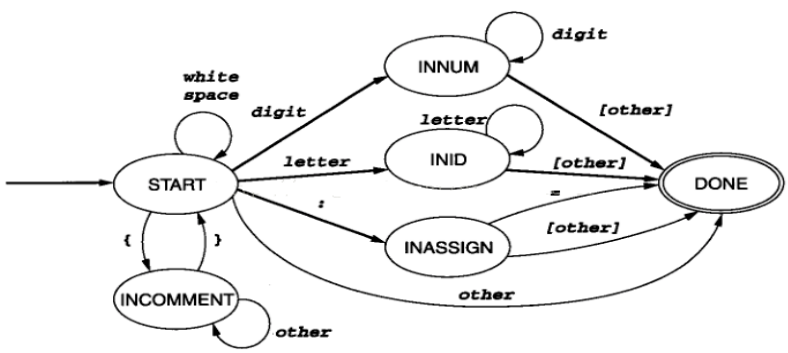
四 川 大 学 计 算 机 学 院、软 件 学 院

实 验 报 告

学号：2014141462082 姓名：胡 研 专业： 计算机 班级： 4 第 1 周

课程名称	编译原理课程设计	实验课时	2
实验项目	Tiny 词法分析	实验时间	2017.4.3
实验目的	熟悉 Tiny 语言的词法 手工构造 Tiny 的词法分析器 构造 DFA,设计数据类型、数据结构		
参考资料	《编译原理及实践》 第四版 《程序设计语言编译原理》 第一版		
Tiny 语言词法规则特点	Tiny 的词法规则 1.关键字：read write if end repeat until else 2.注释：大括号“{注释}” 3.类型：整型和布尔类型 4.运算符：+ - * / ( ) < = := 5.Tiny 的正则表达式 ID = letter+ NUM = digit+ letter = [a-zA-Z] digit = [0-9] Tiny 语言的 DFA 		

数据类型、数据结构设计	DFA 方案 初始状态为 start，遇到{则进入注释，直到}结束注释并回到 start,数字则进入 INNUM 并一直循环,直到非数字结束,进入 Done，字母同理进入 INID，遇到=进入赋值语句，读取到下一个字符结束 string lineBuf[]:用于存储每一行的字符，并逐个读取分析。 static int lineno:静态变量，存储行号 char tokenString[MAXTOKENLEN + 1]; 存放读取到的 token extern FILE* source; 外部文件指针，用于读取 extern FILE* listing; 外部文件指针，用于输出 enum StateType 枚举变量，表当前状态 static int linepos = 0; 当前读取位置 static int bufsize = 0; 最大缓冲池大小 #define BUFLEN 256 宏定义，一行最多字节 enum TokensType 枚举变量，表示 token 的类型 static struct { char* str; TokensType tok; }reservedWords[MAXRESERVED] 定义结构，用于读取关键字

