



華南師範大學

本科学生实验（实践）报告

院 系：计 算 机 学 院

实验课程：编译原理

实验项目：TINY 扩充语言的语法树生成

指导老师：黄煜廉

开课时间：2023 ~ 2024 年度第 1 学期

专 业：计算机科学与技术

班 级：计科 1 班

学 生：李达良

学 号：20203231004

华南师范大学教务处

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

一、实验题目

TINY 扩充语言的语法树生成

二、实验内容

(一) 为 Tiny 语言扩充的语法有

1. 实现改写书写格式的新 if 语句;
2. 增加 for 循环;
3. 扩充算术表达式的运算符: += 加法赋值运算符 (类似于 C 语言的 +=)、求余%、乘方[^],
4. 扩充比较运算符: = (等于), > (大于), <= (小于等于), >= (大于等于), <> (不等于) 等运算符,
5. 增加正则表达式, 其支持的运算符有: 或(|)、连接(&)、闭包(#)、括号()、可选运算符(?) 和基本正则表达式。
6. 增加位运算表达式, 其支持的位运算符有 and(与)、or(或)、not(非), 如果对位运算不熟悉, 可以参考 C/C++ 的位运算。

(二) 对应的语法规则分别为:

1. 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 end
```

改写为:

```
if_stmt → if(exp) stmt-sequence else stmt-sequence | if(exp) stmt-  
sequence
```

2. for 语句的语法规则:

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

(2) for_stmt → for identifier:=simple-exp downto simple-exp do stmt-sequence enddo 步长递减 1

3. += 加法赋值运算符、求余%、乘方[^]等运算符的文法规则请自行组织。

4. = (等于), > (大于), <= (小于等于), >= (大于等于), <> (不等于) 等运算符的文法规则请自行组织。

5. 为 tiny 语言增加一种新的表达式——正则表达式, 其支持的运算符有: 或(|)、连接(&)、闭包(#)、括号()、可选运算符(?) 和基本正则表达式, 对应的文法规则请自行组织。

6. 为 tiny 语言增加一种新的语句, ID:=正则表达式

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成

实验时间 2023 年 11 月 21 日

实验指导老师 黄煜廉 实验评分

7. 为 tiny 语言增加一种新的表达式——位运算表达式，其支持的运算符有 and(与)、or(或)、非(not)。

8. 为 tiny 语言增加一种新的语句，ID:=位运算表达式

9. 为了实现以上的扩充或改写功能，还需要注意对原 tiny 语言的文法规则做一些相应的改造处理。

三、实验目的

(1) 要提供一个源程序编辑的界面，以让用户输入源程序（可输入，可保存、可打开源程序）

(2) 可由用户选择是否生成语法树，并可查看所生成的语法树。

(3) 实验 3 的实现只能选用的程序设计语言为：C++

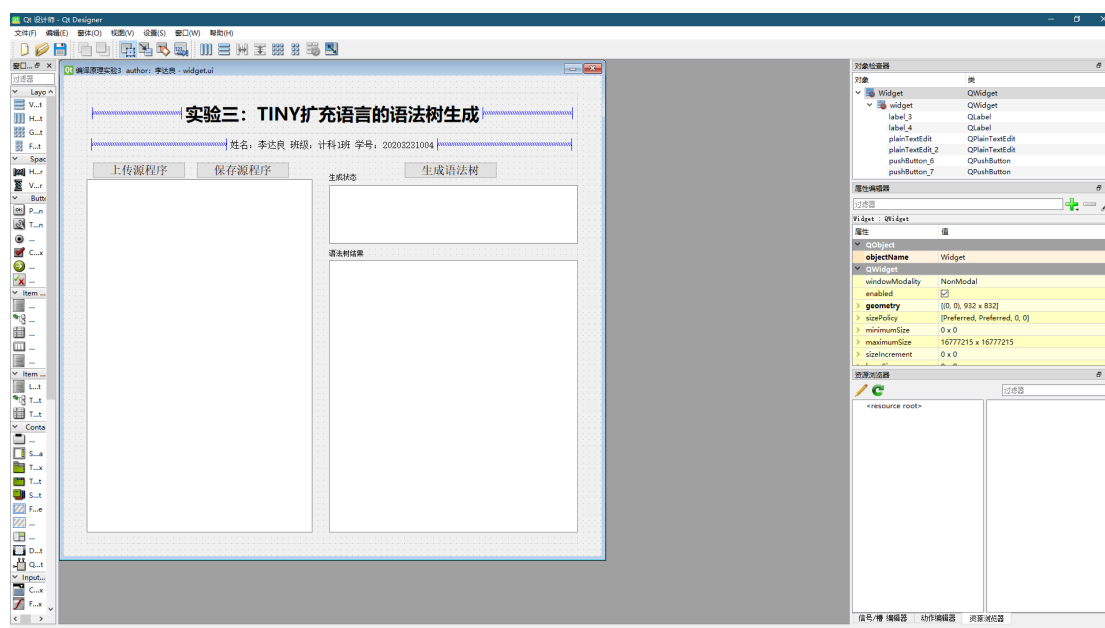
(4) 要求应用程序的操作界面应为 Windows 界面。

(5) 应该书写完善的软件文档

四、实验文档

(1) 程序界面设计

通过 QT 实现 UI 的设计，给予一个 tiny 程序输入框，输入框上方有两个按钮，分别是上传源程序和下载源程序，方便使用者的输入和保存。填写程序后，点击“生成语法树”，在“生成状态”中可以查看是否生成成功，语法错误会有提示，若生成成功，语法树将展示在右下角的方框中，操作简单便捷。



华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

(2) 程序逻辑设计

对于本程序，我们主要分为四大块：

- ①if 语句改写
- ②for 语句新增
- ③新增运算符、比较符号、位运算表达式
- ④新增正则表达式赋值

2.1 if 语句改写

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 end`

改写为：

`if_stmt-->if(exp) stmt-sequence else stmt-sequence | if(exp) stmt-sequence`

我们首先提取左公因子：

`if_stmt-->if(exp) stmt-sequence [else stmt-sequence]`

但是我们会发现一个问题，假如我们进入 if 或者 else 之后，删除掉 end 是无法跳出来 if 或 else 语句的，比如：

