中山大學 SUN YAT-SEN UNIVERSITY

编译原理

作业名称	Oberon-0 逆向工程工具 Rose 实验二：自动生成词法分析程序		任课老师	万老师		
院系	软件学院		方向	通信软件	姓名	李明宽
学号	11331173		完成日期	2013 年 11 月 24 日		
QQ	736459905		E-mail	limkuan@mail2.sysu.edu.cn		

实验二：自动生成词法分析程序

一、 实验目的

本实验通过对词法分析程序自动生成工具 JFlex 的使用，并利用该工具自动生成 Oberon-0 语言的词法分析程序。完成 Oberon-0 源程序的词法分析。

二、 实验过程

1. 总结 Oberon-0 语言的词汇表

根据 Oberon-0 语言的 BNF 定义和语言描述,抽取 Oberon-0 语言的词汇表如下：

关键字/保留字列表	MODULE	程序模块声明	运算符列表	+	加法（取正）
	BEGIN	程序段开始声明		-	减法（取负）
	END	程序段结束声明		*	乘法运算
	CONST	常量声明		>	大于
	TYPE	自定义类型定义		>=	大于等于
	VAR	定义变量		<	小于
	PROCEDURE	定义子过程		<=	小于等于
	RECORD	定义结构体		#	不等于
	ARRAY	定义数组		=	等于
	OF	语句关键字		&	与运算
	WHILE	循环语句开始声明		~	逻辑非操作
	DO	循环语句关键字		:=	赋值运算
	IF	条件语句开始声明		.	域操作运算符
	THEN	条件语句关键字		[	数组下标左括号
	ELSIF	条件语句关键字		]	数组下标右括号
	ELSE	条件语句关键字		(	左括号
	OR	或运算关键字		)	右括号
	DIV	整数除法运算		:	类型声明标志
	MOD	取模运算		;	语句结束标志
	INTEGER	整数类型定义		,	变量间隔标志
	BOOLEAN	布尔类型定义		EOF	源程序结束标志

注：我觉得在词法分析过程中个，关键字和保留字的地位是相同的，都是匹配到相应的词之后返回一个代表这个关键字的 Token，因此不必要对其进行区分，因此把关键字和保留字合并成为一个表。但对于语法分析过程中，关键字一般来说是有实际含义的，

而保留字是使得程序结构完成消除二义性，因此在语法分析的时候处理起来会不一样。

2. 抽取 Oberon-0 语言的词法规则

在 Oberon-0 程序中，用到的词汇正则定义如下：

```
BLANK -> " " | \r | \n | \r\n | \t | \f | \b
COMMENT -> \(\*([^\*]|(\*)+[^\*\)])*(\*)*\*\)
IDENTIFIER -> [a-zA-Z][0-9a-zA-Z]*{0,23}
DECNUMBER -> [1-9][0-9]*{0,11}
OCTNUMBER -> 0[0-7]*{0,11}
```

其余的关键字/保留字是可以直接通过字符串定义的，不需要使用正则表达式定义。

与常见的高级语言如 C/C++、JAVA 以及 Pascal 相比较，它们相同点如下：

- 对于标识符，都不允许以数字开头的字符串作为标识符。
- 对于数的表示，二者都是以 0 开头表示八进制数。

它们的不同点如下（由于对 PASCAL 不是很了解，因此不做比较）：

- 对于标识符，C/C++的标识符允许以”_”开头，标识符中间也允许出现”_”字符，而 Oberon-0 语言中不支持”_”字符。标识符的长度也不一样，C/C++中标识符的长度是不做限制的，而 Oberon-0 语言中最多只能由 24 个字符。
- 注释也不一样，C/C++中使用的是/*…*/类型的注释，而 Oberon-0 语言中使用的是(*…*)括号风格的。
- 对于整形数据的长度，C/C++是通过二进制的表示的数的长度界定的，例如 int 型变量是 4 个字节共 32 位，因此能表示的范围是-2^32~2^32-1，不受这个数值常量是十进制还是八进制的限制，而 Oberon-0 语言所能表示的最大整型数据长度是跟这个数是十进制还是八进制有关。

3. 下载词法分析程序自动生成工具 JFlex

在这里下载安装 JFlex 并且配置好 JFlex 的环境变量，为接下来的实验步骤做准备。由于以前曾经有用过词法分析程序生成工具 JFlex，因此这一步不用很麻烦。

4. 生成 Oberon-0 语言的词法分析程序

在这一步骤中使用 JFlex 编写 Oberon-0 语言的词法分析程序，词法分析生成程序见 src 文件夹中的<lexical.flex>.

