

实验报告

一、实验目的

- 熟悉C语言的词法规则，了解编译器词法分析器的主要功能和实现技术，掌握典型词法分析器构造方法，设计并实现C语言词法分析器
- 了解Flex工作原理和基本思想，学习使用工具自动生成词法分析器
- 掌握编译器从前端到后端各个模块的工作原理，词法分析模块与其他模块之间的交互过程。

二、实验内容

- 根据C语言的词法规则，设计并实现C语言词法分析器
- 词法分析器的输入为C语言源程序，输出为属性字流
- 可以选择编码实现，也可以选择使用自动生成工具

三、实验过程

- 实验环境
 - jflex-1.8.2
 - jdk-17.0.2
- 实验步骤
 - 将需要识别的单词分为7类：Keyword, Identifier, Operator, Delimiter, IntergerConstant, FloatConstant, CharacterConstant, StringListeral
 - 写出匹配这7类单词的正则表达式

```
Keyword="auto"|"break"|"case"|"char"|"const"|"continue"|"default"|"do"|"double"|"else"|"enum"|"extern"|"float"|"for"|"goto"|"if"|"inline"|"int"|"long"|"register"|"restrict"|"return"|"short"|"signed"|"sizeof"|"static"|"struct"|"switch"|"typedef"|"union"|"unsigned"|"void"|"volatile"|"while"
```

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

```
IntegerConstant=0|[1-9][0-9]*
```

```
FloatConstant=[0-9]+(\.[0-9]+)
```

```

Operator="+","-","*","/","%","++","--",">","<","==",">=","
<="","!=","&&","||","!","&","~","^","<<",">>","=","+=","-
=","*=","/=","%=","&=","|=","^=","<<=",">>=","?":"."

Delimiter=",",";","\\","'","(",")","[","]","{","}" cant be #

CharacterConstant='\((\\S)|[^\'])\''

StringLiteral="\S*[^\S]"

```

- 写出匹配空白符、换行符和注释的正则表达式

```

Comment = {TraditionalComment} | {EndOfLineComment} |
{DocumentationComment}

whiteSpace=\ | \t | \f | \b

LineTerminator=\r | \n | \r\n

CommentContent      = ( [^*] | \*+ [^/*] ) *
TraditionalComment  = "/" [^*] ~ "*" / | "/" " "+ "/"
EndOfLineComment    = "//" [^\r\n]* {LineTerminator}?
DocumentationComment = "/*" {CommentContent} "*" + "/"

```

- 写出匹配成功后所要执行的函数

```

%{
    private int count = 0;

    public void genToken(char[] zzBuffer, int type, int count,
int startPos, int endPos) throws IOException{
        FileWriter writer = new FileWriter("test.txt", true);
        String s = String.valueOf(zzBuffer, startPos, endPos -
startPos);
        switch(type) {
            // Keyword
            case 0: writer.write("[@" + count + "," + yycolumn
+ ":" + (yycolumn + endPos - startPos) + "'=" + s + "',<' + s
+ "'>," + yyline + ":" + yycolumn +"]\n"); break;
            // Identifier
            case 1: writer.write("[@" + count + "," + yycolumn
+ ":" + (yycolumn + endPos - startPos) + "'=" + s + "',
<Identifier>," + yyline + ":" + yycolumn +"]\n"); break;
            // Operator

```

```

        case 2: writer.write("[@" + count + "," + yycolumn
+ ":" + (yycolumn + endPos - startPos) + "'=" + s + "'",<" + s
+ "'>," + yyline + ":" + yycolumn +"]\n"); break;
        // Delimiter
        case 3: writer.write("[@" + count + "," + yycolumn
+ ":" + (yycolumn + endPos - startPos) + "'=" + s + "'",<" + s
+ "'>," + yyline + ":" + yycolumn +"]\n"); break;
        // IntegerConstant
        case 4: writer.write("[@" + count + "," + yycolumn
+ ":" + (yycolumn + endPos - startPos) + "'=" + s + "'",
<IntegerConstant>," + yyline + ":" + yycolumn +"]\n"); break;
        // FloatConstant
        case 5: writer.write("[@" + count + "," + yycolumn
+ ":" + (yycolumn + endPos - startPos) + "'=" + s + "'",
<FloatConstant>," + yyline + ":" + yycolumn +"]\n"); break;
        // CharacterConstant
        case 6: writer.write("[@" + count + "," + yycolumn
+ ":" + (yycolumn + endPos - startPos) + "'=" + s + "'",
<CharacterConstant>," + yyline + ":" + yycolumn +"]\n");
break;

        // StringLiteral
        case 7: writer.write("[@" + count + "," + yycolumn
+ ":" + (yycolumn + endPos - startPos) + "'=" + s + "'",
<StringLiteral>," + yyline + ":" + yycolumn +"]\n"); break;
    }
    writer.flush();
    writer.close();
}
%}

