基于有限自动机的词法分析器

基于c语言

软件学院

陈睿 141250013

141250013@smail.nju.edu.cn

2016.10.27

目录

[1. 目标 3](#_Toc465531356)

[2. 内容描述 3](#_Toc465531357)

[3. 构思与方法 3](#_Toc465531358)

[4. 假设 3](#_Toc465531359)

[4.1. 实验环境 3](#_Toc465531360)

[4.2. C语言子集定义 4](#_Toc465531361)

[5. 有关自动机描述 4](#_Toc465531362)

[5.1. 基于语言分类的正则表达式 4](#_Toc465531363)

[5.2. 基于正则表达式构造有限自动机 5](#_Toc465531364)

[6. 数据结构描述 7](#_Toc465531365)

[7. 核心算法描述 8](#_Toc465531366)

[8. 测试用例 9](#_Toc465531367)

[8.1. Test 1 9](#_Toc465531368)

[8.2. Test 2 10](#_Toc465531369)

[9. 问题与解决 12](#_Toc465531370)

[10. 总结与感受 12](#_Toc465531371)

1. 目标

主要目标是实现对自定义的程序语言的词法分析器程序构造。我选择了C语言当中部分具有代表性的子集，定义了一部分正则表达式，实现了一个简单的词法分析器。同本实验，可以更好地理解从RE->NFA->DFA的过程，并应用到具体的程序语言词法分析中。

1. 内容描述

本报告主要描述了一个简单的词法分析器构造过程，包括构思、实现过程中的理论推导、具体的核心算法和数据结构的描述、最终成品、测试结果展示以及在其中遇到的困难与解决方法，还有最后的总结和感受。

1. 构思与方法

从整体思路来看，首先需要确定C语言的子集，然后对定义的语言子集进行分类，写出各类别的正则表达式，根据正则表达式手动构建有限自动机，最后根据有限自动机实现代码。

具体来看，代码部分比较简单，有如下步骤：

1. 读入一段基于定义语言的C语言代码
2. 逐个分析读入的字符，并记录产生的token
3. 输出结果
4. 假设
   1. 实验环境
5. Mac OS
6. Clion + MinGW (gcc)
   1. C语言子集定义

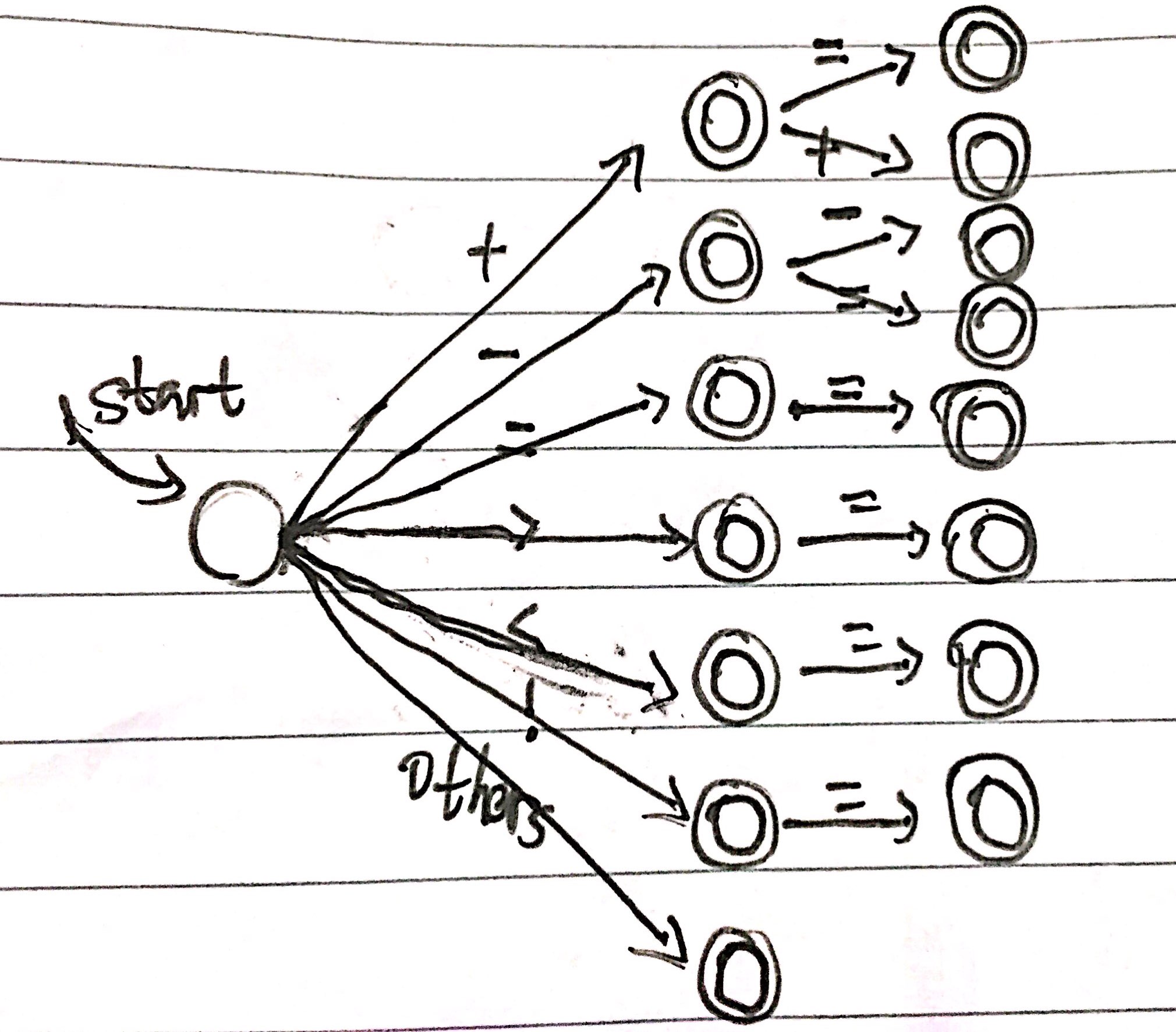
|  |  |  |
| --- | --- | --- |
| **符号** | 单目符号 | ( ) , ; [ ] { } 等 |
| 双目符号 | += == -= !=  ++ -- >= <= |
| 注释 | // /\*…\*/ |
| 空白符 | \n \t space |
| **字符串** | 标识符 | 字母开头，后面可以加任意数量的字母和数字 |
| 保留字 | else if int double char  return void while for |
| **数字** | 整数 | 0或者[1-9]开头，后面加任意数字 |
| 浮点数 | [0-9].开头，后面加只至少一个数字 |

