**电子科技大学 信息与软件工程 学院**

**标 准 实 验 报 告**

**（实验）课程名称 编译技术**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**组队排序1 学生姓名：赵茂源 学 号：2019091601001**

**组队排序2 学生姓名：唐昊哲 学 号：2019091602020**

**组队排序3 学生姓名：吴晓玉 学 号：2019091603007**

**指导教师：周尔强**

**实验地点：信软楼西303 实验时间：2021.10.26**

**一、实验室名称：信软 303**

**二、实验项目名称：递归下降语法分析**

**三、实验学时：4学时**

**四、实验内容及步骤：**

**1. 学习所提供的“表达式文法”的递归下降处理**

**理解 lex.l、rdparser.c的内容**

**在eclipse中建立工程并调试运行**

**2. 学习sysy文法**

**3. 编写SysY文法的递归下降程序**

**(1)编写不生成“语法树”的递归下降程序 rdcheck.c**

**(2)将rdcheck.c改造为生成语法树的递归下降程序rdparser.c**

**(3)改进 词法分析程序、showAst函数、main函数等，使递归下降程序rdparser最终从命令行读取要分析的程序test.c,分析后调用showAst打印该程序的结构。**

**五、实验运行结果：**

**rdcheck.c代码分析：**

**引用之前编写的词法分析器来获得语法树的节点——词法单元，通过对词法单元的分类来对应到产生式中，根据产生式的规则来递归下降不断“尝试”匹配相应的单词，并且在此过程中不构建语法树。**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include “yy.lex.h”

enum yytokentype **{**

IF **=** 245**,**

ELSE **=** 246**,**

WHILE **=** 247**,**

RETURN **=** 248**,**

PRINT **=** 249**,**

SCAN **=** 250**,**

CMP **=** 251**,**

ASSIGN **=** 252**,**

INT **=** 253**,**

STR **=** 254**,**

VOID **=** 255**,**

STRING **=** 256**,**

ID **=** 257**,**

NUMBER **=** 258**,**

EOL **=** 259

**};**

extern int yylex**();**

extern int yylval**;**

extern char**\*** yytext**;**

extern FILE**\*** yyin**;**

int tok**;**

void advance**();**//读入下一个词法单元

int program**();**

int program\_1**();**

int external\_declaration**();**

int decl\_or\_stmt**();**

int declarator\_list**();**

int declarator\_list\_1**();**

int intstr\_list**();**

int intstr\_list\_1**();**

int initializer**();**

int declarator**();**

int parameter\_list**();**

int parameter\_list\_1**();**

int parameter**();**

int type**();**

int statement**();**

int statement\_list**();**

int statement\_list\_1**();**

int expression\_statement**();**

int expr**();**

int cmp\_expr**();**

int cmp\_expr\_1**();**

int add\_expr**();**

int add\_expr\_1**();**

int mul\_expr**();**

int mul\_expr\_1**();**

int primary\_expr**();**

int expr\_list**();**

int expr\_list\_1**();**

int id\_list**();**

int id\_list\_1**();**

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

**{**

yyin **=** fopen**(**"test.c"**,** "r"**);**

advance**();**

int r **=** program**();**

**if** **(**r **==** 1**)**

printf**(**"True!\n\n"**);**

**else**

printf**(**"Fault!\n\n"**);**

system**(**"pause"**);**

**return** 0**;**

**}**

void advance**()** **{**

tok **=** yylex**();**

printf**(**"%s\n"**,** yytext**);**

**}**

//program

// : external\_declaration

// | program external\_declaration

// ;

int program**()** **{**

**if** **(**external\_declaration**())**

**if** **(**program\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int program\_1**()** **{**

**if** **(**external\_declaration**())** **{**

**if** **(**program\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//external\_declaration

// : type declarator decl\_or\_stmt

// ;

int external\_declaration**()** **{**

**if** **(**type**())**

**if** **(**declarator**())**

**if** **(**decl\_or\_stmt**())**

**return** 1**;**

**return** 0**;**

**}**

//decl\_or\_stmt

// : '{' statement\_list '}'

// | '{' '}'

// | ',' declarator\_list ';'

// | ';'

