

**院 系：计 算 机 学 院**

**实验课程：编译原理**

**实验项目：TINY扩充语言的语法分析**

**指导老师：黄煜廉**

**专 业：计算机科学与技术**

**班 级：2016级 4 班**

**学 生：陈铭海**

**学 号：20162180198**

**华南师范大学教务处**

**TINY扩充语言的语法分析**

**一．实验目的**

设计一个应用软件，可以对扩充的TINY语言进行语法分析，并可以生成语法树

1. **实验内容**

扩充的语法规则有：实现 while、do while、for语句、大于>比较运算符号以及求余计算式子，具体文法规则自行构造。

(1) While-stmt --> while exp do stmt-sequence endwhile

(2) Dowhile-stmt-->do stmt-sequence while(exp);

(3) for-stmt-->for identifier:=simple-exp to simple-exp do stmt-sequence enddo 步长递增1

(4) for-stmt-->for identifier:=simple-exp downto simple-exp do stmt-sequence endd步长递减1

(5) 大于>比较运算符号以及求余计算式子的文法规则请自行组织。

(6) 把TINY语言原有的if语句书写格式：

if\_stmt-->if exp then stmt-sequence end | if exp then stmt-sequence else stmt-sequence en 改写为：if\_stmt-->if(exp) stmt-sequence | if(exp) stmt-sequence else stmt-sequence

**三．设计思路**

1. 根据TINY语言的语法规则，写出文法规则

(1). 大于>比较运算符号以及求余计算式子的文法规则如下：

exp-->simple-exp[comparison-op simple-exp]

comparision--> < | > | =

simple-exp-->term{addop term}

addop--> + | -

term-->factor{mulop factor}

mulop--> \* | / | %

factor-->(exp) | number | identfier

2. 再使用老师课堂所教的递归下降法，根据文法规则进行编写。非终结符号递归调用， 终结符号匹配。

**四．程序源代码**

本程序参照书本附录B的代码进行编写，其中的工具函数直接使用书中的代码，程序的主要修改是parse.cpp源代码如下：（请看文件中的parse.h和 parse.cpp）

1. **parse.cpp**

static TreeNode \* stmt\_sequence(void);

static TreeNode \* statement(void);

static TreeNode \* if\_stmt(void);

static TreeNode \* repeat\_stmt(void);

static TreeNode \* assign\_stmt(void);

static TreeNode \* read\_stmt(void);

static TreeNode \* write\_stmt(void);

static TreeNode \* exp(void);

static TreeNode \* simple\_exp(void);

static TreeNode \* term(void);

static TreeNode \* factor(void);

static TreeNode \* While\_stmt(void);

static TreeNode \* Dowhile\_stmt(void);

static TreeNode \* for\_stmt(void);

static void syntaxError(char \* message)

{

fprintf(listing, "\n>>> ");

fprintf(listing, "Syntax error at line %d: %s", lineno, message);

Error = TRUE;

}

static void match(TokenTypeInProjece expected)

{

if (token == expected) token = getToken();

else {

syntaxError("unexpected token -> ");

printToken(token, tokenString);

fprintf(listing, " ");

}

}

TreeNode \* stmt\_sequence(void)

{

TreeNode \* t = statement();

TreeNode \* p = t;

while ((token != ENDFILE) && (token != END) &&

(token != ELSE) && (token != UNTIL) &&

(token != WHILE) &&

(token != ENDWHILE) && (token != ENDDO))

{

TreeNode \* q;

match(SEMI);

q = statement();

if (q != NULL) {

if (t == NULL) t = p = q;

else /\* now p cannot be NULL either \*/

{

p->sibling = q;

p = q;

}

}

}

return t;

}

TreeNode \* statement(void)

{

TreeNode \* t = NULL;

switch (token) {

case IF: t = if\_stmt(); break;

case REPEAT: t = repeat\_stmt(); break;

case ID: t = assign\_stmt(); break;

case READ: t = read\_stmt(); break;

case WRITE: t = write\_stmt(); break;

case WHILE: t = While\_stmt(); break;

case DO: t = Dowhile\_stmt(); break;

case FOR: t = for\_stmt(); break;

default: syntaxError("unexpected token -> ");

printToken(token, tokenString);

token = getToken();

break;

