# Toureter Interpreter

——**Lexical Analysis Design Document**

**Written by Xiaoran Zhang**

# Produced by Toureter Group

**All Rights Reserved, Copy Reserved**

目录

[Toureter Interpreter 0](#_Toc345282135)

[Produced by Toureter Group 0](#_Toc345282136)

[1 整体概述 2](#_Toc345282137)

[2. 实现概念介绍 2](#_Toc345282138)

[2.1 词法分析对象介绍 2](#_Toc345282139)

[2.2 词法分析预期结果 4](#_Toc345282140)

[2.3 总体流程图 4](#_Toc345282141)

[2.4 DFAs 5](#_Toc345282142)

[2.4.1 标识符（包括关键字）的DFA 5](#_Toc345282143)

[2.4.2 数字的DFA 6](#_Toc345282144)

[2.4.3“/”符号（包括“/\*”）的DFA 6](#_Toc345282145)

[2.4.4“=”符号（包括“==”）的DFA 7](#_Toc345282146)

[3. 实现方法介绍 7](#_Toc345282147)

[3.1 模块类图: 7](#_Toc345282148)

[3.1.1 Token类 8](#_Toc345282149)

[3.1.2 扫描字符串，跳过空白字符 9](#_Toc345282150)

[3.1.3 词法分析方法 9](#_Toc345282151)

[3.2 具体实现 10](#_Toc345282152)

[4 实例介绍 15](#_Toc345282153)

# 整体概述

Toureter词法分析模块的主要功能是将用户输入的CMM源程序进行词法分析，通过扫描系统源程序将用户的源程序分解成单个的token串，并且将每一个token的内容和token的类型进行分类，以便编译器后续的分析。并且在词法分析过程中对用户输入的源程序进行词法扫描，如果用户的词法有误，将提醒用户进行词法更正。

# 实现概念介绍

## 2.1 词法分析对象介绍

词法分析的对象是CMM语言的源程序，源程序的叙述如下：

CMM语言为C语言的一个子集：

* + 语言结构：顺序结构（赋值语句、输入、输出）、选择语句（if-else）、循环结构（while）。这些语句结构和C语言的结构一样，允许嵌套。
  + 表达式局限于关系表达式和算术表达式，运算的优先级为：算术运算、关系运算，并服从左结合规则。
  + 算术表达式包括整数和实数上的运算、变量以及“（）”、“\*”、“+”、“-”、“/”，运算符的优先级顺序为：“（）”大于“\*”和“/”大于“+”和“-”。
  + 关系运算符包括：“<”、“==”、“<>”。
  + 一条语句以“；”结束；程序由一条语句或者由“{”和“}”嵌套表达的复合语句。
  + 支持多行注释（使用“/\*”和“ \*/ ”）
  + 支持数组运算，数组的下标必须是正整数，使用“[”和“]”表示数组下标。
  + 变量的使用之前需要先声明，声明的方式和C语言一样。

|  |  |  |
| --- | --- | --- |
| 保留字 | 特殊符号 | 其他 |
| if | + | 十进制的整数与实数 |
| else | - |
| while | \* |
| read | / | 标识符（由数字、字母和下划线组成的串，但必须以字母开头、且不能以下划线结尾的串） |
| write | = |
| int | < |
| real | == |  |
| class | <> |  |
| struct | ( |  |
|  | ) |  |
|  | ; |  |
|  | { |  |
|  | } |  |
|  | /\* |  |
|  | \*/ |  |
|  | [ |  |
|  | ] |  |
|  | /\* |  |
|  | \*/ |  |

## 2.2 词法分析预期结果

经过词法分析，预期能够扫描出的符号类型简介如下：

|  |  |
| --- | --- |
| 符号类型 | 符号 |
| Keyword | if，else，while，int，real，read，write |
| Sign | +，-，\*，/，/\*，\*/，=，==，<，<>，[，]，(，)，{，}，;，,， |
| int | 十进制整数，由数字串组成 |
| real | 十进制实数，由数字和小数点组成 |
| identifier | 由字母开头，由数字、字母和\_组成，并且不以下划线结尾的字符串 |

## 2.3 总体流程图

词法分析的总体流程图如下图所示：

## 2.4 DFAs

### 2.4.1 标识符（包括关键字）的DFA

\_

Letter,digit

letter

\_

Letter,digit

others

判断是否为关键字

### 2.4.2 数字的DFA

digit

digit

others

.