// ;

int decl\_or\_stmt**()** **{**

**if** **(**tok **==** '{'**)** **{**

advance**();**

**if** **(**tok **==** '}'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**tok **==** ','**)** **{**

advance**();**

**if** **(**declarator\_list**())**

**if** **(**tok **==** ';'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**tok **==** ';'**)** **{**

advance**();**

**return** 1**;**

**}**

**return** 0**;**

**}**

//declarator\_list

// : declarator

// | declarator\_list ',' declarator

// ;

int declarator\_list**()** **{**

**if** **(**declarator**())**

**if** **(**declarator\_list\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int declarator\_list\_1**()** **{**

**if** **(**tok **==** ','**)** **{**

advance**();**

**if** **(**declarator**())**

**if** **(**declarator\_list\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//intstr\_list

// : initializer

// | intstr\_list ',' initializer

// ;

int intstr\_list**()** **{**

**if** **(**initializer**())**

**if** **(**intstr\_list\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int intstr\_list\_1**()** **{**

**if** **(**tok **==** ','**)** **{**

advance**();**

**if** **(**initializer**())**

**if** **(**intstr\_list\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//initializer

// : NUMBER

// | STRING

// ;

int initializer**()** **{**

**if** **(**tok **==** NUMBER**)** **{**

advance**();**

**return** 1**;**

**}**

**else** **if** **(**tok **==** STRING**)** **{**

advance**();**

**return** 1**;**

**}**

**return** 0**;**

**}**

//declarator

// : ID

// | ID '=' expr

// | ID '(' parameter\_list ')'

// | ID '(' ')'

// | ID '[' expr ']'

// | ID '[' ']'

// | ID '[' expr ']' '=' '{' intstr\_list '}'

// | ID '[' ']' '=' '{' intstr\_list '}'

// ;

int declarator**()** **{**

**if** **(**tok **==** ID**)** **{**

advance**();**

**if** **(**tok **==** '='**)** **{**

advance**();**

**if** **(**expr**())**

**return** 1**;**

**}**

**else** **if** **(**tok **==** '('**)** **{**

advance**();**

**if** **(**parameter\_list**())**

**;**

**if** **(**tok **==** ')'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**tok **==** '['**)** **{**

advance**();**

**if** **(**expr**())**

**;**

**if** **(**tok **==** ']'**)** **{**

advance**();**

**if** **(**tok **==** '='**)** **{**

advance**();**

**if** **(**tok **==** '{'**)** **{**

advance**();**

**if** **(**intstr\_list**())**

**if** **(**tok **==** '}'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**return** 0**;**

**}**

**return** 1**;**

**}**

**}**

**return** 1**;**

**}**

**return** 0**;**

**}**

//parameter\_list

// : parameter

// | parameter\_list ',' parameter

// ;

int parameter\_list**()** **{**

**if** **(**parameter**())**

**if** **(**parameter\_list\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int parameter\_list\_1**()** **{**

**if** **(**tok **==** ','**)** **{**

advance**();**

**if** **(**parameter**())**

**if** **(**parameter\_list\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//parameter

// : type ID

// ;

int parameter**()** **{**

**if** **(**type**())**

**if** **(**tok **==** ID**)** **{**

advance**();**

**return** 1**;**

**}**

**return** 0**;**

**}**

//type

// : INT

// | STR

// | VOID

// ;

int type**()** **{**

**if** **(**tok **==** INT**)** **{**

advance**();**

**return** 1**;**

**}**

**else** **if** **(**tok **==** STR**)** **{**

advance**();**

**return** 1**;**

**}**

**else** **if** **(**tok **==** VOID**)** **{**

advance**();**

**return** 1**;**

**}**

**return** 0**;**

**}**

//statement

// : type declarator\_list ';'

// | '{' statement\_list '}'

// | expr\_statement

// | IF '(' expr ')' statement

// | IF '(' expr ')' statement ELSE statement

// | WHILE '(' expr ')' statement

// | RETURN ';'

// | RETURN expr ';'

// | PRINT ';'

// | PRINT expr\_list ';'

// | SCAN id\_list ';'

// ;

int statement**()** **{**

**if** **(**type**())** **{**

**if** **(**declarator\_list**())**

**if** **(**tok **==** ';'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**tok **==** '{'**)** **{**

advance**();**

**if** **(**statement\_list**())**

**if** **(**tok **==** '}'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**expression\_statement**())**

**return** 1**;**

**else** **if** **(**tok **==** IF**)** **{**

advance**();**

**if** **(**tok **==** '('**)** **{**

advance**();**

**if** **(**expr**())**

**if** **(**tok **==** ')'**)** **{**

advance**();**

**if** **(**statement**())** **{**

**if** **(**tok **==** ELSE**)** **{**

advance**();**

**if** **(**statement**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**}**

**}**

**}**

**}**

**else** **if** **(**tok **==** WHILE**)** **{**

advance**();**

**if** **(**tok **==** '('**)** **{**

advance**();**

**if** **(**expr**())**

**if** **(**tok **==** ')'**)** **{**

advance**();**

**if** **(**statement**())**

**return** 1**;**

**}**

**}**

**}**

**else** **if** **(**tok **==** RETURN**)** **{**

advance**();**

**if** **(**expr**())**

**;**

**if** **(**tok **==** ';'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**tok **==** PRINT**)** **{**

advance**();**

**if** **(**expr\_list**())**

**;**

**if** **(**tok **==** ';'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**tok **==** SCAN**)** **{**

advance**();**

**if** **(**id\_list**())**

**if** **(**tok **==** ';'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**return** 0**;**

**}**

//statement\_list

// : statement

// | statement\_list statement

// ;

