词法分析

17341197 张富瑞

一.实验目的

1.实验目的：通过扩充已有的样例语言TINY语言的词法分析程序，为扩展TINY语言TINY + 构造词法分析程序，从而掌握词法分析程序的构造方法

2.实验内容：了解样例语言TINY及TINY编译器的实现，了解扩展TINY语言TINY + ，用C语言在已有的TINY词法分析器基础上扩展，构造TINY + 的词法分析程序。

3.实验要求：将TINY + 源程序翻译成对应的TOKEN序列，并能检查一定的词法错误。

二.实验思路

1.关键字分析

- Tiny 关键字

  if then else end repeat until read write

- Tiny+添加的关键字

  true false or and not int bool string while do

且这些关键字都需要小写。

2.Token

为之后分析的方便，不能够单单的将将关键字的种类设为token，需要进行细化。这里将上面的tiny+的关键字的token分类如下：

(TK_TRUE, true), (TK_FALSE, false), (TK_OR, or)

(TK_AND, and), (TK_NOT, not), (TK_INT, int),

(TK_BOOL, bool), (TK_STRING, '...'), (TK_WHILE, while),

(TK_DO, do), (TK_IF, if), (TK_THEN, then),

(TK_ELSE, else), (TK_END, end), (TK_REPEAT, repeat),

(TK_UNTIL, until), (TK_READ, read), (TK_WRITE, write),

同时token中还有其他的符号需要定义：

> <= >= , '

{ } ; := + - * / ( ) < =

token的种类定义为：

(TK_GTR, >), (TK_LEQ, <=), (TK_GEQ, >=)

(TK_COMMA, ,), (TK_SEMICOLON, ;), (TK_ASSIGN, :=),

(TK_ADD, +), (TK_SUB, -), (TK_MUL, *),

(TK_DIV, /), (TK_LP, (), (TK_RP,)),

(TK_LSS, <), (TK_EQU, =)

3.字符处理思路

通过读取单个字符的方式读取整个文件的内容，在每一个字符的读取的过程中，通过dfa的状态转换，来分割出单个的token，同时可以判断出一下token的错误。

三.实验过程

```
        in.get(ch);
        if (in.fail() || in.eof()) {
            if (!word.empty()){
                figureWord();
            }
            //跳出while
            break;
        }
        charNumber++;
```

进行每一个字符的读取，当读到文件末尾的时候退出。word是临时存储字符串变量，用来记录当前正在处理的token的名字。

```
//处理当前ch为sign 但是没判断为sign 且有word的情况  单个sign
        if(word.size() != 0&&type != TokenType::ANNOTATION&& type !=
TokenType::STRING&&
        isSign == false&&Utils::isValidSign(ch)){
            figureWord();
        }
```

这里处理当前的字符为“+-*/”这类的符号，当字符为符号的时候，如果当前还有保留word，且type不是注解也不是字符串的时候，我们需要利用当前符号分割，将word识别成对应的token。

```
//当前的type已经为sign(前一个字符为sign)  :1 ;  不是单个sign := ; 到下一步的时候处理为:=
       if(isSign&&Utils::nameToToken(word) != TokenType::NONE &&
                    Utils::nameToToken(word + ch) == TokenType::NONE){
            figureWord();
        }
```

这里处理长度为2的符号的问题，上一个字符已经是符号，如果当前字符加上上一个字符不能够组成新的正常的符号，那么我们将前一个符号进行处理，得到一个token。

```
        if(word.size() == 0){
            if (ch == '{'){
                type = TokenType::ANNOTATION;
            }else if (ch == '\''){
                type = TokenType::STRING;
            }else if (isdigit(ch)){
                type = TokenType::NUMBER;
            }else if (isalpha(ch)){
                type = TokenType::ID;
            }else if (Utils::isValidSign(ch)){
                isSign = true;
            }else if (!Utils::isSeparator(ch)){
                log.error("unknown symbol: " + ch, lineNumber, charNumber);
            }
        }else{
            if (type == TokenType::STRING && ch == '\'') {
                word += ch;
                figureWord();
                continue;
            }
            if (type == TokenType::ANNOTATION && ch == '}') {
                word += ch;
                figureWord();
                continue;
            }
        }
```

这些是新的token判断的开始，如果新的token开始处理时满足条件，则类型转换成相应的token类型，进行下一步的处理。

```
//处理分割符 这里去掉了注释多行的问题
        if (type != TokenType::ANNOTATION && type != TokenType::STRING) {
            if (Utils::isSeparator(ch)) {
                if (!word.empty())figureWord();
                if (ch == '\n')newLine();
                continue;
            }
        }
```