1. 有关自动机描述
   1. 基于语言分类的正则表达式

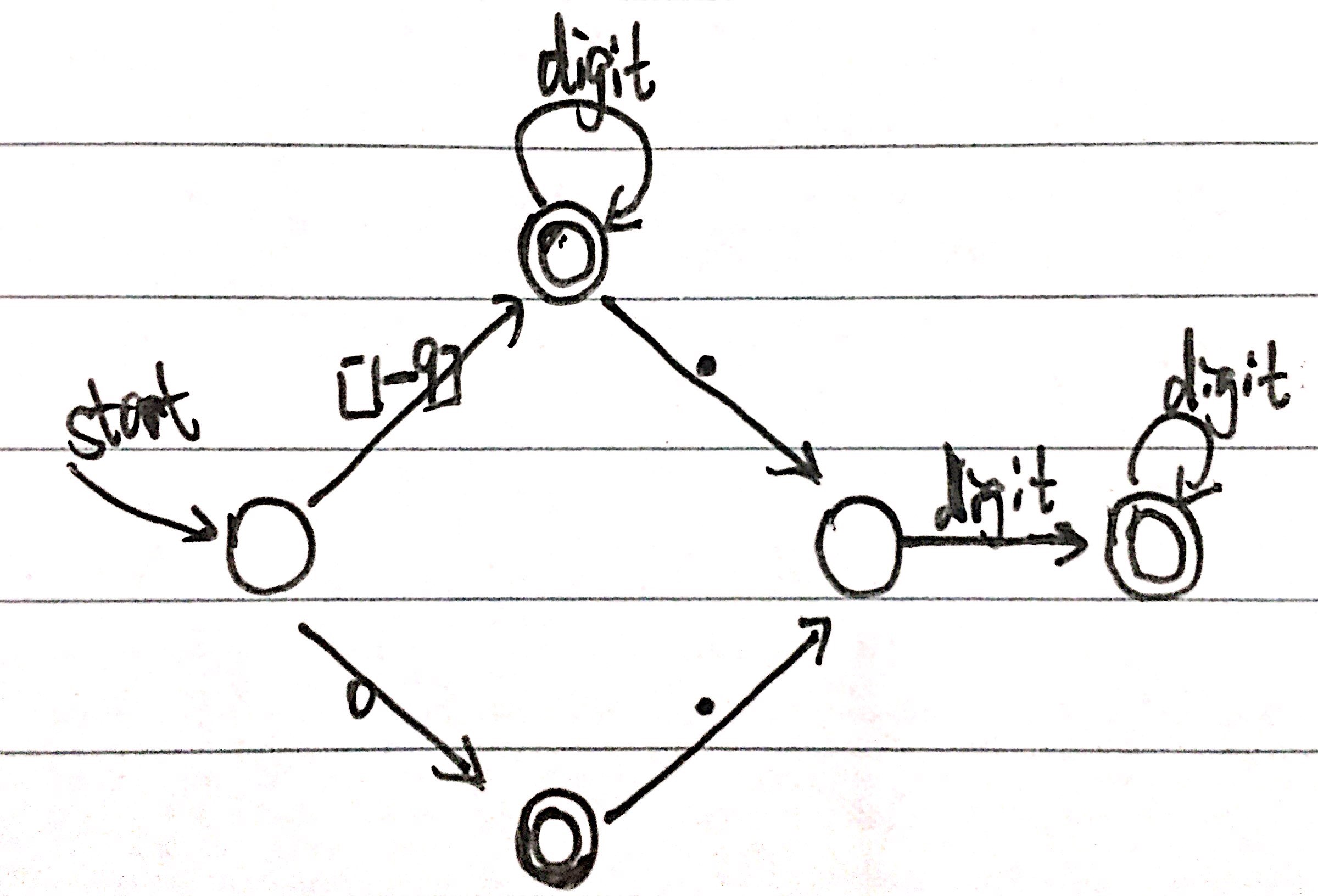
|  |  |  |
| --- | --- | --- |
| **符号** | **单目符号** | single\_char->(,|{|}|(|)|;|...) |
| 双目符号 | double\_char->(+=|==|-=|>=|<=|++|--|!=) |
| 注释 | comment\_single->//[.\*]\n  comment\_multi->/\*(([~\*~/]|[\*~/]|[~\*/])\*)\*/ |
| 空白符 | blank->( \n|\t|space ) |
| **字符串** | 标识符 | char->[a-zA-Z]  digit->[0-9]  str->char(char|digit)\* |
| 保留字 | reserve\_word-> (else|if|int|double|char|return|void|while|for) |
| **数字** | 整数 | digit->[0-9]  integer->([1-9](digit)\*|0) |
| 浮点数 | digit->[0-9]  double->digit.digit(digit)\* |

