

编译原理课程实验报告

实验 1：词法分析

姓名	朱明彦	院系	计算机学院	学号	1160300314
任课教师	辛明影	指导教师	辛明影		
实验地点	格物 208	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

要求：阐述词法分析系统所要完成的功能

- (1) 词法分析是编译的第一阶段。词法分析器的主要任务是读入源程序的输入字符、将它们组成词素，生成并输出一个词法单元序列，每个词法单元对应于一个词素。这个词法单元序列被输入到语法分析器进行语法分析。
- (2) 词法分析器通常还要和符号表进行交互。当词法分析器发现了一个标识符词素时，他要将这个词素添加到符号表中。
- (3) 词法分析器在编译器中负责读取源程序，因此它还会完成一些识别词素之外的其他任务。如过滤掉源程序中的注释和空白（空格、换行符、制表符以及在输入中用于分隔词法单元的其他字符）；另一个任务是将编译器生成的错误消息与源程序的位置联系起来。

二、文法设计

得分

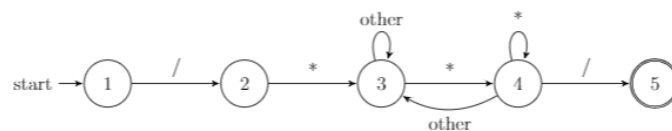
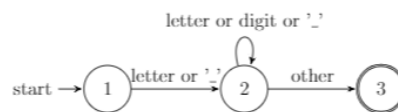
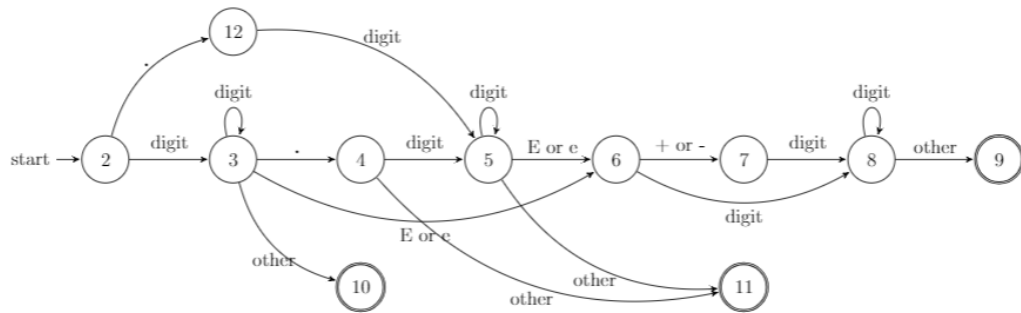
要求：对如下内容展开描述

- (1) 给出各类单词的词法规则描述（正则文法或正则表达式）
- (2) 各类单词的转换图

- (1) 各类单词的词法规则

- a) `digit -> [0-9]`
- b) `letter -> [A-Za-z]`
- c) `identifier -> [letter | '_'] [letter | digit | '_'] *`
- d) `arithmetic_op -> '+' | '-' | '*' | '/' | '%'`
- e) `relation_op -> '!=' | '>' | '<' | '>=' | '<=' | '=='`
- f) `logical_op -> '&' | '|' | '&&' | '||' | '^' | '!'`
- g) `delimiters -> '=' | ';' | ',' | '(' | ')' | '[' | ']' | '{' | '}'`
- h) `number -> digit+(.(digit+)?)?(('e' | 'E') ('+' | '-')?digit+)?`
- i) `comment -> /*([^*]*)*/`
- j) `keyword -> 'int' | 'float' | 'bool' | 'char' | 'struct' | 'if' | 'else' | 'while' | 'do' | 'break' | 'continue' | 'true' | 'false'`

