

实验说明

161250194

张贝贝

0 前言

- 本次实验一选取了方案一的实现方式，先构造了 DFA，再根据 DFA 进行编程实现词法分析。
- 语法规则使用了 TINY 语言的语法规则，原因是这个语言小而精，词法不那么复杂，但是具备了一个语言的基本操作，分析后非常容易理解。

1 TINY 语言

1.1 TINY 定义

- 这是 TINY 在 BNF 中的定义

```
program → stmt-sequence  
stmt-sequence → stmt-sequence ; statement | statement  
statement → if-stmt | repeat-stmt | assign-stmt | read-stmt | write-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  
read-stmt → read identifier  
write-stmt → write exp  
exp → simple-exp comparison-op simple-exp | simple-exp  
comparison-op → < | =  
simple-exp → simple-exp addop term | term  
addop → + | -  
term → term mulop factor | factor  
mulop → * | /  
factor → ( exp ) | number | identifier
```

简化之：

- 仅是一个由分号分隔开的语句序列，它既无过程也无声明。
- 所有的变量都是整型变量。
- 只有两个控制语句：if 语句和 repeat 语句，这两个控制语句本身也可包含语句序列。If 语句有一个可选的 else 部分且必须由关键字 end 结束。
- read 语句和 write 语句完成输入/输出。
- 在花括号中可以有注释，但注释不能嵌套。

- 表达式局限于布尔表达式和整型算术表达式：
 - 布尔表达式由对两个算术表达式的比较组成，该比较使用<与=比较算符。
 - 算术表达式可以包括整型常数、变量、参数以及 4 个整型算符+、-、*、/，此外还有一般的数学属性。

1.2 TINY 词类别

key word: if,then,else,end,repeat,until,read,write

special symbol: :=,=,<;

