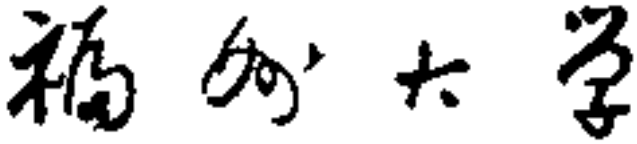


《编译系统设计实践》

实验项目一：词法分析实验

学号：041701320

姓名：杨鑫杰

年级：2017 级

学院：数计学院

专业：软件工程

本组其它成员：学号 221701114 姓名 张玉麟

学号 221701117 姓名 余嘉宸

学号 221701121 姓名 沈明炜

学号 221701131 姓名 郑志成

实验时间：2019－2020 学年第二学期

任课教师：陈晖

实验项目 1：词法分析程序实验

一、实验的目的与任务

词法分析的目的是将输入的源程序进行划分，给出基本符号（token）的序列，并掠过注解和空格等分隔符号。基本符号是与输入的语言定义的词法所规定的终结符。

本实验要求学生编制一个读单词过程，从输入的源程序中，识别出各个具有独立意义的单词，即基本保留字、标识符、常数、运算符、分隔符五大类。并依次输出各个单词的内部编码及单词符号自身值。（遇到错误时可显示“Error”，然后跳过错误部分继续进行）。

二、功能描述

读取源程序文件，扫描输入的输入流进行区分，识别出各个具有独立意义的单词，就是基本保留字、标识符、常数、运算符、分隔符五大类，给出基本符号（token）的序列并可以掠过注解和空格等分隔符号，之后依次输出各个单词的内部编码及单词符号自身值。（遇到错误时可显示“Error”，然后跳过错误部分继续进行）

三、程序结构描述

1. ReadFile 类

读取 yxj_test.txt 文件，该类主要实现文本的预处理、载入双缓冲区、分段保存、每次载入不超过 1K。

剔除无用的空白、跳格、回车和换行等编辑性的字符。

区分标号区、捻接续行和给出句末符等。

```
public static void readFile(char[] bf,int fwd)
```

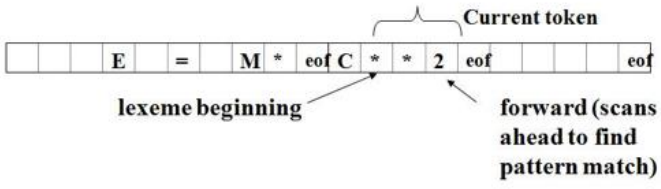
读取文件内容

```
public static void readFile(char[] bf,int fwd) {
    File file=new File(fileName);
    Reader reader=null;
    String str="";
    try {
        reader=new InputStreamReader(new FileInputStream(file));
        int ch;
        while((ch=reader.read())!=-1) {
            if((char)ch!='\r'&&(char)ch!='\n'&&(char)ch!=' '&&(char)ch!='\t') {//预处理换行、空格
                bf[fwd]=(char) ch;//将字符存入缓冲区
                fwd++;
                str+=(char)ch;
                System.out.print((char)ch);//测试
            }
            if(bf[fwd]=='~')//每次最多载入1K个字符
                break;
        }
        bf[fwd]='~';//EOF符号结尾
        reader.close();
    }catch(Exception e) {
        e.printStackTrace();
    }
    WriteFile.new_write(str);
}
```

2. Cache 类

该类为缓冲区的实现。

扫描缓冲区分两半，双缓冲结构，避免 id 的第一个字符在数组末尾，余下的还没进入数组。



1K EOF 1K

```
public static boolean cache(char[] bf,int forword)
```

定义缓冲区

```
public static boolean cache(char[] bf,int forword) {
    boolean flag=true;//判断是否到达了字符串末尾
    switch (bf[forword]) {
    case '~'://缓存区操作
        if(forword==1023) {
            forword++;
            ReadFile.readFile(bf, forword);
        }
        else if (forword==2047) {
            forword=0;
            ReadFile.readFile(bf, forword);
        }
        else{//字符串末尾的EOF
            flag=false;
            break;
        }
    default:
        break;
    }
    return flag;
}
```