- (2) 各类单词的 DFA 转换图



对于其余的单词，无需使用自动机即可进行判断，故此处没有赘述其 DFA。

三、系统设计

得分

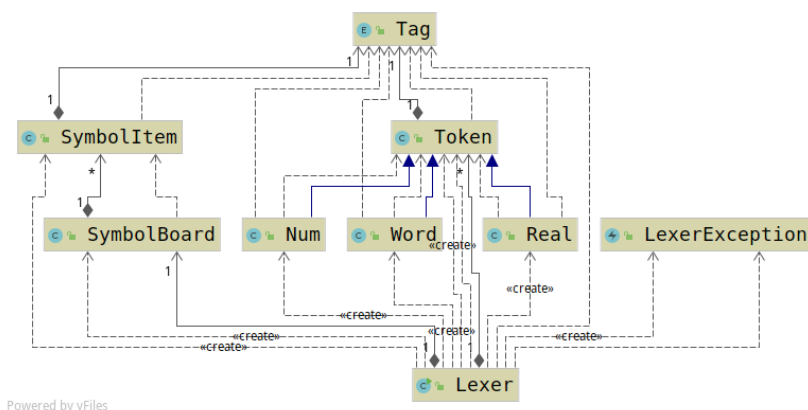
要求：分为系统概要设计和系统详细设计。

(1) 系统概要设计：给出必要的系统宏观层面设计图，如系统框架图、数据流图、功能模块图等以及相应的文字说明。

(2) 系统详细设计：对如下工作进行展开描述

- ✓ 核心数据结构的设计
- ✓ 主要功能函数说明
- ✓ 程序核心部分的程序流程图

(1) 系统概要设计



上图为词法分析器部分的 UML 图，其中 **Lexer** 为词法分析器的主类，**Tag** 为不同词素类型的枚举类型；

Token 为词素的父类，主要针对各种算符的词素；**Num** 为针对整数常数的词素类型，**Real** 是针对浮点型常数的词素类型，**Word** 为针对标识符和字符串类型的词素类型；

LexerException 为针对各种词法分析中可能的错误的异常类型；**SymbolItem** 为符号表中的基本条目，**SymbolBoard** 为符号表类型。

(2) 系统详细设计

(1) 核心数据结构设计

主要有三种核心数据结构的使用，分别是 **LIST**，**Queue** 和 **Set**；其中 **LIST** 主要用于存储待输出的 **Token** 序列和错误信息；**Queue** 用于缓冲已经读入的字符，特别是为了判断一些算符和标识符提前读入的字符；**Set** 用于存储各类需要判断的符号。

(2) 主要功能函数说明

isDigit() 用于判断一个字符是不是数字。

isLetter() 用于判断一个字符是不是字母。

reconID() 用于判断一个串是不是标识符类型或者关键字的自动机，如果是则返回识别出来的词素。

reconNumber() 用于判断一个串是不是数字类型的自动机，其中返回类型区分整数常数和浮点数常数两种不同的词素，当出现数字格式错误时抛出 **LexerException** 异常。

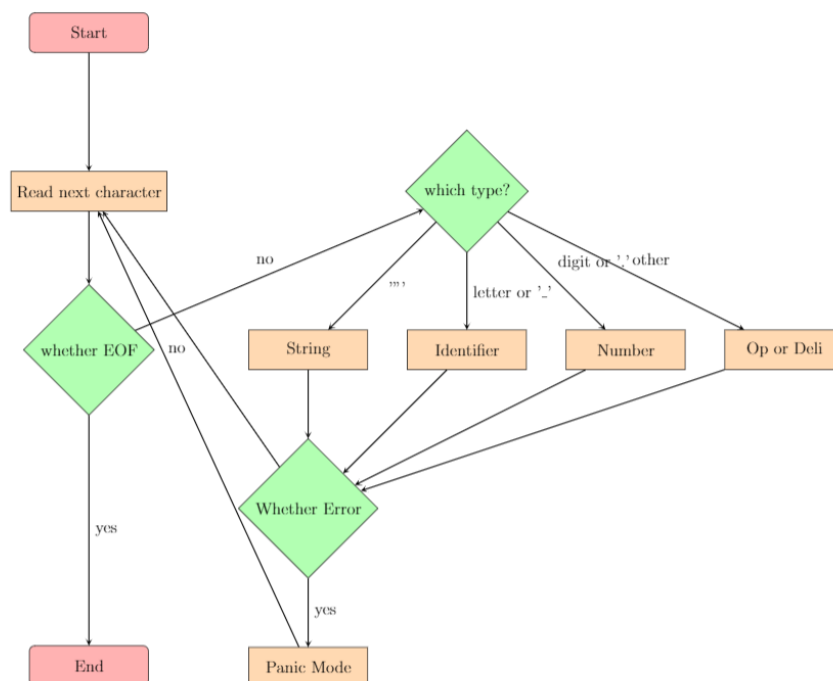
reconComment() 用于判断一个串是不是注释，如果是则一直读到注释结束或者文件结束为止。

