# 目录

[1 目录 1](#_Toc513585235)

[2 词法分析详细设计 2](#_Toc513585236)

[2.1 词法分析概述 2](#_Toc513585237)

[2.2 预处理 3](#_Toc513585238)

[2.3 lex介绍 3](#_Toc513585239)

[2.3.1 lex概述 3](#_Toc513585240)

[2.3.2 lex源程序的结构 4](#_Toc513585241)

[2.3.3 lex具体用法 4](#_Toc513585242)

[2.3.3.1 转义字符 4](#_Toc513585243)

[2.3.3.2 “环境” 4](#_Toc513585244)

[2.3.3.3 lex匹配原则 4](#_Toc513585245)

[2.3.4 flex安装、配置及使用 5](#_Toc513585246)

[2.3.5 转化为C++ 5](#_Toc513585247)

[2.4 单词整理 5](#_Toc513585248)

[2.5 数据结构 7](#_Toc513585249)

[2.5.1 自身定义数据结构 7](#_Toc513585250)

[2.5.1.1 charRec 7](#_Toc513585251)

[2.5.1.2 lineBuffer 7](#_Toc513585252)

[2.5.1.3 lexicalErrorInformatin 7](#_Toc513585253)

[2.5.1.4 yycolumn 7](#_Toc513585254)

[2.5.2 引用外部的数据结构 7](#_Toc513585255)

[2.5.2.1 yylloc 7](#_Toc513585256)

[2.5.3 lex内置数据结构 8](#_Toc513585257)

[2.5.3.1 yyin 8](#_Toc513585258)

[2.5.3.2 yytext 8](#_Toc513585259)

[2.5.3.3 yyleng 8](#_Toc513585260)

[2.5.3.4 yylineno 8](#_Toc513585261)

[2.5.3.5 yylval 8](#_Toc513585262)

[2.6 错误处理 9](#_Toc513585263)

[2.6.1 错误种类及处理对策 9](#_Toc513585264)

[2.6.2 错误信息格式设计 9](#_Toc513585265)

[2.6.2.1 普通错误信息格式 9](#_Toc513585266)

[2.6.2.2 行超长错误格式 10](#_Toc513585267)

[2.6.2.3 标识符超长错误格式 10](#_Toc513585268)

[2.7 函数和过程 10](#_Toc513585269)

[2.7.1 自身定义函数或过程 10](#_Toc513585270)

[2.7.1.1 添加词法错误信息 10](#_Toc513585271)

[2.7.1.2 检查长度限制 11](#_Toc513585272)

[2.7.1.3 初始化终结符的位置信息 12](#_Toc513585273)

[2.7.2 引用的外部函数或过程 12](#_Toc513585274)

[2.7.2.1 将整型数字转化为字符串 12](#_Toc513585275)

[2.7.3 lex内置函数或过程 13](#_Toc513585276)

[2.7.3.1 yywrap 13](#_Toc513585277)

[2.7.3.2 yylex 13](#_Toc513585278)

[2.7.3.3 yyless 13](#_Toc513585279)

[2.8 正则定义式 13](#_Toc513585280)

[2.9 规则动作详细设计 14](#_Toc513585281)

[2.9.1 概述 14](#_Toc513585282)

[2.9.2 环境介绍 14](#_Toc513585283)

[2.9.3 INITIAL环境 14](#_Toc513585284)

[2.9.3.1 空白符 14](#_Toc513585285)

[2.9.3.2 缓存下一行 15](#_Toc513585286)

[2.9.3.3 各种单词通用设计 15](#_Toc513585287)

[2.9.3.4 单引号，进入CH环境 16](#_Toc513585288)

[2.9.3.5 双斜杠，进入SCOM环境 16](#_Toc513585289)

[2.9.3.6 左花括号，进入MCOM环境 16](#_Toc513585290)

[2.9.3.7 非法字符 16](#_Toc513585291)

[2.9.4 CH环境 17](#_Toc513585292)

[2.9.4.1 文件尾 17](#_Toc513585293)

