**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**

**学生姓名：袁佳琪 学 号：2021090909029**

**指导教师：周尔强**

**实验地点：信软楼304 实验时间：2022.11.4**

**一、实验室名称：信软楼304**

**二、实验项目名称：递归下降语法分析**

**三、实验学时：4学时**

**四、实验目的、原理、内容及步骤：**

**目的：**通过本实验加深对编译技术中重点算法和编译技术的理解，提高学生的编程能力培养好的程序设计风格。了解和掌握递归下降分析法的基本原理，根据给出的文法能够完成递归下降程序的实现。

**原理：**在不含左递归的文法G中，如果对每一个非终结符的候选式没有公共左因子，则可以构造一个自上而下的分析程序，这个分析程序由一组递归过程组成，每个过程对应文法的一个非终结符，这样的分析程序称为递归下降分析程序。

递归下降分析器编译思想是简单的，从识别符号开始，在语法规则支配下进行语法分析，它逐个扫视源程序中的所有字符，根据文法和当前输入字符预测到下一个语法成份U时，便确定U为目标，并调用分析和识别U的子程序，在分析U的过程中，又有可能确立其它(或自身)子目标并调用相应子程序，如此继续下去。

**内容：**

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 vscode/Clion 中建立工程并调试运行

2、学习所提供的文法

与词法分析所提供的文法作比较

3、编写 rdgram 所提供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。

**实验步骤：**

1.由实验一词法分析程序改进sysy\_lex.l，使可以识别多行注释、十六进制int型float型以及指数形式常数。部分代码如下：

|  |
| --- |
| [0-9]+ { return num\_INT; }  [0-9]+[eE][+-]?[0-9]+ {return num\_FLOAT;}  [0-9]\*"."[0-9]\*([eE][+-]?[0-9]+)? { return num\_FLOAT;}  [\_A-Za-z]+[\_a-zA-Z0-9]\* {return Y\_ID;}  "0"+[xX0-9A-Fa-f]+ {return num\_INT;}  "0"[xX](([0-9A-Fa-f]\*"."[0-9A-Fa-f]\*)|[0-9a-fA-F]\*)[Pp][+-]?[0-9]+ {return num\_FLOAT;}  "/\*"((("\*"[^/])?|[^\*])\*\n?)\*("\*/") {} |

2.非终结符 Block 和 ArraySubscripts 对应的递归下降函数已实现，可直接调用，函数的声明如下：

past rd\_block();

past rd\_array\_subscripts();

3.设定词法分析的结构：

// 单词类别

enum yytokentype {

num\_INT = 258,

num\_FLOAT = 259,

Y\_ID = 260,

...

};

typedef struct \_TokenType{

enum yytokentype token; // 单词类别

union {

int ivalue;

float fvalue;

char\* svalue;

}attr; // 单词属性

}TokenType;

4.相关变量及回溯函数设定

extern TokenType cur\_token;//全局变量，表示当前的单词

TokenType advance(); //读取下一个单词，将其保存于 cur\_token 中，同时返回该单词

// 下面两个函数用于可能的回溯处理，即如果有多个分支需要尝试，在尝试语法分析前，可调用get\_cur\_tok\_index获取当前单词的位置，

// 之后在分析时可能会调用若干次advance函数，如果分析失败，则可通过set\_cur\_tok\_index函数重置当前单词的位置。

int get\_cur\_tok\_index(); //所有单词已经读入单词数组中，该函数获取当前单词在数组中的下标

void set\_cur\_tok\_index(int ind); //将 cur\_token 设置为下标指定的token值

5.抽象语法树的设定如下：

typedef struct \_ast ast;

typedef struct \_ast \*past;

struct \_ast{

int ivalue;

float fvalue;

char\* svalue;

node\_type nodeType;

past left;

past right;

past if\_cond;

past next;

};

past newAstNode();// 创建一个结点

past newID(char\* value); //创建一个结点，并将结点中的 svalue 设置为 value

past newInt(int value); //创建一个结点，并将结点中的 ivalue 设置为 value

6. 根据实验所给文法，完成非终结符 CallParams、RelExp、Stmt对应的递归下降函数并添加递归需要用到的其他函数。我完成的代码如下：

past rd\_addexp();

past rd\_eqexp();

past rd\_landexp();

past rd\_lorexp();

past rd\_mulexp();

past rd\_primaryexp();

past rd\_lval();

past rd\_exp();

past rd\_unary();

past rd\_addexp();

past rd\_eqexp();

past rd\_landexp();

past rd\_lorexp();

past rd\_mulexp();

past rd\_primaryexp();

past rd\_lval();

past rd\_exp();

past rd\_unary();

past newExpr(int oper, past left, past right);

past newExpr(int oper, past left, past right){

  past res = newAstNode();

  res->left = left;

  res->right = right;

  res->ivalue = oper;

  return res;