```

- 在词法规则部分对整体进行匹配

```

<YYINITIAL> {
    {Keyword}      { count++; genToken(zzBuffer, 0, count,
zzCurrentPos, zzMarkedPos); }
    {Identifier}   { count++; genToken(zzBuffer, 1, count,
zzCurrentPos, zzMarkedPos); }
    {Operator}     { count++; genToken(zzBuffer, 2, count,
zzCurrentPos, zzMarkedPos); }
    {Delimiter}    { count++; genToken(zzBuffer, 3, count,
zzCurrentPos, zzMarkedPos); }
    {IntegerConstant} { count++; genToken(zzBuffer, 4,
count, zzCurrentPos, zzMarkedPos); }
    {FloatConstant}  { count++; genToken(zzBuffer, 5,
count, zzCurrentPos, zzMarkedPos); }
    {CharacterConstant} { count++; genToken(zzBuffer, 6,
count, zzCurrentPos, zzMarkedPos); }
    {StringLiteral}   { count++; genToken(zzBuffer, 7,
count, zzCurrentPos, zzMarkedPos); }
    {Comment}         {}
}

{LineTerminator}|{WhiteSpace} {}

```

- 添加生成选项

```

%public
%class MyScanner
%function scan
%type void

%char
%line
%column
%unicode

```

- 使用Jflex生成词法分析器并添加调用函数

```

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException
    {
        if (args[0].equals("")) {

```

```

        System.out.println("Need parameter.");
        return;
    }

    FileWriter writer = new FileWriter("out.txt", false);
    writer.write("");
    writer.flush();
    writer.close();
    MyScanner scanner = new MyScanner(new
    FileReader(args[0]));
    System.out.println("Start...");
    scanner.scan();
    System.out.println("Finish...");
}
}

```

四、运行效果截图

- C语言源代码

```

/*jhgkhjkh
kljlkjlj
*/
#include<stdio.h>
int main() {
    int i;
    float mini = 123.45542;
    sd[0] = '\';
    sd[1] = 's'

    // kljlkj

    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow){
    LPSTR msg;
    UINT opt;

    if(checkVbox()){
        msg = "Virtual box detected !!!";
        opt = MB_ICONEXCLAMATION;
    } else {
        msg = "Virtual box not detected !!!";
        opt = MB_OK;
    }

    MessageBox(NULL, msg, "Virtual Box", opt);

    return 0;
}

```

- 输出属性字流（部分）

```
test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[ @ 1,0:1 = '#', < '#' >, 3:0]
[ @ 2,1:8 = 'include', < Identifier >, 3:1]
[ @ 3,8:9 = '<', < ' ' >, 3:8]
[ @ 4,9:14 = 'stdio', < Identifier >, 3:9]
[ @ 5,14:15 = ',', < ' ' >, 3:14]
[ @ 6,15:16 = 'h', < Identifier >, 3:15]
[ @ 7,16:17 = '>', < ' ' >, 3:16]
[ @ 8,0:3 = 'int', < 'int' >, 4:0]
[ @ 9,4:8 = 'main', < Identifier >, 4:4]
[ @ 10,8:9 = '(', < ' ' >, 4:8]
[ @ 11,9:10 = ')', < ' ' >, 4:9]
[ @ 12,11:12 = '[', < ' ' >, 4:11]
[ @ 13,1:4 = 'int', < 'int' >, 5:1]
[ @ 14,5:6 = 'i', < Identifier >, 5:5]
[ @ 15,6:7 = ':', < ' ' >, 5:6]
[ @ 16,1:6 = 'float', < 'float' >, 6:1]
[ @ 17,7:11 = 'mini', < Identifier >, 6:7]
[ @ 18,12:13 = '=', < ' ' >, 6:12]
[ @ 19,14:23 = '123.45542', < FloatConstant >, 6:14]
[ @ 20,23:24 = ',', < ' ' >, 6:23]
[ @ 21,1:3 = 'sd', < Identifier >, 7:1]
[ @ 22,3:4 = '[', < ' ' >, 7:3]
[ @ 23,4:5 = '0', < IntegerConstant >, 7:4]
[ @ 24,5:6 = ']', < ' ' >, 7:5]
[ @ 25,7:8 = '=', < ' ' >, 7:7]
[ @ 26,9:13 = '"', < CharacterConstant >, 7:9]
[ @ 27,13:14 = ',', < ' ' >, 7:13]
[ @ 28,1:3 = 'sd', < Identifier >, 8:1]
[ @ 29,3:4 = '[', < ' ' >, 8:3]
[ @ 30,4:5 = '1', < IntegerConstant >, 8:4]
[ @ 31,5:6 = ']', < ' ' >, 8:5]
[ @ 32,7:8 = '=', < ' ' >, 8:7]
[ @ 33,9:12 = 's', < CharacterConstant >, 8:9]
[ @ 34,1:7 = 'return', < 'return' >, 12:1]
[ @ 35,8:9 = '0', < IntegerConstant >, 12:8]
[ @ 36,9:10 = ',', < ' ' >, 12:9]
[ @ 37,0:1 = ']', < ' ' >, 13:0]
[ @ 38,0:3 = 'int', < 'int' >, 15:0]
[ @ 39,4:10 = 'WINAPI', < Identifier >, 15:4]
[ @ 40,11:18 = 'WinMain', < Identifier >, 15:11]
[ @ 41,18:19 = '[', < ' ' >, 15:18]
[ @ 42,19:28 = 'INSTANCE', < Identifier >, 15:19]
[ @ 43,29:38 = 'hinstance', < Identifier >, 15:29]
[ @ 44,38:39 = ':', < ' ' >, 15:38]
```

五、实验心得体会

由于对正则表达式的使用不够熟练，大部分时间都花费在了写正则表达式上，并且被一些细节问题纠缠了很久。另一个时间开销比较大的方面是学习Jflex的使用方法。由于使用手册是全英文的，网上的中文翻译水平参差不齐，导致花费了较多时间。