[2.9.4.2 单引号或换行符，进入INITIAL环境 17](#_Toc513585294)

[2.9.4.3 其它字符 18](#_Toc513585295)

[2.9.5 SCOM环境 18](#_Toc513585296)

[2.9.5.1 文件尾 18](#_Toc513585297)

[2.9.5.2 换行符，进入INITIAL环境 18](#_Toc513585298)

[2.9.5.3 其它字符 19](#_Toc513585299)

[2.9.6 MCOM环境 19](#_Toc513585300)

[2.9.6.1 文件尾 19](#_Toc513585301)

[2.9.6.2 缓存下一行 19](#_Toc513585302)

[2.9.6.3 右花括号，进入INITIAL环境 20](#_Toc513585303)

[2.9.6.4 其它字符 20](#_Toc513585304)

[2.10 对外接口 20](#_Toc513585305)

[2.10.1 入口 20](#_Toc513585306)

[2.10.2 出口 20](#_Toc513585307)

# 词法分析详细设计

## 词法分析概述

在预处理的详细设计中提到{loc}了需要将源程序的最前面加入一个换行符，同时{loc}中介绍了行号yylineno，实际上，由于我们在最前面加入了一个换行符，yylineno指示的行号会比初始程序中真正的行号大1，在下述文档中，我们仍以yylineno表示行号，在代码实现时，需要用yylineno-1。

## 预处理

PASCAL-S大小写不敏感，因此需要将PASCAL-S源程序中的所有大写字母转化为小写字母。

同时也需要在PASCAL-S源程序的最前面加入一个换行符，我们先不用考虑原因（后文会有解释），总之这是为了方便lex预读取行，便于提供更加详细的报错信息。

需要设计一个函数，可以从输入文件读入PASCAL-S源程序，在最开始加入一个换行符，并将所有的大小字母转化为小写字母。

预处理的结果保存在”preProcessed.pas”中，其后真正的利用lex进行词法分析的输入文件实际上就是这个保存了预处理后的PASCAL-S源程序的文件。

* 函数接口

1. **void** preProcess(string inName);

* 返回值

无

* 参数列表

|  |  |
| --- | --- |
| 参数 | 描述 |
| string inName | 输入文件名 |

* 伪代码

1. **void** preProcess(string inName){
2. 将输入文件关联到输入文件流fin;
3. 将"preProcessed.pas"关联到输出文件流fout;
4. string str;
5. **while** (从输入文件流获取一行，保存到str中){
6. 将str中的大写字母转化为小写字母;
7. fout << endl << str;
8. }
9. 关闭输入文件流;
10. 关闭输出文件流;
11. }

## lex介绍

### lex概述

LEX是LEXical compiler的缩写，是UNIX环境下非常著名的工具软件，其主要功能是根据LEX源程序生成一个用C语言描述的词法分析程序。LEX源程序是词法分析程序的规格说明文件，其文件名约定为lex.l，经过LEX编译程序的编译，生成一个C语言程序lex.yy.c。

### lex源程序的结构

一个LEX源程序由声明、翻译规则和辅助过程3部分组成，各部分之间用双百分号“%%”隔开。

声明部分可以包括变量的声明、符号常量的声明和正规定义。正规定义中定义的名字可以出现在翻译规则的正规表达式中。希望出现在输出文件lex.yy.c中的C或C++语言声明语句必须用符号“%{”和“%}”括起来。

翻译规则部分是由正规表达式和相应的动作组成的具有如下形式的语句序列：

P1 {动作1}

P2 {动作2}

……

Pn {动作n}

其中Pi（i=1,2，……，n）是一个正规表达式，描述一种记号的模式；动作i是用C或C++语言描述的程序段，表示当一个符号串匹配模式Pi时，词法分析程序应执行的动作。

辅助过程是对翻译规则的补充。翻译规则部分中某些动作需要调用的过程或函数，如果不是C或C++语言的库函数，则要在此给出具体的定义。这些过程或函数也可以再另一个程序文件中定义，然后再和词法分析程序链接在一起即可。

