湖南大學
HUNAN UNIVERSITY

课程实验报告

课　程　名　称：编译技术

实验项目名称：词法分析器构造工具的实现

专　业　班　级：软件 2105 班

姓　　名：马小梅

学　　号：202126010530

指　导　教　师：马小梅

完　成　时　间：2024 年 5 月 8 日

信息科学与工程学院

实验题目：词构建词法分析器构造工具的实现

实验目的：基于第四章知识，用正则语言描述出 TINY 语言的词法，然后得出语言的词法分析器的完整代码。

实验环境：

Windows 系统，codeblocks

实验内容及操作步骤：

1 TINY 语言特点：

- 语句序列用分号隔开；
- 所有变量都是整型变量，且不需要声明；
- 只有两个控制语句，if 和 repeat;
- if 判断语句必须以 end 结束，且有可选的 else 语句；
- read 和 write 完成输入输出；
- 花括号表示注释，但不允许嵌套注释；
- 有<和=两个比较运算符，有+，-，，/简单运算符；

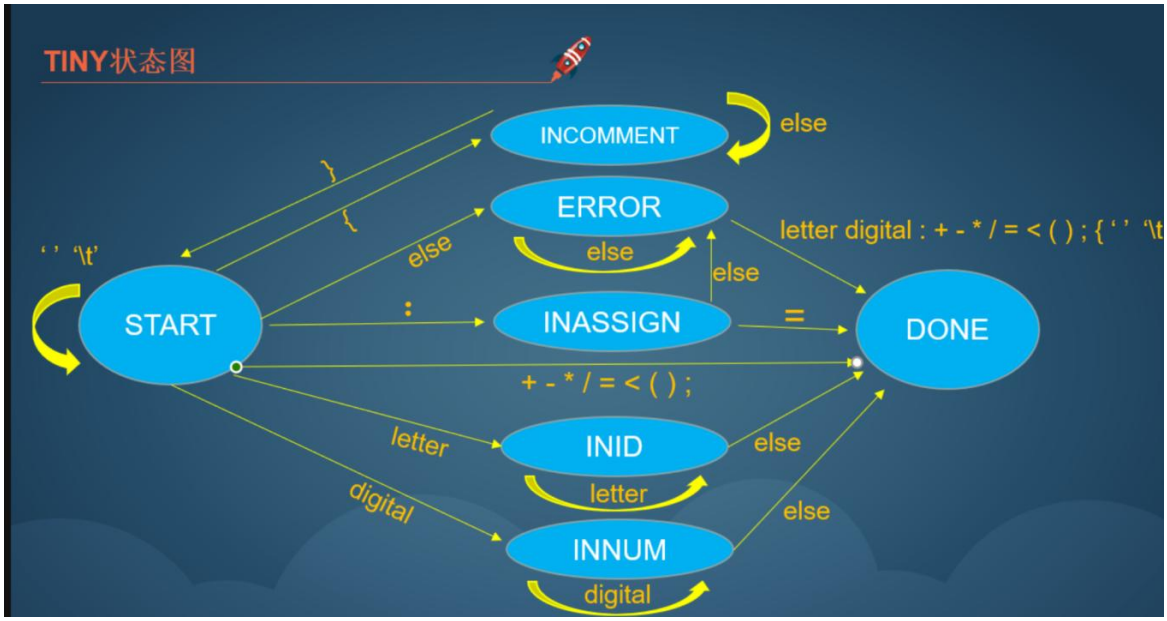
2 TINY 可识别的词法单元

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：( )
- 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：;

3 根据 TINY 关键字画出的 DFA 状态转换图如下图所示：



其中，START 状态：当读入空格或者缩进符时状态不变，当读入数字时进入状态 INNUM，读入字母时进入状态 INID，读入：时进入识别：=的 INASSIGN 状态，输入左{ 时进入注释状态 INCOMMENT，读入+-*/( )=<时直接进入 DONE 状态，读入其他字符时进入 ERROR 状态；