3. Lexer 类

该类为词法分析的实现。

```
public void analyse(char[] bf)
```

识别当前字符

```
public void analyse(char[] bf) {
    ReadFile.readFile(bf, forward);//先载入一轮
    while(Cache.cache(bf, forward)){
        int fwd=forward;
        while(fwd==forward) {//处理同一个字符的循环
            switch (state) {
            case 0:
                state=relop(bf);
                break;
            case 9:
                state=id(bf);
                break;
            case 12:
                state=number(bf);
                break;
            case 20:
                state=number(bf);
                break;
            case 25:
                state=number(bf);
                break;
            case 28:
                state=delim(bf);
                break;
            case 31:
                //System.out.print("\nhhh");
                state=Op(bf);
                break;
            default:
                break;
            }
        }
        state=0;
    }
`
```

`public int relop(char[] bf)` 比较运算符 DFA

```java
public int relop(char[] bf) {
    int state=0;
    char c;
    while(toolFunction.isRelop(c=bf[forward])) {
        switch (c) {
        case '<':
            state=1;
            c=bf[++forward];
            switch (c) {
            case '=':
                state=2;
                word="<=";
                type="relop";
                break;
            case '>':
                state=3;
                word="<>";
                type="relop";
                break;
            default:
                state=4;
                retract(1);//回退
                word="<";
                type="relop";
                break;
            }
            break;
        case '=':
            state=5;
            word="=";
            type="赋值";
            break;
        case '>':
            state=6;
            c=bf[++forward];
            switch (c) {
            case '=':
                state=7;
                word=">=";
                type="relop";
                break;
            default:
                state=8;
                retract(1);//回退
                word=">";
                type="relop";
                break;
            }
            break;
        default:
            state=fail();
            break;
        }
        forward++;
        System.out.print("\n<"+word+","+type+">");
        WriteFile.write("\n<"+word+","+type+">");
    }
    if(state==0)
        state=fail();
    return state;
}
```

public int id(char[] bf) 标识符、关键字的 DFA

```
public int id(char[] bf) {
    int state=9;
    char c;
    String str="";
    if(toolFunction.isLetter(c=bf[forward])) {
        state=10;
        str+=c;
        forward++;
        type="id";
        while(toolFunction.isLetter(c=bf[forward])||toolFunction.isDigit(c=bf[forward])) {
            state=10;
            str+=c;
            forward++;
            for (String s : Key) {//判断类别是标识符还是关键字
                if(str.equals(s)){
                    type="key";
                    break;
                }
                //System.out.print("\nword:  "+word+"  s:  "+s+"  结果："+(word.equals(s)));
            }
            if (type=="key") {
                break;
            }
        }
        state=11;
        retract(1);//回退
        word=str;
        forward++;
        System.out.print("\n<"+word+","+type+">");
        WriteFile.write("\n<"+word+","+type+">");
    }
    else
        state=fail();
    return state;
}
```