### lex具体用法

**这里主要对我们在编写词法分析器时涉及的lex的使用细节、用到的一些lex内置变量和函数等进行说明。具体的用法还需查阅相关文档。**

#### 转义字符

在书写正规定义式/表达式时，**”\[]^-?.\*+|()$/{}%<>**，这些字符具有特殊函数，不能用来匹配自身，如果需要匹配自身，可以通过引号”或转移字符\来指示。

#### “环境”

LEX中有环境的概念，利用环境可以很方便地识别字符常量、字符串、单行注释、多行注释等内容。所有不带环境说明的正规表达式都是默认在INITIAL环境下。LEX在运行时，可以根据利用BEGIN语句在环境之间进行转移。在不同环境中识别同一个单词可以执行不同的动作。

以多行注释（以下用MCOM指代其环境）为例，当在INITIAL环境中遇到{时，就说明接下来开始一段多行注释的内容，此时就进入多行注释环境MCOM，在该环境中，不管遇到什么符号，都可以忽略，直到遇到}，该符号表示多行注释结束，显然此时应该返回到INITIAL环境中，开始正常的单词和符号的识别。

#### lex匹配原则

由LEX生成的词法分析程序在识别单词符号时，遵循以下匹配原则：

* **最长匹配原则**：当有几条规则都适用时，实施匹配最长输入串的那个规则。
* **有限匹配原则**：当有几条规则都适用，并且匹配长度相同时，则实施排在最前面的那条规则。

也就是说，词法分析程序依次尝试每一条规则，尽可能地匹配最长的输入符号串，并且，排在前面的规则的优先级高于排在其后的规则的优先级。另外，如果有一些内容不匹配任何规则，则LEX将会把它拷贝到标准输出。

### flex安装、配置及使用

到官网下载flex的exe安装程序，安装成功后，将其所在目录添加到系统环境变量，即可在命令行中进行调用。

假如我们的LEX源程序是lex.l，那么只需要在命令行，进入到目录中后，键入下列命令：

1. flex lex.l

即可获得词法分析程序lex.yy.c。

### 转化为C++

由于我们的编译程序的目标语言是C++，所以需要将LEX输出的lex.yy.c改名为lex.yy.cpp；同时lex自动生成的一些函数接口的声明，需要用extern “C”进行包括，extern "C"的主要作用就是为了能够正确实现C++代码调用其他C语言代码。加上extern "C"后，会指示编译器这部分代码按C语言的进行编译，而不是C++的。