INNUM 状态：当读入数字时状态保持不变，当读入非数字字符时进入 DONE 状态；

INID 状态，INASSIGN 状态同 INNUM 状态一样识别非字母非=；

ERROR 状态：当读入+-*/<=()时，进入 DONE 状态，表示识别出错误符号+-*/<=()，读入其他字符时仍停留在 ERROR 状态；

DONE 状态:进入这个状态表示 TINY 标记识别完毕，需要开始输出识别的词法单元。

4 代码说明

4.1 一些重要变量的声明

```
enum TokenType{    //规定TINY语言可能出现的标记
    ERR,NONE,      //错误，空词
    IF,THEN,ELSE,END,REPEAT,UNTIL,READ,WRITE,    //TINY的保留词
    ID,NUM,        //单词，数字
    ASSIGN,PLUS,MINUS,MULTI,DIV,LESS,LPAR,RPAR,COLON,EQ    //TINY的特殊符号
};

enum StateType{    //规定状态机的所有状态
    START,DONE,ERROR,    //开始状态，结束状态，错误状态
    INID,INNUM,INASSIGN,INCOMMENT    //单词状态，数字状态，:=符号状态，注释状态
};
```

4.2 判断输入字符是否为字母，数字，操作符，空格或缩进符，关键词，设计思路是主要利用 if 条件语句进行判断输出。
isID()识别一个字母
isNUM()识别一个数字
isOperator()识别一个除了:=的 TINY 特殊符号
isWhiteSpace()识别一个空格或者缩进符

```
bool isID(char c){    //识别一个字符是否为字母
    if((c >= 'a' && c <= 'z')||(c >= 'A' && c <= 'Z'))
    return true;
    return false;
}

bool isNUM(char c){   //识别一个字符是否为数字
    if(c >= '0' && c <= '9')
    return true;
    return false;
}

bool isOperator(char c){   //识别一个字符是否为TINY特殊运算符(除了:=)
    if(c == '+'||c == '-'||c == '*'||c == '/'||c == '='||c == '<'||c == '('||c == ')'||c == ';')
    return true;
    return false;
}

bool isWhiteSpace(char c){  //识别一个字符是否为空格或者缩进符
    if(c == ' ' || c == '\t')
    return true;
    return false;
}
```

4.3showWord()输出当前代码行识别出的**词法单元<标记 token,字符串 string。**

实现思路主要是利用 switch 分支根据标记分别输出对应的词法分析语句：
保留字输出“reserved word:”
单词输出“ID，name=”
数字输出“NUM，val=”
错误输出“ERROR，error=”
特殊字符直接输出

```
void showWord(TokenType tok,string s){   //输出当前代码行识别出的标记以及识别字符串
    if(tok == NONE)
        return;
    switch(tok){
        case IF:
        case THEN:
        case ELSE:
        case END:
        case REPEAT:
        case UNTIL:
        case READ:
        case WRITE:
            cout << "      " << lineno << ":  " << "reserved word: " << s << endl;break;  //以上标记为TINY保留字
        case ID:
            cout << "      " << lineno << ":  " << "ID, name= " << s << endl;break;  //标记为单词
        case NUM:
            cout << "      " << lineno << ":  " << "NUM, val= " << s << endl;break;  //标记为数字
        case ERR:
            cout << "      " << lineno << ":  " << "ERROR, error= " << s << endl;break;  //标记为错误
        case ASSIGN:
        case PLUS:
        case MINUS:
        case MULTI:
        case DIV:
        case LESS:
        case LPAR:
        case RPAR:
        case COLON:
        case EQ:
            cout << "      " << lineno << ":  " << s << endl;break;  //以上标记为TINY特殊字符
        default: break;
    }
}
```

4.4 词法扫描当前代码行，即 DFA 状态转换图的代码解释，根据当前所处状态和输入字符判断下一状态走势并标识。每次扫描一行代码的字符串，所以主循环是 while(linepos < linesize)取字符分析直到 line 的最后一个字符被分析完毕。使用 **switch()实现一个状态转换函数，case 的情况就是当前状态 state 的情况，每个 case 表示的 state 状态又因为当前识别字符串的不同又进行分支**。总的分析情况就不赘述了，完全与上面给出的 DFA 状态转换图一致。**其中 saveflag = true 表示识别的字符需要加入识别字符串 ans 中去，每进入一次 DONE 状态都代表一个标记被识别，就要给 token 赋对应的值，而且对应的识别字符串 ans 输出后需要 ans.clear()清空初始化以迎接下一次识别。**

