



东南大学国家示范性软件学院

College of Software Engineering  
Southeast University

# 编译原理

## 实验报告

实验名称： Lexical Analyzer

学生姓名： 孟越

学生学号： 71Y15102

东南大学 软件学院 制



## 一、实验目的

1. 理解词法分析原理及相关理论
2. 巩固课上知识即词法分析器工作的整个过程，加深对词法分析的理解
3. 编写一个简单的人工 lex，提升编程代码水平

## 二、实验说明

**1.编程环境：** windows7 & vs2010 & C++

**2.设计说明：** 1.自己设计一些正则表达式

2.人工画出 NFA,合并

3.简化为 mDFA

4.最后使用代码实现。

**3.数据结构算法说明：** 主要依据是 mDFA，一个字符一个字符地读取，使用 switch 实现不同状态下的代码执行从而实现状态转换图，最后直接输出即可。

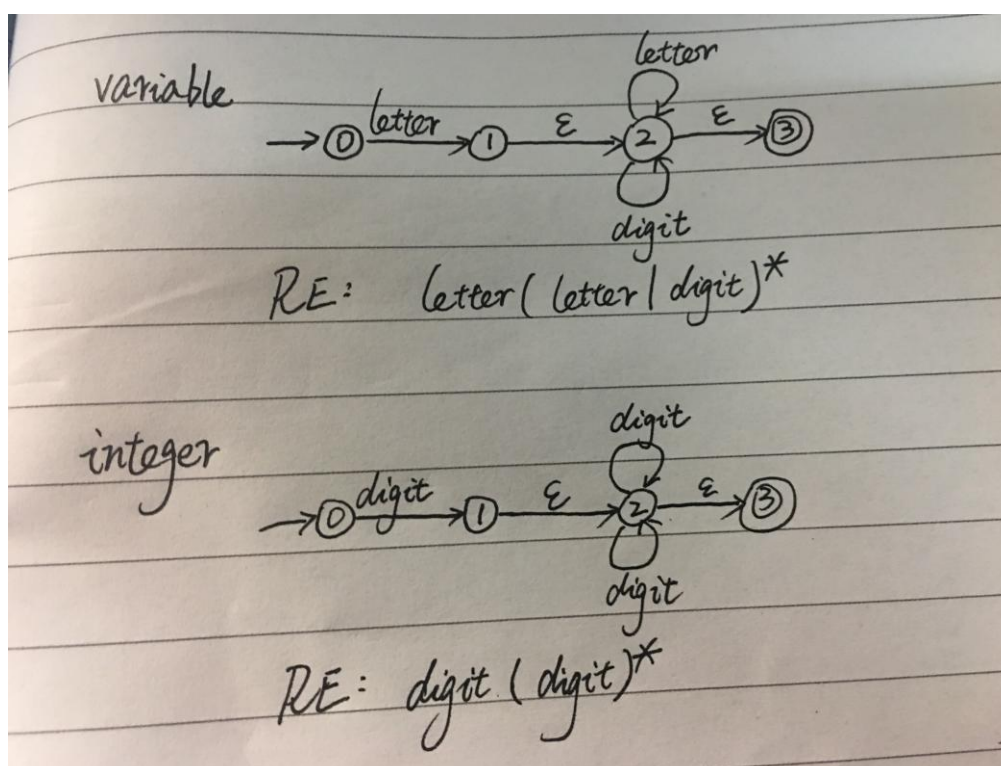


### 三、实验过程

1. 设计一段需要分析的程序代码

```
int int1=2;for(;int1<5;){int1=int1+2;}
```

2. 根据程序代码中所需要解析到的词设计需要的正则表达式并画出它们的 NFA



以下关键字在代码中只使用一次,不需分类,较简单,直接给出 DFA. (也可以分类,这里没有必要)

