编译器构造实验

Lab5—实验2

自动生成词法分析程序（JFlex）

姓名：郝裕玮

班级：计科1班

学号：18329015

目录

[1 编写一个正确的 Oberon-0 源程序 3](#_Toc106702760)

[2 编写上述 Oberon-0 源程序的变异程序 4](#_Toc106702761)

[3 讨论 Oberon-0 语言的特点 5](#_Toc106702762)

[4 讨论 Oberon-0 文法定义的二义性 6](#_Toc106702763)

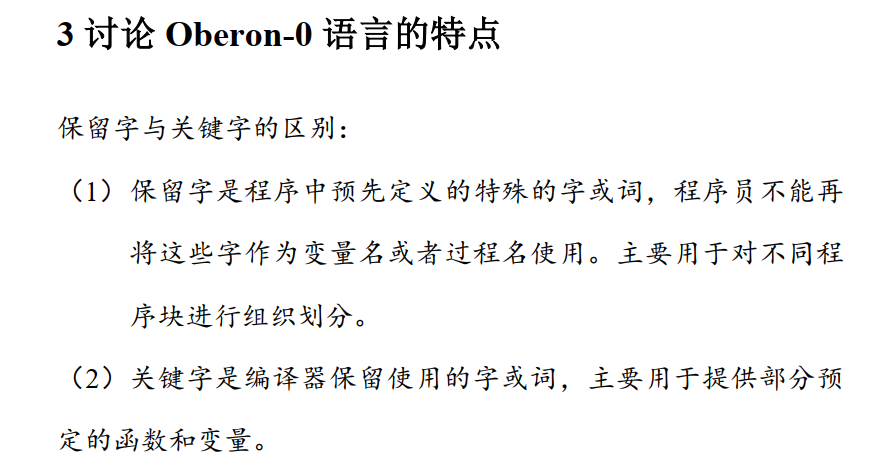
[5 实验心得 15](#_Toc106702764)

# 1 总结 Oberon-0 语言的词汇表

|  |  |  |
| --- | --- | --- |
| 关键字 | | INTEGER, BOOLEAN, CONST, TYPE, VAR, RECORD, ARRAY, Read, Write, WriteLn |
| 保留字 | | MODULE, PROCEDURE, OF, BEGIN, END, IF, THEN, ELSE, ELSIF, WHILE, DO |
| 数值常量 | | [1-9][0-9]\* | 0[0-7]\* |
| 运算符 | 算术运算符 | +, -, \*, DIV, MOD |
| 逻辑运算符 | &, OR, ~ |
| 关系运算符 | =, #, >, >=, <, <= |
| 赋值运算符 | := |
| 选择运算符 | [ ], . |
| 括号 | (, ) |
| 类型运算符 | : |
| 标识符 | | letter(letter|digit)\* |
| 标点符号 | | ;, , |
| 注释 | | (\*, \*) |

单词分类的理由：根据单词本身起到的功能和作用来进行分类即可。

区分保留字和关键字的方法：正如我在实验1的Oberon-0报告中所说：



所以保留字的主要功能是对代码块进行范围划分（例如MODULE, PROCEDURE, BEGIN, END, IF, ELSE等都是用于范围界定），关键字的主要功能是是提供部分预定的函数和变量（例如INTEGER, Read, Write都是具有特定功能，而非用于代码范围划分）。

# 2 抽取 Oberon-0 语言的词法规则

* Keyword —> "INTEGER" | "BOOLEAN" | "CONST" | "TYPE" | "VAR" | "RECORD" | "ARRAY" | "Read" | "Write" | "WriteLn"
* ReservedWord —> "MODULE" | "PROCEDURE" | "OF" | "BEGIN" | "END" | "IF" | "THEN" | "ELSE" | "ELSIF" |"WHILE" | "DO"
* Decimal —> [1-9][0-9]\*
* Octal —> 0[0-7]\*
* Number —> Decimal | Octal
* Operator —> "+" | "-" | "\*" | "DIV" | "MOD" | "&" | " OR " | "~" | "=" | "#" | ">" | " >=" | "<" | "<=" | ":=" | "[" | "]" | "." | "(" | ")" | ":"
* Letter —> [a-zA-z]
* Digit —> [0-9]
* Identifier —> Letter(Letter | Digit)\*
* Punctuation —> ";" | ","
* Comment —> "(\*" [^\*] ~"\*)" | "(\*" "\*"+ ")"