## 单词整理

分析了PASCAL-S的语法后，以表格的形式给出所有PASCAL-S程序中可能出现的单词的正则表达式、记号和属性。PASCAL-S有效单词表如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 描述 | 正则表达式 | 属性 | 记号 |
| 标识符 | **[a-z][a-z0-9]\*** | **该标识符本身** | **IDENTIFIER** |
| 无符号整数 | **[0-9]+** | **该整数本身（字符串表示）** | **UINUM** |
| 无符号浮点数 | **[0-9]+\.[0-9]+** | **该浮点数本身（字符串表示）** | **UFNUM** |
| 字符常量 | **‘.’** | **该字符常量本身（不包含两侧的单引号）** | **CHAR** |
| 关系运算符 | **>=** | **>=** | **RELOP** |
| **>** | **>** |
| **<=** | **<=** |
| **<>** | **<>** |
| **<** | **<** |
| 关系运算符：相等  常量初始化：赋值 | **=** | **=** | **=** |
| 算术运算符：加法 | **+** | **+** | **ADDOP** |
| 逻辑运算符：或 | **or** | **or** |
| 算术运算符：减法（双目）  算术运算符：取相反数（单目） | **-** | **-** | **-** |
| 算术运算符：乘法 | **\*** | **\*** | **MULOP** |
| 算术运算符：除法 | **/** | **/** |
| 算术运算符：整除 | **div** | **div** |
| 算术运算符：取余 | **mod** | **mod** |
| 逻辑运算符：且 | **and** | **and** |
| 逻辑运算符：非 | **not** | **not** | **NOT** |
| 赋值符号 | **：=** | **：=** | **ASSIGNOP** |
| 范围连接符 | **..** | **..** | **RANGEDOT** |
| 界符 | **(** | **(** | **(** |
| **)** | **)** | **)** |
| **[** | **[** | **[** |
| **]** | **]** | **]** |
| **:** | **:** | **:** |
| **,** | **,** | **,** |
| **;** | **;** | **;** |
| **.** | **.** | **.** |
| 关键字 | **program** | **program** | **PROGRAM** |
| 关键字 | **const** | **const** | **CONST** |
| 关键字 | **var** | **var** | **VAR** |
| 关键字 | **array** | **array** | **ARRAY** |
| 关键字 | **of** | **of** | **OF** |
| 关键字 | **procedure** | **procedure** | **PROCEDURE** |
| 关键字 | **function** | **function** | **FUNCTION** |
| 关键字 | **begin** | **begin** | **BEGIN** |
| 关键字 | **end** | **end** | **END** |
| 关键字 | **if** | **if** | **IF** |
| 关键字 | **then** | **then** | **THEN** |
| 关键字 | **for** | **for** | **FOR** |
| 关键字 | **to** | **to** | **TO** |
| 关键字 | **do** | **do** | **DO** |
| 关键字 | **else** | **else** | **ELSE** |
| 关键字 | **repeat** | **repeat** | **REPEAT** |
| 关键字 | **until** | **until** | **UNTIL** |
| 关键字 | **while** | **while** | **WHILE** |
| 关键字 | **integer** | **integer** | **TYPE** |
| 关键字 | **real** | **real** |
| 关键字 | **char** | **char** |
| 关键字 | **boolean** | **boolean** |

注意，上表并没有包含全部的需要用到的正则表达式，其余的情况不属于PASCAL-S的有效单词，将在后续的翻译规则部分的具体实现中进行讲述。

## 数据结构

### 自身定义数据结构

#### charRec

1. string charRec; //在字符环境中，缓存所读入的字符

charRec是string类型的变量，用于缓存在字符环境中，所读入的所有字符，以便后续处理。

#### lineBuffer

1. **char** lineBuffer[10005]; //保存当前行的所有内容

lineBuffer是一个长度为10005的char型数组，用于保存当前行的所有内容。

#### lexicalErrorInformatin

1. vector<string> lexicalErrorInformation;//用于保存词法错误信息的列表

lexicalErrorInformatin是一个string的容器，按顺序保存所有的词法错误信息，以便后续输出。

#### yycolumn

1. **int** yycolumn; //表示当前符号所在某一行的具体位置（或者说列数）

### 引用外部的数据结构

#### yylloc

1. **extern** YYLTYPE yylloc;

YYLTYPE是YACC中用于表示终结符或非终结符的位置的结构。其默认定义如下：

1. **typedef** **struct**{
2. **int** first\_line;
3. **int** first\_column;
4. **int** last\_line;
5. **int** last\_column;
6. }YYLTYPE;

其中first\_line表示当前文法符号代表的内容的起始行，first\_column表示当前文法符号代表的内容的起始列，last\_line表示当前文法符号代表的内容的终止行，last\_column表示当前文法符号代表的内容的终止列。

词法分析程序需要给所有的终结符初始化位置信息。

### lex内置数据结构

#### yyin

1. **FILE** \*yyin;

lex提供了一个FILE\*类型的指针yyin，将输入文件打开后获得的文件指针赋值给yyin，即可实现从文件输入。

#### yytext

1. **char** \*yytext;

匹配成功的单词由该指针指向，因此我们可以利用yytext获取当前匹配成功的单词。

#### yyleng

1. **int** yyleng;

表示当前匹配的单词的长度

#### yylineno

1. **int** yyleng;

