湖南大學
HUNAN UNIVERSITY

课程实验报告

课　程　名　称：编译原理

实验项目名称：构建词法分析器构造工具的实现

专　业　班　级：软件 2005

姓　　名：邹佳骏

学　　号：202026010501

指　导　教　师：杨金民

完　成　时　间：2023 年 5 月 15 日

信息科学与工程学院

实验题目：

试验三、构建词法分析器构造工具的实现

实验目的：

深入理解 TINY 语言，并学会用正则语言描述出 TINY 语言的词法，写出 TINY 语言的词法分析器的完整代码

实验环境：

PC、Windows 操作系统、DEV C++。

实验内容及操作步骤：

实验内容：

1. 基于第四章知识，用正则语言描述出 TINY 语言的词法，然后得出 TINY 语言的词法分
析器的完整代码：

2. 以用 TINY 语言写出的源程序 sample.tny 作为输入，输出出其词序列。以此验证 TINY
语言词法分析器的正确性。

Sample.tny 源程序包含下面 10 行代码：

```
1. { Sample program in TINY language - computes factorial}
2. read x; { input an integer }
3. if 0 < x then { don't compute if x <= 0 }
4. fact := 1;
5. repeat
6. fact := fact * x;
7. x := x - 1
8. until x = 0;
9. write fact { output factorial of x }
10. end
```

操作步骤：

TINY 语言的词法单元有以下几种：

- 关键字：if, then, else, end, repeat, until, read, write
- 标识符：由字母开头，后跟字母或数字的字符串，如 x, fact, myName
- 数字：由一个或多个数字组成的字符串，如 0, 1, 123
- 注释：由一对花括号包围的任意字符，如{this is a comment}
- 赋值符号：:=
- 比较符号：<或=
- 算术符号：+ - * /
- 括号：(或)
- 分号：;

