**编译原理课程实验报告**

**实验1：词法分析**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 王一航 | | 院系 | | 软件工程 | | | 学号 | | | 1153710204 | | |
| 任课教师 | | 陈鄞 | | | | 指导教师 | 廖阔 | | | | | | |
| 实验地点 | | 格物楼213 | | | | 实验时间 | 2019/04/14 | | | | | | |
| 实验课表现 | | 出勤、表现得分 | |  | | 实验报告  得分 |  | | 实验总分 | | |  | |
| 操作结果得分 | |  | |
| **一、需求分析** | | | | | | | | | | 得分 | | |  |
| 基本功能: （已实现的功能标记为**粗体绿色**）  1. **标识符**（由大小写字母、数字以及下划线组成，但必须以字母或者下划线开头）  2. **关键字**（①类型关键字：整型、浮点型、布尔型、记录型；②分支结构中的if和else； ③循环结构中的do和while；④过程声明和调用中的关键字）  3. **运算符**（①算术运算符；②关系运算符；③逻辑运算）  4. **界符**（①用于赋值语句的界符，如“=”；②用于句子结尾的界符，如“；”；③用于数组表示的界符，如“[”和“]”；④用于浮点数表示的界符“.”）  5. **常数**（无符号整数和浮点数，包括科学计数法，字符串常数等）  6. **注释**（/\*……\*/形式）  额外功能:  1识别**字符常数**、**八进制**和**十六进制数**。  2能够应用**子集构造法从输入的NFA自动生成DFA**。  要求:  通过文件导入FA转换表和测试用例，可以通过用户界面显示并编辑测试用例。测试用例要涵盖“实验内容”中列出的各类单词，并包含各种单词拼写错误。  系统的输出分为两部分：一部分是打印输出词法分析器的DFA转换表。另一部分是打印输出源程序对应的token序列。 | | | | | | | | | | | | | |
| **二、文法设计** | | | | | | | | | | 得分 | | |  |
| 要求：对如下内容展开描述  给出各类单词的词法规则描述（正则文法或正则表达式）  标识符： **(A|B|…|Z|a|b|…|z|\_)( A|B|…|Z|a|b|…|z|\_| 0|1|2|…|9)\***  关键字：  **char | long | short | float | double | const | Boolean |**  **void |null| false |true| enum| int| do| while |if |else|**  **for| then | break | continue | class | static |final |extends|**  **new| return | signed| struct | union| unsigned| goto | switch|**  **case |default |auto| extern | register| sizeof | typedef | volatile**  运算符：  >|>=|<|<=|==|!=| | |&| || |&&|!|^|+|-|\*|/|%|++|--|+=|-=|\*=|/=|  界符：  ,|=|;| [|]| (|)|{|}|.|"|'  常数：**(0|1|2|…|9)(0|1|2|…|9)\*(( . (0|1|2|…|9) (0|1|2|…|9)\*)|ε)**  **(( E(+|-|ε)( (0|1|2|…|9) (0|1|2|…|9)\*) )|ε)**  注释：/\*(其他)\*/  八进制数：**0(1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)\***  十六进制数：**0x(1|...|9|a|...|f|A|…|F)(0|...|9|a|...|f |A|…|F)\***  各类单词的转换图  标识符  identifier.gv  关键字  由于画出 NFA 并不便于修改和查看，因此对关键字的识别在代码中通过字符串比较实现  运算符  operator.gv  界符  delimeter.gv  常数  unsigned.gv  注释  comment.gv  八进制  octal.gv  十六进制  hex.gv  字符串常量  string.gv | | | | | | | | | | | | | |
| **三、系统设计** | | | | | | | | | | 得分 | | |  |
| 要求：分为系统概要设计和系统详细设计。  **（1）系统概要设计：**   1. 系统框架图      1. 数据流图      1. 功能模块图     **（2）系统详细设计：**  **一、核心数据结构的设计**  **输入 NFA 结构设计：**  使用 JSON 来表示 NFA 转换表，最外部结构为字典，该字典的键表示 NFA 的所有状态，值的结构亦为字典，结构描述如下：  键 “edges” 表示从该键表示的状态出发的所有边集合，键表示输入字符，值表示，在该输入字符下可以达到的状态集合，结构为列表。  如果某一个状态为接受状态，则有一个键为 “accept”，值为字符串，表示该状态可以接受一个 token。  例：    可以看到，状态 0 存在一条出边，表示在状态 0 的时候，如果遇到输入 “0”，则可以转移到状态 1；在状态 1 的时候，如果遇到输入 “0”，可以转移到状态 2，如果遇到输入 “1” 也可以转移到状态 2，以此类推；在状态 2 的时候，如果遇到输入 “0”，“1”，...，“7” 又可以转换为状态 2，并且状态 2 也是一个接受状态，可以接受一个描述为“oct”（即 8 进制数）的 token。  **输出 DFA 设计**：（参考 NFA 转换 JSON）  **二、主要功能函数说明**   1. MergeNFA 将不同词法结构的 NFA 合并为一个完整的 NFA，用来进行 DFA 转换 2. ImportNFA 从 JSON 文件中输入 NFA 3. DFA.parse 使用 DFA 对待分析代码进行分析，并输出 Token 序列 4. DFA.subsetContruction 使用子集构造法对 NFA 进行转换 5. NFA.epsilon\_closure 对一个状态集合中的所有通过空串能够到达的所有状态集合进行计算 6. NFA.move 对一个状态集合在输入字符为 a 的情况下所能到达的所有状态集合进行计算 7. FA.visualize 对自动机进行可视化展示，输出转换图   **三、程序核心部分的程序流程图**   1. **子集构造法**      1. **空输入的状态转换** | | | | | | | | | | | | | |