int statement\_list**()** **{**

**if** **(**statement**())**

**if** **(**statement\_list\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int statement\_list\_1**()** **{**

**if** **(**statement**())** **{**

**if** **(**statement\_list\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//expression\_statement

// : ';'

// | expr ';'

// ;

int expression\_statement**()** **{**

**if** **(**tok **==** ';'**)** **{**

advance**();**

**return** 1**;**

**}**

**else** **if** **(**expr**())**

**if** **(**tok **==** ';'**)** **{**

advance**();**

**return** 1**;**

**}**

**return** 0**;**

**}**

//expr

// : cmp\_expr

// ;

int expr**()** **{**

**if** **(**cmp\_expr**())**

**return** 1**;**

**return** 0**;**

**}**

//cmp\_expr

// : add\_expr

// | cmp\_expr CMP add\_expr

// ;

int cmp\_expr**()** **{**

**if** **(**add\_expr**())**

**if** **(**cmp\_expr\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int cmp\_expr\_1**()** **{**

**if** **(**tok **==** CMP**)** **{**

advance**();**

**if** **(**add\_expr**())**

**if** **(**cmp\_expr\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//add\_expr

// : mul\_expr

// | add\_expr '+' mul\_expr

// | add\_expr '-' mul\_expr

// ;

int add\_expr**()** **{**

**if** **(**mul\_expr**())**

**if** **(**add\_expr\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int add\_expr\_1**()** **{**

**if** **(**tok **==** '+' **||** tok **==** '-'**)** **{**

advance**();**

**if** **(**mul\_expr**())**

**if** **(**add\_expr\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//mul\_expr

// : primary\_expr

// | mul\_expr '\*' primary\_expr

// | mul\_expr '/' primary\_expr

// | mul\_expr '%' primary\_expr

// | '-' primary\_expr

// ;

int mul\_expr**()** **{**

**if** **(**tok **==** '-'**)**

advance**();**

**if** **(**primary\_expr**())**

**if** **(**mul\_expr\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int mul\_expr\_1**()** **{**

**if** **(**tok **==** '\*' **||** tok **==** '/' **||** tok **==** '%'**)** **{**

advance**();**

**if** **(**primary\_expr**())**

**if** **(**mul\_expr\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//primary\_expr

// : ID '(' expr\_list ')'

// | ID '(' ')'

// | '(' expr ')'

// | ID

// | NUMBER

// | STRING

// | ID ASSIGN expr

// | ID '=' expr

// | ID '[' expr ']'

// | ID '[' expr ']' '=' expr

// ;

int primary\_expr**()** **{**

**if** **(**tok **==** ID**)** **{**

advance**();**

**if** **(**tok **==** '('**)** **{**

advance**();**

**if** **(**expr\_list**())**

**;**

**if** **(**tok **==** ')'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**tok **==** ASSIGN**)** **{**

advance**();**

**if** **(**expr**())**

**return** 1**;**

**}**

**else** **if** **(**tok **==** '='**)** **{**

advance**();**

**if** **(**expr**())**

**return** 1**;**

**}**

**else** **if** **(**tok **==** '['**)** **{**

advance**();**

**if** **(**expr**())**

**if** **(**tok **==** ']'**)** **{**

advance**();**

**if** **(**tok **==** '='**)** **{**

advance**();**

**if** **(**expr**())**

**return** 1**;**

**else**

**return** 0**;**

**}**

**return** 1**;**

**}**

**}**

**else** **return** 1**;**

**}**

**else** **if** **(**tok **==** '('**)** **{**

advance**();**

**if** **(**expr**())**

**if** **(**tok **==** ')'**)** **{**

advance**();**

**return** 1**;**

**}**

**}**

**else** **if** **(**tok **==** NUMBER**)** **{**

advance**();**

**return** 1**;**

**}**

**else** **if** **(**tok **==** STRING**)** **{**

advance**();**

**return** 1**;**

**}**

**return** 0**;**

**}**

//expr\_list

// : expr

// | expr\_list ',' expr

// ;

