegerA(); //无符号整数'

void ConditionalExpression(); //条件表达式

void RelationalOperator(); //关系运算符

#include<iostream>

#include<string>

#include<cstdlib>

#include"Parser.h"

using namespace std;

extern vector<Statements> text;

string Dyd = "./.dyd";

string Dys = "./.dys";

string Var = "./.var";

string Pro = "./.pro";

string Err = "./.err";

int main(int argc, char\*argv[])

{

Dyd.insert(2, argv[1]);

Dys.insert(2, argv[1]);

Pro.insert(2, argv[1]);

Var.insert(2, argv[1]);

Err.insert(2, argv[1]);

Statements::ReadFile();

Program();

Variables::WriteFile();

Processes::WriteFile();

Error::WriteFile();

system("pause");

return 0;

}

#include"Parser.h"

vector<Statements> text;

vector<Variables> variable;

vector<Processes> processes;

vector<Error> errors;

vector<Statements>::iterator pointer;

extern string Dys;

extern string Dyd;

extern string Pro;

extern string Var;

extern string Err;

bool ifparameter = false;

string nowprocess = "main";

int nowlevel = 0;

int location=0;

int line = 1;

//读文件

void Statements::ReadFile()

{

ifstream File\_Input;

string temp;

File\_Input.open(Dyd);

if (!File\_Input)

{

cout << "File Open Failed" << endl;

exit(0);

}

ofstream File\_Output;

File\_Output.open(Dys);

if (!File\_Output)

{

cout << "File Open Failed" << endl;

exit(0);

}

while (!File\_Input.eof())

{

getline(File\_Input, temp);

File\_Output << temp << endl;

stringstream input(temp);

Statements new\_statement;

input >> new\_statement.statement >> new\_statement.code;

text.push\_back(new\_statement);

}

File\_Input.close();

File\_Output.close();

}

//将当前输入指针切换到下一个

void Advance()

{

pointer++;

if ((\*pointer).code == EOLN)

{

line++;

Advance();

}

}

//Variables的函数

Variables::Variables(string vname, string vproc, bool vkind, int vlev, int vadr, string vtype="integer")

{

this->vname = vname;

this->vproc = vproc;

this->vkind = vkind;

this->vlev = vlev;

this->vtype = vtype;

this->vadr = vadr;

}

//判断对象是否未定义

bool Variables::IfNotDefined()

{

bool notdefined=true;

for (Variables i : variable)

{

if (this->vname == i.vname && this->vproc == i.vproc &&this->vkind==i.vkind && this->vlev == i.vlev )

{

notdefined = false;

break;

}

}

return notdefined;

}

//判断类是否为形参

bool Variables::IfParameter()

{

return this->vkind;

}

//写变量名表文件

void Variables::WriteFile()

{

ofstream File\_Output;

File\_Output.open(Var);

if (!File\_Output)

{

cout << "File Open Failed" << endl;

exit(0);

}

File\_Output.fill(' ');

File\_Output <<left<< setw(16) << "vname";

File\_Output <<left<< setw(16) << "vproc";

File\_Output << left<<setw(16) << "vkind";

File\_Output <<left<< setw(16) << "vlev";

File\_Output <<left<< setw(16) << "vtype";

File\_Output <<left<< setw(16) << "vadr" << endl;

for (Variables i : variable)

{

File\_Output << left<<setw(16) << i.vname;

File\_Output <<left<< setw(16) << i.vproc;

File\_Output <<left<< setw(16) << i.vkind;

File\_Output <<left<< setw(16) << i.vlev;

File\_Output << left<<setw(16) << i.vtype;

File\_Output<<left<<setw(16)<<i.vadr<< endl;

}

File\_Output.close();

}

//Processes的函数

Processes::Processes(string pname, int plev, string ptype= "integer",int fadr=0,int ladr=0)

{

this->pname = pname;

this->plev = plev;

this->ptype = ptype;

this->ladr = fadr;

this->ladr = ladr;

}

//写过程名表文件

void Processes::WriteFile()

{

//初始化fadr和ladr

bool flag = true;

for (Processes & i : processes)

{

for (Variables j : variable)

{

if (i.pname == j.vproc)

{

if (flag)

{

i.fadr = j.vadr;

i.ladr = j.vadr;

flag = false;

}

else

i.ladr = j.vadr;

//break;

}

}

flag = true;

}

ofstream File\_Output;

File\_Output.open(Pro);

if (!File\_Output)

{

cout << "File Open Failed" << endl;

exit(0);

}

File\_Output.fill(' ');

File\_Output << left << setw(16) << "pname";

File\_Output << left << setw(16) << "ptype";

File\_Output << left << setw(16) << "plev";

File\_Output << left << setw(16) << "fadr";

File\_Output << left << setw(16) << "ladr" << endl;

for (Processes i : processes)

{

File\_Output << left << setw(16) << i.pname;

File\_Output << left << setw(16) << i.ptype;

File\_Output << left << setw(16) << i.plev;

File\_Output << left << setw(16) << i.fadr;

File\_Output << left << setw(16) << i.ladr<<endl;