}

//75 CallParams: Exp

//76           | Exp Y\_COMMA CallParams

past rd\_call\_paras() {

  //printf("rd\_call\_paras\n");

  past res = rd\_exp();

  if(res == NULL){return NULL;}

   int oper = cur\_token.token;

  while(oper==Y\_COMMA){

    advance();

    res->next = rd\_call\_paras();

    oper = cur\_token.token;

   }

   return res;

}

/\*

   84 RelExp: AddExp

   85       | AddExp Y\_LESS RelExp

   86       | AddExp Y\_GREAT RelExp

   87       | AddExp Y\_LESSEQ RelExp

   88       | AddExp Y\_GREATEQ RelExp

\*/

past rd\_relexp(){

  // 代码实现

    //printf("rd\_relexp\n");

    past l = rd\_addexp();

     if(l ==NULL) {return NULL;}

    int oper = cur\_token.token;

   while(oper == Y\_LESS || oper==Y\_GREAT || oper == Y\_LESSEQ || oper == Y\_GREATEQ){

      advance();

      past r = rd\_addexp();

      l = newExpr(oper,l,r);

      oper = cur\_token.token;

      }

      return l;

}

/\*

   81 AddExp: MulExp

   82       | AddExp Y\_ADD MulExp

   83       | AddExp Y\_SUB MulExp

\*/

past rd\_addexp(){

     //printf("rd\_addexp\n");

     past l =rd\_mulexp();

     if(l==NULL){return NULL;}

     int oper = cur\_token.token;

     while(oper==Y\_ADD||oper==Y\_SUB){

      advance();

      past r = rd\_mulexp();

      l = newExpr(oper,l,r);

      l->nodeType = BINARY\_OPERATOR;

      oper = cur\_token.token;

     }

     return l;

}

/\*

   89 EqExp: RelExp

   90      | RelExp Y\_EQ EqExp

   91      | RelExp Y\_NOTEQ EqExp

\*/

past rd\_eqexp(){

    //printf("rd\_eqexp\n");

    past res = newAstNode();

    past l = NULL;

     l = rd\_relexp();

    if(l == NULL){return NULL;}

    int cur = get\_cur\_tok\_index();

    int oper = cur\_token.token;

    advance();

    if(oper==Y\_EQ){

      past r = rd\_eqexp();

      res->svalue = "=";

      res->left = l;

      res->right = r;

      res->nodeType = BINARY\_OPERATOR;

    }else if(oper==Y\_NOTEQ){

      past r = rd\_eqexp();

      res->svalue = "!=";

      res->left = l;

      res->right = r;

      res->nodeType = BINARY\_OPERATOR;

    }else{

       set\_cur\_tok\_index(cur);

       res = l ;

    }

    return res;

}

/\*

   92 LAndExp: EqExp

   93        | EqExp Y\_AND LAndExp

\*/

past rd\_landexp(){

    //printf("rd\_landexp\n");

    past res = newAstNode();

    past l =NULL;

    l = rd\_eqexp();

    if (l==NULL){return NULL;}

    l->nodeType = BINARY\_OPERATOR;

    int cur = get\_cur\_tok\_index();

    int oper = cur\_token.token;

    advance();

    if(oper==Y\_AND){

      past r = rd\_landexp();

      res->svalue = "&&";

      res->left = l;

      res->right = r;

      res->nodeType = BINARY\_OPERATOR;

    }else{

    set\_cur\_tok\_index(cur);

    res = l ;

    }

    return res;

}

/\*

   94 LOrExp: LAndExp

   95       | LAndExp Y\_OR LOrExp

\*/

past rd\_lorexp(){

    //printf("rd\_lorexp\n");

    past res = newAstNode();

    past l = NULL;

    l = rd\_landexp();

    if(l==NULL){return NULL;}

    l->nodeType = BINARY\_OPERATOR;

    int oper = cur\_token.token;

    if(oper==Y\_OR){

    advance();

    past r = rd\_lorexp();

    res->ivalue=Y\_OR;

    res->left = l;

    res->right = r;

    res->nodeType = BINARY\_OPERATOR;

