

# PA1报告

---

刘明君18307130064

韩晓宇18300750006

## 1 核心代码工作原理

### 1.1 flex文件格式

```
definitions
%%
rules
%%
user code
```

在本次实验中，我们需要完成`cool.flex`中的`rules`部分，同时我们可以在`definitions`部分对我们在`rules`部分中需要使用的事物进行定义，方便后续使用。

### 1.2 definitions

#### 1.2.1 基本使用

在`definitions`的`%{...%}`中，我们可以利用`c`语法来包含一些我们所需要的头文件，声明一些变量等，形如：

```
%{
    #include <stringtab.h>
    ...
    #define yylval cool_yylval
    ...
    extern int curr_lineno;
%}
```

在这之后，我们可以在definitions中对一部分名称进行定义，具体规则如下：

name	definition
------	------------

注：在最开始运行make dotest时会报错，经网上查询解决方法为在该部分开始前加入%option noyywrap语句

### 1.2.2 工作

在该部分，我们的主要工作是进行了一些名称定义，以便简化我们之后rules部分的代码。

首先，由于我们在最后输出结果的时候需要单独列出cool代码中的关键字，且经过标准程序测试后发现这些关键字大小写不敏感，故我们先使用名称定义来定义每个关键字的所有形式，方便之后处理，形式如下：

CLASS	[Cc][Ll][Aa][Ss][Ss]
ELSE	[Ee][Ll][Ss][Ee]
...	

需要注意，true/false关键字开头字母必须为小写，否则会被解析成TYPEID。

除此之外，我们还定义了一些常用的正则组合，比如大小写字母，字符等：

UPPER	[A-Z]
LOWER	[a-z]
CHR	[a-zA-Z0-9_]

以上便是我们在这个部分的主要工作，需要注意的是，状态定义会放在之后另行介绍。

## 1.3 rules

### 1.3.1 基本使用

在该部分，我们的主要工作是解析文法，对于每一种形式，给出对应的行为，具体形式如下：

pattern	action
---------	--------

### 1.3.2 状态定义

我们可以在definitions部分中定义我们所需要的状态，比如 `sCOMMENT` 为单行注释状态，定义方式如下：

```
%S sCOMMENT
```

我们会根据遇到的每一种可能的状态，分别介绍我们代码的工作原理。

### 1.3.3 INITIAL状态

该状态为初始的默认状态，一般用来处理cool代码正文。

#### 关键字

若遇到我们之前所定义的关键字，直接返回大写关键字即可，需要注意，`true/false`关键字需要设置 `yyval.boolean` 的值，并且返回 `BOOL_CONST`。

#### [0-9]+

用来匹配数字，需要将其放入 `inttable` 中，并返回 `INT_CONST`。

#### {UPPER}{CHR}\*

用来匹配 `TYPEID`，需要将其放入 `idtable` 中，并返回 `TYPEID`。

#### {LOWER}{CHR}\*

同上，不过此处返回 `OBJECTID`。

`".|'|"@|"~|"*|"/|"+|"-|"<|=|"{|"}|";|"(|)"|":|",|"`

cool语言中的合法字符，直接解析返回字符本身即可

#### [\t\f\r\v]+

匹配cool代码中的各类空格，匹配完后扔掉即可。

`\n`

匹配换行符，遇到后需要 `curr_lineno+1` 表示行数增加。

`{DARROW}`

匹配定义中的箭头

`<=`

匹配 `<=`

`<-`

匹配 `ASSIGN`

`"(*"`

匹配多行注释开始符号，同时需要跳到 `mCOMMENT` 状态，`nestcom++` 表示当前处在第一个多行注释之中。

`"*)"`

匹配多行注释结束符号，但在 `INITIAL` 状态中遇到则表示有无法匹配 `(*)` 符号，这种情况是一种错误，需要设置 `yylval.error_msg` 并返回 `ERROR`。

`"--"`

匹配单行注释开始符号，同时需要跳到 `sCOMMENT` 状态。

`"\"`

匹配双引号，表示字符串开始符号，同时需要跳到 `STR` 状态，并使用 `string_buf` 来记录接下来的字符串。

`.`

表示匹配除了上述形式之外的其他符号，这些符号不属于 `cool` 语法，故需要设置 `yylval.error_msg` 并返回 `ERROR`。

### 1.3.4 sCOMMENT状态

该状态为单行注释状态，在其中我们仅需要考虑几种情况。

#### EOF

合法结束，回到 `INITIAL` 状态。

#### `\n`

遇到换行符，表示单行注释结束，回到 `INITIAL` 状态，同时 `curr_lineno+1`。

.