public int number(char bf[]) 常数的 DFA

```java
public int number(char bf[])
{
    String str="";
    char c = bf[forward];
    if(toolFunction.isDigit(c))
    {
        state=13;
        while(true)
        {
            str+=c;
            c=bf[++forward];
            switch(state)
            {
            case 13:
                if(toolFunction.isDigit(c))
                {
                    state = 13;
                }
                else if(c=='E')
                {
                    state = 16;
                }
                else if(c=='.')
                {
                    state = 14;
                }
                else {
                    state = 27;
                }
                break;
            case 14:
                if(toolFunction.isDigit(c))
                {
                    state = 15;
                }
                break;
            case 15:
                if(toolFunction.isDigit(c))
                {
                    state = 15;
                }
                else if(c=='E')
                {
                    state = 16;
                }
                else {
                    state = 24;
                }
                break;
            case 16:
                if(c=='+'||c=='-')
                {
                    state = 17;
                }
                else if(toolFunction.isDigit(c))
                {
                    state = 18;
                }
                break;
            case 17:
                if(toolFunction.isDigit(c))
                {
                    state = 18;
                }
                break;
            case 18:
                if(toolFunction.isDigit(c))
                {
                    state = 18;
                }
                else if(!toolFunction.isDigit(c)){
                    state = 19;
                }
                break;
            }
            if(state == 19 || state == 24 || state == 27) {
                break;
            }
        }
        word=str;
        type="num";
        System.out.print("\n<"+word+","+type+">");
        WriteFile.write("\n<"+word+","+type+">");
    }
    else {
        state=fail();
    }
    return state;
}
```

```
public int delim(char bf[])
```
界符的 DFA

```
public int delim(char bf[]) {
    state=28;
    if(toolFunction.isDelim(bf[forward])) {
        state=29;
        String str="";
        str+=bf[forward];
        word=str;
        type="界符";
        forward++;
        System.out.print("\n<"+word+","+type+">");
        WriteFile.write("\n<"+word+","+type+">");
    }
    else {
        state=fail();
    }
    return state;
}
```

```
public int Op(char bf[])
```
算数运算符的 DFA

```
public int Op(char bf[]) {
    //System.out.print("\nhhh");
    if(toolFunction.isOp(bf[forward])) {
        String str="";
        str+=bf[forward];
        word=str;
        type="op";
        forward++;
        System.out.print("\n<"+word+","+type+">");
        WriteFile.write("\n<"+word+","+type+">");
    }
    else {
        state=fail();
    }
    return state;
}
```

```
public void retract(int n)
```
带*的终结状态回退，多扫描的拿掉

```
public void retract(int n) {
    forward-=n;
}
```

```
public int fail()
```
连接几个 DFA

```
public int fail() {
    int start=state;
    switch (start) {
    case 0:   start = 9;  break;
    case 9:   start = 12; break;
    case 12:  start = 20; break;
    case 20:  start = 25; break;
    case 25:  start = 28; break;
    case 28:  start = 31; break;
    case 31:
        System.out.print("\n compiler error!");
        WriteFile.write("\n compiler error!");/*recover();*/
        break;
    default:
        System.out.print("\n compiler error!!");
        WriteFile.write("\n compiler error!");
        break;
    }
    return start;
}
```

4. toolFunction 类

该类的功能是判断字符类型（是否为字母、数字、比较运算符、算术运算符、分隔符）

```
public static boolean isLetter(char c)
```
判断是否为字母

```
public static boolean isLetter(char c)
{
    if(c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z')
    {
        return true;
    }
    else return false;
}
```

```
public static boolean isDigit(char ch)
```
判断是否为数字

```
public static boolean isDigit(char ch)
{

    if(ch >= '0' && ch <= '9')
    {
        return true;
    }
    else return false;
}
```

```
public static boolean isRelop(char c)
```
判断是否是比较运算符

```
public static boolean isRelop(char c) {//判断是否是比较运算符
    if(c=='<'||c=='='||c=='>')
        return true;
    else return false;
}
```

```
public static boolean isDelim(char c)
```
判断是否为分隔符

```
public static boolean isDelim(char c) {//判断是否为分隔符
    if(c=='{' || c=='}' || c=='(' || c==')' || c==';' || c=='[' || c==']' || c=='.')
        return true;
    else {
        return false;
    }
}
```

```
public static boolean isOp(char c)
```
判断是否为算术运算符

```
public static boolean isOp(char c) {
    if(c=='+' || c=='-' || c=='*' || c=='/')
        return true;
    else {
        return false;
    }
}
```

5. WriteFile 类

输出文件不存在则创建，同时新内容覆盖原有文件内容，将结果输出到 yxj_output.txt 文件中

```
public static void write(String str)
```
文件不存在则创建并且写入文件

```
public static void write(String str) {
    try {
        File file=new File("yxj_output.txt");
        if(!file.exists()) {
            file.createNewFile();
        }
        FileWriter fileWriter=new FileWriter(file.getName(),true);//使用true，即进行append file
        fileWriter.write(str);
        fileWriter.close();
    }catch(IOException e) {
        e.printStackTrace();
    }
}
```

```
public static void new_write(String str)
```
覆盖原文件的内容

```
public static void new_write(String str) {
    try {
        File file=new File("yxj_output.txt");
        if(!file.exists()) {
            file.createNewFile();
        }
        FileWriter fileWriter=new FileWriter(file.getName());//覆盖原文件中的内容
        fileWriter.write(str);
        fileWriter.close();
    }catch(IOException e) {
        e.printStackTrace();
    }
}
```