    }else{

    res = l;

    }

    return res;

}

/\*

   77 MulExp: UnaryExp

   78       | MulExp Y\_MUL UnaryExp

   79       | MulExp Y\_DIV UnaryExp

   80       | MulExp Y\_MODULO UnaryExp

\*/

past rd\_mulexp(){

    //printf("rd\_mulexp\n");

    past res = newAstNode();

    past l = rd\_unary();

    if(l==NULL)

    { return NULL;}

    int oper = cur\_token.token;

    if(oper==Y\_MUL||oper==Y\_DIV||oper==Y\_MODULO){

        advance();

        past r = rd\_unary();

        res->ivalue = oper;

        res->left = l;

        res->right = r;

        res->nodeType = BINARY\_OPERATOR;

        return res;

    }else{ return l;}

}

/\*

   65 PrimaryExp: Y\_LPAR Exp Y\_RPAR

   66           | LVal

   67           | num\_INT

   68           | num\_FLOAT

\*/

past rd\_primaryexp(){

    //printf("rd\_primaryexp\n");

    past res = NULL;

    int oper = cur\_token.token;

    if(oper==Y\_LPAR){

      advance();

      res = rd\_exp();

      advance();

    }else if(oper == num\_INT){

       res = newAstNode();

       res->ivalue = cur\_token.attr.ivalue;

       res->nodeType = INTEGER\_LITERAL;

       advance();

    }else if(oper == num\_FLOAT){

       res = newAstNode();

       res->fvalue = cur\_token.attr.fvalue;

       res->nodeType = FLOATING\_LITERAL;

       advance();

    }else{

       res = rd\_lval();

    }

    return res;

}

/\*

   61 LVal: Y\_ID

   62     | Y\_ID ArraySubscripts

\*/

past rd\_lval(){

    //printf("rd\_lval\n");

    past res = newAstNode();

    int oper = cur\_token.token;

    if(oper!=Y\_ID){return NULL;}

    res = newID(cur\_token.attr.svalue);

    res->nodeType = DECL\_REF\_EXPR;

    advance();

    int cur = get\_cur\_tok\_index();

    past n = rd\_array\_subscripts();

        if(n!=NULL){

        res->next = n;

    }else{set\_cur\_tok\_index(cur);}

    return res;

}

past rd\_exp(){

 //printf("rd\_exp\n");

past res =  rd\_addexp();

if(res->right!=NULL){

  res->nodeType = BINARY\_OPERATOR;

}else{res->nodeType = DECL\_REF\_EXPR;}

return res;

}

/\*

   69 UnaryExp: PrimaryExp

   70         | Y\_ID Y\_LPAR Y\_RPAR

   71         | Y\_ID Y\_LPAR CallParams Y\_RPAR

   72         | Y\_ADD UnaryExp

   73         | Y\_SUB UnaryExp

   74         | Y\_NOT UnaryExp

\*/

past rd\_unary(){

    //printf("rd\_unary\n");

    past res = newAstNode();

    int oper = cur\_token.token;

    int cur = get\_cur\_tok\_index();

    past l = rd\_primaryexp();

    if(l!=NULL){

      return l;

    }else{

      set\_cur\_tok\_index(cur);

    }

    if(oper==Y\_ID){

        char\* IDinfo = cur\_token.attr.svalue;

        advance();

        advance();

        int oper2 = cur\_token.token;

        if(oper2==Y\_RPAR){

        res = newID(IDinfo);

        advance();

        }else{

        res = newID(IDinfo);

        past n = rd\_call\_paras();

        res->next = n;

        advance();

        }

    }else if(oper==Y\_AND||oper==Y\_SUB||oper==Y\_NOT){

        res = newAstNode();

        res->ivalue=oper;

        advance();

        past r = rd\_unary();

        res->right = r;

    }else{return NULL;}

    return res;

}

/\*

   49 Stmt: LVal Y\_ASSIGN Exp Y\_SEMICOLON

   50     | Y\_SEMICOLON

   51     | Exp Y\_SEMICOLON

   52     | Block

   53     | Y\_WHILE Y\_LPAR LOrExp Y\_RPAR Stmt

   54     | Y\_IF Y\_LPAR LOrExp Y\_RPAR Stmt

   55     | Y\_IF Y\_LPAR LOrExp Y\_RPAR Stmt Y\_ELSE Stmt

   56     | Y\_BREAK Y\_SEMICOLON

   57     | Y\_CONTINUE Y\_SEMICOLON

   58     | Y\_RETURN Exp Y\_SEMICOLON

   59     | Y\_RETURN Y\_SEMICOLON

\*/

past rd\_stmt(){

  //printf("rd\_stmt\n");

  past res = newAstNode();

  int cur = get\_cur\_tok\_index();

  int oper = cur\_token.token;

  if(oper==Y\_SEMICOLON){

    res->ivalue = Y\_SEMICOLON;

    advance();