* 1. 基于正则表达式构造有限自动机

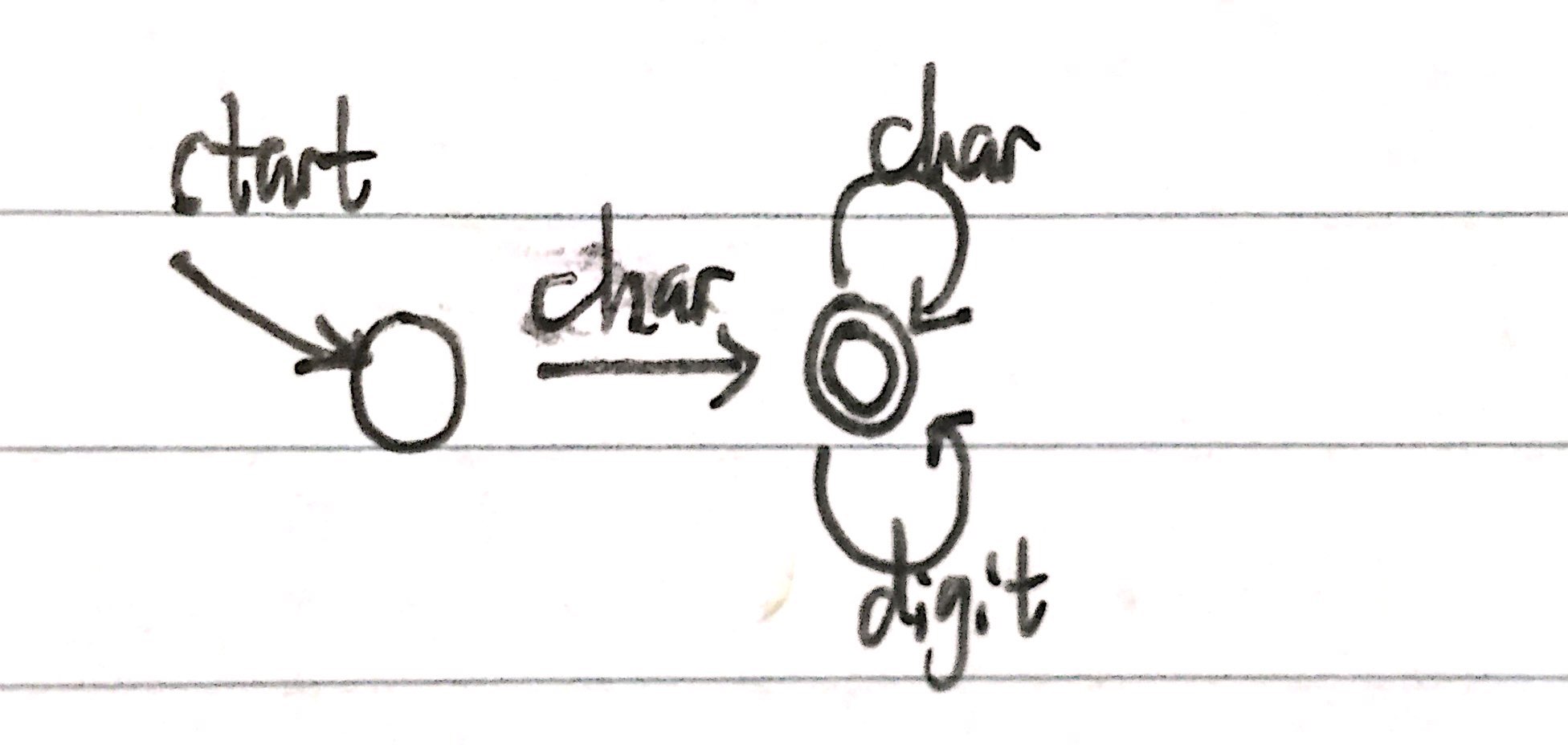
1. 符号



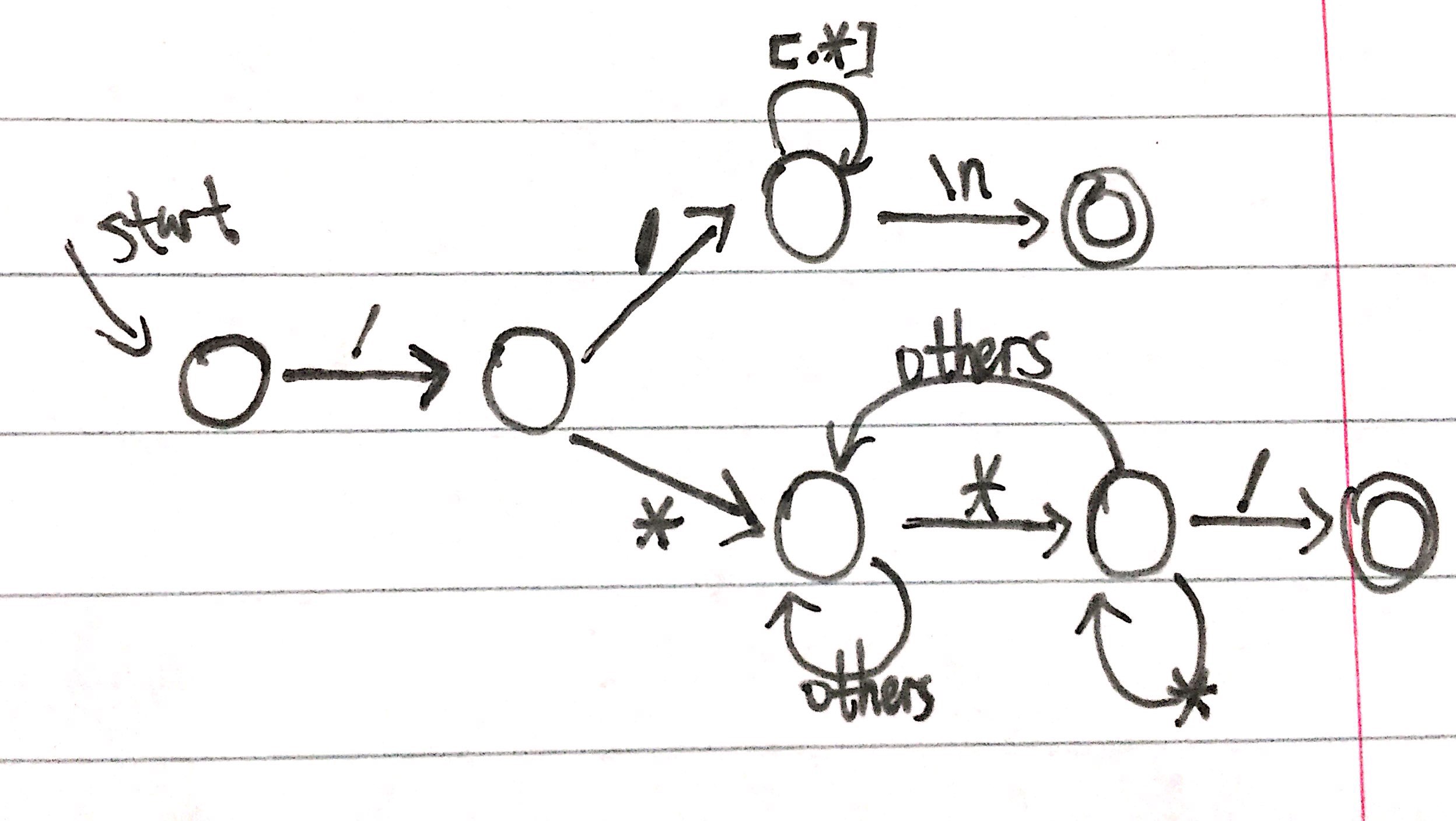
1. 数字