| **四、系统实现及结果分析** | | | | | | | | | | 得分 | | |  |
| 1. **系统实现过程中遇到的问题；**    1. 如何对 NFA 进行有效表示    2. 如何将多个不同的词法单位的 NFA 进行合并 2. **针对某测试程序输出其词法分析结果；**   测试用例：    分析结果：  TOKEN: <comment, /\*  Multilines comment  \*/>  TOKEN: <comment, /\* Inline comment \*/>  TOKEN: <keywords, int>  TOKEN: <identifier, main>  TOKEN: <left round bracket, (>  TOKEN: <right round bracket, )>  TOKEN: <left curly bracket, {>  TOKEN: <keywords, int>  TOKEN: <identifier, a>  TOKEN: <assignment, =>  TOKEN: <unsigned, 1>  TOKEN: <semicolon, ;>  TOKEN: <keywords, int>  TOKEN: <identifier, b>  TOKEN: <assignment, =>  TOKEN: <unsigned, 1e8>  TOKEN: <semicolon, ;>  TOKEN: <keywords, float>  TOKEN: <identifier, c>  TOKEN: <assignment, =>  TOKEN: <unsigned, 0.4>  TOKEN: <semicolon, ;>  TOKEN: <keywords, int>  TOKEN: <identifier, d>  TOKEN: <left square bracket, [>  TOKEN: <right square bracket, ]>  TOKEN: <assignment, =>  TOKEN: <left curly bracket, {>  TOKEN: <unsigned, 1>  TOKEN: <comma, ,>  TOKEN: <unsigned, 2>  TOKEN: <comma, ,>  TOKEN: <unsigned, 3>  TOKEN: <right curly bracket, }>  TOKEN: <semicolon, ;>  TOKEN: <identifier, char>  TOKEN: <identifier, e>  TOKEN: <left square bracket, [>  TOKEN: <right square bracket, ]>  TOKEN: <assignment, =>  TOKEN: <string, "Hello world">  TOKEN: <semicolon, ;>  TOKEN: <identifier, c>  TOKEN: <assignment, =>  TOKEN: <unsigned, 1.0>  TOKEN: <numeric opreater, +>  TOKEN: <unsigned, 3>  TOKEN: <numeric opeater, ->  TOKEN: <unsigned, 1e8>  TOKEN: <numeric opeater, \*>  TOKEN: <unsigned, 0.5>  TOKEN: <assignment, />  TOKEN: <unsigned, 0.005>  TOKEN: <semicolon, ;>  TOKEN: <keywords, int>  TOKEN: <identifier, f>  TOKEN: <assignment, =>  TOKEN: <hex, 0xdeadbeef>  TOKEN: <keywords, int>  TOKEN: <identifier, g>  TOKEN: <assignment, =>  TOKEN: <oct, 01234567>  TOKEN: <identifier, printf>  TOKEN: <left round bracket, (>  TOKEN: <string, "Goodbye world!">  TOKEN: <right round bracket, )>  TOKEN: <semicolon, ;>  TOKEN: <right curly bracket, }>   1. **输出针对此测试程序对应的词法错误报告；**   测试用例：    分析结果：  TOKEN: <keywords, int>  TOKEN: <identifier, a>  TOKEN: <left square bracket, [>  TOKEN: <right square bracket, ]>  TOKEN: <assignment, =>  [0:7] '6','1','8','0','4','5','3','7','9','2' excepted, got 'a'  [0:8] '6','1','8','0','4','5','3','7','9','2' excepted, got ';'  分析：  由于 “1.a” 是一个非法的常数，在自动机扫描到 “1.” 的时候，期望的下一个字符为数字，但是测试用例中下一个字符为 “a”，因此程序输出错误信息：“[0:7] '6','1','8','0','4','5','3','7','9','2' excepted, got 'a'”，前面的 “[0:7]”，表示词法错误发生在第 0 行，第 7 列。   1. **对实验结果进行分析。**   实现了实验指导书中要求的所有基本功能以及附加功能，词法分析结果正确，可以对 NFA 进行自动化转换，并且实现了错误检测功能。   1. **感想**   在实现的过程中，查阅了大量资料，反复翻阅龙书的词法分析部分，发现有很多书上的东西在 Mooc 中并没有涉及，例如：   1. 如何自动化地将正则表达式转换为 NFA 2. 如何化简 DFA 3. 如何直接将正则表达式转换为 DFA 等等。   这些知识对我们深入理解编译系统也是非常重要的，希望能在课程结束以后进行深入学习。 | | | | | | | | | | | | | |
| 指导教师评语：  日期： | | | | | | | | | | | | | |