目录

[一、 前言 2](#_Toc106800087)

[二、 开发过程 2](#_Toc106800088)

[（一） 设计思路 2](#_Toc106800089)

[（二） 实现过程 2](#_Toc106800090)

[三、 设计方案 5](#_Toc106800091)

[（一） 功能组成 5](#_Toc106800092)

[（二） 界面组成 5](#_Toc106800093)

[（三） 例子结果展示 5](#_Toc106800094)

[（四） 代码实现 9](#_Toc106800095)

[1.保存源程序 9](#_Toc106800096)

[2.打开源程序 10](#_Toc106800097)

[3.生成语法树 10](#_Toc106800098)

[4.查看语法树 22](#_Toc106800099)

[5.退出系统 22](#_Toc106800100)

# 前言

本次实验是实现TINY扩充语言的语法分析，用户可以通过输入正确的源代码然后点击生成语法树以及点击查看语法树进行查看。本次实验主要是在原来TINY语言的基础上扩充文法，实现更多语法的分析。

# 开发过程

## 设计思路

1. 首先完成页面布局
2. 保存，打开存有源程序的功能按钮的实现
3. 完成生成语法树的功能的实现
4. 完成查看语法树的功能的实现
5. 退出系统

## 实现过程

1. 界面设计如下：



1. 保存，打开存有源程序的功能按钮的实现

使用Qt的文件QfileDialog、QFile和QTextStream实现保存和打开功能。

fileDialog.getSaveFileName()函数实现获取保存文件的路径，再使用QFile和QTextStream将文法规则写入文件。

QFileDialog::getOpenFileName()函数实现获取打开文件的路径名，再使用

QFile和QTextStream将文法规则读取出来并显示在编辑框上。

1. 完成生成语法树的功能的实现
2. 首先先完成TINY语言语法扩充的文法规则，其文法规则如下：

program->stmt-sequence

stmt-sequence-> stmt-sequence**;**statement | statement

statement->if-stmt|repeat-stmt|assign-stmt|read-stmt|write-stmt

|Dowhile-stmt|for-stmt|regular-stmt

if-stmt->**if** exp **then** stmt-sequence **end**

|**if** exp **then** stmt-sequence **else** stmt-sequence **end**

repeat-stmt->**repeat** stmt-sequence **until** exp

assign-stmt->**identifier =** exp | **identifier -=** exp

read-stmt->**read** **identifier**

write-stmt->**write** exp

Dowhile-stmt->**do** stmt-sequence **while(**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**

regular-stmt-> **identifier :=** regular-exp

regular-exp->regular-exp regular-op regular-term | regular-term

regular-op -> **|**

regular-term->regular-term **&** regular-term2 | regular-term2

regular-term2-> regular-factor**#** | regular-factor

regular-factor->**(**regular-exp**)** | **number** | **identifier**

exp->exp **or** exp-term | exp-term

exp-term->exp-term **and** exp-factor | exp-factor

exp-factor->**not** comparison-exp | comparison-exp

comparison-exp->simple-exp comparison-op simple-exp | simple-exp

comparison-op-> **<** | **<=** | **>** | **>=** | **==** | **<>**

simple-exp->simple-exp addop term | term

addop-> **+** | **-**

term-> term mulop term2 | term2

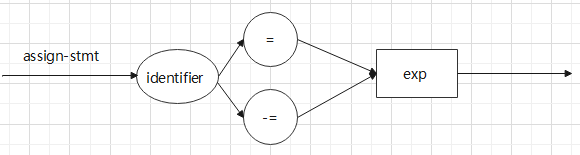
mulop-> **\*** | **/** | **%**

term2-> term2 **^** factor | factor

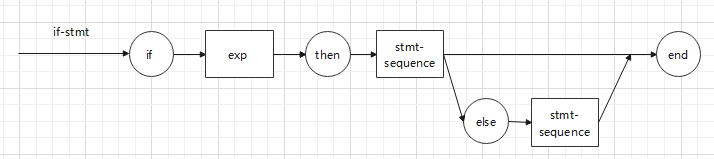
factor-> **(**exp**)** | **number** | **identifier**