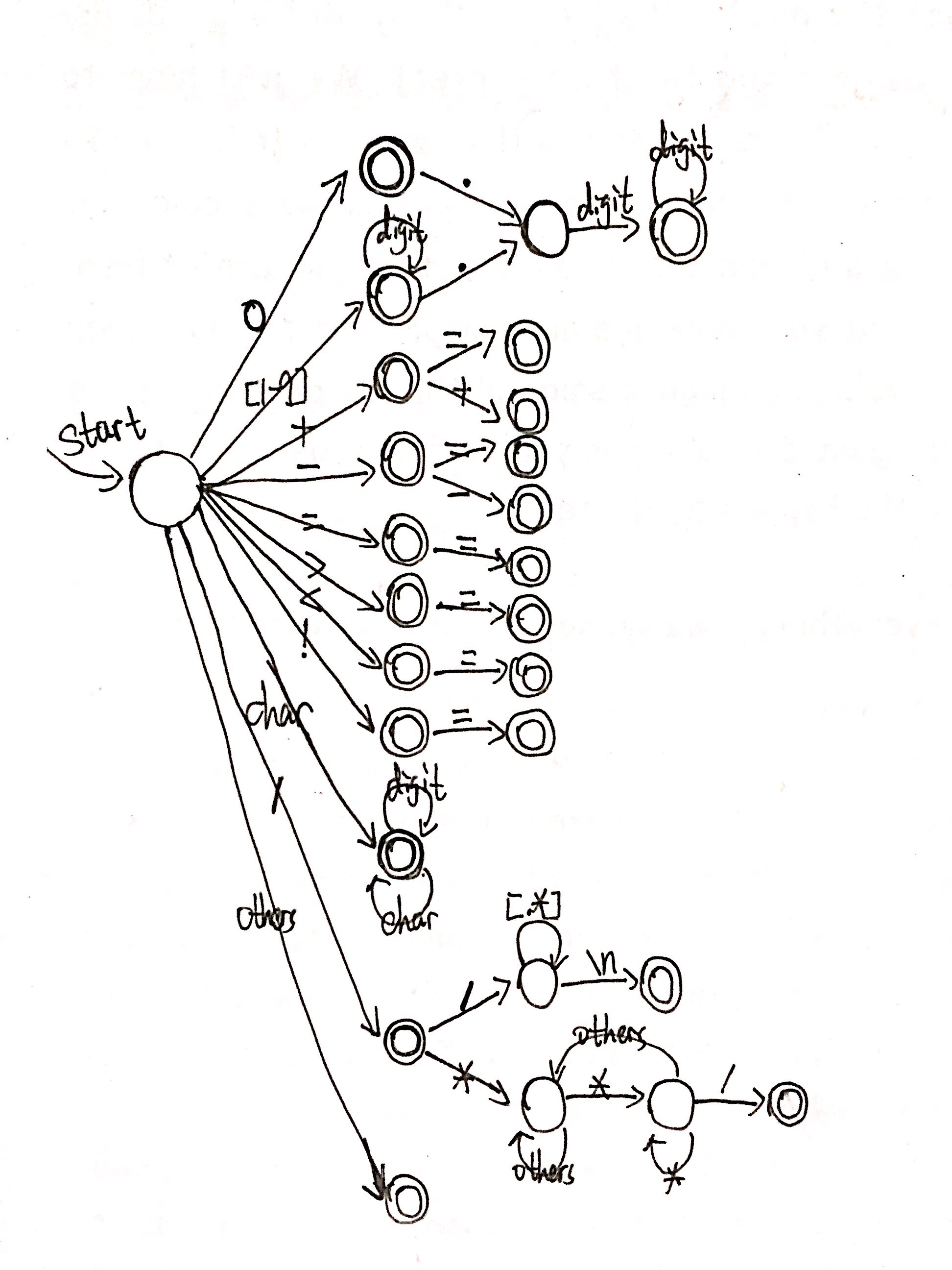
1. 标识符



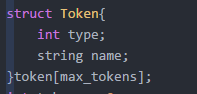
1. 注释



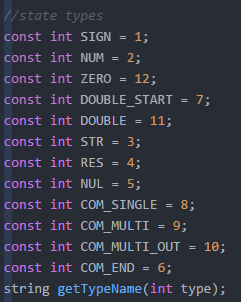
