1. comment 处理

第一次想的比较简单，想通过 \/*[.|\n]*\/

这样一个正则表达式来匹配，在测试用例的时候发现不能处理嵌套，于是转而用

```
\/\* {adjust();comment_level_ = 1; begin(StartCondition__::COMMENT);}
<COMMENT> {
  \*\/ {adjustStr();if(--comment_level_ == 0) begin(StartCondition__::INITIAL);}
  \n {adjustStr();}
  \/\* {adjustStr(); comment_level_++;}
  . {adjustStr();}
}
```

可以处理嵌套 comment 和换行符，到最外层嵌套时终止。

2. string 处理

刚开始同样想通过 \"[^\"]\" 这样的表达式来匹配，测试时发现诸多问题，转而用和 comment 一样的 StartCondition，在处理异常的时候才发现 string 里面有这么多例外。

```
\"      {adjust();string_buf_ = ""; begin(StartCondition__::STR);}
<STR>{
  \" {adjustStr();begin(StartCondition__::INITIAL);setMatched(string_buf_); return
Parser::STRING;}
  \\n {adjustStr();string_buf_ += '\n';}
  \\t {adjustStr();string_buf_ += '\t';}
  \\[0-9]{3} {adjustStr(); string_buf_ += (char)atoi(matched().c_str() + 1);}
  \\\" {adjustStr(); string_buf_ += '\"';}
  \\\\ {adjustStr(); string_buf_ += '\\';}
  \\[ \n\t\f]+\\ {adjustStr();}
  \\\^[A-Z] {adjustStr(); string_buf_ += matched()[2] - 'A' + 1;}

  . {adjustStr();string_buf_ += matched();}
}
```

3. error 处理

不能匹配所有已知规则的 token 统一报错

4. eof 处理

```
<<EOF>> return 0;
```

5. other features

无