Oberon-0与 Pascal 、 C/C++、 Java 等常见高级程序设计语言的词法规则相比的异同点：

（1）Oberon-0不区分变量名大小写。JAVA、C/C++区分变量名大小写。

（2）Oberon-0 的不同过程PROCEDURE使用 BEGIN 和 END 来区分。Java、C/C++中不同代码块用{}来区分。

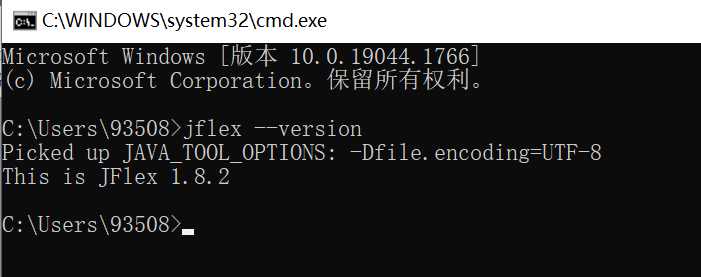
（3）Oberon-0没有浮点数，只支持整除和取余。而Pascal 、 C/C++、 Java 等常见高级程序设计语言支持浮点数除法。

（4）Oberon-0的不等于是”#”， 而Pascal 、 C/C++、 Java 等常见高级程序设计语言的不等于是”!=”。

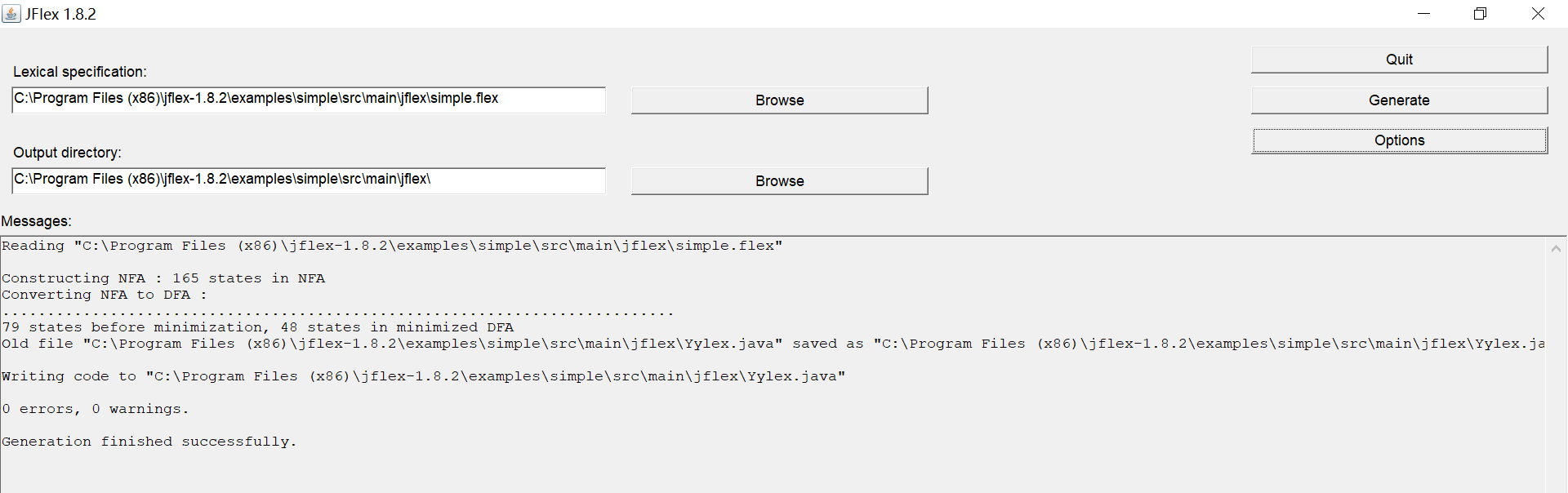
（5）Oberon-0的注释方法为(\* \*)。而Pascal 、 C/C++、 Java 等常见高级程序设计语言的注释方法为”//”。