1. 根据语法图写递归下降子程序，其中消除左递归和左公因子用EBNF来改造文法，其中部分语法图如下：

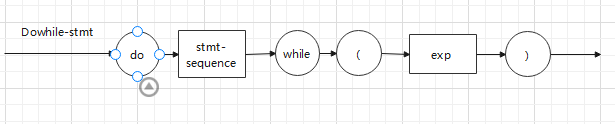
赋值语句的语法图：



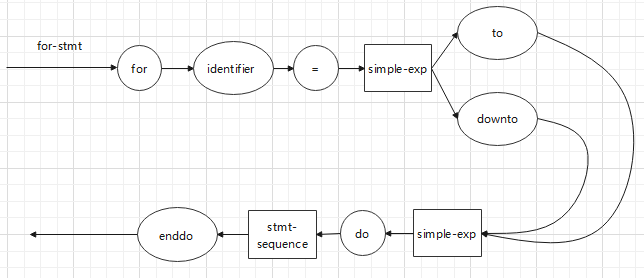
if语句的语法图：



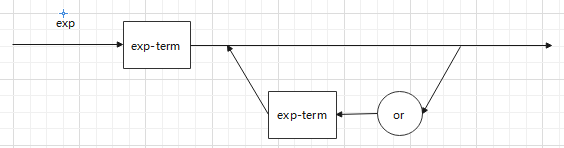
Dowhile语句的语法图：



for语句的语法图



表达式的语法图



剩余的其他语句的语法图类似，这里不一一画出

1. 由于是在原TINY的语法上进行扩充，所以只需修改它原来的代码以及增加新的文法规则的递归下降子程序。本次实现只需要生成语法树即可，所以只需要用到原TINY语言的扫描程序和生成语法树以及打印语法树。
2. 完成查看语法树的功能的实现

将生成的语法树用变量treePrint来保存，点击按钮“查看语法树”，将treePrint的内容显示到编辑框上。

1. 退出按钮功能

exit(0)

# 设计方案

## 功能组成

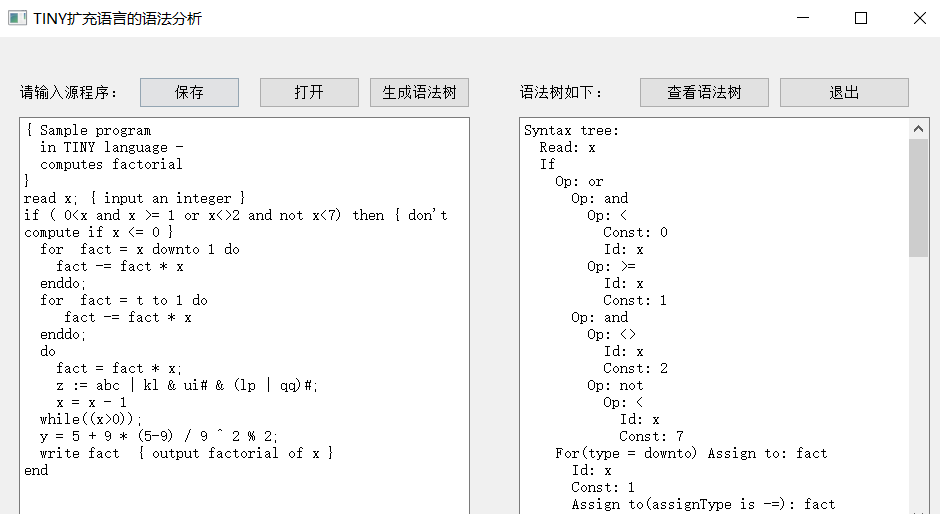
1. 保存源程序
2. 打开源程序
3. 生成语法树
4. 查看语法树
5. 退出系统

