编号：CS65

实习	一	二	三	四	五	六	七	八	九	十	总评	教师签名
成绩												

武汉大学计算机学院

《编译原理》课程

语法分析

实习报告

编　　号：20202021605

实习题目：mini 语言语法分析

专业（班）：计算机科学与技术

学生学号：2018302110169

学生姓名：孙嘉曦

任课教师：杜　卓　敏

2021 年　　6月　　22 日

目　　录

第一部分　　语言语法规则

首先,mini 语言支持的语句如下:算术表达式、逻辑表达式、赋值语句、if-then 分支结构、while 循环结构。

Mini 语言支持的符号如词法分析中所述。Mini 语言程序以#为终止符，划分为一个程序段。

Mini 语言在分析过程中，若正确分析了全部源程序，则输出语法树。反之，出现错误时，会返回错误位置。

第二部分　　文法定义

//一个程序,可能包含许多程序段;程序由变量声明和陈述语句组成

Program → (VariableDeclaration | Statement) [Program]

//变量声明,可只声明不初始化,可初始化,可声明多变量

VariableDeclaration → Type VariableDefine VariableDeclaration ';'

| ',' VarDefine VarDeclaration

| ε

//变量定义,给出变量名,或者后接初始化

VariableDefine → Id

| Id '=' Expression

//语句,包含变量声明\表达式\空\分号\以及 if 和 while 语句

Statement → Id '='Expression ';'

| Expression ';'

| ';'

| ε

| IfStmt

|Whlie

//if 控制结构,if 嵌套

IfControl → 'if' '(' Expression ')' '{' Statement '}' IfControl

| 'else' '{' Statement '}'

//while 控制结构

WhileControl → 'while' '(' Expression')' '{' Statement '}'

//仅 int 类型,可扩充

Type → 'int'

//表达式统一处理以及优先级结构,在算法中详细解释

Expression → <LHS> <OP> <RHS> | <LHS>

<LHS> → <PrimaryExpression>

<RHS> → <PrimaryExpression>

<PrimaryExpression> → <Variable>

| <LiteralExpression>

|(Expression)

<OP> -> !

| * / %

| + -

| << >>

| < <= >= >

| == !=

| &

| ^

| |

| &&

| ||

第三部分　　语法分析算法

语法分析的总体过程是通过递归调用子程序,子程序中满足条件返回相应的语法分析树节点。由文法定义，我将过程划分为两个阶段，避免了消除左递归产生更加复杂多样的产生式。

第一部分：

将表达式作为一个整体，先处理其他部分，忽视表达式的多样性。以下产生式（暂时忽略右半部），用于第一部分的分析

Program → ； VariableDeclaration → ； Type →； VariableDefine → ； Statement → ； IfControl →； WhileControl → 。

Vn / Vt	Program	VariableDeclaration	VariableDefine	Statement	IfControl	WhileControl
type	1	1	**E1**	**E2**	**E3**	**E22**
,	1	2	**E4**	**E5**	**E6**	**E23**
Id	2	**E7**	1	1	**E8**	**E24**
Expression	2	**E9**	**E10**	2	**E11**	**E25**
;	2	**E12**	**E13**	3	**E14**	**E26**
If	2	**E15**	**E16**	5	1	**E27**
else	2	**E17**	**E18**	5	2	**E28**
#	**E19**	3	**E20**	4	**E21**	**E29**
while	2	**E32**	**E31**	6	**E30**	1

第二部分：

在具体的实现过程中，用如下产生式对表达式部分进行处理。已列出关键表达式：

//把每一个表达式分为左半部分,运算符,和右半部分

表达式可以仅为左半部分

Expression → <LHS> <OP> <RHS> | <LHS>

//左表达式和右表达式都可以产生初始表达式

<LHS> → <PrimaryExpression>

<RHS> → <PrimaryExpression>

//初始表达式可以是变量\字母\表达式

<PrimaryExpression> → <Variable>

| <LiteralExpression>

|(Expression)

以下部分算法用于解析运算符的优先级并产生相应的子程序节点。

这个算法是这样理解的，它有三个参数，左表达式运算符优先级，左表达式节点，节点类型。返回的是一个新的子节点。

对于当前的运算符，有一个优先级 tokprec。

我们假定左式为 a+b，那么，+的优先级预定为 1，如果当前运算符为*，则 tokprec 为 2。2>1，那么显然对于 a+b*c，我们不能对 a+b 直接返回 AST，而是要对 b*c 和之后的式子进行结合。