# 3 下载词法分析程序自动生成工具 JFlex

验证JFlex是否安装成功：



运行JFlex 附带的输入源文件例子：





易知成功生成Java 语言的词法分析程序源代码。

# 4 生成 Oberon-0 语言的词法分析程序

JFlex输入源文件代码如下所示（包含注释）：

//用户代码

import java.io.\*;

import exceptions.\*;

//选项与声明

%%

//定义生成词法分析器Java文件的文件名

%class OberonScanner

//使生成的类是public

%public

//设置扫描函数的返回类型

%type String

//使扫描函数声明抛出异常

%yylexthrow LexicalException

//其中用户代码部分直接被复制到扫描函数中，并且在每次文件结束时执行。这个用户代码应该返回表示文件结束的值

%eofval{

    return "EOF";

%eofval}

//支持字符集，防止溢出

%unicode

//行计数器，yyline记录当前行数

%line

//列计数器，yycolumn记录当前列数

%column

//不区分字符大小写，符合oberon-0的词法规则

%ignorecase

//获取当前行数或列数

%{

    int get\_line(){ return yyline;}

    int get\_column(){ return yycolumn;}

%}

//正则表达式

Keyword = "INTEGER" | "BOOLEAN" | "CONST" | "TYPE" | "VAR" | "RECORD" | "ARRAY" | "Read" | "Write" | "WriteLn"

ReservedWord = "MODULE" | "PROCEDURE" | "OF" | "BEGIN" | "END" | "IF" | "THEN" | "ELSE" | "ELSIF" |"WHILE" | "DO"

Decimal = [1-9][0-9]\*

Octal = 0[0-7]\*

Number = {Decimal} | {Octal}

Operator = "+" | "-" | "\*" | "DIV" | "MOD" | "&" | " OR " | "~" | "=" | "#" | ">" | " >=" | "<" | "<=" | ":=" | "[" | "]" | "." | "(" | ")" | ":"

Identifier = [:jletter:][:jletterdigit:]\*

Punctuation = ";" | ","

Comment = "(\*" [^\*] ~"\*)" | "(\*" "\*"+ ")"

WhiteSpace = " " | \r | \n | \r\n | [ \t\f]

//异常情况的正则表达式

WrongInteger = {Number} + {Identifier} +

WrongOctal = 0[0-7]\*[9|8][0-9]\*

WrongComment = "(\*" ([^\\*]|"\*"+[^\)])\*|([^\(]|"("+[^\\*])\*"\*)"

//词法规则

%%

<YYINITIAL>{

    {Keyword}                           {return "Keyword";}

    {ReservedWord}                      {return "ReservedWord";}

    {Number}                            {

                                            if(yylength() > 12){

                                                throw new IllegalIntegerRangeException();

                                            }

                                            else{

                                                return "Number";

                                            }

                                        }

    {Operator}                          {return "Operator";}

    {Identifier}                        {

                                            if(yylength() > 24){

                                                throw new IllegalIdentifierLengthException();

                                            }

                                            else{

                                                return "Identifier";