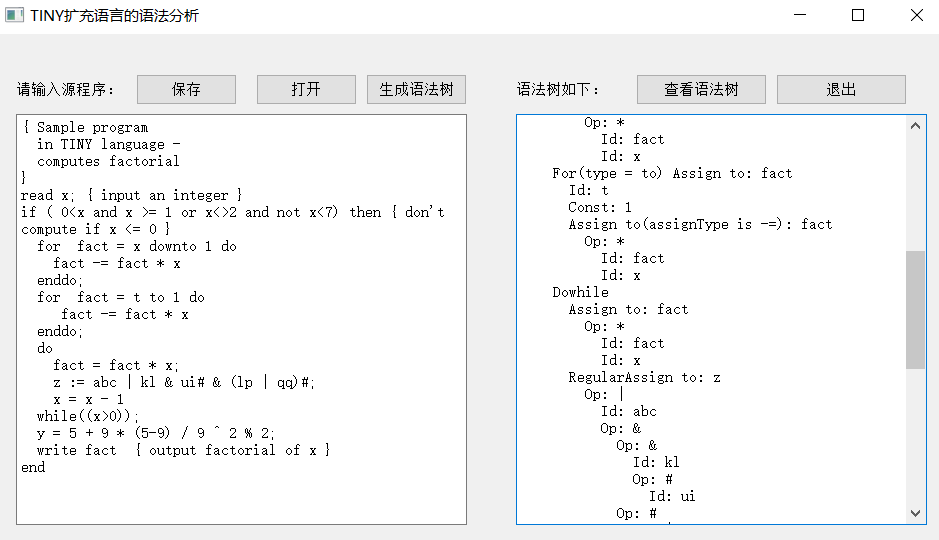
## 界面组成

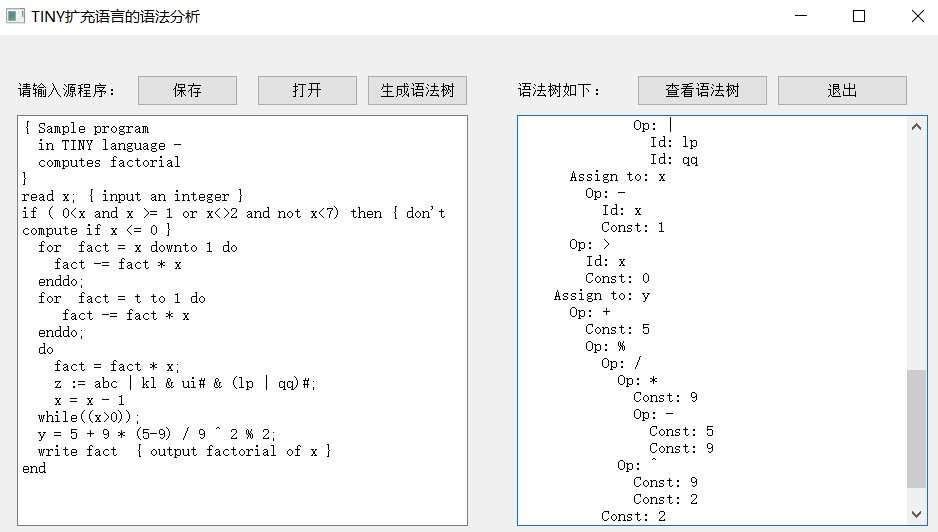


## 例子结果展示

例子1：（例子1包括了题目所有要求的文法规则的测试）









其语法树如下：

Syntax tree:

