 

**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **编译技术** |
| **学生姓名：** | **熊腾浩** |
| **学生学号：** | **201930383212** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2021-2022第一学期** |

**软件学院**

**2021年11月**

# XXXX

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **地 点：** | | | B7 楼 | | 233 房； | | **实验台号：** | |  | |
| **实验日期与时间：** | | | 2021.11.5 | | | | **评 分：** | |  | |
| **预习检查纪录：** | | |  | | | | **实验教师：** | |  | |
| **电子文档存放位置：** | | |  | | | | | | | |
| **电子文档文件名：** | | |  | | | | | | | |
| **批改意见：** | | |  | | | | | | | |
| 出勤情况（20%） | 课堂态度（20%） | | 实验报告质量（60%） | | | | | |
| 任务完成情况 | | 源代码注释 | | 报告内容 | |
| □ 出勤  □ 缺勤  □ 早退  □ 请假 | □ 认真  □ 不认真 | | □ 完成  □ 未完成 | | □ 优秀  □ 良好  □ 通过  □ 未通过 | | □ 优秀  □ 良好  □ 通过  □ 未通过 | |

# 报告内容

1、实验目标

Implementing a Scanner for decaf

You are to write a lexical analyzer/scanner for the language decaf

Goals

1. The input of the scanner is a source code file and the output of the

scanner is a stream of tokens.

2. Your scanner should go for longest possible match i.e. a string ‘:=’is to be identified as ‘ass-symbol’and not as ‘:’and ‘=’.

3. Token is represented as (Kind, Value). We use the following symbols to denote different kinds of tokens KEY denotes reserved words SYM denotes special symbols ID denotes identifiers NUM denotes numeric constants STR denotes string constants

4. Check lexical errors: giving meaning error messages and the lines where errors occur. The kinds of lexical errors are:

- Illegal character, that is, scanner may recognize a character that is not in the alphabet of Decaf, such as @ is an illegal character

- The right bracket of a STRING is lost, such as "scanner

2、实验环境

（实验使用的软件/硬件环境）

Ubuntu 18.04 LTS，openjdk 11.0.11 2021-04-20，javac 11.0.11

3、关键数据结构和核心算法

3.1 数据结构

无

3. 2 核心算法

 // comment

        if (peek == '/') {

            readch();

            if (peek == '/') {

                while (peek != '\n' && peek != '\r' && peek != '\0') {

                    readch();

                }

            } else if (peek == '\*') {

                while (true) {

                    readch();

                    if (peek == '\0') {

                        throw new Error(

                            String.format("ERROR %d:%d: comment not terminated",

                                          line, column));

                    }

                    if (peek == '\*') {

                        if (readch('/')) {

                            return Word.Comment;

                        }

                    }

                    if (peek == '/') {

                        if (readch('\*')) {

                            throw new Error(String.format(

                                "ERROR %d:%d: comment not terminated", line,

                                column));

                        }

                    }

                    if (peek == '\n') {

                        line++;

                        column = 0;

                    }

                }

            }

            if (peek == '\n' || peek == '\r') {

                return Word.Comment;

            }

        }

// string

        if (peek == '"') {

            StringBuffer b = new StringBuffer();

            for (;;) {

                if (readch('"')) {

                    String s = b.toString();

                    Word w = (Word) words.get(s);

                    if (w != null) return w;

                    w = new Word(s, Tag.STRING);

                    words.put(s, w);

                    return w;

                } else if (peek == '\n') {

                    throw new Error(String.format("Error: %d:%d: Missing \"",

                                                  line, column));

                } else {

                    b.append(peek);

                }

            }

        }

// number

        if (Character.isDigit(peek)) {

            int v = 0;

            if (peek == 0) {

                // hex number

                readch();

                if (peek == 'x' || peek == 'X') {

                    for (;;) {

                        readch();

                        if (peek >= '0' && peek <= '9')

                            v = v \* 16 + peek - '0';

                        else if (peek >= 'a' && peek <= 'f')

                            v = v \* 16 + peek - 'a' + 10;

                        else if (peek >= 'A' && peek <= 'F')

                            v = v \* 16 + peek - 'A' + 10;

                        else

                            break;

                    }

                    return new HexNum(v);

                } else if (peek == '.') {

                    double x = v;

                    double d = 10;

                    for (;;) {

                        readch();

                        if (peek == 'e' || peek == 'E') {

                            // double number

                            readch();