digit

others

### 2.4.3“/”符号（包括“/\*”）的DFA

\*

/

\*

\*

others

others

### 2.4.4“=”符号（包括“==”）的DFA

=

=

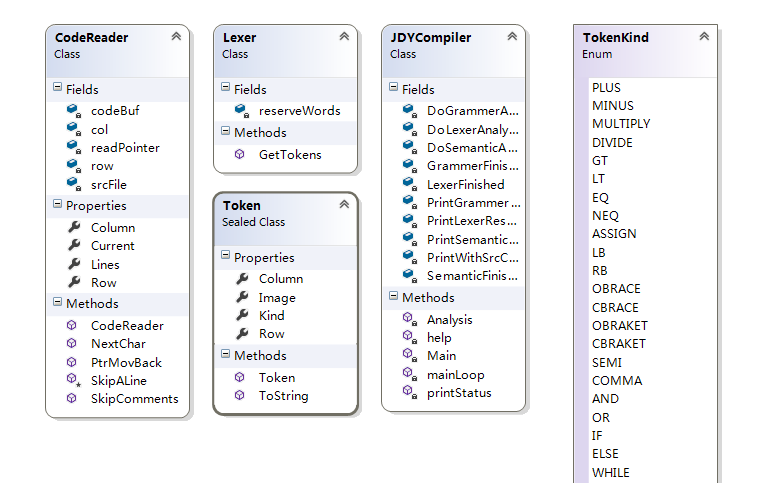
识别“==”

others

识别“=”

# 实现方法介绍

## 模块类图:



* 1. **所需结构**

### 3.2.1 Token类

用于存放Token的类：

internal sealed class Token:ICloneable

{

public readonly static TokenKind[] CONST\_TOKEN = new TokenKind[] { TokenKind.INT\_CONSTANT, TokenKind.REAL\_CONSTANT, TokenKind.TRUE, TokenKind.FALSE };

//token内容

public string Image { get; private set; }

public int Column { get; set; }

public int Row { get; set; }

public TokenKind Kind { get; private set; }

public Token(string image, TokenKind kind)

{

this.Image = image;

this.Kind = kind;

}

public override string ToString()

{

string type="";

if (this.Kind < TokenKind.INT\_CONSTANT && this.Kind > TokenKind.OR)

type = " reserved word:";

else if (this.Kind == TokenKind.INT\_CONSTANT)

type = " int const:";

else if (this.Kind == TokenKind.REAL\_CONSTANT)

type = " real const:";

else if (this.Kind == TokenKind.IDENTIFIER)

type = " ID:";

else if (this.Kind == TokenKind.EOF)

type = " EOF";

return string.Format(" {0}:{1} {2}", this.Row, type, this.Image);

}

public object Clone()

{

Token newToken = new Token(this.Image, this.Kind) { Row = this.Row, Column = this.Column };

return newToken;

}

}

### 3.2.2 扫描字符串，跳过空白字符

扫描字符串：

public static Tokens GetTokens(CodeReader codeReader) {

Tokens tokens = new Tokens();

char nextChar;

while ((nextChar=codeReader.NextChar()) != '\0') {

Token newToken = null;

bool Recongnized = true;

switch (nextChar)

{...}}

### 3.2.3 词法分析方法

词法分析方法：

switch (nextChar)

{

case '\*': newToken = new Token("\*", TokenKind.MULTIPLY); break;

case '(': newToken = new Token("(", TokenKind.LPARENT); break;

case ')': newToken = new Token(")", TokenKind.RPARENT); break;

case '{': newToken = new Token("{", TokenKind.OBRACE); break;

case '}': newToken = new Token("}", TokenKind.CBRACE); break;

case '[': newToken = new Token("[", TokenKind.OBRAKET); break;

case ']': newToken = new Token("]", TokenKind.CBRAKET); break;

case ';': newToken = new Token(";", TokenKind.SEMI); break;

case ',': newToken = new Token(",", TokenKind.COMMA); break;

case '^': newToken = new Token("^", TokenKind.BITXOR); break;

case '!': newToken = new Token("!", TokenKind.NOT); break;

case '%': newToken = new Token("%", TokenKind.MOD); break;

case '~': newToken = new Token("~", TokenKind.BITNOT); break;

case '.': newToken = new Token(".", TokenKind.DOT); break;

case '&':…

… }

## 3.3 具体实现

while ((nextChar=codeReader.NextChar()) != '\0') {

Token newToken = null;

bool Recongnized = true;

switch (nextChar)