表示当前行号。需要在LEX源程序的声明部分写入%option yylineno，开启该变量的使用。

#### yylval

1. **class** Type{
2. **public**:
3. string str;//记号的具体属性
4. string token;//记号
5. Type(){}
6. };
7. #define YYSTYPE Type\*
8. YYSTYPE yylval

YYSTYPE指的是YACC中各终结符号、非终结符号节点附带的属性类型，可以是一个普通变量，也可以是一个复杂的结构体。在YACC中，我们直接将其作为了语法分析树的节点类型，因此是一个较为复杂的结构类型。上述代码所展现的**类Type**的定义已经被简化，只留下了词法分析程序需要用到的域。

YYSTYPE是指向Type类的指针类型。

yylval变量起到了在词法分析程序和语法分析程序间传递信息的作用。

## 错误处理

### 错误种类及处理对策

1. **行长度超过限制**，本编译器设置为10000；遇到该错误时报错，且词法分析器立即停止运行
2. **标识符长度超过限制**，本编译器设置为100；遇到该错误时报错，并对标识符进行截断处理，只取前100个字符，这可能会造成后续语义分析中出现重定义的错误
3. **非法字符**，遇到该错误时报错并跳过该非法字符
4. **读取字符常量时遇到文件尾**，遇到该错误时报错，且词法分析器立即停止运行
5. **读取的字符常量为空**；遇到该错误时报错，并返回"\0"这一字符常量
6. **读取的字符常量不止一个字符**；遇到该错误时报错，并提交第一个字符作为当前识别的字符常量
7. **读取字符常量时，先遇到了换行符而不是右单引号**，也就是右单引号缺失的情况；遇到该错误时报错，如果之前没有识别任何字符，则返回”\0”，如果之前识别了字符，则返回第一个字符
8. **读取多行注释时遇到文件尾**，也就是右花括号缺失的情况；遇到该错误时报错，且词法分析器立即停止运行

### 错误信息格式设计

格式中，{变量}表示显示变量的值。

#### 普通错误信息格式

错误3、4、5、6、7、8均使用该错误信息格式。将如下表所示的信息

|  |  |
| --- | --- |
| 定义 | 描述 |
| string info | 具体错误信息描述 |
| int l | 起始列 |
| int r | 终止列 |
| int yylineno | 行号 |

组织为：

第一行：[{info}] {yylineno}.{l}-{yylineno}.{r}

第二行：{lineBuffer}

第三行：在l位置到r位置之间输出”^”，l位置之间输出空格

这六种错误的info信息如下表所示：

|  |  |  |
| --- | --- | --- |
| 错误编号 | 描述 | info |
| 错误3 | 非法字符 | Invalid character! |
| 错误4 | 读取字符常量时遇到文件尾 | Unexpected end of file when reading a char constant |
| 错误5 | 读取的字符常量为空 | Char constant missing! |
| 错误6 | 读取的字符常量不止一个字符 | Too many characters in a char constant! |
| 错误7 | 读取字符常量时，先遇到了换行符而不是右单引号 | Right quote missing! |
| 错误8 | 读取多行注释时遇到文件尾 | Unexpected end of file when reading a multiple line comment, lacking of a right brace |

#### 行超长错误格式

错误1使用该错误信息格式，将如下表所示的信息

|  |  |
| --- | --- |
| 定义 | 描述 |
| int l | 起始列 |
| int r | 终止列 |
| int yylineno | 行号 |

组织为：[Line length too large, exceed 10000] {yylineno}.{l}-{yylineno}.{r} Lex analyse abort!