```
if (0<x) { don't compute if x <= 0 }
for fact := x downto 1 do
    fact := fact * x
enddo;
write fact { output factorial of x }
```

假如我们不想 write fact 语句在 if 里面，我们发现按照当前语法规则是没办法做到的。因此我们思考一下 C++ 的方法，使用 {} 的形式去解决这个问题，那我们 tiny 中，{} 是表示注释，因此我们考虑替换成 [] 来解决这个问题。对应语法形式如下：

`if_stmt-->if(exp) [stmt-sequence [else stmt-sequence]]`（红色的 [] 代表不是 EBNF 语法的 []）

因此我们可以将对应的程序进行改写：

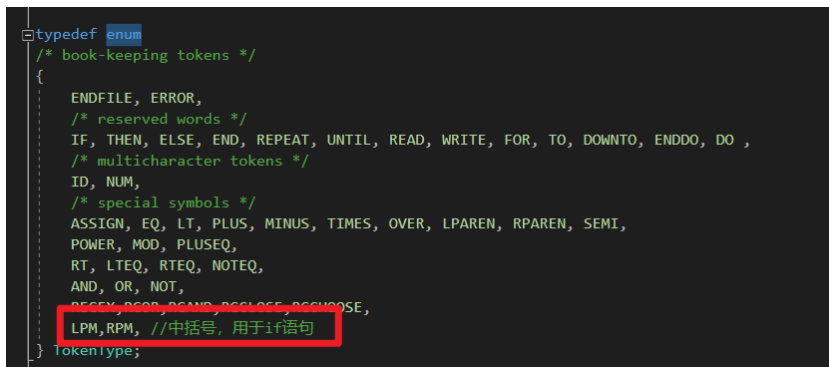
```
TreeNode* if_stmt(void)
{
    TreeNode* t = newStmtNode(IfK);
    match(IF);
    match(LPAREN);
    if (t != NULL) t->child[0] = exp();
    match(RPAREN);
    match(LPM);
    if (t != NULL) t->child[1] = stmt_sequence();
}
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
match(RPM);  
if (token == ELSE) {  
    match(ELSE);  
    match(LPM);  
    if (t != NULL) t->child[2] = stmt_sequence();  
    match(RPM);  
}  
return t;  
}
```

其中，LPM 和 RPM 对应 “[” 和 “]”，注意在 globals.h 添加对应的 enum:



```
typedef enum  
/* book-keeping tokens */  
{  
    ENDFILE, ERROR,  
    /* reserved words */  
    IF, THEN, ELSE, END, REPEAT, UNTIL, READ, WRITE, FOR, TO, DOWNT, ENDDO, DO ,  
    /* multicharacter tokens */  
    ID, NUM,  
    /* special symbols */  
    ASSIGN, EQ, LT, PLUS, MINUS, TIMES, OVER, LPAREN, RPAREN, SEMI,  
    POWER, MOD, PLUSEQ,  
    RT, LTEQ, RTEQ, NOTEQ,  
    AND, OR, NOT,  
    BEGIN, BEGIN, BEGIN, BEGIN, BEGIN,  
    LPM, RPM, //中括号, 用于if语句  
} tokentype;
```

2.2 for 语句新增

for 语句的语法规则:

- (1) for-stmt \rightarrow for identifier:=simple-exp to simple-exp do stmt-sequence enddo 步长递增 1
- (2) for-stmt \rightarrow for identifier:=simple-exp downto simple-exp do stmt-sequence enddo 步长递减 1

我们首先得添加关键字 for:

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
typedef enum
/* book-keeping tokens */
{
    ENDFILE, ERROR,
    /* reserved words */
    IF, THEN, ELSE, END, REPEAT, UNTIL, READ, WRITE, FOR, TO, DOWNTOW, ENDDO, DO,
    /* multicharacter tokens */
    ID, NUM,
    /* special symbols */
    ASSIGN, EQ, LT, PLUS, MINUS, TIMES, OVER, LPAREN, RPAREN, SEMI,
    POWER, MOD, PLUSEQ,
    RT, LTEQ, RTEQ, NOTEQ,
    AND, OR, NOT,
    REGEX, RGOR, RGAND, RGCLOSE, RGCHOOSE,
    LPM, RPM, //中括号, 用于if语句
} TokenType;
```

同时 stmt 的类型也得添加两种，一种是步长递增，一种是步长递减：

```
/****** Syntax tree for parsing *****/
typedef enum { StmtK, ExpK } NodeKind;
typedef enum { IfK, RepeatK, AssignK, ReadK, WriteK, ForToK, ForDowntoK, RegexK } StmtKind;
typedef enum { OpK, ConstK, IdK } ExpKind;

/* ExpType is used for type checking */
typedef enum { Void, Integer, Boolean } ExpType;
```

最后我们添加一个解析 for 语句的函数即可：

```
TreeNode* for_stmt(void)
{
    // 赋值节点
    TreeNode* p = newStmtNode(AssignK);
    match(FOR);
    if ((p != NULL) && (token == ID))
    {
        p->attr.name = copyString(tokenString);
    }
    match(ID);
    match(ASSIGN);
    if (p != NULL)
    {
        p->child[0] = simple_exp();
    }
    // FOR节点
    TreeNode* t = NULL;
    if (token == TO) { // 步长+1
        t = newStmtNode(ForToK);
        t->child[0] = p;
    }
}
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
        match(TO);
    }
    else if (token == DOWNT0) { // 步长-1
        t = newStmtNode(ForDowntoK);
        t->child[0] = p;
        match(DOWNT0);
    }
    else // 出错提示
    {
        syntaxError("Expecting 'to' or 'downto' after assignment in for
statement");
    }
    t->child[1] = simple_exp();
    match(D0);
    t->child[2] = stmt_sequence();
    match(ENDD0);
    return t;
}
```

2.3 新增运算符号、比较符号、位运算表达式

因为位运算、运算符号、比较符号是可以混杂在一起的，所以我们必须一起考虑这些符号的优先级来制定文法规则。

对于比较符号而言，因为原先的 tiny 文法中：

$\text{exp} \rightarrow \text{simple-exp comparison-op simple-exp} \mid \text{simple-exp}$

我们可以看到，comparison-op 其实在一句话中只能用一次，所以其实不需要考虑比较符号内部之间的优先级。那我们就可以参考 C++ 或者 python 中的优先级，知道 tiny 语言中，各符号优先级应该如下（从低到高）：

1. or
2. and
3. < > <= >= <> (各种比较符号)
4. + -
5. * / %
6. not
7. ^

根据上述优先级，我们制定以下文法规则：

$\text{exp} \rightarrow \text{exp orop orexp} \mid \text{orexp}$

$\text{orop} \rightarrow \mid$

$\text{orexp} \rightarrow \text{orexp andop andexp} \mid \text{andexp}$

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
andop -> &
andexp -> simple-exp comparison-op simple-exp | simple-exp
comparison-op -> < | > | <= | >= | <>
simple-exp -> simple-exp addop term | term
addop -> + | -
term -> term mulop notexp | notexp
mulop -> * | / | %
notexp -> notop (power | notexp) | power
notop -> ~
power -> power powop factor | factor
powop -> ^
factor -> (exp) | number | identifier
```

转换成 EBNF 后得到:

```
exp -> orexp {orop orexp}
orop -> or
orexp -> andexp {andop andexp}
andop -> and
andexp -> simple-exp [comparison-op simple-exp]
comparison-op -> < | > | <= | >= | <>
simple-exp -> term {addop term}
addop -> + | -
term -> notexp {mulop notexp}
mulop -> * | / | %
notexp -> {notop} power
notop -> not
power -> factor {powop factor}
powop -> ^
factor -> (exp) | number | identifier
```

对于花括号来说, 我们使用 while 来解析, 对于[]就是用 if 语句, 所以我们可以得到核心代码如下:

```
TreeNode* exp(void)
{
    TreeNode* t = orexp();
    while (token == OR)
    {
        TreeNode* p = newExpNode(OPK);
```


华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分 _____

```
        if (p != NULL) {
            p->child[0] = t;
            p->attr.op = token;
            t = p;
            match(token);
            t->child[1] = orexp();
        }
    }
    return t;
}

TreeNode* orexp(void)
{
    TreeNode* t = andexp();
    while (token == AND)
    {
        TreeNode* p = newExpNode(OpK);
        if (p != NULL) {
            p->child[0] = t;
            p->attr.op = token;
            t = p;
            match(token);
            t->child[1] = andexp();
        }
    }
    return t;
}

TreeNode* andexp(void)
{
    TreeNode* t = simple_exp();
    if (token == LTEQ || token == RTEQ || token == LT || token == RT || token ==
NOTEQ || token == EQ) {
        TreeNode* p = newExpNode(OpK);
        if (p != NULL) {
            p->child[0] = t;
            p->attr.op = token;
            t = p;
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
    }
    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(0pK);
        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 = notexp();
    while ((token == TIMES) || (token == OVER) || (token == MOD))
    {
        TreeNode* p = newExpNode(0pK);
        if (p != NULL) {
            p->child[0] = t;
            p->attr.op = token;
            t = p;
            match(token);
            p->child[1] = notexp();
        }
    }
}
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
    }  
    return t;  
}  
  
TreeNode* notexp(void)  
{  
    if (token == NOT)  
    {  
        match(NOT);  
        TreeNode* p = newExpNode(OpK);  
        p->attr.op = NOT;  
        if (token == NOT)  
        {  
            p->child[0] = notexp();  
        }  
        else  
        {  
            p->child[0] = power();  
        }  
        return p;  
    }  
    else  
    {  
        TreeNode* t = power();  
        return t;  
    }  
}
```

```
TreeNode* power(void)  
{  
    TreeNode* t = factor();  
    while ((token == POWER))  
    {  
        TreeNode* p = newExpNode(OpK);  
        if (p != NULL) {  
            p->child[0] = t;  
            p->attr.op = token;  
            t = p;  
        }  
    }  
}
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
        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;
}
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

2.4 新增正则表达式赋值、加法赋值(+=)

2.4.1 正则表达式赋值

由于正则表达式与算术表达式的操作不一样，正则表达式是不用计算，因此，是不允许出现类似于这样的语句算数表达式和正则表达式混杂在一起的情况。对于正则表达式而言，如果我们也把他放到 `assign-stmt -> identifier := exp` 中的 `exp` 进行处理，就显得整个代码变得复杂了起来。于是我们可以考虑：`ID::=正则表达式`，使用`::=`区分正常的赋值和正则表达式的赋值。

考虑正则表达式的优先级：

1. |
2. &（连接）
3. #（闭包） ?（可选）

因此我们可以指定以下文法规则（EBNF）：

```
regex_stmt -> andreg {orop andreg }  
orop -> |  
andreg -> topreg {andop topreg}  
andop -> &  
topreg -> reg_factor {topop}  
topop -> # | ?  
reg_factor -> (regex_stmt ) | ideifier | number
```

转换成代码如下：

```
TreeNode* regex_stmt(void)  
{  
    TreeNode* t = andreg();  
    while ((token == RGOR))  
    {  
        TreeNode* p = newExpNode(OpK);  
        if (p != NULL) {  
            p->child[0] = t;  
            p->attr.op = token;  
            t = p;  
            match(token);  
            t->child[1] = andreg();  
        }  
    }  
    return t;  
}
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成

实验时间 2023 年 11 月 21 日

实验指导老师 黄煜廉 实验评分

}