注释中的其他符号直接无视即可以，匹配后不需要做任何操作。

### 1.3.5 mCOMMENT状态

该状态为多行注释状态

#### EOF

非法结束，需要返回 `ERROR`，同时回到 `INITIAL` 状态。

#### `"(*"`

`nestcom++`，表示处在多行注释中的个数+1

#### `"*)"`

`nestcom--`，表示处在多行注释中的个数-1，如果减完为0则表示多行注释结束，回到 `INITIAL` 状态。

#### `\n`

遇到换行符，由于是多行注释故注释并未结束，`curr_lineno++` 表示行数+1即可。

.

注释中的其他符号直接无视即可以，匹配后不需要做任何操作。

### 1.3.6 STR状态

该状态为字符串状态，表明目前正在处于字符串之中。

**\0**

字符串中的`\0`为空字符，在此处不合法，故需要跳到`STRbad`状态并返回`ERROR`。

**EOF**

字符串中遇到`EOF`不合法，需要跳到`INITIAL`状态（因为`EOF`已经表示文件结束了，无需跳到`STRbad`状态）并返回`ERROR`。

**“\”**

表示字符串中用来转义的斜杠，此处双斜杠是在`flex`代码中转义的斜杠，实际上在`cool`代码的字符串中表示一个斜杠。在这种情况下跳到`STResp`状态处理转义字符。

**\n**

字符串中遇到换行符非法，应当回到`INITIAL`状态并返回`ERROR`，同时需要将当前行数+1

**"\""**

在该状态遇到引号，表示字符串结束，需要回到`INITIAL`状态，同时将`string_buf_ptr`中存储的字符串放入`stringtable`中，并返回`STR_CONST`。

.

匹配字符串中的其他符号，同时将这些符号放入`string_buf_ptr`中，并记录当前的长度，如果当前的长度为1025，表示已经超过了最大的字符串长度，此时返回`ERROR`，其余情况不需要进行额外操作。

### 1.3.7 STResp状态

该状态主要用来处理字符串中遇到转义字符的情况。

**\0**

我们无法转义空字符，因此在这种情况下需要返回 **ERROR**。

**EOF**

我们也无法转义EOF，因此在这种情况下需要返回 **ERROR**。

**n**

如果遇到n，则该转移字符为\n，表示换行符，我们需要在 `string_buf_ptr` 中加入该转义字符，同时将其当成一个普通字符检查当前字符串的长度是否超过最大值，并回到 **STR** 状态。由于这个换行符实在字符串中，并不是cool代码中的换行符，因此我们不需要让当前行数+1。

同理，遇到b、t、f的处理方式类似，不再赘述。

**\n**

表示转义了一个换行符，实际上本身就是一个换行符，与n的处理方式基本相同，但由于转义的字符是换行符，表明其在cool代码中也有换行操作，故需要让当前行数+1。

.

如果转义其他字符，则就表示其他字符本身，其余处理方式与上文一致。

### 1.3.8 STRbad状态

**\n**

根据handout，在此状态下，我们认为字符串结束，回到初始状态，同时让当前行数+1。

**"\\\"n**

表示转义了换行符，属于特殊情况，仅让当前行数+1即可。

"\\".

对于其他的转义字符，无视即可。

"\""

根据handout，在此状态下，我们认为字符串结束，回到初始INITIAL状态。

EOF

遇到EOF，则直接回到初始INITIAL状态。

.

对于其他字符，无视即可

## 2 测试

### 构造测试用例

首先通过 `make dotest` 测试 `test.cl` 的解析结果。之后构造更多测试样例验证代码的正确性。由于会有比较多的测试，我们先写了一个 `checker.py` 来进行自动化测试，可以将 `tests` 文件夹的所有文件都使用 `bin/lexer` 和我们的解析器进行解析，之后判断是否相同。

我们自行构造了若干测试用例，具体描述如下，同时还在网上找了一些测试输入。

下面 `string` 中表示文件内容，可见字符直接表示，不可见字符通过形如 `\x0` 的方式表示，出现的字符串 `long` 在文件中表示 256 个 `long` 字符串。