int expr\_list**()** **{**

**if** **(**expr**())**

**if** **(**expr\_list\_1**())**

**return** 1**;**

**return** 0**;**

**}**

int expr\_list\_1**()** **{**

**if** **(**tok **==** ','**)** **{**

advance**();**

**if** **(**expr**())**

**if** **(**expr\_list\_1**())**

**return** 1**;**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

//id\_list

// : ID

// | id\_list ',' ID

// ;

int id\_list**()** **{**

**if** **(**tok **==** ID**)** **{**

advance**();**

**if** **(**id\_list\_1**())**

**return** 1**;**

**}**

**return** 0**;**

**}**

int id\_list\_1**()** **{**

**if** **(**tok **==** ','**)** **{**

advance**();**

**if** **(**tok **==** ID**)** **{**

advance**();**

**if** **(**id\_list\_1**())**

**return** 1**;**

**}**

**}**

**else** **return** 1**;**

**return** 0**;**

**}**

**rdparser.c代码分析：**

**在之前构建的rdcheck.c的基础上加入语法树构建的代码即可，方法就是在递归下降的过程中成功匹配了产生式，就在回溯的过程中构建语法树节点，并且连接左右儿子。**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "yy.lex.h"

enum TokenType **{**

IF **=** 245**,**

ELSE **=** 246**,**

WHILE **=** 247**,**

RETURN **=** 248**,**

PRINT **=** 249**,**

SCAN **=** 250**,**

CMP **=** 251**,**

ASSIGN **=** 252**,**

INT **=** 253**,**

STR **=** 254**,**

VOID **=** 255**,**

STRING **=** 256**,**

ID **=** 257**,**

NUMBER **=** 258**,**

EOL **=** 259

**};**

extern int yylex**();**

extern int yylval**;**

extern char**\*** yytext**;**

extern FILE**\*** yyin**;**

int tok**;**

**typedef** struct \_ast ast**;**

**typedef** struct \_ast **\***past**;**

struct \_ast **{**

char **\***nodeType**;**

char **\***content**;**

past left**;**

past right**;**

**};**

void advance**();**

past program**();**

past external\_declaration**();**

past decl\_or\_stmt**();**

past declarator\_list**();**

past intstr\_list**();**

past initializer**();**

past declarator**();**

past parameter\_list**();**

past parameter**();**

past type**();**

past statement**();**

past statement\_list**();**

past expression\_statement**();**

past expr**();**

past cmp\_expr**();**

past add\_expr**();**

past mul\_expr**();**

past primary\_expr**();**

past expr\_list**();**

past id\_list**();**

past newAstNode**(**char **\***nodeType**,** char **\***content**,** past left**,** past right**);**

void showAst**(**past node**,** int nest**);**

void freeAst**(**past node**);**

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

**{**

yyin **=** fopen**(**"test.c"**,** "r"**);**

advance**();**

past root **=** program**();**

showAst**(**root**,** 0**);**

freeAst**(**root**);**

system**(**"pause"**);**

**return** 0**;**

**}**

void advance**()** **{**

tok **=** yylex**();**

printf**(**"%s\n"**,** yytext**);**

**}**

past newAstNode**(**char **\***nodeType**,** char **\***content**,** past left**,** past right**)** **{**

past node **=** **(**past**)**malloc**(sizeof(**ast**));**

**if** **(**node **==** **NULL)** **{**

printf**(**"run out of memory.\n"**);**

exit**(**0**);**

**}**

node**->**nodeType **=** nodeType**;**

node**->**content **=** content**;**

node**->**left **=** left**;**

node**->**right **=** right**;**

**return** node**;**

**}**

void showAst**(**past node**,** int nest**)** **{**

**if** **(**node **==** **NULL)**

**return;**

int i **=** 0**;**

**for** **(**i **=** 0**;** i **<** nest**;** i**++)**

printf**(**" "**);**

printf**(**"%s: %s\n"**,** node**->**nodeType**,** node**->**content**);**

showAst**(**node**->**left**,** nest **+** 1**);**

showAst**(**node**->**right**,** nest **+** 1**);**

**}**

void freeAst**(**past node**)** **{**

**if** **(**node **==** **NULL)**

**return;**

freeAst**(**node**->**left**);**

freeAst**(**node**->**right**);**

free**(**node**);**

**}**

//program

// : external\_declaration

// | program external\_declaration

// ;

past program**()** **{**

past root **=** **NULL;**

**if** **(**root **=** external\_declaration**())**

root **=** newAstNode**(**"program"**,** ""**,** root**,** program**());**

**return** root**;**

**}**

//external\_declaration

// : type declarator decl\_or\_stmt

// ;

past external\_declaration**()** **{**

past root **=** **NULL;**

**if** **(**root **=** type**())** **{**

root **=** newAstNode**(**"external\_declaration"**,** ""**,** root**,** newAstNode**(**"typebody"**,** ""**,** **NULL,** **NULL));**