还需要注意的地方有，当 INID、INNUM、INASSIGN、ERROR 状态分别识别了非字母字符、非数字字符、非 = 字符、非字母、数字、空格、缩进、特殊符号、左{字符后都需要 linepos-- 表示当前字符虽然未被识别但是已经被匹配过，可能是其他状态的可识别字符，需要回溯。最后每到一个代码行的末尾必须强制进入 DONE 状态，因为除了 INCOMMENT 状态外没有哪一个字符串可以换行表示，所以也需要一个 switch()来分析当前状态强制进入 DONE 后的情况。**最后再根据标记 token 和识别字符串 ans 调用 showWord()输出。**

```
void scanToken() {  //词法扫描当前代码行
    ans.clear();    //新的代码行开始初始化识别字符串
    linepos = 0;    //当前代码行字符位置初始化
    linesize = (int)line.length();    //获得当前代码行长度
    while (linepos < linesize) {  //代码行字符位置小于代码行长度时状态机不断识别字符
        saveflag = true;          //字符是否保存到当前识别字符串标志初始化
        char ch = line[linepos];  //ch存储当前字符
        switch (state) {          //状态机
            case START://开始状态
                if (isWhiteSpace(ch))
                    saveflag = false;   //状态不变，字符不保存
                else if (isID(ch))
                    state = INID;      //进入单词状态，字符保存
                else if (isNUM(ch))
                    state = INNUM;     //进入数字状态，字符保存
                else if (ch == ':')
                    state = INASSIGN;   //及进入:=符号状态，字符保存
                else if (ch == '{') {
                    saveflag = false;     //进入注释状态，字符不保存
                    state = INCOMMENT;
                } else if (isOperator(ch)) {
                    state = DONE;     //识别TINY特殊符号，进入结束状态
                    if (ch == '+')
                        token = PLUS;          // +标记为PLUS
                    else if (ch == '-')
                        token =  MINUS;  // -标记为MINUS
                    else if (ch == '*')
                        token =  MULTI;  // *标记为MULTI
                    else if (ch == '/')
                        token =  DIV;    // /标记为DIV
                    else if (ch == '=')
                        token =  EQ;     // =标记为EQ
                    else if (ch == '<')
                        token =  LESS;   // <标记为LESS
                    else if (ch == '(')
                        token =  LPAR;   // (标记为LPAR
                    else if (ch == ')')
                        token =  RPAR;   // )标记为RPAR
                    else
                        token =  COLON;                // ;标记为COLON
                } else
                    state = ERROR;  //其余字符进入错误状态
                break;
            case INID://单词状态
                //字符不为字母时，进入结束状态，字符位置返回，字符不保存，标记为ID
                if (!isID(ch)) {
                    state = DONE;
                    linepos--;
                    saveflag = false;
                    token = ID;
                }
                break;
```

```
            case INNUM://数字状态
                //字符不为数字时，进入结束状态，字符位置返回，字符不保存，标记为NUM
                if (!isNUM(ch)) {
                    state = DONE;
                    linepos--;
                    saveflag = false;
                    token = NUM;
                }
                break;
            case INASSIGN://:=符号状态
                if (ch == '=') {
                    state = DONE;    //字符为 = 时，进入结束状态，字符保存，标记为ASSIGN
                    token = ASSIGN;
                } else {
                    state = ERROR;    //其余字符进入错误状态，字符位置返回，字符不保存
                    linepos--;
                    saveflag = false;
                }
                break;
            case INCOMMENT://注释状态
                if (ch == '}') {
                    state = START;
                    saveflag = false;
                } //字符为 } 时，进入开始状态，字符不保存
                else
                    saveflag = false; //其余字符状态不变，字符不保存
                break;
            case ERROR://错误状态
                //字符为数字、字母、空格、缩进、特殊符号、花括号时进入结束状态，字符位置返回，字符不保存，标记为ERR
                if (isNUM(ch) || isID(ch) || isWhiteSpace(ch) || isOperator(ch) || ch == ':' || ch == '{') {
                    state = DONE;
                    linepos--;
                    saveflag = false;
                    token = ERR;
                }
                break;
            default:
                break;
        }

        linepos++; //字符位置向后偏移
        if (saveflag) //当前字符ch保存标记为真时，将ch加入识别字符串ans
            ans += ch;
        if (linepos == linesize) { //字符位置等于代码行长度，溢出
            switch (state) {
                case START:
                    state = DONE;
                    token = NONE;
                    break;    //开始 -> 结束，标记为NONE
                case INID:
                    state = DONE;
                    token = ID;
                    break;        //单词 -> 结束，标记为ID
                case INNUM:
                    state = DONE;
                    token = NUM;
                    break;     //数字 -> 结束，标记为NUM
                case INASSIGN:
                    state = DONE;
                    token = ERR;
                    break;  // :=  -> 结束，标记为ERR
                case ERROR:
                    state = DONE;
                    token = ERR;
                    break;     //错误 -> 结束，标记为ERR
                default:
                    break;
            }
        }
        // 当前为结束状态时，进入开始状态，输出当前代码行识别出的标记以及识别字符串，清空识别字符串
        if (state == DONE) {
            state = START;
            showWord(identifyReserved(token, ans), ans);
            ans.clear();
        }
    }
}
```