— for  $\rightarrow 0 \xrightarrow{f} 1 \xrightarrow{o} 2 \xrightarrow{r} 3$

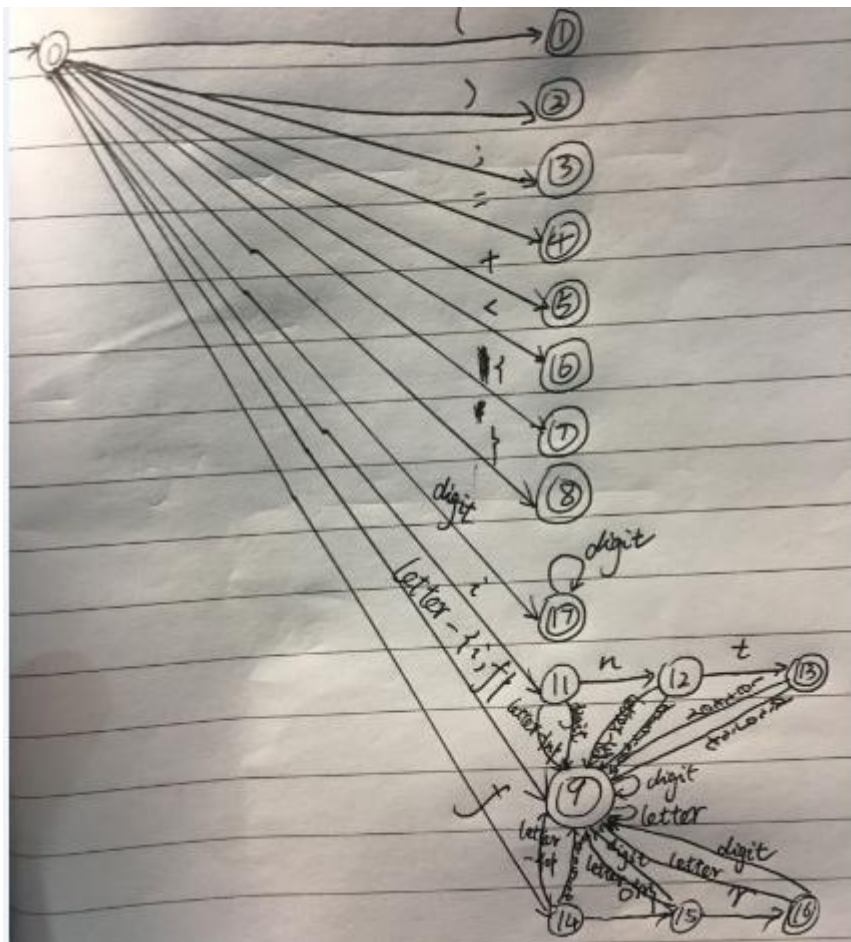
— int  $\rightarrow 0 \xrightarrow{i} 1 \xrightarrow{n} 2 \xrightarrow{t} 3$

—  $((, >, <, \{, \}, =, <, +, ;)^*$

$\rightarrow 0 \xrightarrow{*} 1$



3. 合并以上 NFA 并转化为 mDFA



4. 根据以上状态转换图编写代码

Main 函数:

```
void main()
{
    ifstream inputFile("LexicalInput.txt");
    ofstream outputFile("SyntaxInput.txt");
    //string int_="int int1=2;for(;int1<5;){int1=int1+2;}";
    string int_=" ";
    if(!inputFile.is_open()){
        cout<<"未成功打开文件"<<endl;
    }

    while(getline(inputFile, int_)){
    }
    int_+=" ";
    lexicalAnalyze(int_);
    outputFile<<result<<endl;
    system("pause");
    inputFile.close();
}
```



```
outputFile.close();  
}
```

辅助函数:

```
bool isDigit(char c){  
    if(c>='0' && c<='9'){  
        return true;  
    }  
    return false;  
}  
  
bool isLetter(char c){  
    if(c>='a' && c<='z'){  
        return true;  
    }  
    else if(c>='A' && c<='Z'){  
        return true;  
    }else{  
        return false;  
    }  
}  
  
void output(int state, string val){  
    string token="";  
    if(state==13)  
    {  
        token="(int, "+val+")";  
        cout<<token<<endl;  
        result+=(token+", ");  
    }else if(state==16){  
        token="(for, "+val+")";  
        cout<<token<<endl;  
        result+=(token+", ");  
    }else if(state==1 || state==2 || state==7 || state==8){  
        token="("+val+", "+val+")";  
        cout<<token<<endl;  
        result+=(token+", ");  
    }else if(state==3){  
        token="(;, "+val+")";  
        cout<<token<<endl;  
        result+=(token+", ");  
    }else if(state==4){  
        token="(=, "+val+")";  
        cout<<token<<endl;  
        result+=(token+", ");  
    }else if(state==5){  
        token="(+, "+val+")";
```



```
        cout<<token<<endl;
        result+=(token+", ");
    }else if(state==6){
        token="(<,"+val+"";
        cout<<token<<endl;
        result+=(token+", ");
    }else if(state==17){
        token="(integer,"+val+"";
        cout<<token<<endl;
        result+=(token+", ");
    }else if(state==9){
        token="(variable,"+val+"";
        cout<<token<<endl;
        result+=(token+", ");
    }
}
```