**if** **(**root**->**right**->**left **=** declarator**())**

**if** **(**root**->**right**->**right **=** decl\_or\_stmt**())**

**;**

**}**

**return** root**;**

**}**

//decl\_or\_stmt

// : '{' statement\_list '}'

// | '{' '}'

// | ',' declarator\_list ';'

// | ';'

// ;

past decl\_or\_stmt**()** **{**

past root **=** **NULL;**

**if** **(**tok **==** '{'**)** **{**

advance**();**

**if** **(**root **=** statement\_list**())**

root **=** newAstNode**(**"decl\_or\_stmt"**,** ""**,** root**,** **NULL);**

**if** **(**tok **==** '}'**)**

advance**();**

**}**

**else** **if** **(**tok **==** ','**)** **{**

advance**();**

**if** **(**root **=** declarator\_list**())**

**if** **(**tok **==** ';'**)**

advance**();**

**}**

**else** **if** **(**tok **==** ';'**)**

advance**();**

**return** root**;**

**}**

//declarator\_list

// : declarator

// | declarator\_list ',' declarator

// ;

past declarator\_list**()** **{**

past root **=** **NULL;**

**if** **(**root **=** declarator**())** **{**

root **=** newAstNode**(**"declarator\_list"**,** ""**,** root**,** **NULL);**

**if** **(**tok **==** ','**)** **{**

advance**();**

root**->**right **=** declarator\_list**();**

**}**

**}**

**return** root**;**

**}**

//intstr\_list

// : initializer

// | intstr\_list ',' initializer

// ;

past intstr\_list**()** **{**

past root **=** **NULL;**

**if** **(**root **=** initializer**())** **{**

root **=** newAstNode**(**"intstr\_list"**,** ""**,** root**,** **NULL);**

**if** **(**tok **==** ','**)** **{**

advance**();**

root**->**right **=** intstr\_list**();**

**}**

**}**

**return** root**;**

**}**

//initializer

// : NUMBER

// | STRING

// ;

past initializer**()** **{**

past root **=** **NULL;**

**if** **(**tok **==** NUMBER**)** **{**

root **=** newAstNode**(**"number"**,** strdup**(**yytext**),** **NULL,** **NULL);**

advance**();**

**}**

**else** **if** **(**tok **==** STRING**)** **{**

root **=** newAstNode**(**"string"**,** strdup**(**yytext**),** **NULL,** **NULL);**

advance**();**

**}**

**return** root**;**

**}**

//declarator

// : ID

// | ID '=' expr

// | ID '(' parameter\_list ')'

// | ID '(' ')'

// | ID '[' expr ']'

// | ID '[' ']'

// | ID '[' expr ']' '=' '{' intstr\_list '}'

// | ID '[' ']' '=' '{' intstr\_list '}'

// ;

past declarator**()** **{**

past root **=** **NULL;**

**if** **(**tok **==** ID**)** **{**

root **=** newAstNode**(**"declarator"**,** strdup**(**yytext**),** **NULL,** **NULL);**

advance**();**

**if** **(**tok **==** '='**)** **{**

advance**();**

root**->**right **=** expr**();**

**}**

**else** **if** **(**tok **==** '('**)** **{**

advance**();**

root**->**left **=** parameter\_list**();**

**if** **(**tok **==** ')'**)**

advance**();**

**}**

**else** **if** **(**tok **==** '['**)** **{**

advance**();**

root**->**left **=** expr**();**

**if** **(**tok **==** ']'**)** **{**

advance**();**

**if** **(**tok **==** '='**)** **{**

advance**();**

**if** **(**tok **==** '{'**)** **{**

advance**();**

**if** **(**root**->**right **=** intstr\_list**())** **{**

root**->**right **=** newAstNode**(**"InitListExpr"**,** ""**,** root**->**right**,** **NULL);**