reconString() 用于判断一个串是不是字符串，如果是则一直读到字符串结束，如果出现换行符则抛出异常为非法的字符串。

panicMode() 为用于“恐慌模式”恢复的函数，当出现非法字符等异常情况时，会调用恐慌模式直到遇到界符或换行符。

scan() 为 **Lexer** 的核心函数，其利用当前读入不同字符，分别调用不同的识别串的自动机。

(3) 程序核心部分程序流程图



四、系统实现及结果分析	得分	
<p>要求：对如下内容展开描述。</p> <ol style="list-style-type: none"> (1) 系统实现过程中遇到的问题； (2) 针对某测试程序输出其词法分析结果； (3) 输出针对此测试程序对应的词法错误报告； (4) 对实验结果进行分析。 <p>注：其中的测试样例自行产生。</p> <p>(1) 对于如'!'和'!='这类的符号，仅仅根据当前字符'!'是无法判断出其具体含义的，所以在开始处理的时候读入下一个字符，如果恰好可以和当前字符组成新的有意义的符号则成功，否则应该记录下当前符号，此时引入了 buffer 机制，利用队列(Queue)进行缓存。</p> <p>(2) 测试样例如下：</p> <pre> 1 struct student{ 2 int id; 3 char [] name; 4 }; 5 6 while(num != 0) 7 { 8 num = num + 1; 9 x = .0; /*afsjlkjdsalfjlasdfalsdkjfl kasd 10 hsjdak *****/ 11 string = "Hello world! 12 "; 13 int _aInt = 100; 14 a[10] = 100; 15 bool b = false#; 16 int 123Bcd; 17 int y = 12.2e9; 18 } </pre> <p>词法分析结果</p>		

<STRUCT>	<SEMICOLON>
<ID, student>	<BOOL>
<LP>	<ID, b>
<INT>	<ASSIGN>
<ID, id>	<FALSE>
<SEMICOLON>	<SEMICOLON>
<CHAR>	<INT>
<MLP>	<NUM, 123>
<MRP>	<ID, Bcd>
<ID, name>	<SEMICOLON>
<SEMICOLON>	<INT>
<RP>	<ID, y>
<SEMICOLON>	<ASSIGN>
<WHILE>	<REAL, 1.22E10>
<SLP>	<SEMICOLON>
<ID, num>	<RP>
<NE>	
<NUM, 0>	
<SRP>	
<LP>	
<ID, num>	
<ASSIGN>	
<ID, num>	
<ADD>	
<NUM, 1>	
<SEMICOLON>	
<ID, x>	
<ASSIGN>	
<REAL, 0.0>	
<SEMICOLON>	
<ID, string>	
<ASSIGN>	
<SEMICOLON>	
<INT>	
<ID, _aInt>	
<ASSIGN>	
<NUM, 100>	
<SEMICOLON>	
<ID, a>	
<MLP>	
<NUM, 10>	
<MRP>	
<ASSIGN>	
<NUM, 100>	