#### 标识符超长错误格式

错误2使用该错误信息格式，将如下表所示的信息

|  |  |
| --- | --- |
| 定义 | 描述 |
| int l | 起始列 |
| int r | 终止列 |
| int yylineno | 行号 |

组织为：[Identifier length too large, exceed 100] {yylineno}.{l}-{yylineno}.{r}

## 函数和过程

### 自身定义函数或过程

#### 添加词法错误信息

* 函数接口

1. **void** addLexicalErrorInformation(**char** \*word, string info, **int** l, **int** r);

* 返回值

void

* 参数列表

|  |  |
| --- | --- |
| 参数 | 描述 |
| char \*word | 当前单词指针 |
| string info | 具体错误信息 |
| int l | 起始列数 |
| int r | 终止列数 |

* 函数伪代码

1. **void** addLexicalErrorInformation(**char** \*word, string info, **int** l, **int** r){
2. 按照1.5.2.1{loc}中指示的格式将参数提供的信息组装成一条完整的、结构化的错误信息
3. lexicalErrorInformation.push\_back(错误信息)
4. }

#### 检查长度限制

* 函数接口

1. **bool** CheckAndAddLengthTooLargeErrorInformation(**char** \*text, string type, **int** l, **int** r)

* 返回值

bool，true表示检测到长度超过限制的错误，false表示没有检测到长度超过限制的错误

* 参数列表

|  |  |
| --- | --- |
| 参数 | 描述 |
| char \*text | 当前单词 |
| string type | “line”表示检测行长度，”identifier”表示检测标识符长度 |
| int l | 起始列 |
| int r | 终止列 |

* 函数伪代码

1. **bool** CheckAndAddLengthTooLargeErrorInformation(**char** \*text, string type, **int** l, **int** r){
2. **if**(type=="line"){
3. **if**(当前行长度>10000){
4. 按照1.5.2.2{loc}中说明的格式将已有的信息组织成一条结构化的错误信息;
5. lexicalErrorInformation.push\_back(错误信息);
6. **return** **true**;
7. }
8. **return** **false**;
9. }
10. **else**{ //type=="identifier"
11. **if**(当前标识符长度>100){
12. 按照1.5.2.3{loc}中说明的格式将已有的信息组织成一条结构化的错误信息;
13. lexicalErrorInformation.push\_back(错误信息);
14. **return** **true**;
15. }
16. **return** **false**;
17. }
18. }

#### 初始化终结符的位置信息

这是一个宏，而不是一个函数。

* 宏名

YY\_USER\_ACTION

* 需要用到的变量列表

|  |  |
| --- | --- |
| 变量 | 描述 |
| YYLTYPE yylloc | 当前单词的位置信息，该类型的定义见1.4.2.1{loc} |
| int yycolumn | 当前列数 |
| int yyleng | 当前单词的长度 |

* 伪代码

1. yylloc.first\_line = yylineno;
2. yylloc.last\_line = yylineno;
3. yylloc.first\_column = yycolumn;
4. yylloc.last\_column = yycolumn+yyleng-1;
5. yycolumn += yyleng;

* 备注

该宏会自动执行，不需要自行调用。另外yycolumn的值需要在翻译规则中进行初始化，每次遇到换行符时都要初始化为1。

### 引用的外部函数或过程

#### 将整型数字转化为字符串

* 函数接口

1. **extern** string itos(**int** num);

* 返回值

string，表示转化后的字符串

* 参数列表

|  |  |
| --- | --- |
| 参数 | 描述 |
| int num | 需要转化的整型数字 |

### lex内置函数或过程

#### yywrap

1. **int** yywrap();

这一函数在文件（或输入）的末尾调用。返回值类型为int，如果函数的返回值是1，就停止解析。因此它可以用来解析多个文件。方法是使用 yyin 文件指针指向不同的文件，直到所有的文件都被解析。最后，yywrap()可以返回1来表示解析的结束。**我们的编译器只支持同时编译一个文件**，因此只需简单的将yywrap的实现写成如下的形式：

1. **int** yywrap()
2. {
3. **return** 1;
4. }

#### yylex

1. **int** yylex();

该函数由是lex提供给外部程序的接口，返回值类型为int。如果返回值是0，就表示当前文件分析结束，如果返回值是非0，就对应了某个记号的编号。所以对于语法分析程序来说，其返回值就是记号。因此，语法分析程序可以不断的调用该程序获得记号序列，直到其返回值为0。