}

File\_Output.close();

}

//判断对象是否未在表里

bool Processes::IfInTheTable()

{

for (Processes i : processes)

{

if (this->pname == i.pname && this->plev == i.plev)

return false;

}

return true;

}

//Error的函数

Error::Error(int ErrorLine, string ErrorType, string ErrorDiscription=" ")

{

this->ErrorDisCription = ErrorDiscription;

this->ErrorType = ErrorType;

this->ErrorLine = ErrorLine;

}

void Error::WriteFile()

{

ofstream File\_Output;

File\_Output.open(Err);

if (!File\_Output)

{

cout << "File Open Failed" << endl;

exit(0);

}

for (Error i:errors)

{

File\_Output << "\*\*\*LINE: " << i.ErrorLine << " "<<i.ErrorType<<" "<<i.ErrorDisCription<<endl;

}

File\_Output.close();

}

//程序

void Program()

{

pointer = text.begin();

SubProgram();

}

//分程序

void SubProgram()

{

Processes main("main", 0);

processes.push\_back(main);

if ((\*pointer).code == BEGIN)

{

Advance();

StatementList();

if ((\*pointer).code == 23)

{

Advance();

ExecutionStatementList();

if ((\*pointer).code == END)

Advance();

else

{

Error new\_error(line, LACK\_OF,"end");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,";");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,"begin");

errors.push\_back(new\_error);

}

}

//说明语句表

void StatementList()

{

ExplanatoryStatement();

StatementListA();

}

//说明语句表'

void StatementListA()//此处需添加缺少；错

{

auto next\_pointer = pointer + 1;

if ((\*next\_pointer).code == EOLN) next\_pointer++;

if ((\*pointer).code == 23 && (\*next\_pointer).code == INTEGER)

{

Advance();

ExplanatoryStatement();

StatementListA();

}

else if ((\*pointer).code ==INTEGER)

{

Error new\_error(line-1, LACK\_OF, ";");

errors.push\_back(new\_error);

ExplanatoryStatement();

StatementListA();

}

}

//执行语句表

void ExecutionStatementList()

{

ExecuteStatement();

ExecutionStatementListA();

}

//执行语句表'

void ExecutionStatementListA()

{

auto next\_pointer = pointer + 1;

if ((\*next\_pointer).code == EOLN) next\_pointer++;

if ((\*pointer).code == 23)

{

Advance();

ExecuteStatement();

ExecutionStatementListA();

}

else if ((\*pointer).code != END && (\*next\_pointer).code != END && (\*pointer).code != 23) //未匹配到； 且下方还有执行语句

{

Error new\_error(line-1, LACK\_OF, ";");

errors.push\_back(new\_error);

ExecuteStatement();

ExecutionStatementListA();

}

else

return;

}

//说明语句

void ExplanatoryStatement()

{

auto next\_pointer = pointer + 1;

if ((\*next\_pointer).code == FUNCTION)

FunctionDescription();

else

VariableDescription();

}

//变量说明

void VariableDescription()

{

if ((\*pointer).code == INTEGER)

{

Advance();

Variable();

}

else

{

Error new\_error(line, LACK\_OF, "integer");

errors.push\_back(new\_error);

}

}

//函数说明

void FunctionDescription()

{

nowlevel++;

string TempToStoreLastProcess = nowprocess;

nowprocess = (\*(pointer +2)).statement;

if ((\*pointer).code == INTEGER)

{

Advance();

if ((\*pointer).code == FUNCTION)

{

Advance();

Identifier();

if ((\*pointer).code == 21)

{

Advance();

ifparameter = true;

Parameter();

ifparameter = false;

if ((\*pointer).code == 22)

{

Advance();

if ((\*pointer).code == 23)

{

Advance();

FunctionBody();

}

else

{

Error new\_error(line, LACK\_OF,";");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,")");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,"(");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,"function");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,"integer");

errors.push\_back(new\_error);

}

nowprocess = TempToStoreLastProcess;

nowlevel--;

}

//变量

void Variable() //变量重定义在此处判断

{

//Variables(string vname, string vproc, bool vkind, int vlev, int vadr, string vtype="integer")

Variables new\_variable((\*pointer).statement, nowprocess, ifparameter, nowlevel, location);

if ((\*pointer).statement != nowprocess) //非返回值

{

if ((\*(pointer - 1)).code == INTEGER)

{

if (new\_variable.IfNotDefined() || variable.size() == 0)

{

variable.push\_back(new\_variable);

location++;

}

else //重定义

{

Error new\_error(line, VARIABLE\_ALREADY\_DEFINED);

errors.push\_back(new\_error);

}

}

else

{

if (new\_variable.IfParameter() && new\_variable.IfNotDefined()) //变量为参数

{

variable.push\_back(new\_variable);

location++;

}

else if (new\_variable.IfNotDefined())

{

Error new\_error(line, VARIABLE\_NOT\_DEFINED);

errors.push\_back(new\_error);

}

}

}

Identifier();

}

//标识符

void Identifier()

{

Advance();