(3) 词法错误报告

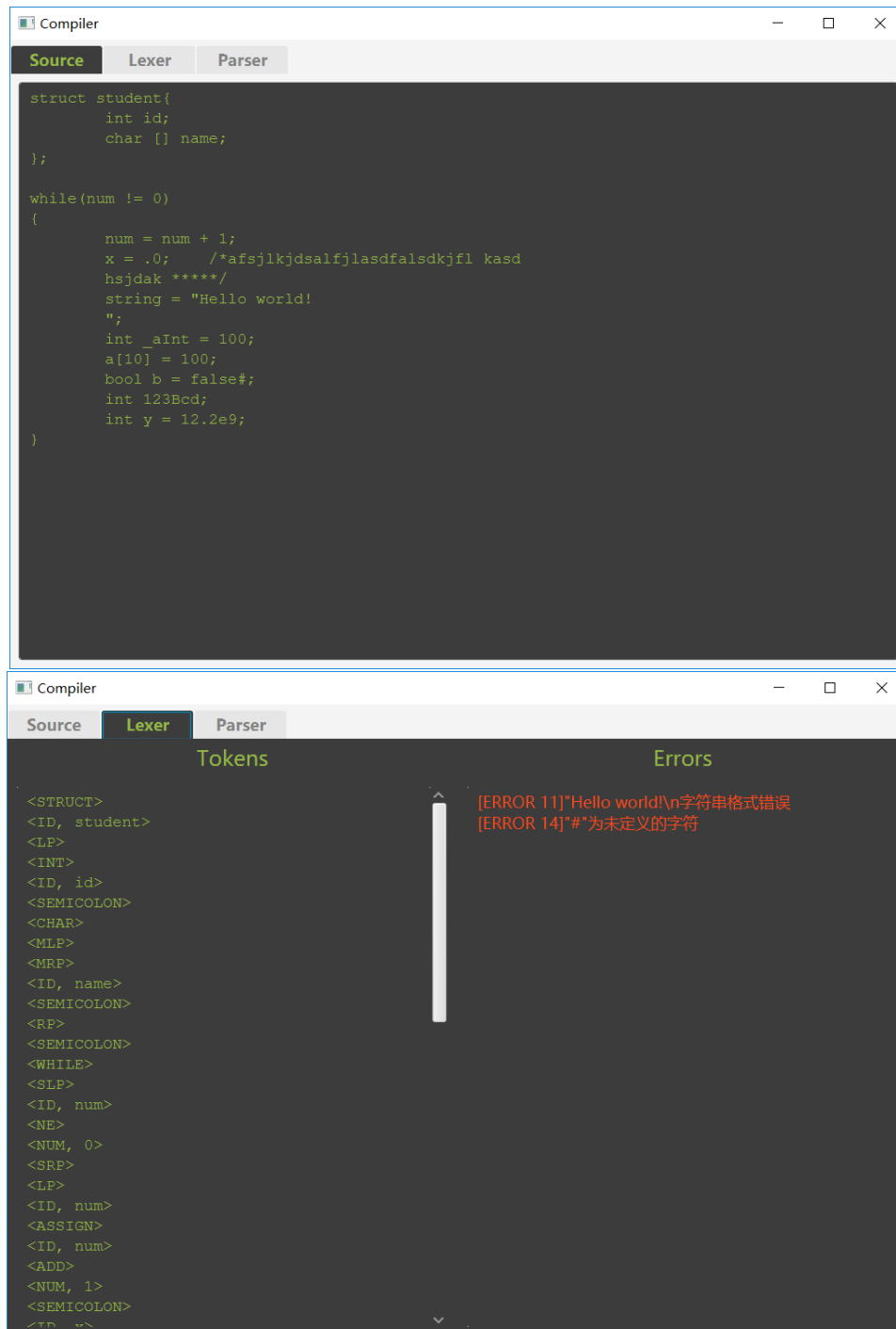
[ERROR 11]"Hello world!\n 字符串格式错误

[ERROR 14]"#"为未定义的字符

(4) 对于这段源程序，其中包含各类关键字、算符、标识符、常数以及注释，在最终的词法分析的 token 序列中可以将各类单词区分开，并且将注释和空白符过滤掉。对于错误信息，输出为错误的位置，以及错误的信息。

此外对于符号表，对于本实验在词法分析时，仅仅可以填充相应的标识符的入口地址，在此处将符号表处理为嵌套结构的 Map 结构，其中 key 为标识符的串，故将输出 token 序列的标识符可以直接引用得到对应的符号表条目。

实验界面



The image displays two screenshots of a compiler application window titled "Compiler".

The top screenshot shows the "Source" tab. It contains the following code:

```
struct student{
    int id;
    char [] name;
};

while(num != 0)
{
    num = num + 1;
    x = .0; /*afsjlkjdsalfjlasdfalsdkjfl kasd
hsjdak *****/
    string = "Hello world!";
    int _aInt = 100;
    a[10] = 100;
    bool b = false#;
    int 123Bcd;
    int y = 12.2e9;
}
```

The bottom screenshot shows the "Lexer" tab. It is divided into two sections: "Tokens" and "Errors".

The "Tokens" section lists the following tokens:

```
<STRUCT>
<ID, student>
<LP>
<INT>
<ID, id>
<SEMICOLON>
<CHAR>
<MLP>
<MRP>
<ID, name>
<SEMICOLON>
<RP>
<SEMICOLON>
<WHILE>
<SLP>
<ID, num>
<NE>
<NUM, 0>
<SRP>
<LP>
<ID, num>
<ASSIGN>
<ID, num>
<ADD>
<NUM, 1>
<SEMICOLON>
<RP>
<RP>
```

The "Errors" section displays the following error messages:

```
[ERROR 11]"Hello world!\n"字符串格式错误
[ERROR 14]"#"为未定义的字符
```

指导教师评语：

日期：