由于实验软装置中未提供主程序 main()函数，因此我自己编写了一个 MAIN 函数，作用是对于输入的 Oberon-0 源程序，分析出每一个词法单元然后在每个词法单元前后加上<>输出。

5. 讨论不同词法分析程序生成工具的差异

- 由于使用的是 JFlex，因此对 JFlex 学习的比较多，下面是 JFlex 的一些特性：
  - JFlex 生成工具的基本框架是：

    用户代码

    %%

    选项与声明

    %%

    词法规则
  - %class 中定义了生成的 Java 文件的类名以及文件名
  - %eofval 中定义了当识别到 EOF 的时候执行的操作
  - %yylexthrow 中定义了当前词法分析程序中要抛出的异常
  - %cup 的声明是为 java_cup 提供接口，因此生成了 next_token()函数。
  - %line 以及%column 是声明了可以使用 yyline 以及 yycolumn 获得当前的位置。
  - %{……用户代码……%}中填写的是用户代码，一般在这里实现 main 函数
  - <YYINITIAL>中定义了对于匹配每一个正则表达式后要执行的操作。

- 其余的两个词法分析程序都差不多，只是 GNU Flex 使用的是 C 语言，而 JFlex 以及 JLex 都是针对 Java 语言设计的，因此在生成代码之后的处理过程中会有很大的不一样。

三、 实验成果展示

1. 目录结构与文件说明

```
├─── bin……编译输出文件夹
│    ├─── exceptions……输出的所有 exceptions 字节码
│    │    ├─── IllegalIdentifierLengthException.class
│    │    ├─── IllegalIntegerException.class
│    │    ……
│    │    ├─── SyntacticException.class
│    │    └─── TypeMismatchedException.class
│    ├─── OberonScanner.class……输出的词法分析程序字节码
│    └─── sym.class……词汇表的字节码
├─── doc……生成的 JavaDoc
│    ├─── allclasses-frame.html
│    ……
│    └─── sym.html
├─── jflex……使用的工具
│    ├─── java-cup-11a.jar……Java_CUP 的工具
│    └─── JFlex.jar……JFlex 工具
├─── src……程序源代码
│    ├─── exceptions……所有的 exceptions
│    │    ├─── IllegalIdentifierLengthException.java
│    │    ├─── IllegalIntegerException.java
│    │    ……
│    │    ├─── SyntacticException.java
│    │    └─── TypeMismatchedException.java
│    ├─── oberon.flex……词法分析程序生成代码
│    ├─── OberonScanner.java……生成的词法分析程序
│    ├─── sym.java……词汇表
│    └─── testcases……测试样例
│         ├─── Sample.obr……自己编写的正确的测试样例
│         ├─── Test.001……Test.001-Test.009 是软装置中提供的样例
│         ├─── Test.002
│         ├─── Test.003
│         ├─── Test.004
│         ├─── Test.005
│         ├─── Test.006
│         ├─── Test.007
│         ├─── Test.008
│         ├─── Test.009……Test.001-Test.009 是词法错误的变异程序
├─── test.bat……运行有词法错误的编译程序脚本
├─── build.bat……构建脚本
├─── lexgen.doc……实验报告文档 DOC 格式
├─── lexgen.pdf……实验报告文档 PDF 格式
├─── readme.txt……说明文件
├─── run.bat……运行正确的 Oberon 程序脚本
├─── doc.bat……生成 JavaDOC 脚本
├─── gen.bat……根据 Jflex 生成词法分析程序脚本
└─── 词汇表.xlsx……Oberon 语言所定义的词汇表
```

2. 样例程序运行截图：

➢ 运行正确的 Oberon-0 测试样例（部分截图）：

```
(*********************************************************************)

 <MODULE>   <sample>  <;>

 <CONST>   <size>   <=>   <100>  <;>
 <VAR>   <n>  <,>   <num>  <,>   <what>  <,>   <max>  <,>   <result>  <,>   <i>
  <:>   <INTEGER>  <;>
 <VAR>   <isexit>   <:>   <BOOLEAN>  <;>

 <TYPE>   <userrecord>   <=>   <RECORD>
                <userid>   <:>   <INTEGER>  <;>
                <score>   <:>   <INTEGER>
         <END>  <;>

 <VAR>   <userlist>   <:>   <ARRAY>   <size>   <OF>   <userrecord>  <;>

 <PROCEDURE>   <max>  <(>  <n>   <:>   <INTEGER>  <;>   <a>   <:>   <ARRAY>   <s
ize>   <OF>   <userrecord>  <;>   <VAR>   <maxindex>   <:>   <INTEGER>  <)>  <;

 <VAR>   <i>   <:>   <INTEGER>  <;>
```