Read: x

If

Op: or

Op: and

Op: <

Const: 0

Id: x

Op: >=

Id: x

Const: 1

Op: and

Op: <>

Id: x

Const: 2

Op: not

Op: <

Id: x

Const: 7

For(type = downto) Assign to: fact

Id: x

Const: 1

Assign to(assignType is -=): fact

Op: \*

Id: fact

Id: x

For(type = to) Assign to: fact

Id: t

Const: 1

Assign to(assignType is -=): fact

Op: \*

Id: fact

Id: x

Dowhile

Assign to: fact

Op: \*

Id: fact

Id: x

RegularAssign to: z

Op: |

Id: abc

Op: &

Op: &

Id: kl

Op: #

Id: ui

Op: #

Op: |

Id: lp

Id: qq

Assign to: x

Op: -

Id: x

Const: 1

Op: >

Id: x

Const: 0

Assign to: y

Op: +

Const: 5

Op: %

Op: /

Op: \*

Const: 9

Op: -

Const: 5

Const: 9

Op: ^

Const: 9

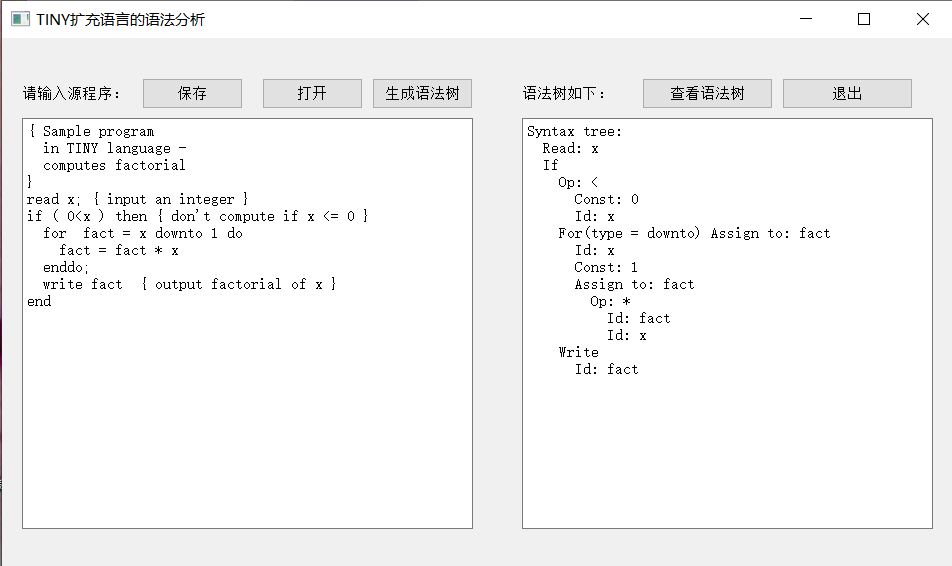
Const: 2

Const: 2

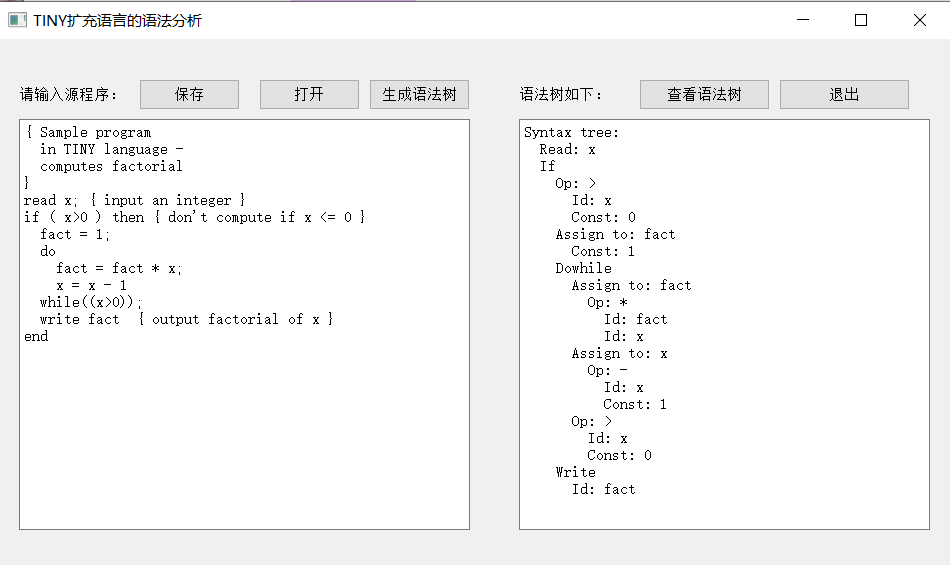
Write

Id: fact

例子2：

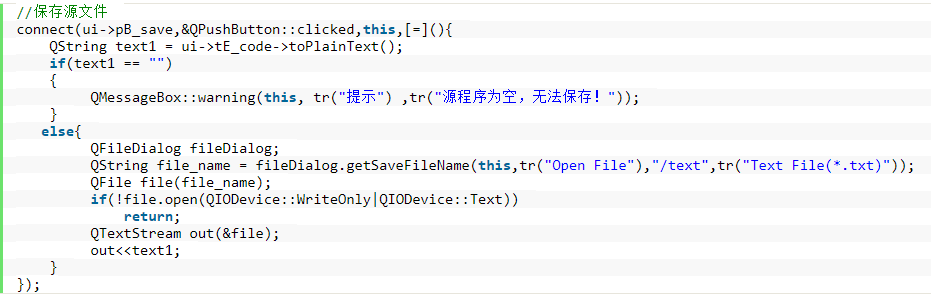


例子3：

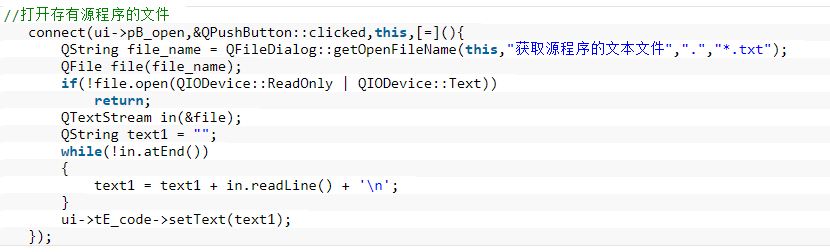


## 代码实现

### 1.保存源程序

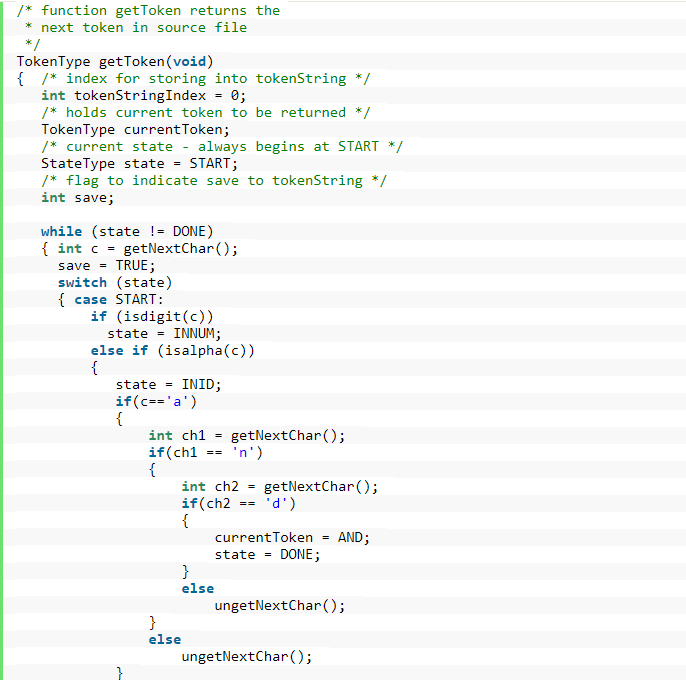


### 2.打开源程序



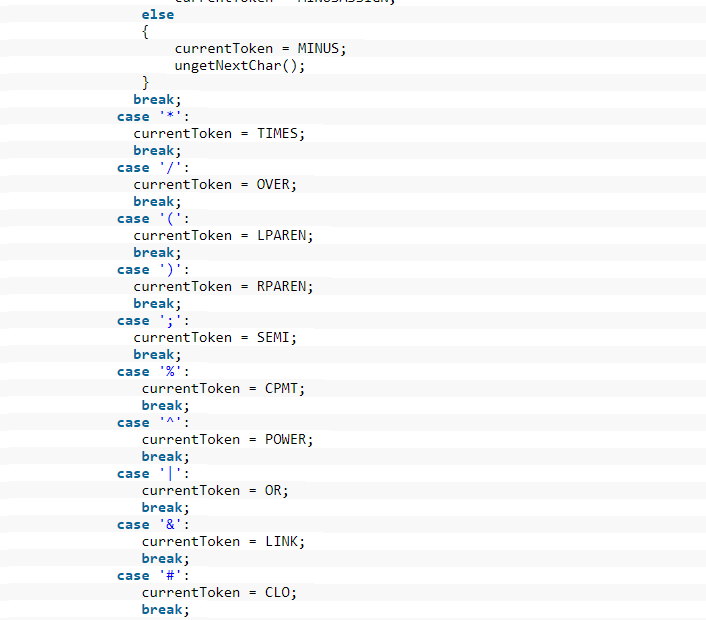
### 3.生成语法树

（1）首先要有扫描程序，获取token，函数为TokenType getToken()