                                            }

                                        }

    {Punctuation}                       {return "Punctuation";}

    {Comment}                           {return "Comment";}

    {WhiteSpace}                        {}

    {WrongInteger}                      {throw new IllegalIntegerException();}

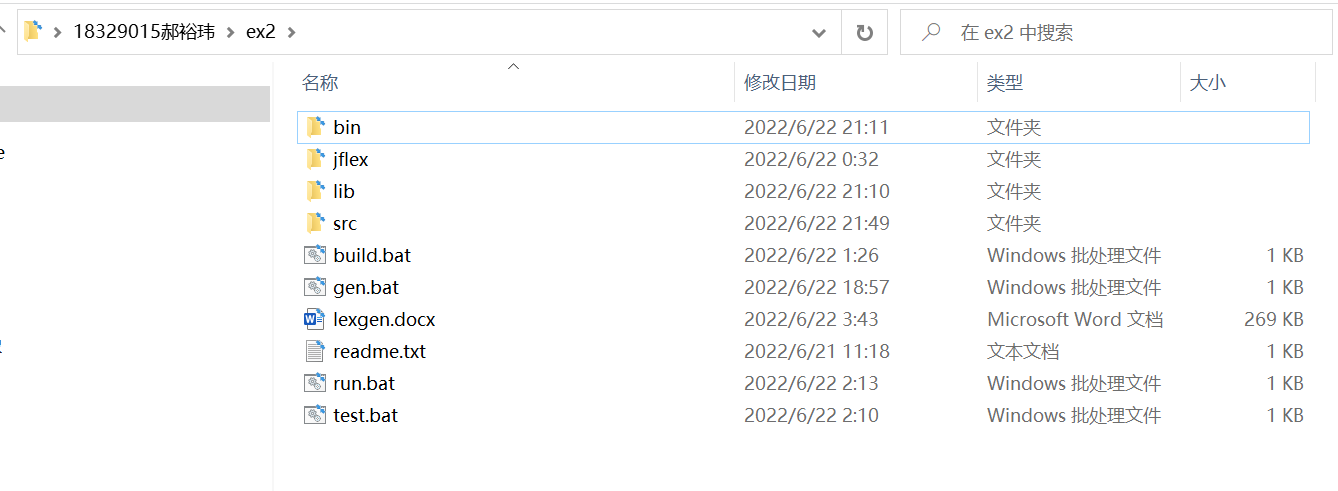
    {WrongOctal}                        {throw new IllegalOctalException();}

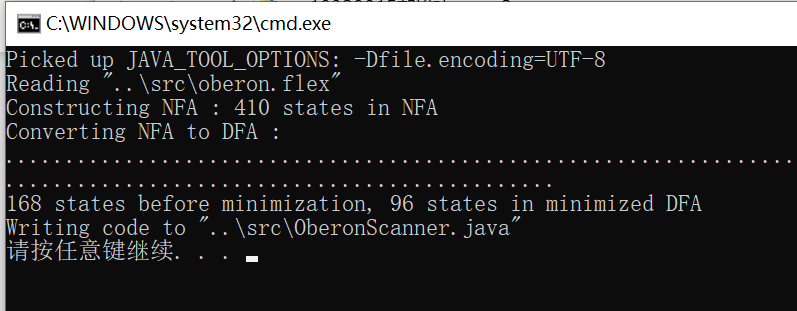
    {WrongComment}                      {throw new MismatchedCommentException();}

