**广州大学学生实验报告**

**开课学院及实验室：**计算机科学与工程实验室 **2020年 11月 11 日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | 计算机科学与网络工程学院 | **年级/专业/班** | **计科182** | **姓名** | 莫广贤 | **学号** | 1806100104 |
| **实验课程名称** | 编译原理实验课 | | | | | **成绩** |  |
| **实验项目名称** | 实验一 词法分析 | | | | | **指导老师** | 唐 卷 |

(\*\*\*报告只能为文字和图片,老师评语将添加到此处,学生请勿作答\*\*\*)

**一、实验目的**

设计实现一个词法分析程序，加深对词法分析原理的理解

**二、实验内容**

（1）设一小型编译程序关于高级语言有如下的规定：

高级语言程序具有四种基本结构：顺序结构﹑选择结构﹑循环结构和过程。为了便于掌握编译的核心内容，突出重点，简化编译程序的结构，同时又涵盖高级语言程序的基本结构，我们选取赋值语句﹑if语句和while语句作为前三种结构的代表，略去了过程结构。实际上，上述三种语句已经基本满足了高级语言的程序设计。因此，我们仅考虑由下面产生式所定义的程序语句：

S→if B then S else S ︱while B do S ︱begin L end︱A

L→S;L︱S

A→i:=E

B→B∧B︱B∨B︱¬ B︱(B) ︱i rop i︱i

E→E+E︱E\*E︱(E) ︱i

其中，各非终结符的含义如下：

S——语句；

L——语句串；

A——赋值句；

B——布尔表达式；

E——算术表达式。

各终结符的含义如下：

i ——整型变量或常数，布尔变量或常数；

rop ——六种关系运算符的代表；

; ——起语句分隔符作用；

:= ——赋值符号；

¬ ——逻辑非运算符“not”；

∧ ——逻辑与运算符“and”；

∨ ——逻辑或运算符“or”；

+ ——算术加运算符；

\* ——算术乘运算符；

( ——左括号；

) ——右括号。

注意，六种关系运算符分别为

<：小于 <=：小于等于 <>：不等于

>：大于 >=：大于等于 =：等于

关于表达式的运算，我们规定由高到低优先顺序为算术运算、关系运算、布尔运算；并且服丛左结合规则。算术运算符优先级的顺序依次为“( )”﹑“\*” ﹑“+” ；布尔运算符优先级的顺序依次为“¬ ”﹑“∧”﹑“∨”；六个关系运算符优先级相同。

我们规定的程序是由一条语句或由begin和end嵌套起来的复合语句组成的，并且规定在语句末要加上“#~”表示程序结束。下面给出的是符合规定的程序示例：

begin

A:=A+B\*C;

C:=A+2;

while A<C and B<D do

while A>B do

if M=N then C:=D

else while A<=D do

A:=D

end#~

（2）该小型编译程序关于单词的内部定义

由于我们规定的程序语句中涉及单词较少，故在词法分析阶段忽略了单词输入错误的检查，而将编译程序的重点放在中间代码生成阶段。词法分析器的功能是输入源程序，输出单词符号。我们规定输出的单词符号格式为如下的二元式：

**(单词种别，单词自身的值)**

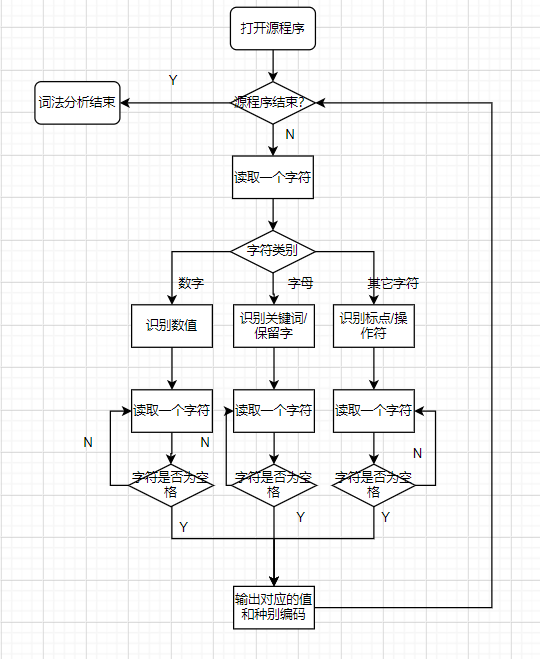
我们对常量，变量，临时变量，保留关键字(if、while、begin、end、else、then、do等)，关系运算符，逻辑运算符，分号，括号等，规定其内部定义如表1所示。