用正则语言表示，TINY 语言的词法可以用以下规则定义：

- 关键字：if|then|else|end|repeat|until|read|write

- 标识符：[a-zA-Z][a-zA-Z0-9]*
- 数字：[0-9]+
- 注释：\{[^}]*\}
- 赋值符号：:=
- 比较符号：<|=
- 算术符号：[+\-*\/]
- 括号：[()]
- 分号：;

```c
// TINY 语言的词法分析器的代码
#include <stdio.h>
#include <ctype.h>
#include <string.h>
// 定义词法单元的类型
enum TokenType {
    ERROR, ID, NUM, COMMENT, ASSIGN, SYMBOL
};
// 定义有限状态自动机的状态
enum StateType {
    START, INID, INNUM, INCOMMENT, INASSIGN, DONE
};
// 定义特殊符号
char symbols[] = "+-*/=<();";
#define MAXTOKENLEN 40// 定义最大词法单元长度
#define BUFSIZE 256// 定义缓冲区大小
char buffer[BUFSIZE];// 定义缓冲区
int bufpos = 0;// 定义缓冲区指针
int lineno = 0;// 定义当前行号
char tokenString[MAXTOKENLEN+1];// 定义当前词法单元
enum TokenType tokenType;// 定义当前词法单元类型
```

```c
// 从文件中读取一个字符，如果到达文件尾，返回 EOF
int getNextChar(FILE *fp) {
    if (bufpos >= BUFSIZE) {
        // 缓冲区已满，需要重新读取
        int size = fread(buffer, sizeof(char), BUFSIZE, fp);
        if (size == 0) {
            // 文件已读完，返回 EOF
            return EOF;
        }
        // 重置缓冲区指针
        bufpos = 0;
    }
    // 返回当前字符，并将指针后移一位
    return buffer[bufpos++];
}

// 将缓冲区指针前移一位，相当于退回一个字符
void ungetNextChar() {
    if (bufpos > 0) {
        bufpos--;
    }
}

// 判断一个字符是否是字母或下划线
int isID(char c) {
    return isalpha(c) || c == '_';
}

// 判断一个字符是否是特殊符号之一
int isSymbol(char c) {
    return strchr(symbols, c) != NULL;
}

// 输出词法单元的类型和值
void printToken() {
    switch (tokenType) {
        case ERROR:
            printf("ERROR: %s\n", tokenString);
            break;
        case ID:
            printf("ID: %s\n", tokenString);
            break;
        case NUM:
            printf("NUM: %s\n", tokenString);
            break;
        case COMMENT:
            printf("COMMENT: %s\n", tokenString);
            break;
        case ASSIGN:
            printf("ASSIGN: %s\n", tokenString);
            break;
        case SYMBOL:
            printf("SYMBOL: %s\n", tokenString);
            break;
        default:
            printf("Unknown token type: %d\n", tokenType);
            break;
    }
}
```

```c
// 扫描并输出一个词法单元，如果到达文件尾，返回 EOF；如果到达行尾，返回 EOLN；否则返回 OK
int scanToken(FILE *fp) {
    // 定义当前状态为 START
    enum StateType state = START;
    // 定义当前词法单元长度为 0
    int tokenLen = 0;
    // 定义当前字符和下一个字符
    int currentChar, nextChar;
    // 定义循环标志为真
    int loop = 1;
    // 循环直到状态为 DONE 或遇到文件尾或行尾
    while (loop && state != DONE) {
    // 获取下一个字符，并判断是否为文件尾或行尾
        nextChar = getNextChar(fp);
        if (nextChar == EOF) {
            // 文件尾，返回 EOF，并退出循环
            return EOF;
        }
        if (nextChar == '\n') {
            // 行尾，增加行号，并判断当前状态
            lineno++;
            switch (state) {
                case INID:
                case INNUM:
                case INASSIGN:
                case ERROR:
                // 这些状态下，行尾表示词法单元结束，转移到 DONE，并退回一个字符
                    state = DONE;
                    ungetNextChar();
                    break;
                case INCOMMENT:
                // 这个状态下，行尾表示注释未结束，输出错误信息，并转移到 DONE
                    state = DONE;
                    tokenType = ERROR;
                    printf("Line %d: Comment not closed\n", lineno);
                    break;
                default:
```

```
                break;
            default:
            // 其他状态下，行尾不影响词法分析，继续循环
                break;
        }
    } else {
        // 不是文件尾或行尾，根据当前状态和当前字符进行相应的处理
        switch (state) {
            case START:
            // 开始状态，根据字符的类型转移到相应的状态，并追加字符
                if (isID(nextChar)) {
                // 字母或下划线，转移到 INID，并设置词法单元类型为 ID
                state = INID;
                tokenType = ID;
                } else if (isdigit(nextChar)) {
                    // 数字，转移到 INNUM，并设置词法单元类型为 NUM
                    state = INNUM;
                    tokenType = NUM;
                } else if (nextChar == ':') {
                    // 冒号，转移到 INASSIGN，并设置词法单元类型为 ASSIGN
                    state = INASSIGN;
                    tokenType = ASSIGN;
                } else if (nextChar == '{') {
                    // 左花括号，转移到 INCOMMENT，并设置词法单元类型为 COMMENT
                    state = INCOMMENT;
                    tokenType = COMMENT;