} /\* end case \*/

return t;

}

TreeNode \* if\_stmt(void)

{

TreeNode \* t = newStmtNode(IfK);

match(IF);

if (t != NULL)

{

match(LPAREN);

t->child[0] = exp();

match(RPAREN);

}

if (t != NULL) t->child[1] = stmt\_sequence();

if (token == ELSE) {

match(ELSE);

if (t != NULL) t->child[2] = stmt\_sequence();

}

return t;

}

TreeNode \* repeat\_stmt(void)

{

TreeNode \* t = newStmtNode(RepeatK);

match(REPEAT);

if (t != NULL) t->child[0] = stmt\_sequence();

match(UNTIL);

if (t != NULL) t->child[1] = exp();

return t;

}

TreeNode \* assign\_stmt(void)

{

TreeNode \* t = newStmtNode(AssignK);

if ((t != NULL) && (token == ID))

t->attr.name = copyString(tokenString);

match(ID);

match(ASSIGN);

if (t != NULL) t->child[0] = exp();

return t;

}

TreeNode \* read\_stmt(void)

{

TreeNode \* t = newStmtNode(ReadK);

match(READ);

if ((t != NULL) && (token == ID))

t->attr.name = copyString(tokenString);

match(ID);

return t;

}

TreeNode \* write\_stmt(void)

{

TreeNode \* t = newStmtNode(WriteK);

match(WRITE);

if (t != NULL) t->child[0] = exp();

return t;

}

TreeNode \* exp(void)

{

TreeNode \* t = simple\_exp();

if ((token == LT) || (token == EQ) || (token == GT)) {//GT是大于号

TreeNode \* p = newExpNode(OpK);

if (p != NULL) {

p->child[0] = t;

p->attr.op = token;

t = p;

}

match(token);

if (t != NULL)

t->child[1] = simple\_exp();

}

return t;

}

TreeNode \* simple\_exp(void)

{

TreeNode \* t = term();

while ((token == PLUS) || (token == MINUS))

{

TreeNode \* p = newExpNode(OpK);

if (p != NULL) {

p->child[0] = t;

p->attr.op = token;

t = p;

match(token);

t->child[1] = term();

}

}

return t;

}

TreeNode \* term(void)

{

TreeNode \* t = factor();

while ((token == TIMES) || (token == OVER)|| (token == REMAINDER))//REMAINDER是求余

{

TreeNode \* p = newExpNode(OpK);

if (p != NULL) {

p->child[0] = t;

p->attr.op = token;

t = p;

match(token);

p->child[1] = factor();

}

}

return t;

}

TreeNode \* factor(void)

{

TreeNode \* t = NULL;

switch (token) {

case NUM:

t = newExpNode(ConstK);

if ((t != NULL) && (token == NUM))

t->attr.val = atoi(tokenString);

match(NUM);

break;

case ID:

t = newExpNode(IdK);

if ((t != NULL) && (token == ID))

t->attr.name = copyString(tokenString);

match(ID);

break;

case LPAREN:

match(LPAREN);

t = exp();

match(RPAREN);

break;

default:

syntaxError("unexpected token -> ");

printToken(token, tokenString);

token = getToken();

break;

}

return t;

}

//扩充While文法

TreeNode \* While\_stmt(void)

{

TreeNode \* t = newStmtNode(WhileK);

match(WHILE);

if (t != NULL) t->child[0] = exp();

match(DO);

if (t != NULL) t->child[1] = stmt\_sequence();

match(ENDWHILE);

return t;

}

//扩充Dowhile文法

TreeNode \* Dowhile\_stmt(void)

{

TreeNode \* t = newStmtNode(DoWhileK);

match(DO);

if (t != NULL) t->child[0] = stmt\_sequence();

match(WHILE);

match(LPAREN);

if (t != NULL) t->child[1] = exp();

match(RPAREN);

return t;

}

//扩充for文法

TreeNode \* for\_stmt(void)

{

TreeNode \* t = newStmtNode(ForK);

match(FOR);

if (t != NULL) t->child[0] = assign\_stmt();

if (token == TO) {

match(TO);

if (t != NULL) t->child[1] = simple\_exp();

}

else {

match(DOWNTO);

if (t != NULL) t->child[1] = simple\_exp();