**if** **(**tok **==** '}'**)**

advance**();**

**}**

**}**

**}**

**}**

**}**

**}**

**return** root**;**

**}**

//parameter\_list

// : parameter

// | parameter\_list ',' parameter

// ;

past parameter\_list**()** **{**

past root **=** **NULL;**

**if** **(**root **=** parameter**())** **{**

root **=** newAstNode**(**"parameter\_list"**,** ""**,** root**,** **NULL);**

**if** **(**tok **==** ','**)** **{**

advance**();**

root**->**right **=** parameter\_list**();**

**}**

**}**

**return** root**;**

**}**

//parameter

// : type ID

// ;

past parameter**()** **{**

past root **=** **NULL;**

**if** **(**root **=** type**())**

**if** **(**tok **==** ID**)** **{**

root **=** newAstNode**(**"parameter"**,** ""**,** root**,** newAstNode**(**"ID"**,** strdup**(**yytext**),** **NULL,** **NULL));**

advance**();**

**}**

**return** root**;**

**}**

//type

// : INT

// | STR

// | VOID

// ;

past type**()** **{**

past root **=** **NULL;**

**if** **(**tok **==** INT **||** tok **==** STR **||** tok **==** VOID**)** **{**

root **=** newAstNode**(**"type"**,** strdup**(**yytext**),** **NULL,** **NULL);**

advance**();**

**}**

**return** root**;**

**}**

//statement

// : type declarator\_list ';'

// | '{' statement\_list '}'

// | expr\_statement

// | IF '(' expr ')' statement

// | IF '(' expr ')' statement ELSE statement

// | WHILE '(' expr ')' statement

// | RETURN ';'

// | RETURN expr ';'

// | PRINT ';'

// | PRINT expr\_list ';'

// | SCAN id\_list ';'

// ;

past statement**()** **{**

past root **=** **NULL;**

**if** **(**root **=** type**())** **{**

root **=** newAstNode**(**"declarator\_statement"**,** ""**,** root**,** **NULL);**

**if** **(**root**->**right **=** declarator\_list**())**

**if** **(**tok **==** ';'**)**

advance**();**

**}**

**else** **if** **(**tok **==** '{'**)** **{**

advance**();**

**if** **(**root **=** statement\_list**())** **{**

root **=** newAstNode**(**"compound\_statement"**,** ""**,** root**,** **NULL);**

**if** **(**tok **==** '}'**)**

advance**();**

**}**

**}**

**else** **if** **(**root **=** expression\_statement**())**

**;**

**else** **if** **(**tok **==** IF**)** **{**

root **=** newAstNode**(**"if\_statement"**,** ""**,** **NULL,** newAstNode**(**"ifbody"**,** ""**,** **NULL,** **NULL));**

advance**();**

**if** **(**tok **==** '('**)** **{**

advance**();**

**if** **(**root**->**left **=** expr**())**

**if** **(**tok **==** ')'**)** **{**

advance**();**

**if** **(**root**->**right**->**left **=** statement**())**

**if** **(**tok **==** ELSE**)** **{**

advance**();**

**if** **(**root**->**right**->**right **=** statement**())**

**;**

**}**

**}**

**}**

**}**

**else** **if** **(**tok **==** WHILE**)** **{**

root **=** newAstNode**(**"while\_statement"**,** ""**,** **NULL,** **NULL);**

advance**();**

**if** **(**tok **==** '('**)** **{**

advance**();**

**if** **(**root**->**left **=** expr**())**

**if** **(**tok **==** ')'**)** **{**

advance**();**

**if** **(**root**->**right **=** statement**())**

**;**

**}**

**}**

**}**

**else** **if** **(**tok **==** RETURN**)** **{**

root **=** newAstNode**(**"return\_statement"**,** ""**,** **NULL,** **NULL);**

advance**();**

root**->**left **=** expr**();**

**if** **(**tok **==** ';'**)**

advance**();**

**}**

**else** **if** **(**tok **==** PRINT**)** **{**

root **=** newAstNode**(**"print\_statement"**,** ""**,** **NULL,** **NULL);**

advance**();**

root**->**left **=** expr\_list**();**

**if** **(**tok **==** ';'**)**

advance**();**

**}**

**else** **if** **(**tok **==** SCAN**)** **{**

root **=** newAstNode**(**"scan\_statement"**,** ""**,** **NULL,** **NULL);**

advance**();**

**if** **(**root**->**left **=** id\_list**())**

**if** **(**tok **==** ';'**)**

advance**();**

**}**

**return** root**;**

**}**

//statement\_list

// : statement

// | statement\_list statement

// ;

past statement\_list**()** **{**

past right **=** statement**();**

**if** **(**right **==** **NULL)**

**return** **NULL;**

**else**

**return** newAstNode**(**"statement\_list"**,** ""**,** statement\_list**(),** right**);**

**}**

//expression\_statement

// : ';'

// | expr ';'