```
TreeNode* andreg(void)
```

```
{
```

```
    TreeNode* t = topreg();
```

```
    while ((token == RGAND))
```

```
    {
```

```
        TreeNode* p = newExpNode(OpK);
```

```
        if (p != NULL) {
```

```
            p->child[0] = t;
```

```
            p->attr.op = token;
```

```
            t = p;
```

```
            match(token);
```

```
            t->child[1] = topreg();
```

```
        }
```

```
    }
```

```
    return t;
```

```
}
```

```
TreeNode* topreg(void)
```

```
{
```

```
    TreeNode* t = reg_factor();
```

```
    while ((token == RGCLOSE) || (token == RGCHOOSE))
```

```
    {
```

```
        TreeNode* p = newExpNode(OpK);
```

```
        if (p != NULL) {
```

```
            p->child[0] = t;
```

```
            p->attr.op = token;
```

```
            t = p;
```

```
            match(token);
```

```
        }
```

```
    }
```

```
    return t;
```

```
}
```

```
TreeNode* reg_factor(void)
```

```
{
```

```
    TreeNode* t = NULL;
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
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 = regex_stmt();  
    match(RPAREN);  
    break;  
default:  
    syntaxError("unexpected token -> ");  
    printToken(token, tokenString);  
    token = getToken();  
    break;  
}  
return t;  
}
```

2.4.2 加法赋值 (+=)

其实只要区分:=和+=即可

```
TreeNode* assign_stmt(void)  
{  
    TreeNode* t = newStmtNode(AssignK);  
    if ((t != NULL) && (token == ID))  
        t->attr.name = copyString(tokenString);  
    match(ID);  
    if (token == ASSIGN)  
    {
```

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
        match(ASSIGN);
        if (t != NULL) t->child[0] = exp();
    }
    else if (token == PLUSEQ)
    {
        match(PLUSEQ);
        if (t != NULL) t->child[0] = exp();
    }
    else if (token == REGEX)
    {
        match(REGEX);
        if (t != NULL) t->child[0] = regex_stmt();
    }
    return t;
}
```

五、本实验 TINY 语言文法总结

program->stmt-sequence
stmt-sequence->stmt-sequence;statement | statement
statement->if-stmt | repeat-stmt | assign-stmt | read-stmt | write-stmt | plusassign-stmt | for-stmt |
regex-stmt
if-stmt->if(exp) [stmt-sequence [else stmt-sequence]] (红色的[]代表不是 EBNF 语法的[])
repeat-stmt->repeat stmt-sequence until exp
assign-stmt->identifier := exp
plusassign-stmt->identifier += exp
read-stmt->read identifier
write-stmt->write exp
for-stmt->for identifier:=simple-exp to simple-exp do stmt-sequence enddo | for
identifier:=simple-exp downto simple-exp do stmt-sequence enddo
exp -> exp orop oexp | oexp
orop -> |
orexp -> oexp andop andexp | andexp
andop -> &
andexp -> simple-exp comparison-op simple-exp | simple-exp
comparison-op -> < | > | <= | >= | <>
simple-exp -> simple-exp addop term | term

华南师范大学实验报告

学生姓名 李达良 学 号 20203231004
专 业 计算机科学与技术 年级、班级 2021 级 1 班
课程名称 编译原理 实验项目 TINY 扩充语言的语法树生成
实验时间 2023 年 11 月 21 日
实验指导老师 黄煜廉 实验评分

```
addop -> + | -  
term -> term mulop notexp | notexp  
mulop -> * | / | %  
notexp -> notop (power | notexp) | power  
notop -> ~  
power -> power powop factor | factor  
powop -> ^  
factor -> (exp) | number | identifier  
regex-stmt->identifier ::= regex_exp  
regex_exp-> regex_exp rorop andreg | andreg  
rorop -> |  
andreg -> andreg randop topreg | topreg  
randop -> &  
topreg -> topreg topop | reg_factor  
topop -> # | ?  
reg_factor -> (regex_exp) | ideifier | number
```

六、实验测试

详见《测试报告》。

七、小结

在实验过程中，我们面临了对原 Tiny 语言文法规则的修改和扩展的挑战，需要确保新的语法规则与原有规则的兼容性。说实话这个实验本身并不是太难，代码量不大，但是需要理解 tiny 原有解释器的逻辑，并且在原 tiny 语言文法上增加新的文法。通过这个过程，我深刻理解了语言设计的复杂性和灵活性。

同时，通过实现源程序编辑界面和语法树生成功能，我学到了如何设计用户友好的界面和实现复杂的数据结构。这为我今后在软件开发领域的工作提供了宝贵的经验。

总体而言，这次实验丰富了我的编程经验，加深了我对编程语言设计和软件工程的理解，是一次富有挑战和收获的实践。