    .                                   {throw new IllegalSymbolException(); }

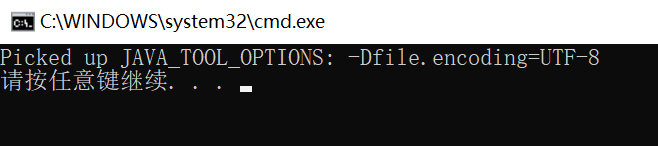
}

在当前文件夹下运行gen.bat来生成词法分析程序OberonScanner.java



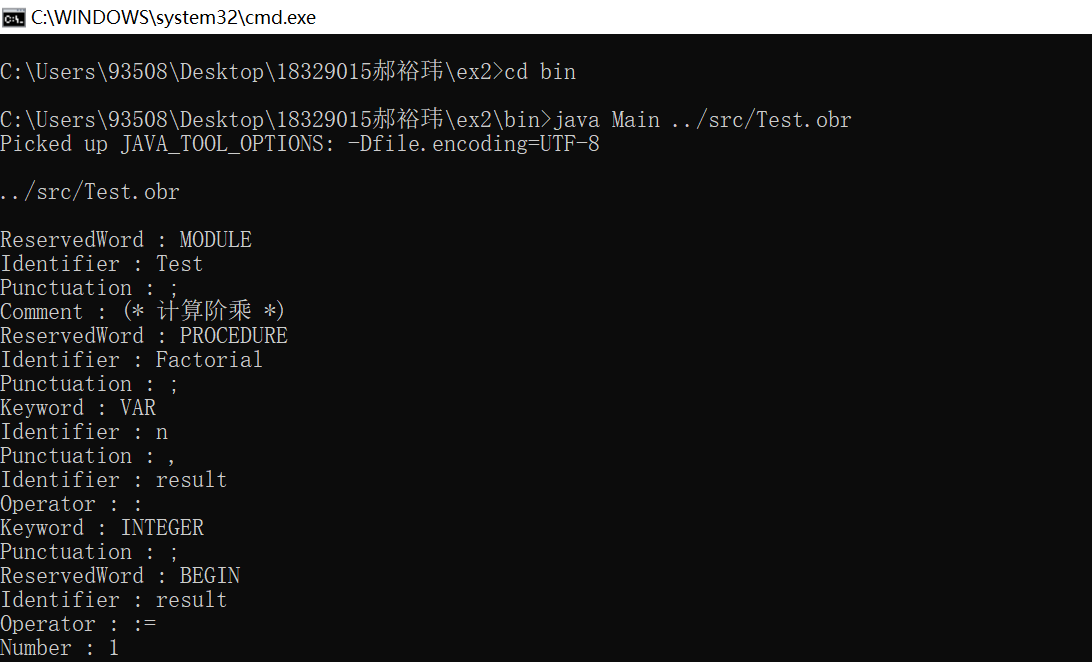


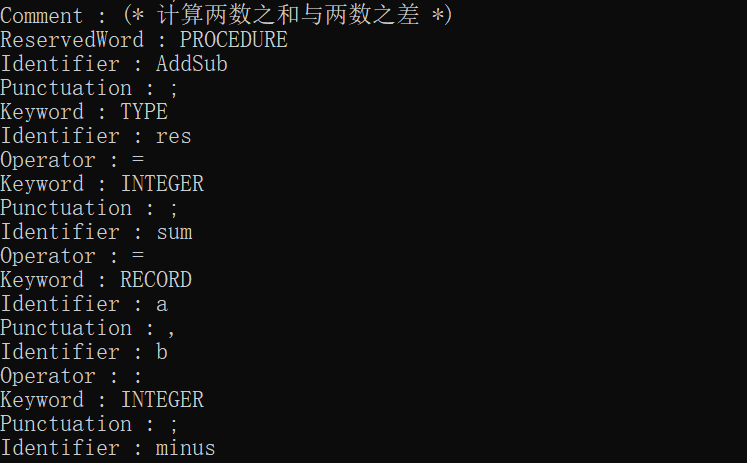
运行build.bat对词法分析程序进行编译

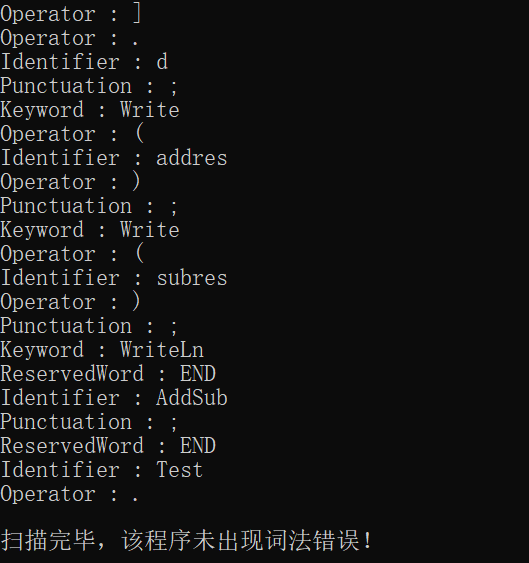


运行run.bat来处理我编写的正确 Oberon-0 例子程序：Test.obr，