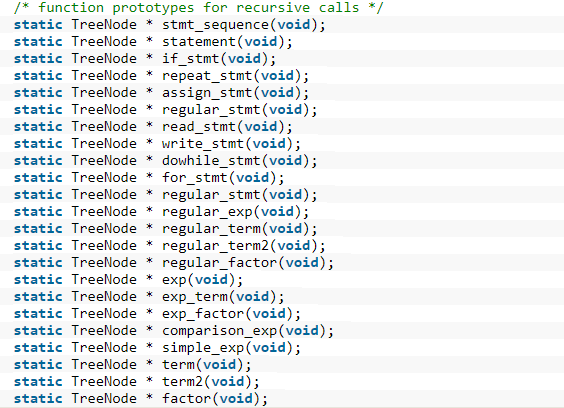




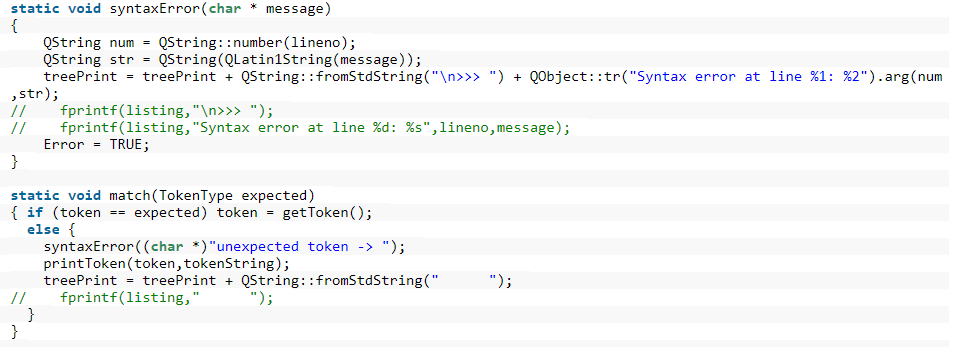


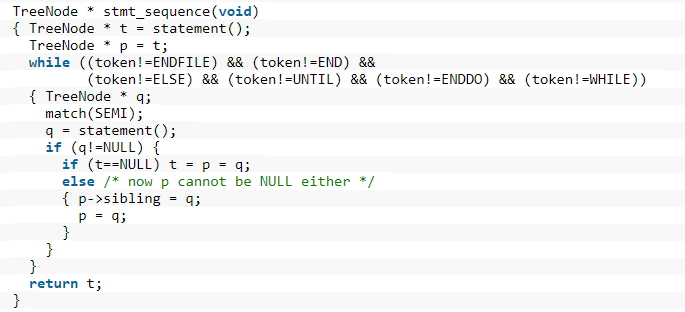
（2）接着是生成语法树的递归下降子程序

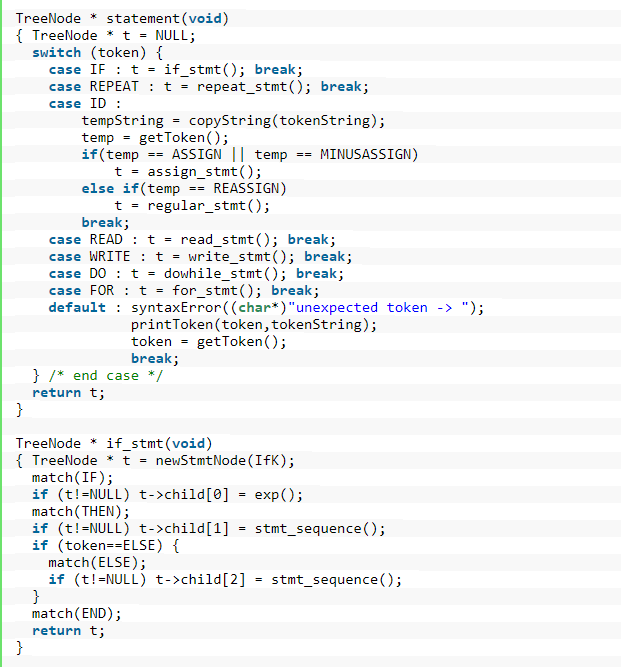
所有递归下降子程序如下：