4.5 代码加载函数，首先打开 test.txt 文件，assert()函数检测 test.txt 是否打开。然后进入一个循环，**不断取出文件中的一行代码字符，并用 line 字符串存储，**lineno 存储当前行号。随后输出当前行号及代码字符串。判定 line.empty()，如果不为 null 调用 **scanToken()函数进行词法扫描。**最后当文件读取到 EOF 表示文件读取完毕时发现状态 state 仍然为 INCOMMENT 说明 TINY 代码存在一个错误“注释未完成”，于是打印提示信息。

```
void codeLoading(){  //代码加载函数
    fstream myfile("test1.txt",ios::in);  //打开 test1.txt
    assert(myfile.is_open());    //检测 test.txt 是否正常打开
    while(!myfile.eof()){   //文件输入未检测到字符EOF结束
        line.clear();   //清空之前的代码行字符串
        getline(myfile,line);  //将当前代码行字符串(包含空格、缩进，不包含\n)赋予line
        cout << "LINE" << ++lineno << ":" << line << endl;  //输出当前代码行号及代码行字符串
        if(!line.empty())    //当前代码行字符串非空时进行词法扫描
            scanToken();
        if(myfile.eof() && state == INCOMMENT)   //当文件输入检测到字符EOF结束且状态仍为注释状态时报告 注释未完成 错误
            cout << "       " << lineno << ":  " << "ERROR, error= Incomplete comment";
    }
}
```

示例 TINY 代码的词法分析运行结果如下图所示：前 6 行待测代码及分析结果如下图所示：

```
{ 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;
```

```
LINE1:{ Sample program in TINY language-computes factorial}
LINE2:read x; { input an integer }
      2:  reserved word: read
      2:  ID, name= x
      2:  ;
LINE3:if 0 < x then { don't compute if x <= 0 }
      3:  reserved word: if
      3:  NUM, val= 0
      3:  <
      3:  ID, name= x
      3:  reserved word: then
LINE4:  fact := 1;
      4:  ID, name= fact
      4:  :=
      4:  NUM, val= 1
      4:  ;
LINE5:  repeat
      5:  reserved word: repeat
LINE6:    fact := fact * x;
      6:  ID, name= fact
      6:  :=
      6:  ID, name= fact
      6:  *
      6:  ID, name= x
      6:  ;
```

```
    x := x - 1
  until x = 0;
  write fact  { output factorial of x}
end
```

```
LINE7:    x := x - 1
      7:  ID, name= x
      7:  :=
      7:  ID, name= x
      7:  -
      7:  NUM, val= 1
LINE8:  until x = 0;
      8:  reserved word: until
      8:  ID, name= x
      8:  =
      8:  NUM, val= 0
      8:  ;
LINE9:  write fact  { output factorial of x}
      9:  reserved word: write
      9:  ID, name= fact
LINE10:end
      10:  reserved word: end
LINE11:
```

收获与体会：

通过这次实验对词法分析的整个过程有了一个全面的梳理，对 DFA 的重要性有了更深的体会，整体实现函数感觉不是很难，主要是对 DFA 状态图的梳理与理解。

实验成绩	