以下截图仅展示部分分析结果：



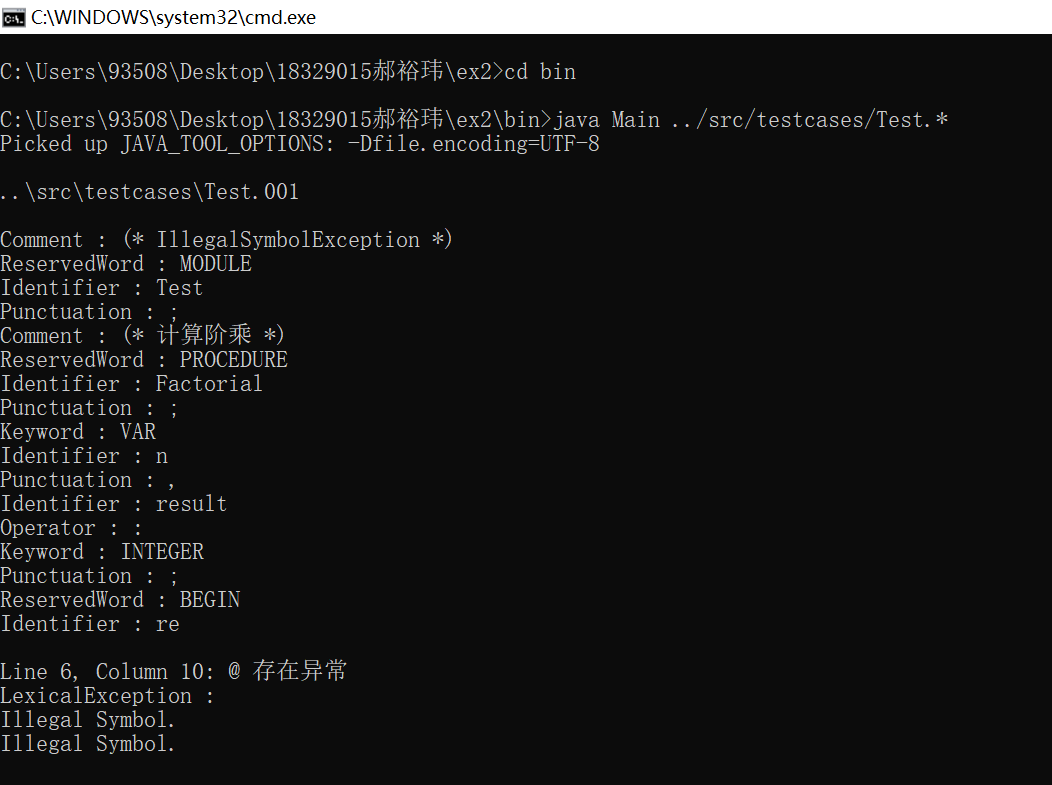




再运行test.bat来检测各种异常的测试用例：

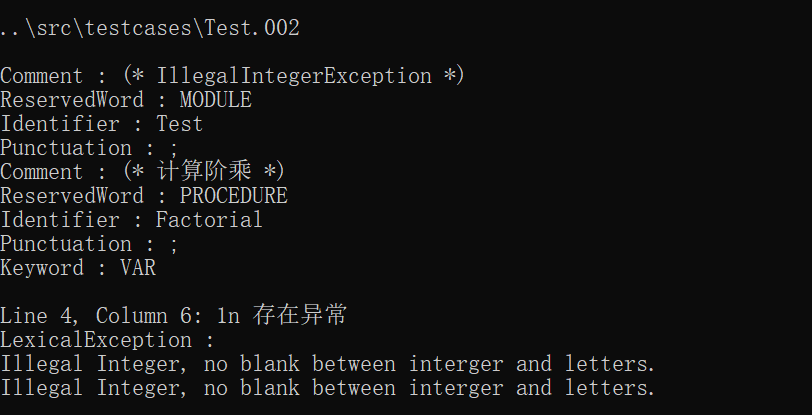
（1）IllegalSymbolException

当识别一个单词时遇到不合法的输入符号（譬如@、 $等符号）则抛出该异常。



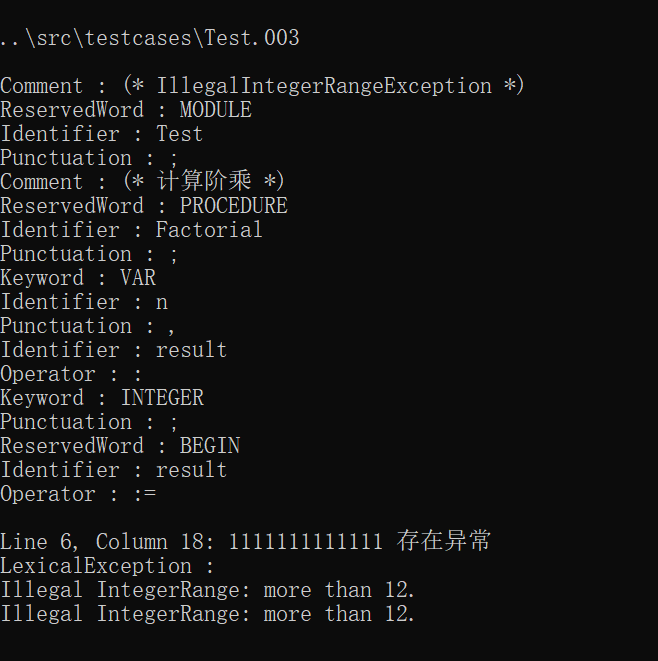
（2）IllegalIntegerException

当整数常量（无论是十进制还是八进制） 与其后的标识符之间无空白分隔时抛出该异常。



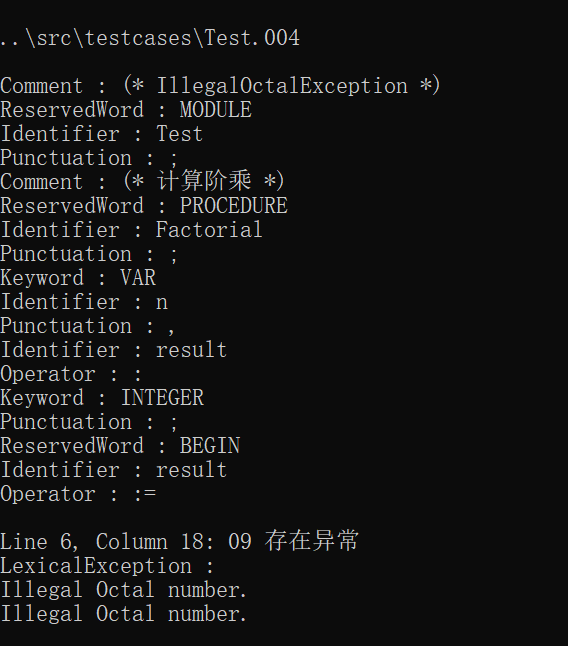