词法分析函数:

```
void lexicalAnalyze(string txt){
    int state=0;
    string val="";
    int i=0;
    while(i<txt.length())
    {
        switch(state)
        {
            case 0:
                if(txt[i]=='('){
                    state=1;
                    val="(";
                    output(state, val);
                    state=0;
                    val="";
                    i++;
                }
                else if(txt[i]==')'){
                    state=2;
                    val=")";
                    output(state, val);
                    state=0;
                    val="";
                    i++;
                }
                else if(txt[i]==';'){
                    state=3;
```



```
        val="\";
        output(state, val);
        state=0;
        val="\";
        i++;
    }
    else if(txt[i]=='='){
        state=4;
        val="\";
        output(state, val);
        state=0;
        val="\";
        i++;
    }
    else if(txt[i]=='+'){
        state=5;
        val="\";
        output(state, val);
        state=0;
        val="\";
        i++;
    }
    else if(txt[i]=='<'){
        state=6;
        val="\";
        output(state, val);
        state=0;
        val="\";
        i++;
    }
    else if(txt[i]=='{'){
        state=7;
        val="\";
        output(state, val);
        state=0;
        val="\";
        i++;
    }
    else if(txt[i]=='}'){
        state=8;
        val="\";
        output(state, val);
        state=0;
        val="\";
    }
```



```
        i++;
    }
    else if(isDigit(txt[i])){
        state=17;
        val+=txt[i++];
    }
    else if(isLetter(txt[i])){
        if(txt[i]=='i'){
            state=11;
            val+="i";
            i++;
        }else if(txt[i]=='f'){
            state=14;
            val+="f";
            i++;
        }else{
            state=9;
            val+=txt[i++];
        }
    }
    else if(txt[i]==' '){
        i++;
    }
    else{
        cout<<"wrong input"<<endl;
        i++;
    }
    break;
case 17:
    if(isDigit(txt[i]))
    {
        state=17;
        val+=txt[i++];
    }else{
        output(17, val);
        state=0;
        val="";
    }
    break;
case 9:
    if(isDigit(txt[i]) || isLetter(txt[i])){
        state=9;
        val+=txt[i++];
    }else{
```





```
        output(9, val);
        state=0;
        val="";
    }
    break;
case 11:
    if(txt[i]=='n'){
        state=12;
        val+='n';
        i++;
    }
    else{
        state=9;
    }
    break;
case 12:
    if(txt[i]=='t'){
        state=13;
        val+='t';
        i++;
    }
    else{
        state=9;
    }
    break;
case 13:
    if(!isDigit(txt[i]) && !isLetter(txt[i])){
        output(13, val);
        state=0;
        val="";
    }else{
        state=9;
    }
    break;
case 14:
    if(txt[i]=='o'){
        state=15;
        val+='o';
        i++;
    }
    else{
        state=9;
    }
    break;
```



```
case 15:
    if(txt[i]=='r'){
        state=16;
        val+='r';
        i++;
    }
    else{
        state=9;
    }
    break;
case 16:
    if(!isDigit(txt[i]) && !isLetter(txt[i])){
        output(16, val);
        state=0;
        val="";
    }else{
        state=9;
    }
    break;
}
}
```