四、符号表的设计和实现

符号表

自身值	类别	属性值
if	**关键字**	1
else	**关键字**	1

••••••	关键字	1
=	比较运算符	2
<	比较运算符	2
••••••	比较运算符	2
+	算术运算符	3
-	算术运算符	3
*	算术运算符	3
/	算术运算符	3
id	标识符	4
number	常数	5
(	分隔符	6
)	分隔符	6
{	分隔符	6
••••••	分隔符	6

五、测试用例

用例 1

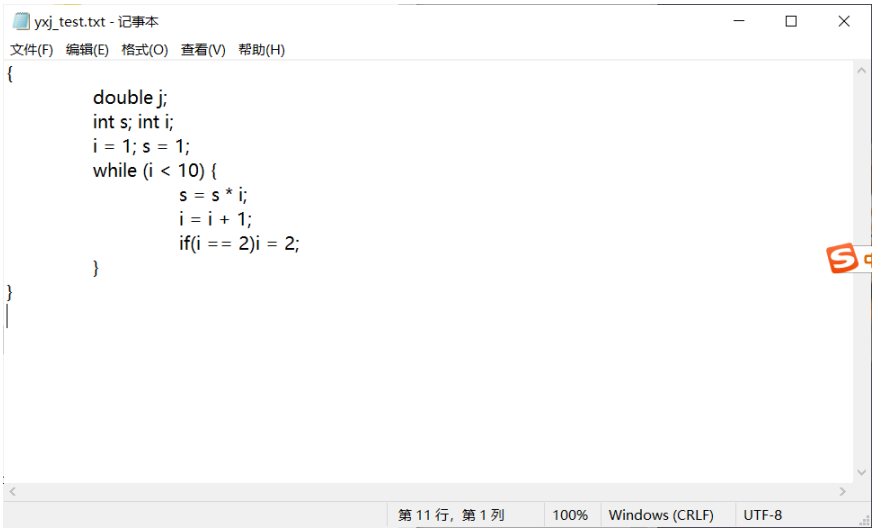
代码：

结果：

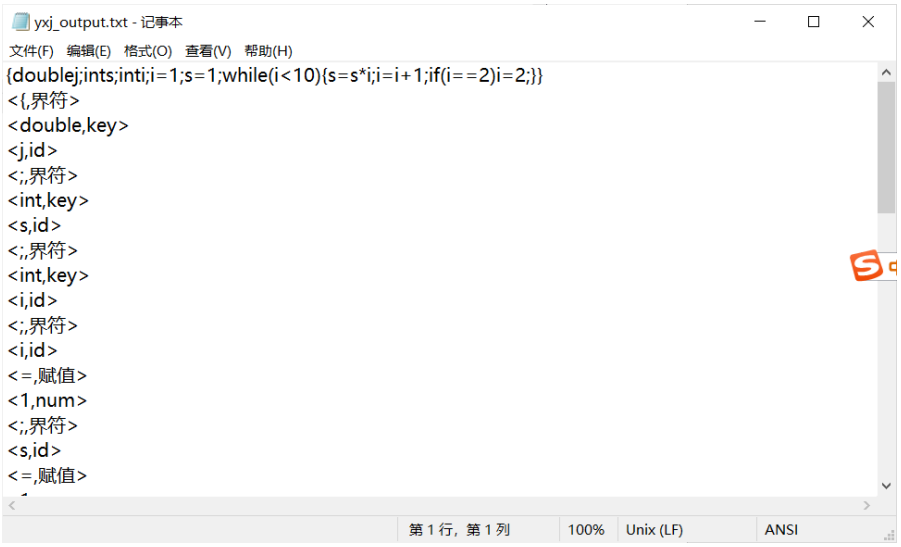

yxj_output.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
if(a>b){a=b+3.0;}
<if,key>
<(,界符>
<a,id>
<>,relop>
<b,id>
<),界符>
<{,界符>
<a,id>
<=,赋值>
<b,id>
<+,op>
<3.0,num>
<;,界符>
<},界符>
第 1 行，第 1 列
100%
Unix (LF)
ANSI

