#### 3.设计思路

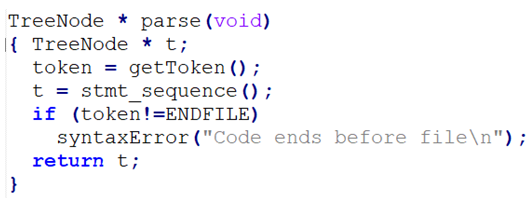
对原来的TINY语言编译器词法分析源代码剖析，并改写语法规则：

1.在main.c中，首先声明了全局头文件：  
#include “globals.h”

2.注意到接下来的宏定义：  
NO*PARSE、NO*ANALYZE、NO*CODE，初始值为FALSE。*  
*将NO*PARSE设置为TRUE可以获得一个仅扫描的编译器，因此NO*PARSE代表是否进行语法分析；将NO*ANALYZE设置为TRUE可以获得一个仅解析的编译器，因此NO*ANALYZE代表是否进行语义分析；将NO*CODE设置为TRUE可以得到一个不生成代码的编译器，因此NO\_CODE代表是否生成代码。  
分析到这里，已经可以确定该编译器源代码可以分为语法分析、语义分析、代码生成三个部分

3.与宏相关的if-else语句，根据#if NO\_PARSE时调用getToken()，而#else时调用parse()，推测getToken()就是词法分析函数，parse()中必然包括类似功能或者parse()中调用了getToken()函数。查找parse()函数，发现它在parseu.c中。

4.parse.c中应有词法分析与语法分析的相关实现，注意到如下代码：



这里调用了getToken()函数，考虑到在进行语法分析前必然要进行词法分析，因此可以确定stmt\_sequence()函数进行语法分析，而词法分析函数就是getToken()

5.parse.c文件中还定义了如下函数：

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);

用于分析if, repeat, 赋值， read，write，逻辑，算式表达式等语句

按照实验要求，我们把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

对exp(void)，simple\_exp(void)，term(void) 改写：新增-= 减法赋值运算符号、求余%、乘方^、>=(大于等于)、<=(小于等于)、>(大于)、<>(不等于)运算符号

并且添加以下函数处理逻辑表达式：

static TreeNode \* logit(void);
  
static TreeNode \* orterm(void);
  
static TreeNode \* andterm(void);
  
static TreeNode \* notterm(void);

添加以下函数处理正则表达式：

static TreeNode \* regex(void);
  
static TreeNode \* reorterm(void);
  
static TreeNode \* reandterm(void);
  
static TreeNode \* reclosterm(void);

6.getToken()函数在scan.c中被定义，因此可以确定，scan.c就是词法分析代码所在的文件。getToken()函数基于状态转换实现，返回下一个词法单元。通过该函数中的状态转移，可以推测TINY函数的赋值符号为:=，{}内为注释，每一句以;结束，标识符只能由字母组成，支持+、-、\*、/、<等简单的运算符。改写parse.c后，也要同步改写getToken()函数，记得先要写在globals.h中扩充保留字和特殊符号的定义：

typedef enum
  
 /\* book-keeping tokens \*/
  
 {ENDFILE,ERROR,
  
 /\* reserved words \*/
  
 IF,THEN,ELSE,END,REPEAT,UNTIL,READ,WRITE,OR,AND,NOT,//新增逻辑符号
  
 /\* multicharacter tokens \*/
  
 ID,NUM,ALPHA,
  
 /\* special symbols \*/
  
 ASSIGN,EQ,LT,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,SEMI,
  
 /\* 新增特殊符号 \*/
  
 MINUSASSIGN,LTEQ,GTEQ,GT,NOTEQ,MOD,POWER, // -=, <=, >=,>,<>,%,^
  
 REOR,REAND,RECLOSURE,REASSIGN // 新增正则表达式符号:或(|) 、连接(&)、闭包(#)
  
 } TokenType;

7.继续分析，char tokenString[MAXTOKENLEN+1]为标识符或保留字，使用字符数组存储。#define BUFLEN 256为源代码行的输入缓冲区的最大长度。每一行输入放进字符数组lineBuf[BUFLEN]中。

8.getNextChar()函数从lineBuf获取下一个非空字符，如果lineBuf已读完，则读取新行。

9.接下来定义了一个预留字的查找表，如if、then、else等，使用结构体实现。reservedLookup()函数使用最为简单的线性查找算法，通过刚刚定义的查找表，查找某一个标识符是否是保留字，从而可以区分保留字和用户自定义的ID。

10.当成功匹配一个词素后结束状态转换循环，若匹配到的词法单元为ID，需要检查该词素是否为保留字。若全局变量TraceScan为TRUE，调用util.c中定义了printToken()函数，将一个词法单元及其词汇表打印到listing文件中。getToken()函数最终返回下一个词法单元，实现词法分析。

11.在结束语法分析，循环调用getToken()以得到所有的词法单元之后，紧接着根据宏定义NO*PARSE、NO*PARSE与NO\_CODE的值，决定是否进行语法分析、语义分析与代码生成(可以细分为若干步)。

12.stmt\_sequence() 用于语法分析，匹配到if, repeat, assign, read, write 等语句时转入对应状态继续分析，其中包括对逻辑表达式和正则表达式的处理。

13.原analyze.c函数用于语义分析，现将其改写用于产生语法树并编写getsyntaxTree() 函数。对应地，在util.c函数中新增打印内容。至此，实验任务基本完成。