// ;

past expression\_statement**()** **{**

past root **=** **NULL;**

**if** **(**tok **==** ';'**)**

advance**();**

**else** **if** **(**root **=** expr**())** **{**

**if** **(**tok **==** ';'**)**

advance**();**

**}**

**return** root**;**

**}**

//expr

// : cmp\_expr

// ;

past expr**()** **{**

**return** cmp\_expr**();**

**}**

//cmp\_expr

// : add\_expr

// | cmp\_expr CMP add\_expr

// ;

past cmp\_expr**()** **{**

past root **=** **NULL;**

root **=** add\_expr**();**

**if** **(**root**)**

**while** **(**tok**==**CMP**)** **{**

root **=** newAstNode**(**"cmp\_expr"**,** strdup**(**yytext**),** root**,** **NULL);**

advance**();**

root**->**right **=** add\_expr**();**

**}**

**return** root**;**

**}**

//add\_expr

// : mul\_expr

// | add\_expr '+' mul\_expr

// | add\_expr '-' mul\_expr

// ;

past add\_expr**()** **{**

past root **=** **NULL;**

root **=** mul\_expr**();**

**if** **(**root**)**

**while** **(**tok **==** '+' **||** tok **==** '-'**)** **{**

root **=** newAstNode**(**"add\_expr"**,** strdup**(**yytext**),** root**,** **NULL);**

advance**();**

root**->**right **=** mul\_expr**();**

**}**

**return** root**;**

**}**

//mul\_expr

// : primary\_expr

// | mul\_expr '\*' primary\_expr

// | mul\_expr '/' primary\_expr

// | mul\_expr '%' primary\_expr

// | '-' primary\_expr

// ;

past mul\_expr**()** **{**

past root **=** **NULL;**

root **=** primary\_expr**();**

**if** **(**root**)** **{**

**while** **(**tok **==** '\*' **||** tok **==** '/' **||** tok **==** '%'**)** **{**

root **=** newAstNode**(**"mul\_expr"**,** strdup**(**yytext**),** root**,** **NULL);**

advance**();**

root**->**right **=** primary\_expr**();**

**}**

**}**

**else** **if** **(**tok **==** '-'**)** **{**

root **=** newAstNode**(**"mul\_expr"**,** strdup**(**yytext**),** **NULL,** **NULL);**

advance**();**

root**->**right **=** primary\_expr**();**

**}**

**return** root**;**

**}**

//primary\_expr

// : ID '(' expr\_list ')'

// | ID '(' ')'

// | '(' expr ')'

// | ID

// | NUMBER

// | STRING

// | ID ASSIGN expr

// | ID '=' expr

// | ID '[' expr ']'

// | ID '[' expr ']' '=' expr

// ;

past primary\_expr**()** **{**

past root **=** **NULL;**

**if** **(**tok **==** ID**)** **{**

root **=** newAstNode**(**"primary\_expr"**,** strdup**(**yytext**),** **NULL,** **NULL);**

advance**();**

**if** **(**tok **==** '('**)** **{**

advance**();**

root**->**left **=** expr\_list**();**

**if** **(**tok **==** ')'**)**

advance**();**

**}**

**else** **if** **(**tok **==** '['**)** **{**

advance**();**

**if** **(**root**->**left **=** expr**())**

**if** **(**tok **==** ']'**)** **{**

advance**();**

**if** **(**tok **==** '='**)** **{**

root **=** newAstNode**(**"Operator"**,** strdup**(**yytext**),** root**,** **NULL);**

advance**();**

root**->**right **=** expr**();**

**}**

**}**

**}**

**else** **if** **(**tok **==** '=' **||** tok **==** ASSIGN**)** **{**

root **=** newAstNode**(**"primary\_expr"**,** strdup**(**yytext**),** root**,** **NULL);**

advance**();**

root**->**right **=** expr**();**

**}**

**}**

**else** **if** **(**tok **==** '('**)** **{**

advance**();**

**if** **(**root **=** expr**())**

**if** **(**tok **==** ')'**)**

advance**();**

**}**

**else** **if** **(**tok **==** NUMBER**)** **{**

root **=** newAstNode**(**"number"**,** strdup**(**yytext**),** **NULL,** **NULL);**

advance**();**

**}**

**else** **if** **(**tok **==** STRING**)** **{**

root **=** newAstNode**(**"string"**,** strdup**(**yytext**),** **NULL,** **NULL);**

advance**();**

**}**

**return** root**;**

**}**

//expr\_list

// : expr

// | expr\_list ',' expr

// ;