用例 2

代码：

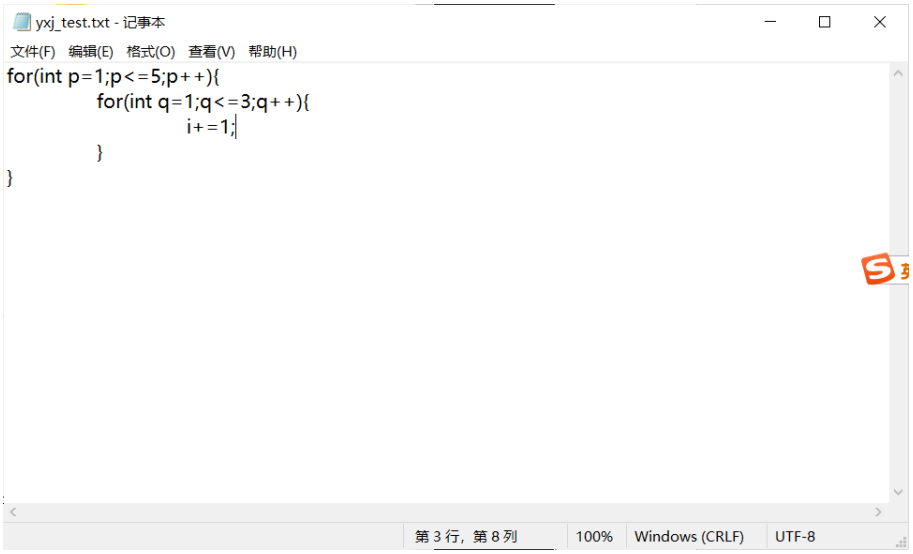

yxj_test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
{
double j;
int s; int i;
i = 1; s = 1;
while (i < 10) {
s = s * i;
i = i + 1;
if(i == 2)i = 2;
}
}
第 11 行，第 1 列
100%
Windows (CRLF)
UTF-8

结果：

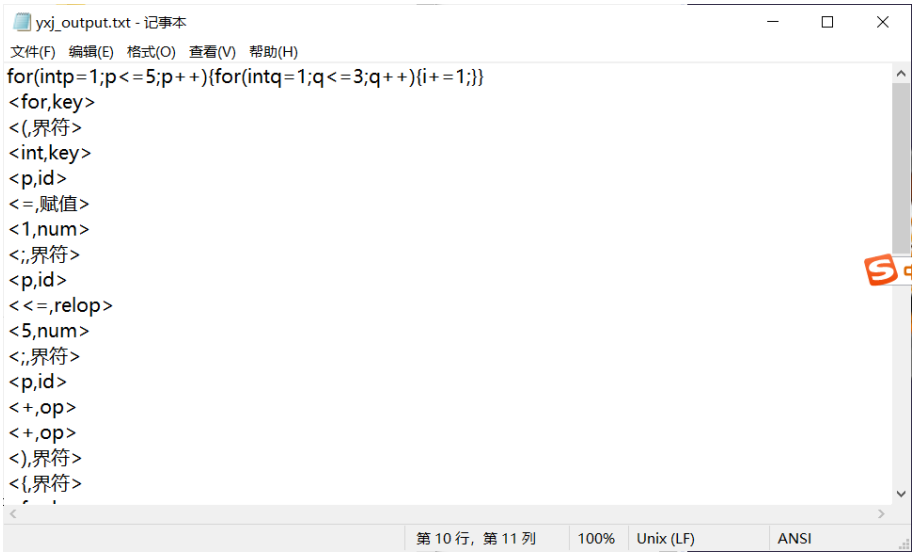

yxj_output.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
{doublej;intsinti;i=1;s=1;while(i<10){s=s*i;i=i+1;if(i==2)i=2;}}
<{,界符>
<double,key>
<j,id>
<;,界符>
<int,key>
<s,id>
<;,界符>
<int,key>
<i,id>
<;,界符>
<i,id>
<=,赋值>
<1,num>
<;,界符>
<s,id>
<=,赋值>
第 1 行，第 1 列
100%
Unix (LF)
ANSI