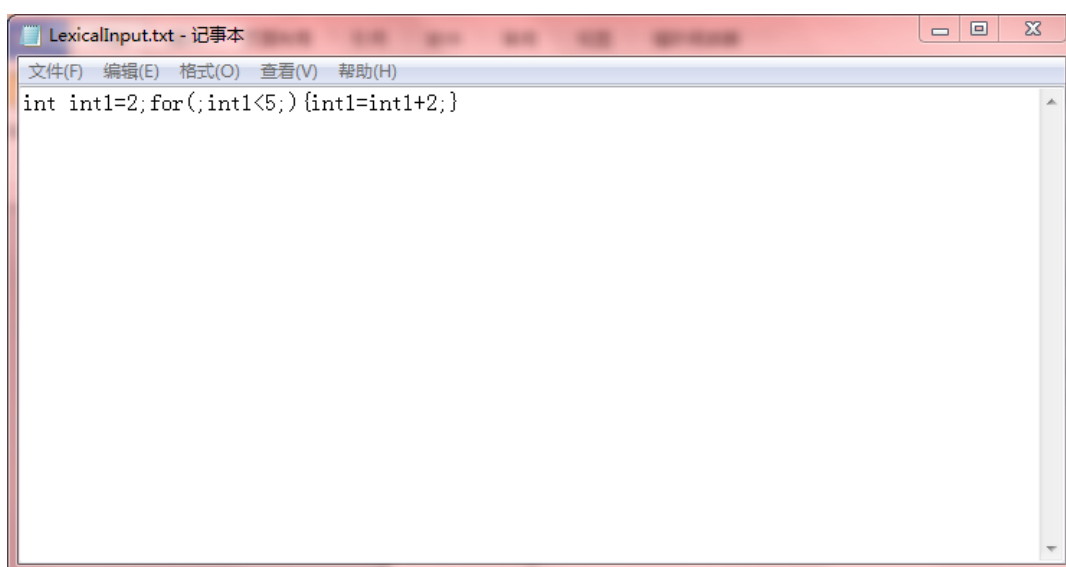


## 四、用户手册

1. **程序规格说明：**输入一段程序代码，扫描，识别出各种字符，输出相应的 token
2. **使用说明：**在项目文件夹下放了 LexicalInput.txt 文件放了一些程序代码和 SyntaxInput.txt 文件作为 token 序列的输出（同时作为语法分析器的输入）直接运行代码程序会对 LexicalInput.txt 文件进行扫描，把相应的程序代码转化为 token 保存在 SyntaxInput.txt 中。

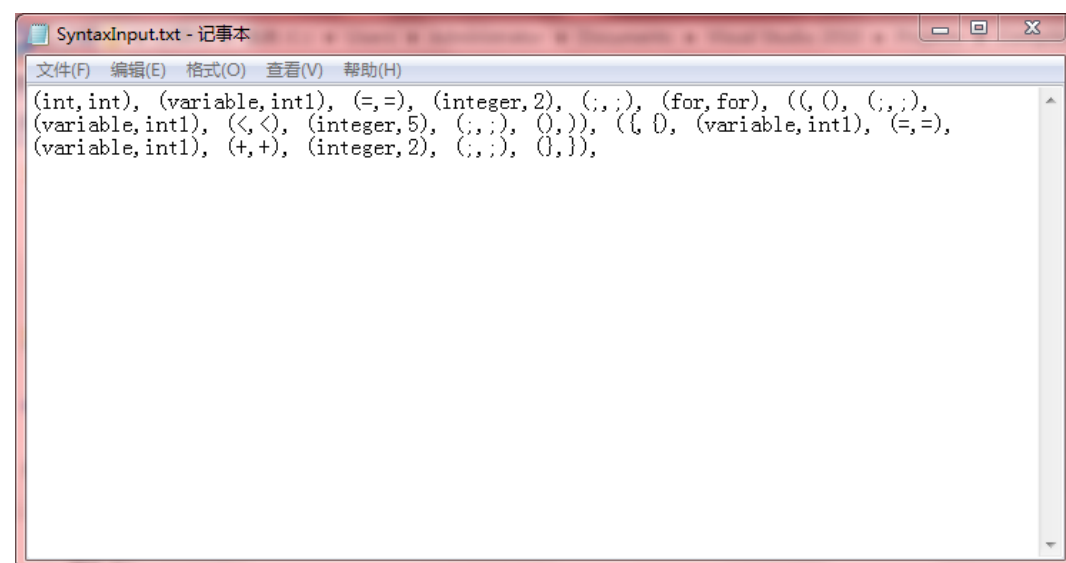
### 3. 例子：

输入：



```
int int1=2;for(;int1<5;){int1=int1+2;}
```

输出：



```
(int,int), (variable,int1), (=,=), (integer,2), (;,;), (for,for), ((,()), (,;),  
(variable,int1), (<, <), (integer,5), (;,;), (}), ({, (variable,int1), (=,=),  
(variable,int1), (+,+), (integer,2), (;,;), (},))
```



## 五、实验总结

### ■ 本次实验分析：

本次实验中，我预先写好了一小段程序代码，然后对程序代码中的符号进行识别输出为 token。只要对每个需要识别的符号分类设计出其正则表达式，再根据课程中学到的只是转化为 mDFA。代码实现的其实就是一个 mDFA，即每读取一个字符状态是怎么转移的，最终状态决定了字符 token 的类型，从而逐一输出。

### ■ 实验的未来展望：

- 1.设计的正则表达式还比较简单，并不够完整，期望以后能设计出针对某种编程语言较为完善完整的 RE
- 2.这次实现的是一个人工 lex，转化到 mDFA 的过程都是人工完成，然后代码仅仅实现了最后依据状态转移图扫描代码的过程，期望以后能够实现完整的 lex。

注：源程序在同文件夹中