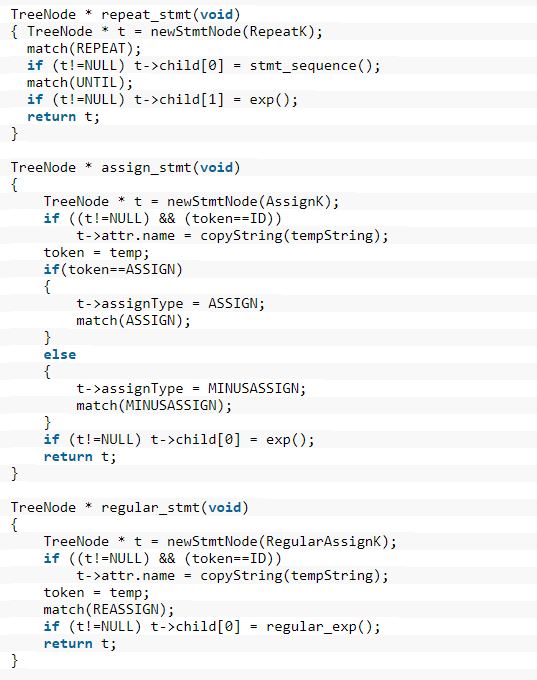


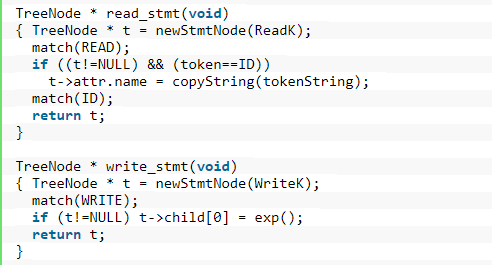
具体实现如下：（包括出错函数和match函数）

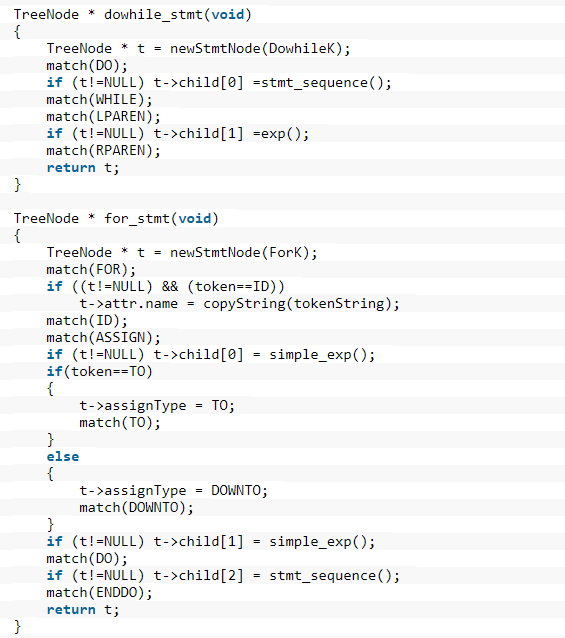


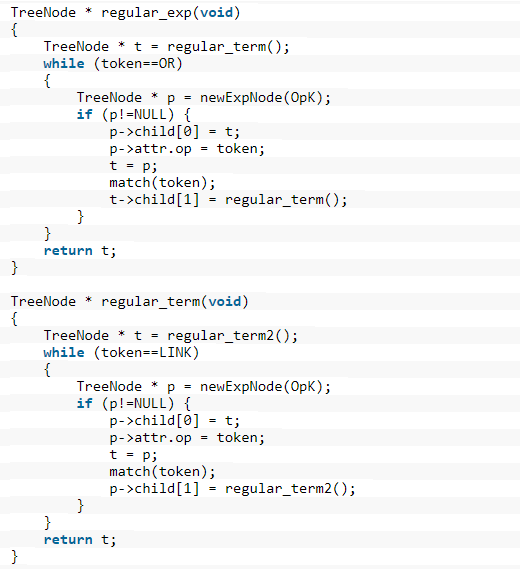


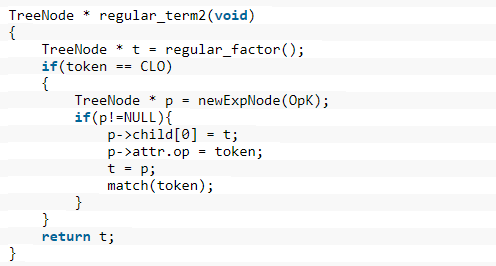