}

match(DO);

if (t != NULL) t->child[2] = stmt\_sequence();

match(ENDDO);

return t;

}

TreeNode \* parse(void)

{

TreeNode \* t;

token = getToken();

t = stmt\_sequence();

if (token != ENDFILE)

syntaxError("Code ends before file\n");

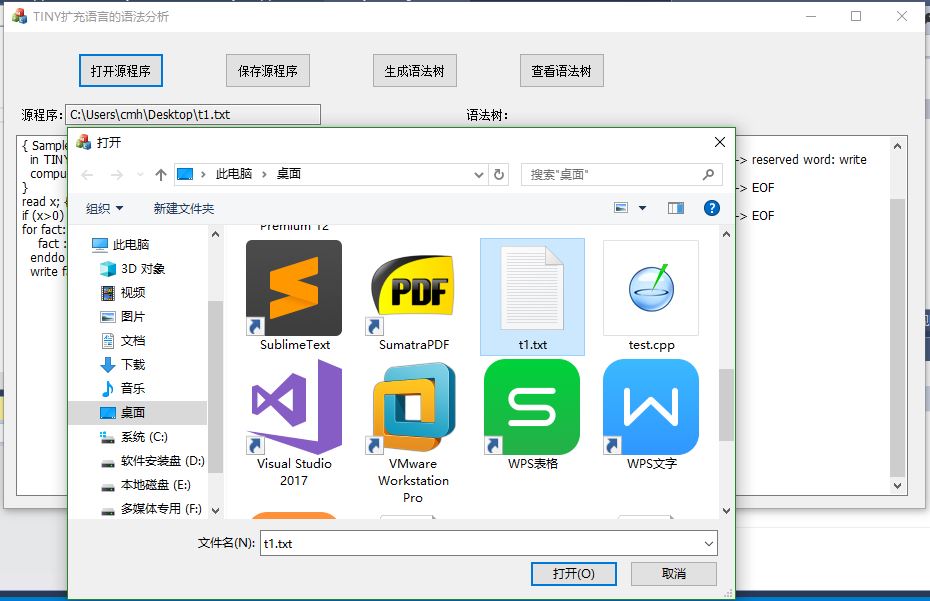
return t;

}

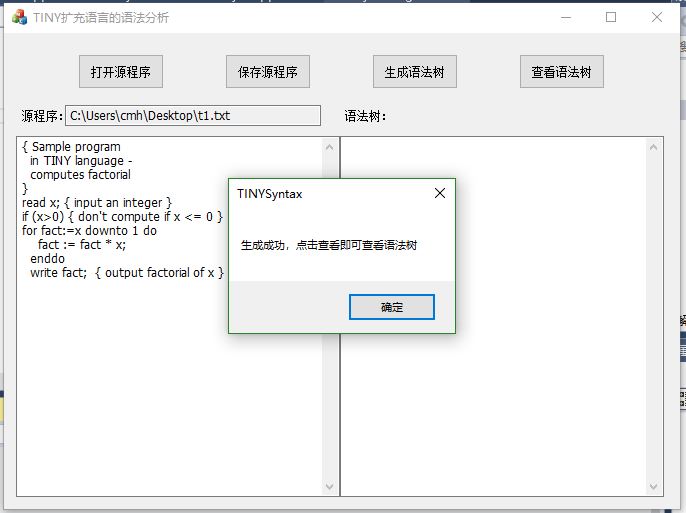
**五．实验结果**

运行程序，截图如下：

1. 打开源程序



1. 生成和查看语法树





**六．实验小结**

本次实验通过对编译原理中TINY语法分析的实现，更加深刻地了解了语法分析的相关知识，对于递归下降法的理解更进一步

经过测试，本程序对于TINY的扩充语法可以很好的进行分析，并且可以提示错误。但再进行测试时，有一些语法不太符合原来的TINY语言了，而且在对if语句的改写后会对if语句最后要带上一个“；”才不会报错。解决方法可以对if语法规则进行重新设计，或者更改程序。

1. **实验小结**

参考资料：

1. :编译原理及实践，机械工业出版社，Kenneth C.Louden著，冯博琴 冯岚等译
2. :黄煜廉老师的ppt