past expr\_list**()** **{**

past root **=** **NULL;**

**if** **(**root **=** expr**())** **{**

root **=** newAstNode**(**"expr\_list"**,** ""**,** root**,** **NULL);**

**if** **(**tok **==** ','**)** **{**

advance**();**

root**->**right **=** expr\_list**();**

**}**

**}**

**return** root**;**

**}**

//id\_list

// : ID

// | id\_list ',' ID

// ;

past id\_list**()** **{**

past root **=** **NULL;**

**if** **(**tok **==** ID**)** **{**

root **=** newAstNode**(**"id\_list"**,** **NULL,** newAstNode**(**"ID"**,** strdup**(**yytext**),** **NULL,** **NULL),** **NULL);**

advance**();**

**if** **(**tok **==** ','**)** **{**

advance**();**

root**->**right **=** id\_list**();**

**}**

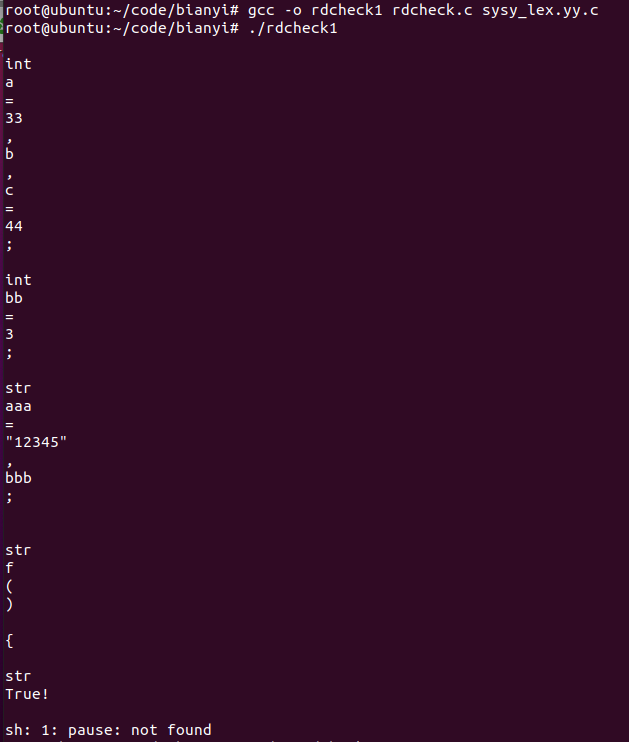
**}**

**return** root**;**

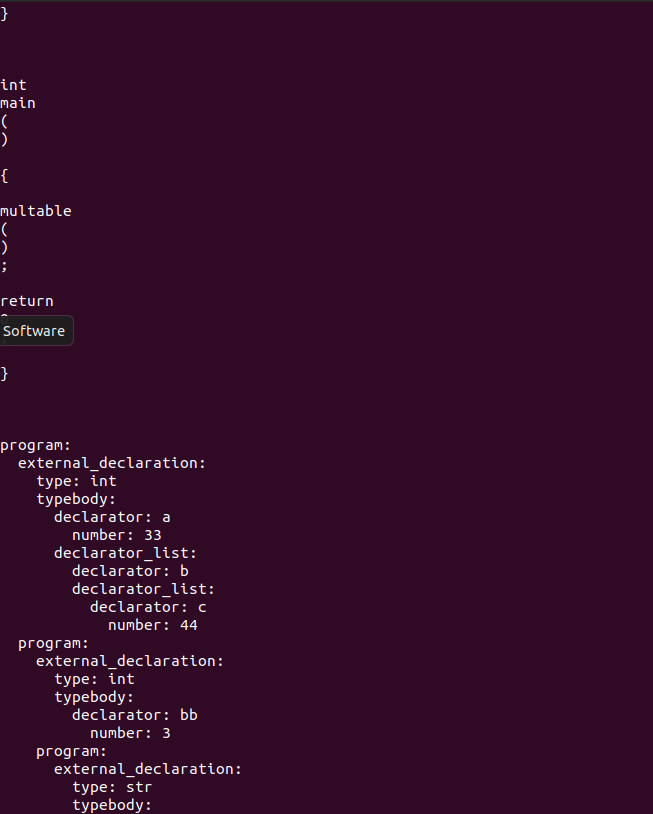
**}**

**实验运行结果：**

**rdcheck.c运行结果：**



**rdparser.c运行结果：**



**六、实验结论与总结：**

通过本次递归下降语法分析实验，我们学习编写了如何利用词法分析器以及产生式规则来判断给定程序的文法是否合法，学会了在递归回溯的过程中构建语法分析树。

**报告评分：**

**指导教师签字：**