我们假定左式为 a*b，优先级 2，若当前运算符为+，优先级较小，对于 a*b 就可以返回一个 LHSChild 作为节点。

```
unique_ptr<Node> Parser::ParseBinOpRHS(int ExprPrec,unique_ptr<Node> LHSChild,node_type NodeType) {
    while (true) {
        string value = nowToken();
        int TokPrec = GetTokPrecedence(nowToken());

        if (TokPrec < ExprPrec)
            return LHSChild;

        auto OPChild = OP();

        auto RHSChild = RHS();
        if (!RHSChild)
            return nullptr;

        int NextPrec = GetTokPrecedence(nowToken());
        if (TokPrec < NextPrec) {
            RHSChild->type = node_type::LHS;
            RHSChild = ParseBinOpRHS(TokPrec + 1, std::move(RHSChild),node_type::RHS);
            if (!RHSChild)
                return nullptr;
        }

        unique_ptr<Node>newLHSChild(new Node("", NodeType));
        newLHSChild->addChildNode(move(LHSChild));
        newLHSChild->addChildNode(move(OPChild));
        newLHSChild->addChildNode(move(RHSChild));
        LHSChild = move(newLHSChild);
    }
}
```

```
unique_ptr<Node> Parser::BinaryExpression(){
    unique_ptr<Node>binaryExpressionNode(new Node("", node_type::BinaryExpression));
    auto LHSChild = LHS();
    if (!LHSChild)return nullptr;
    if (tryEat(token_type::LOGICOR) || tryEat(token_type::LOGICAND) || tryEat(token_type::OR) ||
        tryEat(token_type::XOR) || tryEat(token_type::AND) || tryEat(token_type::EQUALITY) ||
        tryEat(token_type::NOTEQUAL) || tryEat(token_type::LESS) || tryEat(token_type::LESSEQUAL) ||
        tryEat(token_type::MORE) || tryEat(token_type::MOREEQUAL) || tryEat(token_type::LSHIFT) ||
        tryEat(token_type::RSHIFT) || tryEat(token_type::PLUS) || tryEat(token_type::MINUS) ||
        tryEat(token_type::STAR) || tryEat(token_type::SLASH) || tryEat(token_type::MOD) ||
        tryEat(token_type::NOT)) {
        auto Childs = move(ParseBinOpRHS(0, move(LHSChild), node_type::LHS));
        for (int child = 0; child != Childs->childNodes.size(); child++) {
            binaryExpressionNode->addChildNode(move(Childs->childNodes[child]));
        }
    }
    else {
        binaryExpressionNode->addChildNode(move(LHSChild));

    }
    return binaryExpressionNode;
}
```

第四部分　　出错处理出口

由第三部分的表格中标红的内容，在进行 LL（1）分析时可以识别表达式之外的一些错误，并返回错误位置。

1. VariableDefine

type	**E1**	**Error：变量定义过程中可能产生的分隔符错误与变量名错误，分别报错。**
,	**E4**	
Id	**E10**	
Expre’s’sion	**E13**	
;	**E16**	
If	**E18**	
else	**E20**	

2.#

#	**E19**	**E20**	**E21**
Error：在不应结束的位置结束，即读入错误的终结符号。			

3. VariableDeclaration

Id	2	**E7**	**Error：变量声明变量名字为不合**
Expression	2	**E9**	

;	2	**E12**	**法符号，变量表达式不合法，都会导致错误**
If	2	**E15**	
else	2	**E17**	

4.Statement

type	**E2**	**Error：语句错误 Type 和，不能独立在语句中，则报错**
,	**E5**	

5.IfControl

type	**E3**	**Error：if 控制逻辑错误。If 的控制中错误使用了不合语法的符号**
,	**E6**	
Id	**E8**	
Expression	**E11**	
;	**E14**	
#	**E21**	
While	**E30**	

6.WhileControl

type	**E22**	**Error:while 控制逻辑错误。同 if**
,	**E23**	
Id	**E24**	
Expression	**E25**	
;	**E26**	
If	**E27**	
else	**E28**	
#	**E29**	

第五部分　　测试计划

部分测试计划如下：

```
//测试
int a,b,c;//多个变量声明
int data=10;
int dataout=0;//变量声明初始化
a=1; b=data;c=0;//变量赋值
if (a*b+c<=b&&a+c*b==1){dataout=1;}//if及运算符优先级测试
while(b<=10)//while测试
{
    dataout++;
    b--;
}#
;//单个分号测试
#
//空程序测试
#
```