表1 关于单词的内部定义

**三、实验原理**

1.存储结构：使用类来保存信息，并用一维数组来存储保留字表。

2.流程图



**四、实验设备/环境**

1.Windows操作系统

2.使用javascript语言。本次实验程序在node环境下编写，需要先安装好node环境。在源程序文件目录下输入命令”node demo.js”即可运行

**五、实验要求**

请对下列程序进行词法分析，输出单词的种别编码和值：

begin

begin

begin

a:=1

end;

begin

b:=1

end;

q:=10

end;

x:=9;

y:=11;

z:=12

end

#~

**六、实验步骤与结果**

1.实验步骤

依照实验原理中的流程图编写程序，并对程序进行测试。

2.实验结果



**七、总结心得**

通过本次实验，对词法分析程序的开发有了一定的认识。本次实验只是简单的实现了一下识别给定的词，还有很多地方还是可以改善。同时本次实验也锻炼了我调试程序的能力。

**八、程序源代码**

const fs = require("fs");

const data = fs.readFileSync("a.txt");  //读取源码文件

class WD {                      //关键词类

    constructor(chara, value) {

        this.chara = chara;     //关键//关键词字符串

        this.value = value;     //关键词的编码

    }

}

class Reader {

    constructor(text) {

        this.text = text;           //保存要分析的文本

        this.pos = -1;              //初始读取的字符串位置为-1

        this.length = text.length;  //需要分析的文本的字符串长度

        this.flag = true;           //是否能继续分析的标志

        this.reserveList = [        //关键词表

            new WD('if', 0), new WD('then', 1), new WD('else', 2), new WD('while', 3), new WD('begin', 4),

            new WD('do', 5), new WD('end', 6), new WD('and', 39), new WD('or', 40), new WD('not', 41),

        ]

    }

    getChar() {                     //读取一个字符

        this.pos++;

        if (this.pos >= this.length) {

            // console.log('Reached end.');

            return;

        }

        return this.text[this.pos];

    }

    retract() {                     //回退一个字符

        this.pos--;

    }

    letter(c) {                     //判断是否为字母

        return /^[a-zA-Z]$/.test(c);

    }

    digit(c) {                      //判断是否为数字

        return /^[0-9]$/.test(c);

    }

    rop(c) {                        //判断是否为关系运算符

        if (c === '>' || c === '<' || c === '=')

            return true;

    }

    reserve(s) {                    //判断是否为关键字

        for (let i = 0; i < 7; i++) {

            if (s === this.reserveList[i].chara)

                return this.reserveList[i].value;

        }

        return 0;

    }

    judge1(c) {                     //c是字母的时候进行的判断

        let s = '';

        while (this.letter(c) || this.digit(c)) {

            s += c;

            c = this.getChar();

        }

        this.retract();

        if (this.reserve(s) === 0) return `(15,${s})`;

        else return `(${s},${this.reserve(s)})`;

    }

    judge2(c) {                     //c是数字的时候进行的判断

        let s = '';

        while (this.digit(c)) {

            s += c;

            c = this.getChar();

        }

        this.retract();

        return `(57,${s})`;

    }

    judge3(c) {                     //c是:的时候进行的判断

        let s = '';

        s += c;

        c = this.getChar();

        if (c === '=') {

            s += c;

            return `(38,${s})`;

        }

    }

    judge4(c) {                     //c是关系运算符进行的判断

        let s = '';

        while (c === '>' || c === '<' || c === '=') {

            s += c;

            c = this.getChar();

        }

        this.retract();

        const op\_rop = ['<', '>', '=', '<=', '>=', '<>'];

        if (op\_rop.some(item => item === s))

            return `(42,${s})`;

    }

    judge5() {                      //c是#进行的判断

        let t = this.getChar();

        if (t === '~') {

            this.flag = false;

            return 'analyze done.';

        }

    }

    analyze() {

        while (this.pos <= this.length && this.flag) {

            let c = this.getChar();

            if (this.letter(c)) console.log(this.judge1(c));

            else if (this.digit(c)) console.log(this.judge2(c));

            else if (c === ':') console.log(this.judge3(c));

            else if (this.rop(c)) console.log(this.judge4(c));

            else if (c === '#') console.log(this.judge5());

            else if (c === ';') console.log('(8,;)');

            else if (c === '+') console.log('(34,+)');

            else if (c === '\*') console.log('(36,\*)');

            else if (c === '(') console.log('(48,()');

            else if (c === ')') console.log(('(49,))'));

            else if (!/[:#;+\*()\s]/.test(c)) {          //若是非法字符，则直接退出

                console.log(`Error, '${c}' is illegal.`);

                break;

            }

        }

    }

}

const r = new Reader(data.toString());

try {

    r.analyze();

} catch (error) {

    console.log(error);

}