                            int sym = 1;

                            if (peek == '+' || peek == '-') {

                                if (peek == '-') sym = -1;

                                readch();

                            } else {

                                throw new Error(String.format(

                                    "ERROR: %d:%d: Invalid double number", line,

                                    column));

                            }

                            int exp = 0;

                            while (Character.isDigit(peek)) {

                                exp = 10 \* exp + Character.digit(peek, 10);

                                readch();

                            }

                            return new Real(x \* Math.pow(10, exp \* sym));

                        } else {

                            if (!Character.isDigit(peek)) break;

                            x = x + Character.digit(peek, 10) / d;

                            d = d \* 10;

                        }

                    }

                    return new Real(x);

                }

            } else {

                do {

                    v = 10 \* v + Character.digit(peek, 10);

                    readch();

                } while (Character.isDigit(peek));

                if (peek != '.') return new Num(v);

                double x = v;

                double d = 10;

                for (;;) {

                    readch();

                    if (peek == 'e' || peek == 'E') {

                        // double number

                        readch();

                        int sym = 1;

                        if (peek == '+' || peek == '-') {

                            if (peek == '-') sym = -1;

                            readch();

                        } else {

                            throw new Error(String.format(

                                "ERROR: %d:%d: Invalid double number", line,

                                column));

                        }

                        int exp = 0;

                        while (Character.isDigit(peek)) {

                            exp = 10 \* exp + Character.digit(peek, 10);

                            readch();

                        }

                        return new Real(x \* Math.pow(10, exp \* sym));

                    } else {

                        if (!Character.isDigit(peek)) break;

                        x = x + Character.digit(peek, 10) / d;

                        d = d \* 10;

                    }

                }

                return new Real(x);

            }

        }

// identifier or keyword

        if (Character.isLetter(peek)) {

            StringBuffer b = new StringBuffer();

            do {

                b.append(peek);

                readch();

            } while (Character.isLetterOrDigit(peek) ||

                     peek == '\_'); // '\_' is allowed

            String s = b.toString();

            Word w = (Word) words.get(s);

            if (w != null) return w;

            w = new Word(s, Tag.ID);

            words.put(s, w);

            return w;

        }

3. 3 错误处理

字符串常量没有右双引号报错

多行注释嵌套报错

双精度数不符合格式要求报错

多行注释没有结束符报错

3.4 运行结果

运行输入：

class Main {

static void main() {

class Fibonacci f = new class Fibonacci();

Print.print(f.get(ReadInteger()));

}

}

/\*\*

\* Fibonacci

\*/

class Fibonacci {

public int get(int n) {

if (n == 0) {

return 0;

}

if (n == 1) {

return 1;

}

return get(n - 1) + get(n - 2);

}

}

运行输出（已将换行符改为空格）：

(KEY , class) (ID , Main) (SYM , {) (KEY , static) (KEY , void) (ID , main) (SYM , () (SYM , )) (SYM , {) (KEY , class) (ID , Fibonacci) (ID , f) (SYM , =) (KEY , new) (KEY , class) (ID , Fibonacci) (SYM , () (SYM , )) (SYM , ;) (ID , Print) (SYM , .) (KEY , print) (SYM , () (ID , f) (SYM , .) (ID , get) (SYM , () (ID , ReadInteger) (SYM , () (SYM , )) (SYM , )) (SYM , )) (SYM , ;) (SYM , }) (SYM , }) (KEY , class) (ID , Fibonacci) (SYM , {) (KEY , public) (KEY , int) (ID , get) (SYM , () (KEY , int) (ID , n) (SYM , )) (SYM , {) (KEY , if) (SYM , () (ID , n) (SYM , ==) (NUM , 0) (SYM , )) (SYM , {) (KEY , return) (NUM , 0) (SYM , ;) (SYM , }) (KEY , if) (SYM , () (ID , n) (SYM , ==) (NUM , 1) (SYM , )) (SYM , {) (KEY , return) (NUM , 1) (SYM , ;) (SYM , }) (KEY , return) (ID , get) (SYM , () (ID , n) (SYM , -) (NUM , 1) (SYM , )) (SYM , +) (ID , get) (SYM , () (ID , n) (SYM , -) (NUM , 2) (SYM , )) (SYM , ;) (SYM , }) (SYM , })

4、总结

体会：学习了词法分析器的逻辑。

　（包括体会、问题和建议）