用例 3

代码：


yxj_test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
for(int p=1;p<=5;p++){
for(int q=1;q<=3;q++){
i+=1;
}
}
第 3 行，第 8 列
100%
Windows (CRLF)
UTF-8

结果：


yxj_output.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
for(intp=1;p<=5;p++){for(intq=1;q<=3;q++){i+=1;}}
<for,key>
<(,界符>
<int,key>
<p,id>
<=,赋值>
<1,num>
<;,界符>
<p,id>
<<=,relop>
<5,num>
<;,界符>
<p,id>
<+,op>
<+,op>
<),界符>
<{,界符>
第 10 行，第 11 列
100%
Unix (LF)
ANSI

用例 4

代码：


yxj_test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
int a=a+(b*c+b/a)-c;
第 1 行，第 1 列
100%
Windows (CRLF)
UTF-8

结果：

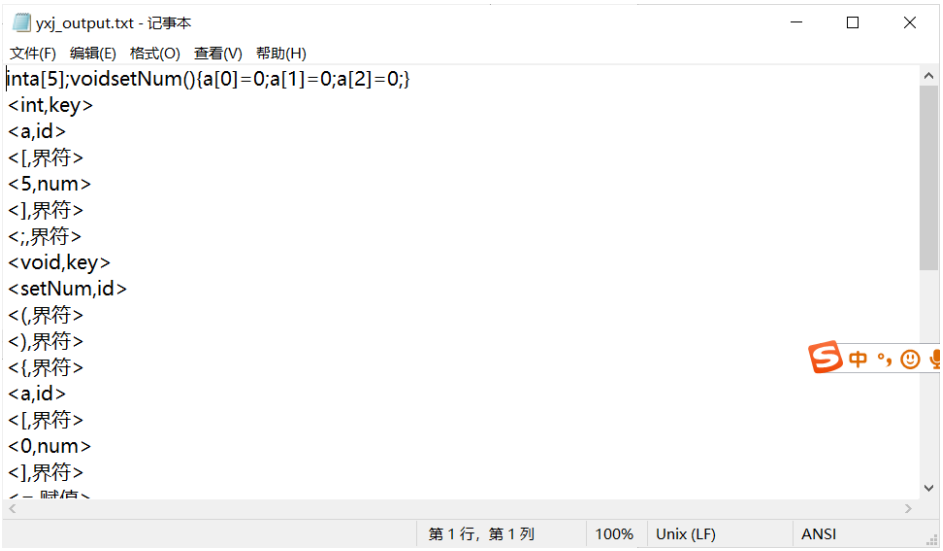

yxj_output.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
inta=a+(b*c+b/a)-c;
<int,key>
<a,id>
<=,赋值>
<a,id>
<+,op>
<(,界符>
<b,id>
<*,op>
<c,id>
<+,op>
<b,id>
</,op>
<a,id>
<),界符>
<-,op>
<c,id>
第 11 行，第 7 列
100%
Unix (LF)
ANSI

用例 5

代码：


yxj_test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
int a[5];
void setNum(){
a[0]=0;
a[1]=0;
a[2]=0;
}
第 1 行，第 1 列
100%
Windows (CRLF)
UTF-8

结果：



六、总结

经过词法分析这个实验，我们小组词法分析的过程有了更深的认识，这个实验的难点感觉就是你要对实验进行全面的分析和设计，不然就会有遗漏，就比如我们这次就漏了很多细节，比如连续赋值时的一个逗号的遗漏等等细节，就只有在测试的时候才知道，还有就是跳过 error 继续运行，没认真看就漏了。所以全面的分析和设计必不可少。再有就是我们团队的配合，我们是在 GitHub 上合作写的，每个人负责各自的部分，一些引用要及时讨论，不然就会出 bug，还有就是命名的问题，刚开始没有统一，后面整合的时候才统一的，有点浪费时间，这些问题希望在接下来的实验中解决，团队也尽快打的磨合。