{

case '\*': newToken = new Token("\*", TokenKind.MULTIPLY); break;

case '(': newToken = new Token("(", TokenKind.LPARENT); break;

case ')': newToken = new Token(")", TokenKind.RPARENT); break;

case '{': newToken = new Token("{", TokenKind.OBRACE); break;

case '}': newToken = new Token("}", TokenKind.CBRACE); break;

case '[': newToken = new Token("[", TokenKind.OBRAKET); break;

case ']': newToken = new Token("]", TokenKind.CBRAKET); break;

case ';': newToken = new Token(";", TokenKind.SEMI); break;

case ',': newToken = new Token(",", TokenKind.COMMA); break;

case '^': newToken = new Token("^", TokenKind.BITXOR); break;

case '!': newToken = new Token("!", TokenKind.NOT); break;

case '%': newToken = new Token("%", TokenKind.MOD); break;

case '~': newToken = new Token("~", TokenKind.BITNOT); break;

case '.': newToken = new Token(".", TokenKind.DOT); break;

case '&':

char prev = codeReader.NextChar();

if (prev == '&')

{

newToken = new Token("&&", TokenKind.AND);

}

else {

newToken = new Token("&", TokenKind.BITAND);

codeReader.PtrMovBack();

}

break;

case '+':

prev = codeReader.NextChar();

if (prev == '+')

newToken = new Token("++", TokenKind.INCREMENT);

else if (prev == '=')

newToken = new Token("+=", TokenKind.INCREASSGIN);

else

{

newToken = new Token("+", TokenKind.PLUS);

codeReader.PtrMovBack();

}

break;

case '-':

prev = codeReader.NextChar();

if (prev == '-')

{

newToken = new Token("--", TokenKind.DECREMENT);

}

else if (prev == '=') {

newToken = new Token("-=", TokenKind.DECREASSGIN);

}

else

{

newToken = new Token("-", TokenKind.MINUS);

codeReader.PtrMovBack();

}

break;

case '|':

prev = codeReader.NextChar();

if (prev == '|')

{

newToken = new Token("||", TokenKind.OR);

}

else {

newToken = new Token("|", TokenKind.BITOR);

codeReader.PtrMovBack();

}

break;

case '/':

prev = codeReader.NextChar();

if (prev == '\*' || prev == '/')

{//注释

codeReader.SkipComments();

}

else

{

newToken = new Token("/", TokenKind.DIVIDE);

codeReader.PtrMovBack();

}

break;

case '>':

prev = codeReader.NextChar();

if (prev == '=')

newToken = new Token(">=", TokenKind.GOE);

else

{

newToken = new Token(">", TokenKind.GT);

codeReader.PtrMovBack();

}

break;

case '<':

prev = codeReader.NextChar();

if (prev == '>')

newToken = new Token("<>", TokenKind.NEQ);

else if (prev == '=')

newToken = new Token("<=", TokenKind.LOE);

else

{

codeReader.PtrMovBack();

newToken = new Token("<", TokenKind.LT);

}

break;

case '=':

prev = codeReader.NextChar();

if (prev == '=')

newToken = new Token("==", TokenKind.EQ);

else

{

codeReader.PtrMovBack();

newToken = new Token("=", TokenKind.ASSIGN);

}

break;

default:

TokenKind kind = TokenKind.UNDETERMINED;

StringBuilder image = new StringBuilder();

if (nextChar == '\_' || nextChar >= 'A' && nextChar <= 'Z' || nextChar >= 'a' && nextChar <= 'z' || nextChar == '$')

{ //parse identifier

do

{

image.Append(nextChar);

nextChar = codeReader.NextChar();

}

while (nextChar == '\_' || nextChar >= 'A' && nextChar <= 'Z' || nextChar >= 'a' && nextChar <= 'z' || nextChar == '$' || nextChar >= '0' && nextChar <= '9');

codeReader.PtrMovBack();

if (reserveWords.ContainsKey(image.ToString())) //if is reserved word

kind = reserveWords[image.ToString()];

else

kind = TokenKind.IDENTIFIER;

}

else if (nextChar >= '0' && nextChar <= '9' || nextChar == '.')