#### yyless

yyless(n)表示将当前所识别的单词的后yyleng-n个字符退回，留作后续匹配分析，仅保留其前n个字符。

## 正则定义式

主要参照{loc}，在LEX的声明部分，写下这些正则定义式，简化翻译规则部分的正则表达式。

|  |  |  |
| --- | --- | --- |
| 正则定义式名 | 正则表达式 | 描述 |
| line | \n.\* | 下一行 |
| letter | [a-z] | 小写字母 |
| digit | [0-9] | 数字 |
| blank\_chars | [ \f\r\t\v]+ | 空白符 |
| identifier | {letter}({letter}|{digit})\* | 标识符 |
| \_integer | {digit}+ | 无符号整数 |
| floater | {digit}+\.{digit}+ | 无符号浮点数 |
| \_type | (integer|real|boolean|char) | 基本类型 |
| relop | (>=|>|<=|<>|<) | 运算符 |
| addop | (\+|or) | 运算符 |
| mulop | (\\*|\/|div|mod|and) | 运算符 |
| delimiter | (\(|\)|\[|\]|:|,|;|\.) | 界符 |

## 规则动作详细设计

### 概述

该部分，除了包括{loc}中提到的单词的识别，还将介绍如何匹配、缓存并返还下一行；如何处理字符常量识别的各种情况；如何处理遇到文件尾的各种情况；如何处理单行注释；如何处理多行注释等；如何处理非法字符等情况。

PASCAL-S的单行注释以//开始，多行注释由一对花括号包括。

### 环境介绍

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 描述 | 进入条件 | 离开条件 |
| INITIAL | 初始环境（默认环境） | lex开始运行，或从CH、SCOM、MCOM返回 | 遇到单引号、双斜杠、左花括号 |
| CH | 字符常量识别环境 | 在INITIAL中遇到单引号 | 遇到单引号、换行符 |
| SCOM | 单行注释识别环境 | 在INITIAL中遇到双斜杠 | 遇到换行符 |
| MCOM | 多行注释识别环境 | 在INITIAL中遇到左花括号 | 遇到右花括号 |

### INITIAL环境

#### 空白符

* 正则表达式

{blank\_chars}

* 动作伪代码

无

* 说明

遇到空白符，无需采取任何动作。

#### 缓存下一行

* 正则表达式

{line}

* 动作伪代码

1. {
2. **if**(检测当前行长度超过限制)//CheckAndAddLengthTooLargeErrorInformation
3. **return** 0;
4. 将当前行保存到lineBuffer中;
5. yycolumn初始化为1;
6. 将当前行全部退回到输入区，只保留一开始的换行符;
7. }

* 说明

下一行的起始标志是换行符，因此为了识别第一行，在预处理中，我们在整个源程序的第一行之前加入了一个换行符；另外加入换行符也会影响到yylineno及行号，真正的行号应为yylineno-1。

缓存下一行的主要目的是为了输出更加详细的词法错误信息。可以先指明错误类型，然后输出错误所在行，最后用”^”符号指出错误的具体位置。

#### 各种单词通用设计

* 正则表达式

|  |  |  |  |
| --- | --- | --- | --- |
| program | const | var | array |
| of | procedure | function | begin |
| if | then | for | to |
| do | else | repeat | until |
| while | {\_type} | not | {relop} |
| {addop} | {mulop} | - | = |
| := | \.\. | {delimiter} |  |

* 动作伪代码

1. {
2. yylval指向新创建的节点;
3. 根据当前的单词种类，对yylval的记号和属性域进行赋值;
4. yylval的行号域赋值为当前行号;
5. **return** 当前记号;
6. }

* 说明

各单词的记号和属性信息在{loc}中已详细说明，不再赘述。

#### 单引号，进入CH环境

* 正则表达式

‘

* 动作伪代码

1. {
2. 进入CH环境;
3. charRec初始化为空字符串;
4. }

#### 双斜杠，进入SCOM环境

* 正则表达式

\/\/

* 动作伪代码

1. {
2. 进入SCOM环境;
3. }

#### 左花括号，进入MCOM环境

* 正则表达式

\{

* 动作伪代码

1. {
2. 进入MCOM环境;
3. }

#### 非法字符

* 正则表达式

.