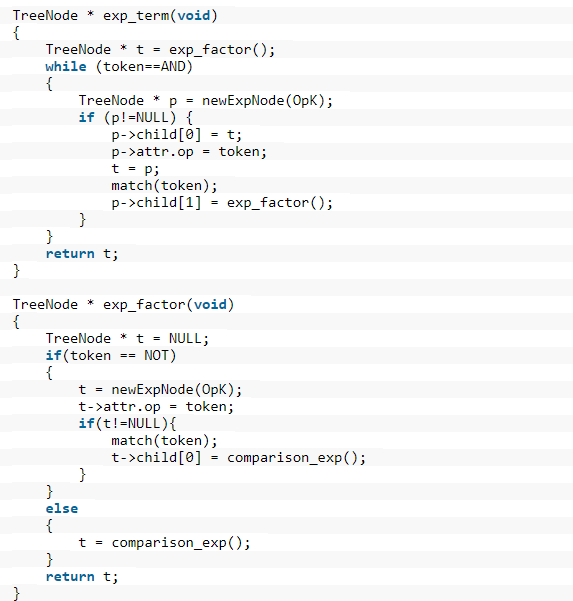


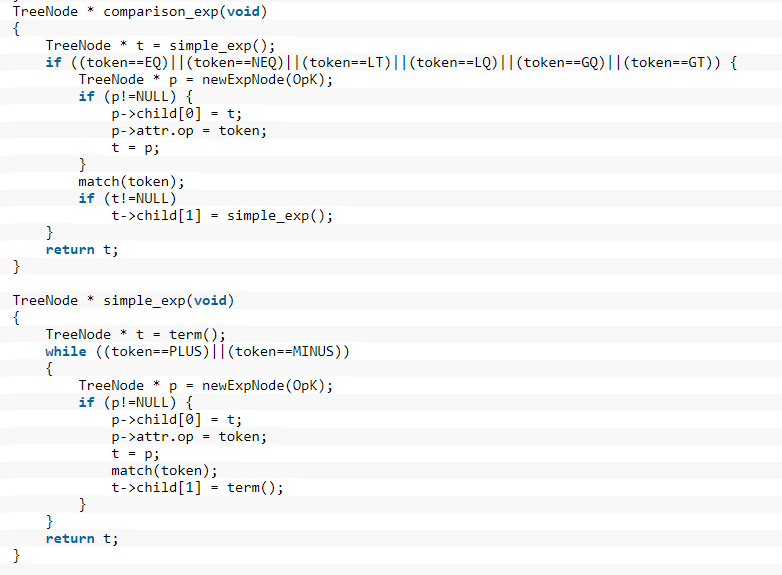




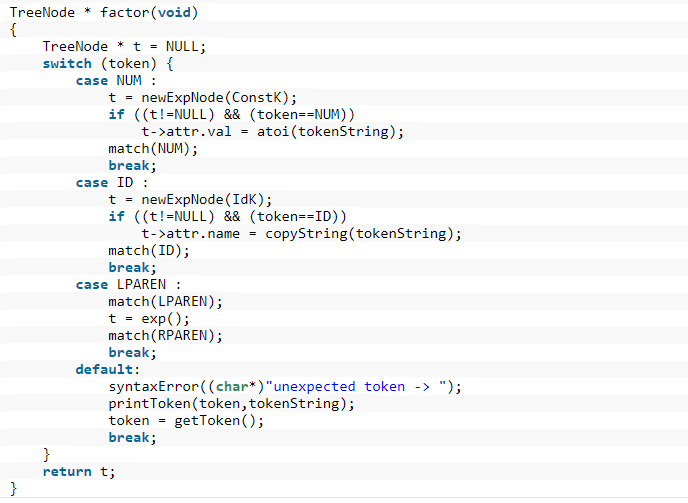








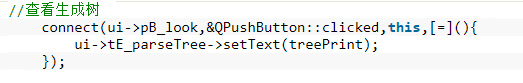




（3）执行生成语法树



## 4.查看语法树



### 5.退出系统