1. 总有限状态机



1. 数据结构描述
2. Token



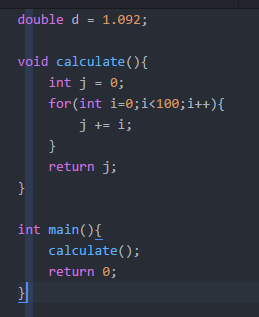
1. States



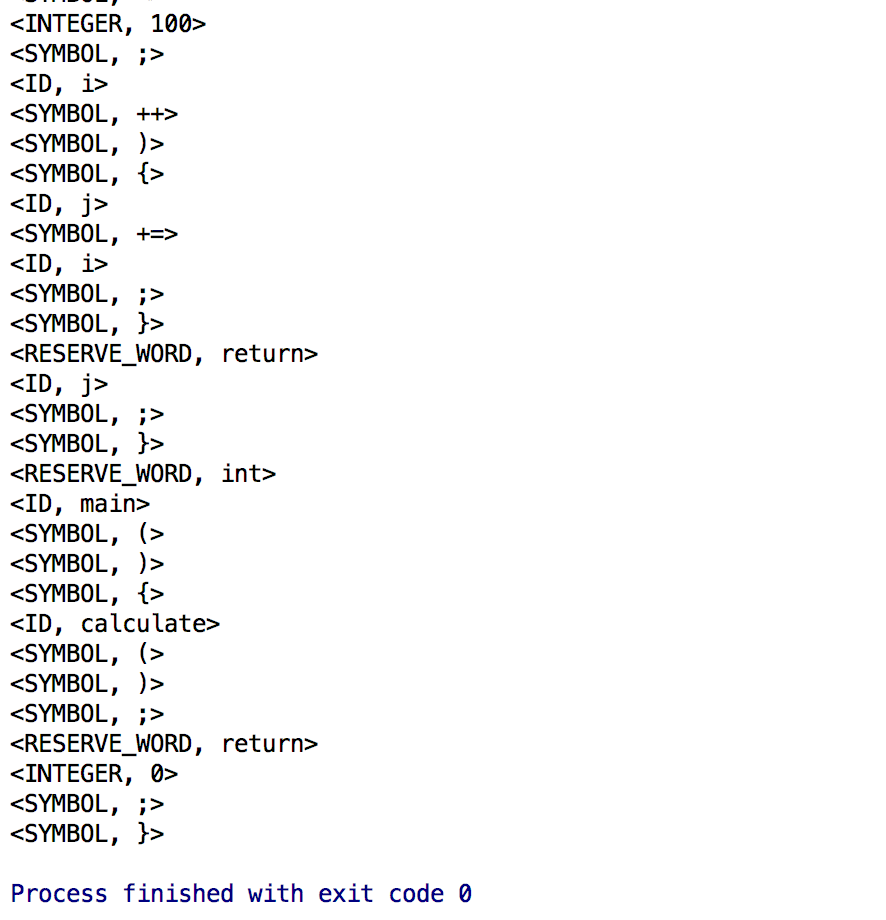
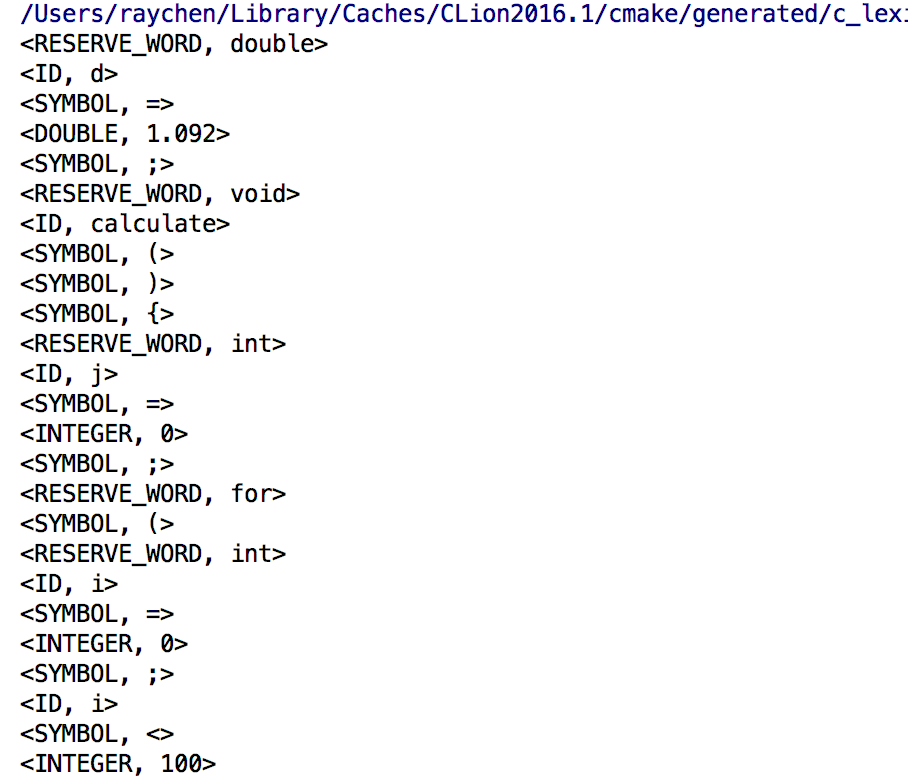
1. 核心算法描述
2. Analyze函数负责依据有限自动机，进行逐字分析输入的语言，并保存Token