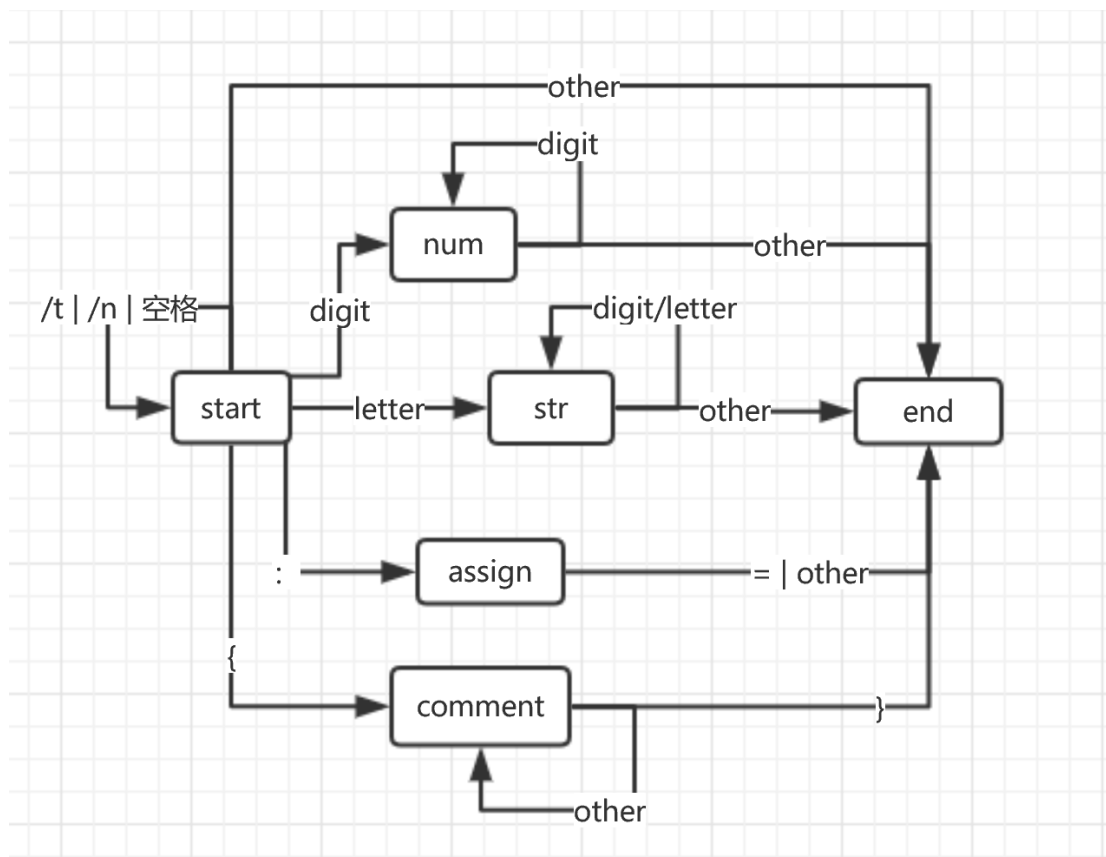
comment:{xxx}

calculation symbol:+,-,*,/

number: all of num

ID: all of letter or num, must begin with letter

2 根据语法规则获得 DFA



3 设计思想概述

3.1 算法

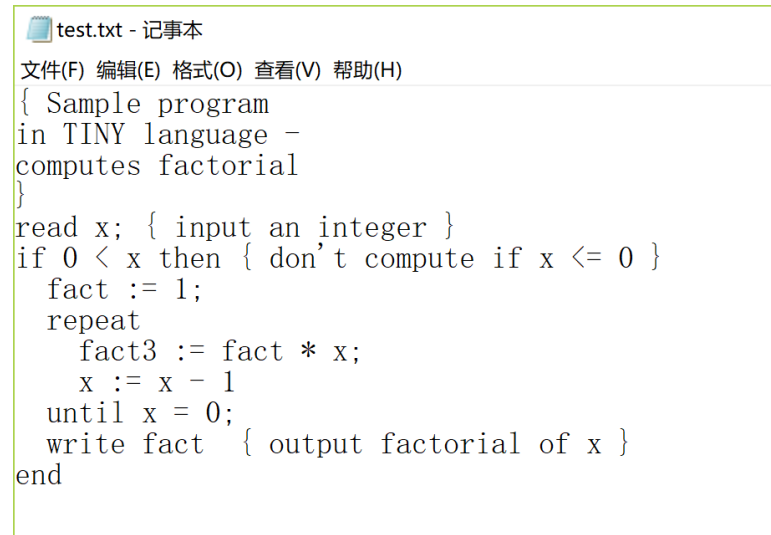
- 词法分析过程概述：循环读输入文件的每一个字符，然后根据读入的字符判断状态，到达 **end** 后，将获得的 **token** 进行识别，再打印。
- 核心算法：以构造的 DFA 为基础，根据读入字符判断状态。
- 核心算法说明（以获得数字的 **token** 为例）：开始后，如果读入数字，则记为 **num** 状态，并记录当前的数字，然后读入一个字符，如果还是数字，则状态不变，合并当前数字，继续循环；如果不是数字，则退回到上一个字符，并改变状态为 **end**，不合并当前字符，打印最终的数字 **token**。然后继续循环读入下一个字符。直到读完输入文件，循环结束。

3.2 数据结构

词法分析器比较简易，没有采用复杂的数据结构，主要使用了表驱动来辨别不同的词类别。

4 测试

4.1 测试文件（test.txt 在项目的 src 文件下）：



```
test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
{ Sample program
in TINY language -
computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact3 := fact * x;
    x := x - 1
  until x = 0;
  write fact { output factorial of x }
end
```

4.2 测试结果（输出在控制台；输出每一行语句与对应的词法分析结果；截图在项目的 src 文件下）：

line1: { Sample program

line2: in TINY language -

line3: computes factorial

line4: }

注释: { Sample program in TINY language - computes factorial}

line5: read x; { input an integer }

关键字: read

标识符: x

特殊符号: ;

注释: { input an integer }

line6: if 0 < x then { don't compute if x <= 0 }

关键字: if

数字: 0

特殊符号: <

标识符: x

关键字: then

注释: { don't compute if x <= 0 }

line7: fact := 1;

标识符: fact

特殊符号: :=

数字: 1

特殊符号: ;

line8: repeat

关键字: repeat

line9: fact3 := fact * x;

标识符: fact3

特殊符号: :=

标识符: fact

算数符号: *

标识符: x

特殊符号: ;

```
line10:      x := x - 1
标识符: x
特殊符号: :=
标识符: x
算数符号: -
数字: 1

line11:  until x = 0;
关键字: until
标识符: x
特殊符号: =
数字: 0
特殊符号: ;

line12:  write fact { output factorial of x }
关键字: write
标识符: fact
注释: { output factorial of x }

line13: end
关键字: end
-----
```

5 最后

5.1 过程中存在的困惑

对于选择一门具体的语言来进行词法分析我是有过犹豫的。因为它的表达不够符号化，所以难以进行复杂的 **GE-NFA-DFA** 分析。但是因为它的可读性、理解性以及现实意义比较高，所以最终还是选择了一门典型的语言来进行词法分析。

5.2 收获

实验一确实让我更深入的理解了词法分析的过程和词法分析器的具体代码实现，对编译原理和编译器的实现有了更深层次的了解。我相信对编译的学习将对我以后的其他编程活动有很大的帮助。

5.3 感谢

之前由于阅读实验要求 **PPT** 不仔细，导致这份报告迟交，耽误了助教大大的工作，很抱歉。也很感谢您愿意接受我的补交，感恩~