➢ 运行有词法错误的变异程序 Test.001：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Illegal symbol. *)
 <MODULE>   <test>  <;>
\
exceptions.IllegalSymbolException:
Illegal Symbols have been found.
The error position is Line 3Column 0
```

➢ 运行有词法错误的变异程序 Test.002：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Illegal integer number. *)
 <MODULE>   <test>  <;>
     <CONST>   <temp>   <=>  25id
exceptions.IllegalIntegerException:
The integer is illegal.
The error position is Line 3Column 17
```

➢ 运行有词法错误的变异程序 Test.003：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Illegal length of integer number.(decimal) *)
 <MODULE>   <test>  <;>
     <CONST>   <temp>   <=>  123456789012345678900
exceptions.IllegalIntegerRangeException:
The integer's length is too long.
The error position is Line 3Column 17
```

➢ 运行有词法错误的变异程序 Test.004：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Illegal length of integer number.(Octal) *)
 〈MODULE〉  〈test〉 〈;〉
     〈CONST〉  〈temp〉  〈=〉 012345601234560
exceptions.IllegalIntegerRangeException:
The integer's length is too long.
The error position is Line 3Column 17
```

➢ 运行有词法错误的变异程序 Test.005：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Illegal Octal number. *)
 〈MODULE〉  〈test〉 〈;〉
     〈CONST〉  〈temp〉  〈=〉 0123456789
exceptions.IllegalOctalException:
The octal integer  is illegal.
The error position is Line 3Column 17
```

➢ 运行有词法错误的变异程序 Test.006：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Illegal length of identifier. *)
 〈MODULE〉  〈test〉 〈;〉
     〈TYPE〉  〈number〉  〈=〉  〈INTEGER〉 〈;〉
     〈VAR〉  〈firstnum〉 〈,〉  〈secondnum〉 〈,〉  〈choice〉 〈,〉 abcdefghijklmno
pqrstuvwxyz
exceptions.IllegalIdentifierLengthException:
Identifier's length is too long.
The error position is Line 4Column 37
```

➢ 运行有词法错误的变异程序 Test.007：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Mismatched comment. *)
(*...
MODULE Test;
    TYPE NUMBER = INTEGER;
    VAR firstnum, secondnum, choice : NUMBER;

    PROCEDURE MaximumNum (a, b: NUMBER
exceptions.MismatchedCommentException:
Comment is Mismatched.
The error position is Line 2Column 0
```

➢ 运行有词法错误的变异程序 Test.008：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Mismatched comment. *)
(*...*
MODULE Test;
    TYPE NUMBER = INTEGER;
    VAR firstnum, secondnum, choice : NUMBER;

    PROCEDURE MaximumNum (a, b: NUMBER
exceptions.MismatchedCommentException:
Comment is Mismatched.
The error position is Line 2Column 0
```

➢ 运行有词法错误的变异程序 Test.009：

```
(* An Oberon-0 Program to provide three functions: 1-MaximumNum, 2-MinimumNum, 3
-Factorial.*)
(* Mutant: Multiple lexical errors. *)
 <MODULE>   <test>  <;>
     <CONST>   <temp1>   <=>  \
exceptions.IllegalSymbolException:
Illegal Symbols have been found.
The error position is Line 3Column 18
```

四、 实验心得

经过这次实验，再次详细了解了 JFlex 这样一个词法分析程序生成工具的使用方法，完成了对 Oberon-0 语言的词法分析，能够使用 next_token()函数获得下一个 Token 词法单元，为接下来进行语法分析奠定了基础。

同时，完成了这个实验，对编译原理中的词法分析又有更进一步的理解，以及更加熟悉了正则表达式的用法。例如在编写 JFlex 过程中，刚开始一直都纠结怎么样匹配注释，或者判断一个注释不匹配，经过仔细的思考后，才想到了解决办法，把对应的正则表达式写出来之后发现，其实如果能真正理解正则表达式每一个符号的意思并且对它的结构分辨清楚，其实一个在复杂的正则表达式也是可以理解的。

接下来才是这次实验的重头戏，使用 JavaCUP 工具生成语法分析程序，要抓紧时间好好加油了！

Thanks\(^o^)/~