1. 测试用例
   1. Test 1
2. 测试输入

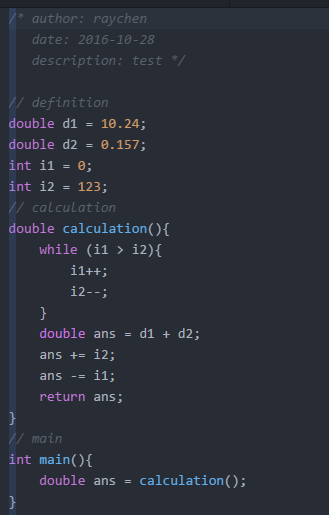


1. 测试输出

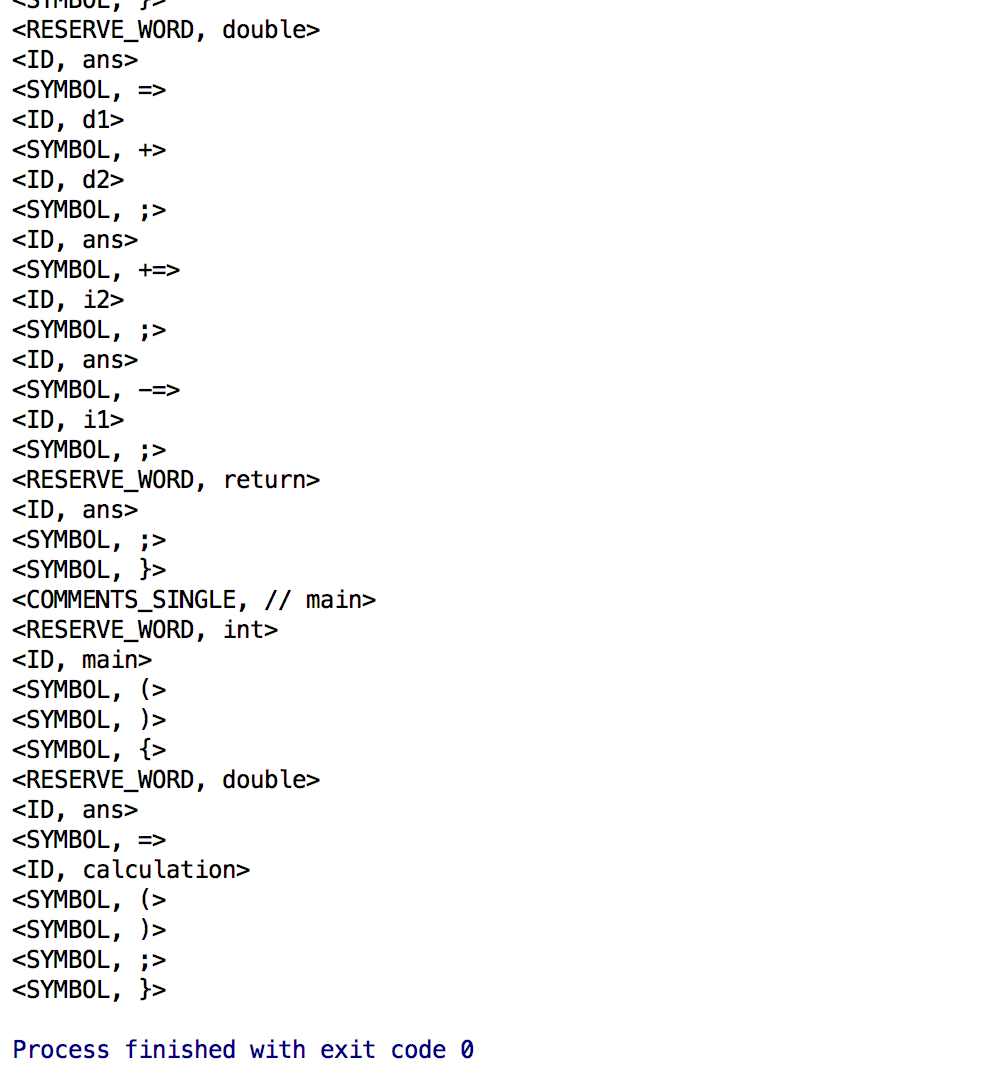
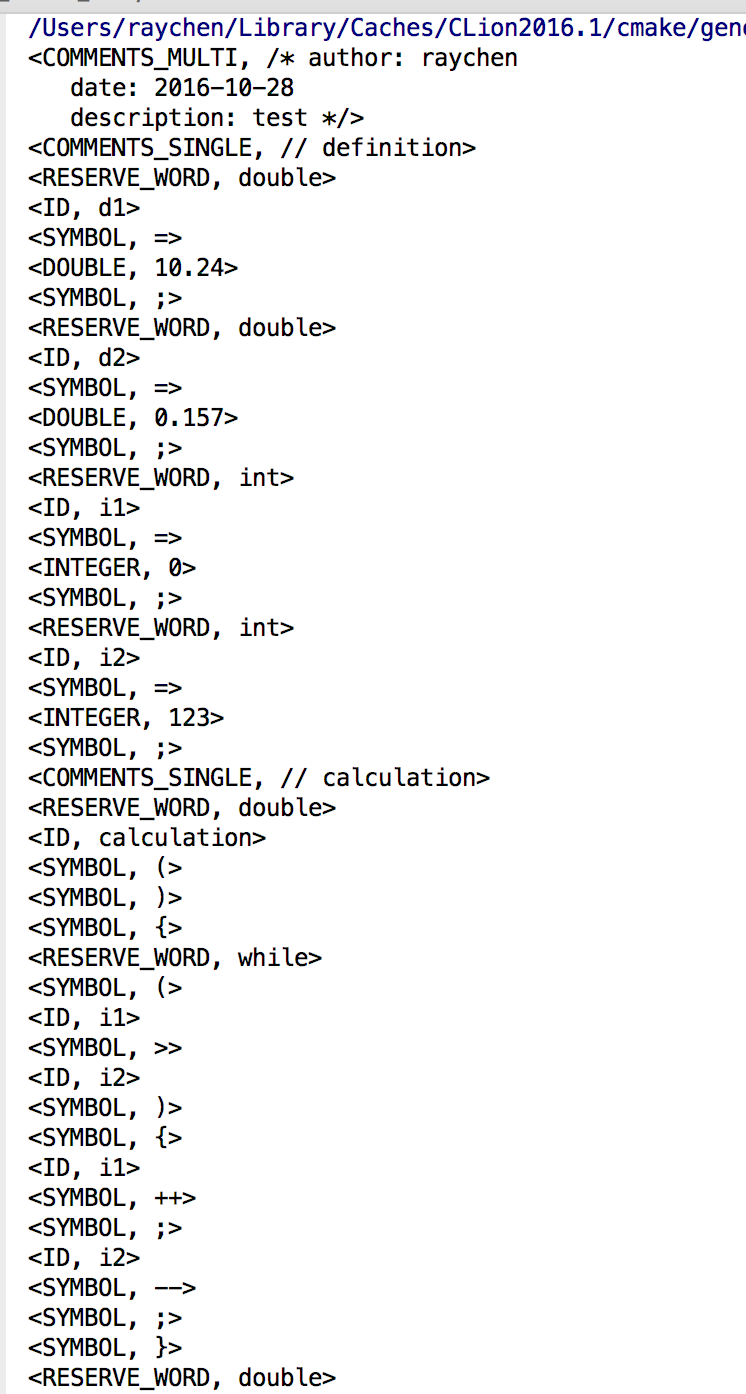


* 1. Test 2

1. 测试输入



1. 测试输出



1. 问题与解决
2. 保留字判断，如何从标识符中区分出保留字？

解决：对于判断一个标识符是不是保留字，我采取了先当成一般标识符去识别，得到token后再去判断该标识符是否为保留字，减少了中间判断过程。

1. 整体代码结构，判断流程、状态转换比较复杂。如何解决？

解决：我采取了先对当前状态进行判断，在对应各状态内对各种input的类型进行处理。层次比较清晰，状态的进一步转换也易懂。

1. 总结与感受

通过本次实验，我更深入理解了编译原理课程所讲内容，词法分析过程，同时为写词法分析生成器奠定基础。