（3）IllegalIntegerRangeException

当识别出的整数常量（无论是十进制还是八进制）大于本文档约定的整数常量值最大限制时抛出此异常。



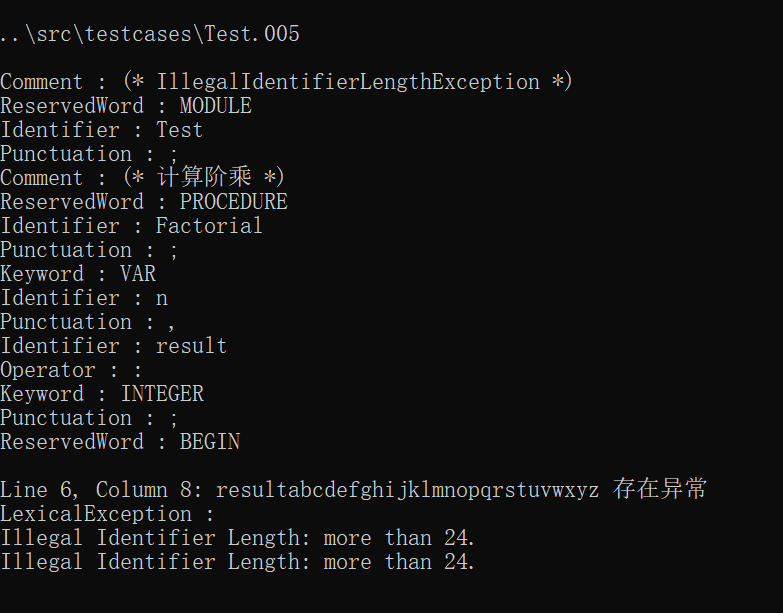
（4）IllegalOctalException

当 0 开头的整数常量中含有 0～7 之外的符号（包括 8 和 9）时抛出该异常。（图见下页）



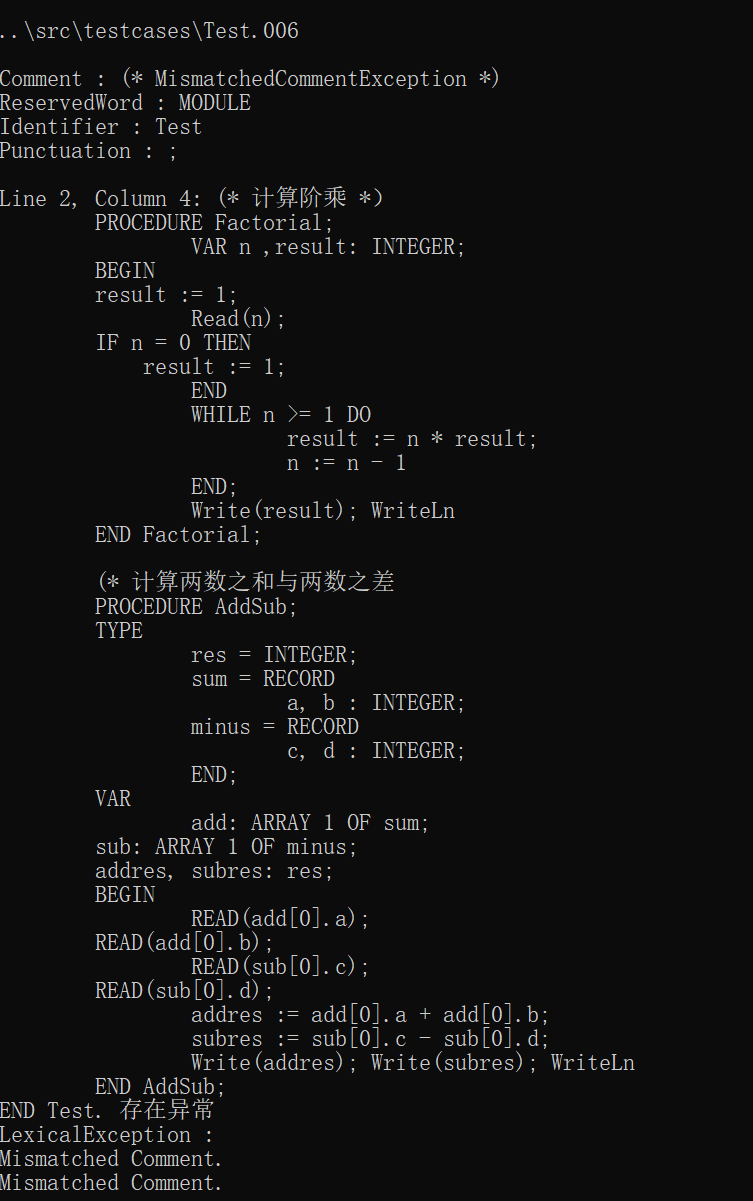
（5）IllegalIdentifierLengthException

当识别出的一个标识符长度超过最大限制时抛出该异常。



（6）MismatchedCommentException

当“(\*”开头的注释直至扫描到最后一个符号都找不到配对的“\*)”时抛出该异常。



综合上述6个词法异常的测试样例可知，JFlex生成的Java词法分析程序可成功检测出实验要求的6种不同的词法异常。

# 5 讨论不同词法分析程序生成工具的差异

1. JFlex和JLex生成Java语言，GNU Flex生成C语言。所以这三者的词法规则必定有所不同（C和Java的差异）。
2. 三者的代码书写规则有不同之处。

（3）JFlex和JLex将代码分为用户代码，选项与声明，词法规则。GNU Flex将代码分为定义段(definitions)，规则段(rules)、用户代码段(user code)。

# 6 实验心得

通过实验2让我对JFlex的使用更加熟练，并对正则表达式和词法规则的结合有了更进一步的认识，对词法分析这一步骤也更加理解其本质。