}

//由于标识符一次性读入，故不需要下列产生式起作用

/\*

//标识符'

void IdentifierA()

{

if ((\*pointer).code == 11)// 可能存在问题

Number();

else

Letter();

IdentifierA();

}

//字母

void Letter()

{

}

//数字

void Number()

{

}

\*/

//参数

void Parameter()

{

Variable();

}

//函数体

void FunctionBody()

{

Processes new\_process((\*(pointer - 6)).statement, nowlevel);

if(new\_process.IfInTheTable())

processes.push\_back(new\_process);

if ((\*pointer).code == BEGIN)

{

Advance();

StatementList();

if ((\*pointer).code == 23)

{

Advance();

ExecutionStatementList();

if ((\*pointer).code == END)

Advance();

else

{

Error new\_error(line, LACK\_OF,"end");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,";");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,"begin");

errors.push\_back(new\_error);

}

}

//执行语句

void ExecuteStatement()

{

if ((\*pointer).code == READ)

ReadSentence();

else if ((\*pointer).code == WRITE)

WriteSentence();

else if ((\*pointer).code == IF)

ConditionalStatement();

else if((\*pointer).code==IDENTIFIER)

AssignmentStatement();

}

//读语句

void ReadSentence()

{

if ((\*pointer).code == READ)

{

//Processes new\_process((\*pointer).statement, nowlevel);

//if (new\_process.IfInTheTable())

// processes.push\_back(new\_process);

Advance();

if ((\*pointer).code == 21)

{

Advance();

Variable();

if ((\*pointer).code == 22)

Advance();

else

{

Error new\_error(line, LACK\_OF,")");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,"(");

errors.push\_back(new\_error);

Advance();

}

}

else

{

Error new\_error(line, LACK\_OF,"read");

errors.push\_back(new\_error);

}

}

//写语句

void WriteSentence()

{

if ((\*pointer).code == WRITE)

{

//Processes new\_process((\*pointer).statement, nowlevel);

//if (new\_process.IfInTheTable())

// processes.push\_back(new\_process);

Advance();

if ((\*pointer).code == 21)

{

Advance();

Variable();

if ((\*pointer).code == 22)

Advance();

else

{

Error new\_error(line, LACK\_OF,")");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF,"(");

errors.push\_back(new\_error);

Advance();

}

}

else

{

Error new\_error(line, LACK\_OF,"write");

errors.push\_back(new\_error);

}

}

//赋值语句

void AssignmentStatement()

{

Variable();

if ((\*pointer).code == 20)

{

Advance();

ArithmeticExpression();

}

else

{

Error new\_error(line, LACK\_OF, ":=");

errors.push\_back(new\_error);

}

}

//条件语句

void ConditionalStatement()

{

if ((\*pointer).code == IF)

{

Advance();

ConditionalExpression();

if ((\*pointer).code == THEN)

{

Advance();

ExecuteStatement();

if ((\*pointer).code == ELSE)

{

Advance();

ExecuteStatement();

}

else

{

Error new\_error(line, LACK\_OF, "else");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF, "then");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF, "if");

errors.push\_back(new\_error);

}

}

//算术表达式

void ArithmeticExpression()

{

Item();

ArithmeticExpressionA();

}

//算术表达式'

void ArithmeticExpressionA()

{

if ((\*pointer).code == 18)

{

Advance();

Item();

ArithmeticExpressionA();

}

}

//项

void Item()

{

Factor();

ItemA();

}

//项'

void ItemA()

{

if ((\*pointer).code == 19)

{

Advance();

Factor();

ItemA();

}

/\*

else if ((\*pointer).code == IDENTIFIER)

{

Error new\_error(line - 1, LACK\_OF, "\*");

errors.push\_back(new\_error);

Factor();

ItemA();

}

\*/

else

return;

}

//因子

void Factor()

{

auto next\_pointer = pointer + 1;

if ((\*pointer).code == CONSTANT)

Constant();

else if ((\*next\_pointer).code==21) //对应函数调用

FunctionCall();

else

Variable();

}

//常数

void Constant()

{

//UnsignedInteger();

Advance();

}

//函数调用

void FunctionCall()

{

Identifier();

if ((\*pointer).code == 21)

{

Advance();

ArithmeticExpression();

if ((\*pointer).code == 22)

Advance();

else

{

Error new\_error(line, LACK\_OF,")");

errors.push\_back(new\_error);

}

}

else

{

Error new\_error(line, LACK\_OF, "(");

errors.push\_back(new\_error);

Advance();

}

}

/\*

//无符号整数

void UnsignedInteger()

{

Number();

UnsignedIntegerA();

}

//无符号整数'

void UnsignedIntegerA()

{

Number();

UnsignedIntegerA();

}

\*/

//条件表达式

void ConditionalExpression()

{

ArithmeticExpression();

RelationalOperator();

ArithmeticExpression();

}

//关系运算符

void RelationalOperator()

{

int code = (\*pointer).code;

switch (code)

{

case 12:

case 13:

case 14:

case 15:

case 16:

case 17:

Advance();

break;

default:

return;

}

}