这里处理换行，空格等分隔符的问题，如果当前处理的token不为空，则进行处理，得出token的类型。

```
void Scanner::figureWord() {
    //新建一个临时的Token
    Token token;
    token.token = word;
    token.type = type;
    token.line = lineNumber;
    token.offset = charNumber - 1;

    //enum
```

```
    TokenType t = Utils::nameToToken(word);

    //转成正常的type
    if (token.type == TokenType::NONE)token.type = t;
    else if (token.type == TokenType::ID && t != TokenType::NONE)token.type = t;

    //if word is error
    if (token.type == TokenType::NONE)log.error("unknown token: " + word,
lineNumber, charNumber - 1);

    //if word is not annotation 放入token list
    if (token.type != TokenType::ANNOTATION)list.push_back(token);
    word.clear();
    type = TokenType::NONE;
    isSign = false;
}
```

处理token的过程。用word去获取token以及和现在暂存的token进行比较，选择合适的token，之后放入token的队列中。

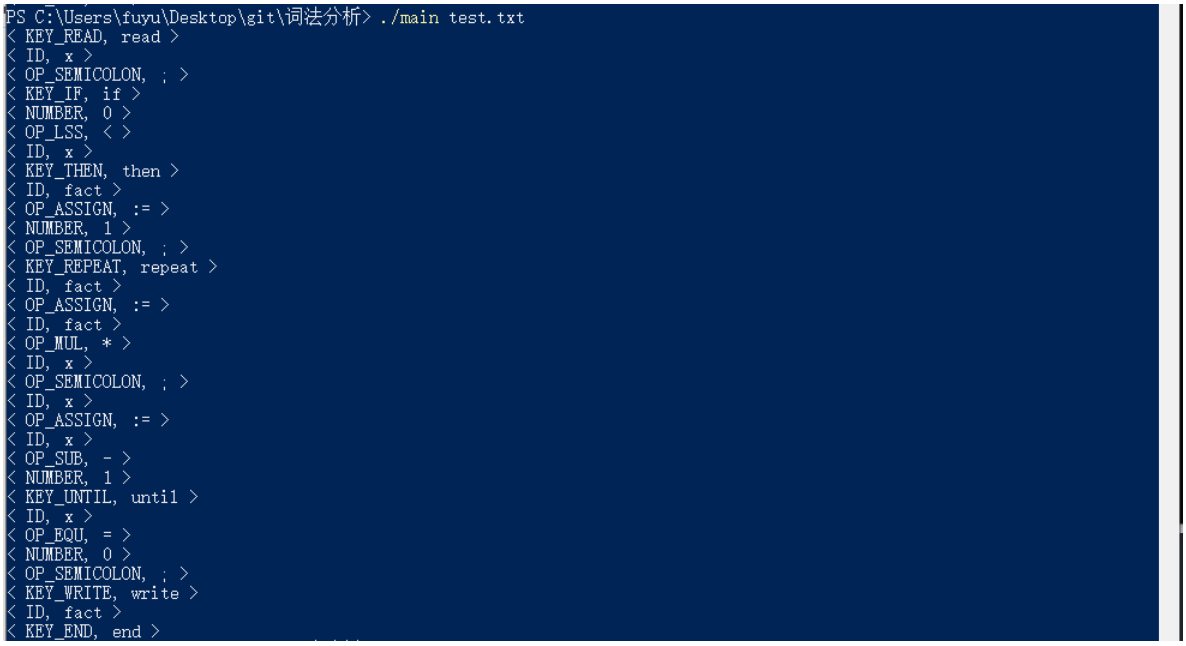
四.实验结果

测试tiny程序

```
{ 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
```

token结果


PS C:\Users\fuyu\Desktop\git\词法分析> ./main test.txt
< KEY_READ, read >
< ID, x >
< OP_SEMICOLON, ; >
< KEY_IF, if >
< NUMBER, 0 >
< OP_LSS, < >
< ID, x >
< KEY_THEN, then >
< ID, fact >
< OP_ASSIGN, := >
< NUMBER, 1 >
< OP_SEMICOLON, ; >
< KEY_REPEAT, repeat >
< ID, fact >
< OP_ASSIGN, := >
< ID, fact >
< OP_MUL, * >
< ID, x >
< OP_SEMICOLON, ; >
< ID, x >
< OP_ASSIGN, := >
< ID, x >
< OP_SUB, - >
< NUMBER, 1 >
< KEY_UNTIL, until >
< ID, x >
< OP_EQU, = >
< NUMBER, 0 >
< OP_SEMICOLON, ; >
< KEY_WRITE, write >
< ID, fact >
< KEY_END, end >

加入不能够识别的符号"&"


PS C:\Users\fuyu\Desktop\git\词法分析> ./main test.txt
ERROR IN LINE 3:1
ERROR IN LINE 3:2 unknown token: &&
You have 2 errors.