                } else if (isSymbol(nextChar)) {
                    // 特殊符号，直接转移到 DONE，并设置词法单元类型为 SYMBOL
                    state = DONE;
                    tokenType = SYMBOL;
                } else if (isspace(nextChar)) {
                    // 空白符号，忽略并保持在 START 状态
                } else {
                    // 其他字符，直接转移到 DONE，并设置词法单元类型为 ERROR
                    state = DONE;
                    tokenType = ERROR;
                }
                // 追加字符到词法单元中
                tokenString[tokenLen++] = nextChar;
                break;
            case INID:
                // 标识符状态，如果字符是字母、数字或下划线，保持状态并追加字符；否则转移到 DONE，并退回一个字符
                if (isID(nextChar) || isdigit(nextChar)) {
                // 保持状态并追加字符
                    tokenString[tokenLen++] = nextChar;
                } else {
                    // 转移到 DONE，并退回一个字符
                    state = DONE;
                    ungetNextChar();
                }
                break;
            case INNUM:
                // 数字状态，如果字符是数字或小数点，保持状态并追加字符；否则转移到DONE，并退回一个字符
                if (isdigit(nextChar) || nextChar == '.') {
                    // 保持状态并追加字符
                    tokenString[tokenLen++] = nextChar;
                } else {
                    // 转移到 DONE，并退回一个字符
                    state = DONE;
                    ungetNextChar();
                }
                break;
            case INCOMMENT:
                // 注释状态，如果字符是右花括号，转移到 DONE；否则保持状态并追加字符
                if (nextChar == '}') {
                    // 转移到 DONE
                    state = DONE;
                } else {
                    // 保持状态并追加字符
                    tokenString[tokenLen++] = nextChar;
                }
                break;
            case INASSIGN:
                // 赋值符号状态，如果字符是等号，转移到 DONE；否则转移到 ERROR，并退回一个字符
                if (nextChar == '=') {
```

```
                case INASSIGN:
                    // 赋值符号状态，如果字符是等号，转移到 DONE；否则转移到 ERROR，并退回一个字符
                    if (nextChar == '=') {
                        // 转移到 DONE，并追加字符
                        state = DONE;
                        tokenString[tokenLen++] = nextChar;
                    } else {
                        // 转移到 ERROR，并退回一个字符
                        state = ERROR;
                        ungetNextChar();
                    }
                    break;
                default:
                    // 其他状态，不应该出现，输出错误信息，并退出循环
                    printf("Unknown state: %d\n", state);
                    loop = 0;
                    break;
            }
        }
        // 如果词法单元长度超过最大长度，输出错误信息，并截断词法单元
        if (tokenLen >= MAXTOKENLEN) {
            printf("Token too long: %s\n", tokenString);
            tokenString[MAXTOKENLEN] = '\0';
        }
    }
    // 如果状态是 DONE，输出词法单元的类型和值，并返回 OK
    if (state == DONE) {
        printToken();
        return OK;
    }
    // 否则，返回未知的结果
    return UNKNOWN;
}
```

```
请输入文件名:
3.txt
C:\Users\lenovo\Desktop\编译原理与技术\小班课\第二次\3.txt
1       int x,fact;
2       read x; { input an integer }
3       if 0 < x then { don't compute if x <= 0 }
4       fact := 1;
5       repeat
6       fact := fact * x;
7       x := x - 1
8       until x = 0;
9       write fact { output factorial of x }
10      end
词法分析完成
1 1     [int : KEY_INT]
1 5     [x : ID]
1 6     [, : OP_COMMA]
1 7     [fact : ID]
1 11    [; : OP_SEMICOLON]
2 1     [read : KEY_READ]
2 6     [x : ID]
2 7     [; : OP_SEMICOLON]
3 1     [if : KEY_IF]
3 4     [0 : NUMBER]
3 6     [< : OP_LSS]
3 8     [x : ID]
3 10    [then : KEY_THEN]
4 1     [fact : ID]
4 6     [:= : OP_ASSIGN]
4 9     [1 : NUMBER]
4 10    [; : OP_SEMICOLON]
5 1     [repeat : KEY_REPEAT]
6 1     [fact : ID]
6 6     [:= : OP_ASSIGN]
6 9     [fact : ID]
6 14    [* : OP_MUL]
6 16    [x : ID]
6 17    [; : OP_SEMICOLON]
7 1     [x : ID]
7 3     [:= : OP_ASSIGN]
7 6     [x : ID]
7 8     [- : OP_SUB]
7 10    [1 : NUMBER]
8 1     [until : KEY_UNTIL]
```

```
7 10    [1 : NUMBER]
8 1     [until : KEY_UNTIL]
8 7     [x : ID]
8 9     [= : OP_EQU]
8 11    [0 : NUMBER]
8 12    [; : OP_SEMICOLON]
9 1     [write : KEY_WRITE]
9 7     [fact : ID]
10 0    [end : KEY_END]
```

实验成绩	