* 动作伪代码

1. {
2. 添加非法字符错误;//错误3
3. }

### CH环境

#### 文件尾

* 正则表达式

<CH><<EOF>>

* 动作伪代码

1. {
2. 添加读取字符常量时遇到文件尾的错误;//错误4
3. }

#### 单引号或换行符，进入INITIAL环境

* 正则表达式

<CH>("'"|"\n")

* 动作伪代码

1. {
2. len=charRec记录的字符串的长度;
3. **if**(当前识别的是单引号 && len==0){
4. 添加字符常量为空的错误信息;//错误5
5. 返回到初始环境;
6. yylval指向新创建的节点;
7. yylval的属性为"\0";
8. yylval的记号为**CHAR**;
9. yylval的行号域赋值为当前行号;
10. **return** **CHAR**;
11. }
12. **else** **if**(当前识别的是单引号 && len==1){
13. yylval指向新创建的节点;
14. 返回到初始环境;
15. yylval的属性为charRec[0];
16. yylval的记号为**CHAR**;
17. yylval的行号域赋值为当前行号;
18. **return** **CHAR**;
19. }
20. **else** **if**(当前识别的是单引号 && len>1){
21. 添加字符常量过长的错误信息;//错误6
22. yylval指向新创建的节点;
23. 返回到初始环境;
24. yylval的属性为charRec[0];
25. yylval的记号为**CHAR**;
26. yylval的行号域赋值为当前行号;
27. **return** **CHAR**;
28. }
29. **else**{//当前识别的是换行符
30. 添加右单引号缺失的错误信息;//错误7
31. yylval指向新创建的节点;
32. 返回到初始环境;
33. **if**(len==0)
34. yylval的属性为"\0";
35. **else**
36. yylval的属性为charRec[0];
37. yylval的记号为**CHAR**;
38. yylval的行号域赋值为当前行号;
39. **return** **CHAR**;
40. }
41. }

#### 其它字符

正则表达式

<CH>.

动作伪代码

1. {
2. charRec后面拼上yytext[0];
3. }

### SCOM环境

#### 文件尾

* 正则表达式

<SCOM><<EOF>>

* 动作伪代码

1. {
2. **return** 0;//表示分析结束
3. }

* 说明

单行注释遇到文件尾是很正常的情况。

#### 换行符，进入INITIAL环境

* 正则表达式

<SCOM>"\n"

* 动作伪代码

1. {
2. 返回到初始环境;
3. 将换行符退回;
4. 当前行号减1;
5. }

#### 其它字符

* 正则表达式

<SCOM>.

* 动作伪代码

无

### MCOM环境

#### 文件尾

* 正则表达式

<MCOM><<EOF>>

* 动作伪代码

1. {
2. 添加多行注释遇到文件尾的错误信息;
3. **return** 0;//词法分析终止
4. }

#### 缓存下一行

* 正则表达式

<MCOM>{line}

* 动作伪代码

1. {
2. **if**(检测当前行长度超过限制)//CheckAndAddLengthTooLargeErrorInformation
3. **return** 0;
4. 将当前行保存到lineBuffer中;
5. yycolumn初始化为1;
6. 将当前行全部退回到输入区，只保留一开始的换行符;
7. }

#### 右花括号，进入INITIAL环境

* 正则表达式

<MCOM>"\}"

* 动作伪代码

1. {
2. 返回到初始环境;
3. }

#### 其它字符

* 正则表达式

<MCOM>.

* 动作伪代码

无

## 对外接口

### 入口

* FILE\* yyin，该指针指向打开的输入文件

### 出口

* int yylex()，该函数返回记号编号
* YYSTYPE yylval，该结构体指针用于在词法分析程序和语法分析程序间共享记号、及其属性和行号等信息