  }else if (oper==Y\_WHILE){

    advance();

    advance();

    past con = rd\_lorexp();

    advance();

    past st = rd\_stmt();

    res->left = con;

    res->right= st;

    res->nodeType = WHILE\_STMT;

    //res->right->ivalue = '=';

  }else if(oper==Y\_IF){

    advance();

    advance();

    past con = rd\_lorexp();

    //con->ivalue = Y\_LESS;

    con->nodeType = BINARY\_OPERATOR;

    advance();

    past st1 = rd\_stmt();

    st1->ivalue = Y\_ASSIGN;

    if(cur\_token.token==Y\_ELSE){

      advance();

      past st2 = rd\_stmt();

      res->nodeType = IF\_STMT;

      res->if\_cond = con;

      res->left = st1;

      //con->left->nodeType = BINARY\_OPERATOR;

      res->right = st2;

    }else{

      res->nodeType = IF\_STMT;

      res->if\_cond = con;

      res->left= st1;

    }

  }else if(oper==Y\_BREAK){

    res->nodeType = BREAK\_STMT;

    advance();

    advance();

  }else if(oper==Y\_CONTINUE){

    res->nodeType = CONTINUE\_STMT;

    advance();

    advance();

  }else if(oper==Y\_RETURN){

    advance();

    if(cur\_token.token==Y\_SEMICOLON){

      res->nodeType = RETURN\_STMT;

      advance();

    }else{

      past n = rd\_exp();

      res->nodeType = RETURN\_STMT;

      res->next = n;

      advance();

    }

  }else {

    past l = rd\_lval();

    if(l!=NULL){

      advance();

      past r = rd\_exp();

      advance();

      res->nodeType = BINARY\_OPERATOR;

      res->left = l;

      res->right = r;

      res->ivalue ='=';

    }else{

        set\_cur\_tok\_index(cur);

        past l =rd\_block();

        if(l!=NULL){

          res = l;

        }else{

          set\_cur\_tok\_index(cur);

          res = rd\_exp();

          advance();

        }

    }

  }

  return res;

}

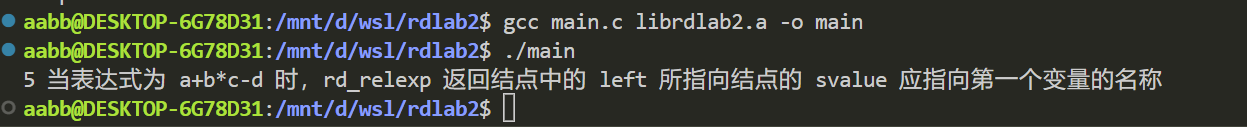
**五、实验运行结果：**

最终，我在icoding平台上通过了所有测试用例：



虽然在icoding平台已通过，但在本地测试环境中有未通过的用例：

但我认为递归下降是最左推导，不停地添加右子树，left为运算表达式的语法树a+b\*c, left的svalue为NULL。



**六、实验结论与总结：**

通过编写生成语法树的递归下降程序main.c，成功做到识别sysy语法并生成语法树。此次实验的代码量相当大，并且难以修改，通过反复测试和修改，我最终成功通过了icoding的测试用例。并且也对生成的语法树有了更清晰的认知，比如：是先生成左子树还是右子树，该在哪个位置保存节点的类型，该用什么方式保存标识符相关的属性值等等。在递归下降的分析程序中，也需要用到回溯，需要合理判断那些地方需要回溯，应该回溯到哪个位置。表达式的生成中需要判断运算符是左结合还是右结合，使用while还是If更好，If和while表达式的条件和执行语句该怎么连接，这些都是需要细心思考的问题。

经此实验我对C语言编写递归程序更加熟悉，进一步理解编译器中语法树的设定和节点的类别，学会了与词法分析器的交互，也对递归下降分析法有了更深层次的理解，而不仅仅是停留在书本知识。

**报告评分：**

**指导教师签字：**