### 1.cl

一个很长的串，一些 `\0,\00,\000` 相关的串。

几个 `>=` 一类的二元符号。

文件结尾有个转义掉的 `\n`，接着EOF。



## EOF

EOF系列的文件末尾都没有 `\n`

### **eof1.cl**

`--` 应该正常结束，不报错

### **eof2.cl**

`"` 字符串中包含EOF。

### **eof3.cl**

`"\` 报错字符 `\` 不能出现在文件末尾，即包含被转义过的 EOF（handout里面没提到）。

### **eof4.cl**

`\` 报错未知符号。

### **eof5.cl**

`-` 只有一个减号。

### **eof6.cl**

`(*` 报错多行注释包含EOF。

### **eof7.cl**

`"\x0"` 报错两个字符串分别包含空字符和EOF。

## eof8.cl

`"\x0"` 报错字符串包含空字符。

## eof9.cl

`"\x0\"` 报错字符串包含空字符，这里同时出现了多种错误，应输出第一个遇到的错误。

## eof10.cl

`"long\"` 报错字符串过长，这里同时出现了多种，应输出第一个遇到的错误。

## error4.1.cl

键盘中数字键上面的特殊字符

`"long", "long, "long\"` 这里都输出字符串过长。

```
*) *) **) **) )*) **)*) * ) ) *
```

## error4.1\_2.cl

`"long\"asd"` 这里过长之后有一个转义后的 `"` 需要吸收掉，只输出一个过长的串。这里最开始写的时候进入了错误字符串就无条件只看下一个 `"` 了，是有问题的。

## error4.1\_3.cl

```
"long\"asd", "long\n\"asd", "long\\asd", "long\\"asd", "long\\\\"asd", "long\\"as  
d"
```

前三行过长，第四行过长，之后 `asd` 的 object，未结束的字符串。第五行过长

## **error4.1\_4.cl**

过长的串之后有些转义后的 `\x16`（换行符）。

## **let\_stmt.cl**

`LET_STMT`，`let_stmt` 不特殊处理。

## **null1.cl**

`\x0\x0cs\x0\x0\sca` 未知字符 `\x0`

## **null2.cl**

`"\x0\x0cs\x0\x0\sca"` 字符串里有空字符。

## **null3.cl**

一些不超过最长长度但是包含空字符的串。

## **null4.cl**

开头就有空字符的串。

## **null5.cl**

开头就有空字符的超长串。

## **null6.cl**

超长，末尾有空字符，并且没有结束 `"` 的串。

## **null7.cl**

超长，末尾有空字符，并且没有结束 `"` 的串。

## **null8.cl**

`"\\x0"` 输出转义过的空字符。（handout中没提到）

## **null9.cl**

`"\\x0` 输出转义过的空字符。（handout中没提到）

## **right1.cl**

一些关键字。

## **str1.cl**

`\x0\x1\x2` 一直到 `\x127`。

## **str2.cl**

引号中的 `\x0\x1\x2` 一直到 `\x127`。

## **str3.cl**

`"\\x0\\x1\\x2` 一直到 `\\x127` 但是没有后面的引号。

## 遇到的问题

在 `eof1.cl` 中单行注释如果遇到 EOF 应该正常结束，而不是报错，这里在理解manual时候发生了一些偏差。

`eof2.cl` 中如果想让一个字符串碰到 EOF 而报错的话，要么在行末加一个转义字符 `\`，要么使用程序来生成测试文件。因为正常的编辑器在保存文件时都会在文件的末尾增加一个换行符。

`eol3.cl` 中自然需要测试如果是 `"\"` 而文件末尾没有换行符的情况，这时需要输出 `"\"` 出现在文件末尾“错误，这是 `manual` 里面没有提到的。

`error4.1_2.cl` 中，字符串过长以后，我们的实现方式是进入一个 `STRbad` 的状态，并且吸收掉后面的字符，直到碰到 `"` 或者换行，这里如果 `"` 实际是一个被转义的引号，就不能结束这个字符串。也就是说字符串报错之后也需要正常处理转义的情况。

`null8.cl` 中测试了被转义的空字符 `\\x0`，这里需要报错“遇到转义后的空字符”错误，这个也是 `manual` 里面没有提到的。（而且这两个空字符的情况报错的末尾有个句号，和 `manual` 里面说的不一样）

在网上找到的测试中，我们发现多行注释的逻辑是进行嵌套，而不是匹配第一个 `*)`。