{

bool beforeDot = true;

bool beforeExpo = true;

while (true)

{

if (nextChar == '.')

if (beforeDot)

{

beforeDot = false;

}

else

{

Recongnized = false;

codeReader.PtrMovBack();

}

else if (nextChar == 'e')

if (beforeExpo)

{

beforeExpo = false;

beforeDot = true;

}

else

{

Recongnized = false;

codeReader.PtrMovBack();

}

else if (nextChar < '0' || nextChar > '9')

{

Recongnized = false;

codeReader.PtrMovBack();

}

if (Recongnized)

{

image.Append(nextChar);

nextChar = codeReader.NextChar();

}

else

{

break;

}

}

kind = beforeDot&&beforeExpo ? TokenKind.INT\_CONSTANT : TokenKind.REAL\_CONSTANT;

}

if (image.Length > 0) {

newToken = new Token(image.ToString(), kind);

}

break;

}

if (newToken != null)

{

tokens.Add(newToken);

newToken.Column = codeReader.Column-newToken.Image.Length;

newToken.Row = codeReader.Row;

}

}

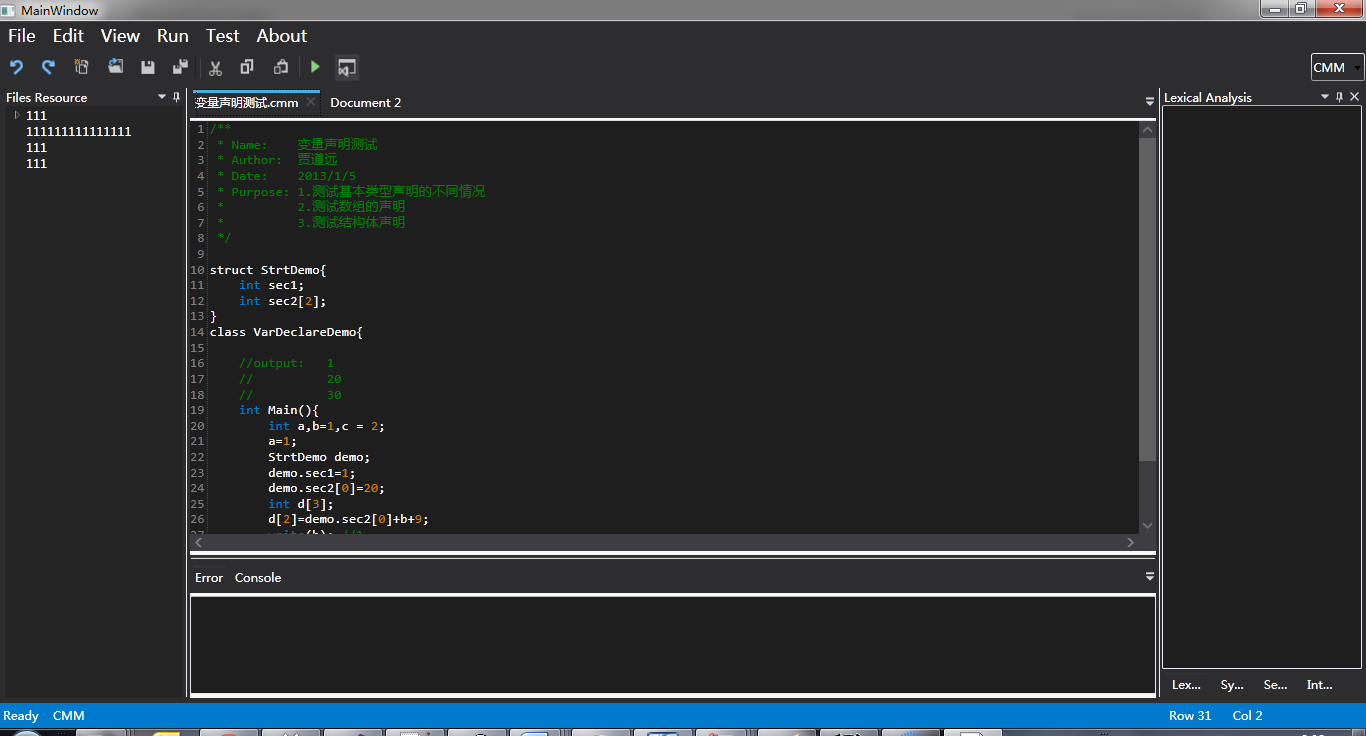
tokens.Add(new Token("", TokenKind.EOF){Column=codeReader.Column,Row=codeReader.Row});

return tokens;

}

# 4 实例介绍

输入以下内容：



得到如下结果，完成词法分析过程：