关键代码的分析	

```
static char getNextChar(void)       //获取下一个非空字符，lineBuf满了就装到一个新行里
{
    if (!(linepos < bufsize))
    {
        lineno++;
        if (fgets(lineBuf, BUFLEN - 1, source))  //fgets(_Out_writes_z_(_MaxCount) char * _Buf, _In_ int _MaxCount, _Inout_ FILE * _File)
            //                  输出文件，最大个数，读入文件指针
        {
            if (EchoSource)     //是否回显读取的内容
            {
                fprintf(listing, "%4d:%s", lineno, lineBuf);
            }
            bufsize = strlen(lineBuf);
            linepos = 0;
            return lineBuf[linepos++];
        }
        else
        {
            EOF_flag = true;
            return EOF;
        }
    }
    else return lineBuf[linepos++];

                                    }



TokensType getToken(void) //getToken 方法用于识别读取到的内容，并返回token
{
    int tokenStringIndex = 0;
    TokensType currentToken;  //创建一个Token用于表示当前的记号
    StateType state = START;  //入口
    int save;
    while (state != DONE)
    {
```

```
            int c = getNextChar();
            save = true;
            switch (state)
            {
            case START:
                if (isdigit(c))    //如果是数字，state表示在num中
                {
                    state = INNUM;
                }
                else if (isalpha(c))   //字母，进入ID可能是关键字或者ID
                {
                    state = INID;

                                                }

                            ……省略部分重复代码



case INCOMMENT:
                save = false;
                if (c == EOF)      //结束符号
                {
                    state = DONE;
                    currentToken = ENDFILE;
                }
                else if (c == '}')
                {
                    state = START;    //注释结束，继续读取
                }
                break;
            case INASSIGN:
                state = DONE;
                if (c == '=')
                {
                    currentToken = ASSIGN;
                }
                else
                {
                    ungetNextChar();     //回退一个字符
                    save = false;
```

```
                currentToken = ERROR;
            }

                                    break;



static TokensType reservedLookup(char* s)     //查找关键字
{
    int i;
    for (int i = 0; i < MAXRESERVED; i++)     //遍历寻找关键字
    {
        if (!strcmp(s, reservedWords[i].str))          //strcmp（S1,S2）
S1 > S2 返回值大于0,S1 <S2 返回值小于0，相等返回0
        {
            return reservedWords[i].tok;     //返回关键字
        }
    }
    return ID;                                //没有找到，返回ID

                                }


void printToken(TokensType token, char *tokenString)
                        //打印读取到的符号或者关键字
{
    switch (token)
    {
    case IF:
    case THEN:
    case ELSE:
    case END:
    case REPEAT:
    case UNTIL:
    case READ:
    case WRITE:
        fprintf(listing, "\treserved word:%s\n", tokenString);
        break;
    case ID:
        fprintf(listing, "\tID, name = %s\n", tokenString);
        break;
```

```
        case NUM:
            fprintf(listing, "\tNUM, val = %s\n", tokenString);
            break;
        case ASSIGN:
        case EQ:
        case LT:
        case PLUS:
        case MINUS:
        case TIMES:
        case OVER:
        case LPAREN:
        case RPAREN:
        case SEMI:
            fprintf(listing, "\t%s\n", tokenString);
        }

                                    }



int _tmain(int argc, _TCHAR* argv[])       //主函数
{

    fprintf(listing, "%s\n",
"***************************************");
    fprintf(listing, "%s\n", "TINY Lexical Analazer Result:");
    fprintf(listing, "%s\n",
"***************************************");

    TokensType token;                       //创建一个Token开始循环读取
    do
    {
        token = getToken();                 //调用getToken开始词法扫描
    } while (token != ENDFILE);
    fprintf(listing, "\t%s\n", "EOF");  //输出结束符
    fclose(listing);                        //关闭文件
    printf("%s\n", "已完成扫描并保存到myresult.txt...");
    return 0;

                                    }
```

结　论（运行结果实例、分析）

输入示例

example.tiny - 记事本

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

```
{ Sample program in TINY language - computes factorial }
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
        fact := 1;
        repeat
                fact := fact * x;
                x := x - 1
        until x = 0;
        write fact { output factorial of x }
end
```

运行结果

myresult.txt - 记事本

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

```
****************************************
TINY Lexical Analazer Result:
****************************************
        5:      reserved word:read
        5:      ID, name = x
        5:      ;
        6:      reserved word:if
        6:      NUM, val = 0
        6:      <
        6:      ID, name = x
        6:      reserved word:then
        7:      ID, name = fact
        7:      :=
        7:      NUM, val = 0
        7:      ;
        8:      ID, name = i
        8:      :=
        8:      ID, name = x
        8:      ;
        9:      reserved word:repeat
        10:     ID, name = fact
        10:     :=
        10:     ID, name = fact
        10:     +
        10:     ID, name = i
        10:     ;
```

myresult.txt - 记事本

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

```
        8:      ID, name = i
        8:      :=
        8:      ID, name = x
        8:      ;
        9:      reserved word:repeat
        10:     ID, name = fact
        10:     :=
        10:     ID, name = fact
        10:     +
        10:     ID, name = i
        10:     ;
        11:     ID, name = i
        11:     :=
        11:     ID, name = i
        11:     -
        11:     NUM, val = 1
        12:     reserved word:until
        12:     ID, name = i
        12:     =
        12:     NUM, val = 0
        12:     ;
        13:     reserved word:write
        13:     ID, name = fact
        14:     reserved word:end
        15:     EOF
```

运行结果分析

实验结果达到了实验的要求，对于变量定义语句，IF 语句，重复语句，读写语句都得到了正确的分析结果，正确的输出了行号，并且实现了注释的去除，最后在执行 end 语句处结束了词法扫描

小　结	顺利完成实验，熟悉了 Tiny 语言和其词法。根据语言和词法规则，顺利构造 DFA。成功用 C++语言，根据构造的 DFA，增强了自己的编程能力和水平技巧，使用了较大的数据结构，熟悉了 C++语言,实践了课本上 Tiny 语言的语法分析,实现了 Tiny 词法分析器。
指导老师评　议	成绩评定：　　指导教师签名：