编译原理实验报告（一）--词法分析器

19335218 向鼎

摘要：手动实现一个简单的C语言词法分析器

1.导言

1.1 什么是词法分析器

“词法分析（lexical analysis）是计算机科学中将字符序列转换为单词（Token）序列的过程。进行词法分析的程序或者函数叫作词法分析器（Lexical analyzer，简称Lexer），也叫扫描器（Scanner）。词法分析器一般以函数的形式存在，供语法分析器调用。”

1.2 支持的单词类型

- 关键字：int, main, void, if, else, char, float, double, for, while, break, switch, case, default, return.
- 标识符：用户自己定义的变量id
- 数字常量：0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 以及小数
- 界符："<=", ">=", "==", "=", ">", "<", "+", "++", "+=", "-", "--", "-=", "*", *=, "/", "/=", "{", "}", "(", ")", "[", "]", ",", ";".
- 字符常量：以单引号括起来的字符, 如：'c '
- 字符串常量：以双引号括起来的字符串, 如："hello"

2. 实验过程

2.1 实验环境

编译语言：C++

编译环境：Clion

操作系统：Win10

2.2 实验步骤

- **自动机设计**：明确好需要分析的单词范围之后，着手设计自动机，画出如下的状态转换图，其中蓝色的状态表示为终止状态，红色的表示为检测到错误。

- **单词记录表**：一共设计了7张表格用于储存分析器的分析结果，其中结果表ans，以<类型，在表中的位置>的形式顺序储存分析出的每个词语。

```
    vector<string> TOKEN_k; //关键字表
    vector<string> TOKEN_p; //界符表
    vector<string> TOKEN_i; //标识符表
    vector<string> TOKEN_c; //数字常量表
    vector<string> TOKEN_charc; //字符常量表
    vector<string> TOKEN_strc; //字符串常量表
    vector<pair<string,int>> ans; //结果表，以<类型，在表中的位置>的形式储存
```

- **状态转移函数**：对于当前的输入字符，根据其当前状态判断下一状态。

```
    void next_state(char ch){ //自动机状态转移函数
        if(state == 0){
            if(is_char(ch)){state=1;str.push_back(ch);} // δ(0,字母)=1
            else if(is_digit(ch)){state=3;str.push_back(ch);} // δ(0,数字)=3
            else if(ch == '='){state=8;str.push_back(ch);} // δ(0,'=')=8
            else if(ch == '>'){state=11;str.push_back(ch);} // δ(0,'>')=11
            else if(ch == '<'){state=14;str.push_back(ch);} // δ(0,'<')=14
            else if(ch == '+'){state=17;str.push_back(ch);} // δ(0,'+')=17
            else if(ch == '-'){state=21;str.push_back(ch);} // δ(0,'-')=21
            else if(ch == '*'){state=25;str.push_back(ch);} // δ(0,'*')=25
            else if(ch == '/'){state=28;str.push_back(ch);} // δ(0,'/')=28
            else if(ch == '{'){state=31;str.push_back(ch);} // δ(0,'{')=31
            else if(ch == '}'){state=32;str.push_back(ch);} // δ(0,'}')=32
            else if(ch == '('){state=33;str.push_back(ch);} // δ(0,'(')=33
            else if(ch == ')'){state=34;str.push_back(ch);} // δ(0,')')=34
            else if(ch == '['){state=35;str.push_back(ch);} // δ(0,'[')=35
            else if(ch == ']'){state=36;str.push_back(ch);} // δ(0,']')=36
            else if(ch == ','){state=37;str.push_back(ch);} // δ(0,',')=37
            else if(ch == ';'){state=38;str.push_back(ch);} // δ(0,';')=38
            else if(ch == 39){state=39;str.push_back(ch);} // δ(0,''')=39
            else if(ch == 34){state=41;str.push_back(ch);} // δ(0,'"')=41
        }
        else if(state == 1){
            if(is_char(ch) || is_digit(ch)){state=1;str.push_back(ch);} // δ(1,字符或数
字)=1
            else{
                state = 2;
                ptr_i = ptr_i - 1; // δ(1,其他)=2，终止态
            }
        }
        else if(state == 3){
            if(is_digit(ch)){state=3;str.push_back(ch);} // δ(3,数字=3
            else if(ch == '.'){state=4;str.push_back(ch);} // δ(3,'.')=4
            else{
                state=7;
                ptr_i = ptr_i-1; // δ(3,其他)=7，终止态
            }
        }
```

```
        else if(state == 4){
            if(is_digit(ch)){state=5;str.push_back(ch);} // δ(4,数字)=5
            else{
                error = true; //其他情况，编译出错，终止态
            }
        }
        else if(state == 5){
            if(is_digit(ch)){state=5;str.push_back(ch);} // δ(5,数字)=5
            else{
                state=6;
                ptr_i = ptr_i - 1; // δ(5,其他)=6，终止态
            }
        }
        else if(state == 8){
            if(ch == '='){state=9;str.push_back(ch);} // δ(8,'=')=9
            else{
                state=10;
                ptr_i = ptr_i - 1; // δ(8,其他)=10，终止态
            }
        }
        else if(state == 11){
            if(ch == '='){state=12;str.push_back(ch);} // δ(11,'=')=12
            else{
                state=13;
                ptr_i = ptr_i - 1; // δ(11,其他)=13，终止态
            }
        }
        else if(state == 14){
            if(ch == '='){state =15;str.push_back(ch);} // δ(14,'=')=15
            else{
                state=16;
                ptr_i = ptr_i - 1; // δ(14,其他)=16，终止态
            }
        }
        else if(state ==17){
            if(ch == '+'){state=18;str.push_back(ch);} // δ(17,'+')=18
            else if(ch == '='){state=19;str.push_back(ch);} // δ(17,'=')=19
            else{
                state=20;
                ptr_i = ptr_i - 1; // δ(17,其他)=20，终止态
            }
        }
        else if(state == 21){
            if(ch == '-'){state=22;str.push_back(ch);} // δ(21,'-')=22
            else if(ch == '='){state=23;str.push_back(ch);} // δ(21,'=')=23
            else{
                state=24;
                ptr_i = ptr_i - 1; // δ(21,其他)=24，终止态
            }
```

```
        }
        else if(state == 25){
            if(ch == '='){state=26;str.push_back(ch);} // δ(25,'=')=26
            else{
                state=27;
                ptr_i = ptr_i - 1; // δ(25,其他)=27，终止态
            }
        }
        else if(state == 28){
            if(ch == '='){state=29;str.push_back(ch);} // δ(28,'=')=29
            else{
                state=30;
                ptr_i = ptr_i - 1; // δ(28,其他)=30，终止态
            }
        }
        else if(state == 39){
            if(ch == 39){state=40;str.push_back(ch);} // δ(39,''')=40，终止态
            else{
                state=39;
                str.push_back(ch); // δ(39,其他)=39
            }
        }
        else if(state == 41){
            if(ch == 34){state=42;str.push_back(ch);} // δ(41,'"')=42，终止态
            else{
                state=41;
                str.push_back(ch); // δ(41,其他)=41
            }
        }
    }
```

- **词法分析器主体**：按字符读入输入文件，每当遇到一个终止状态时，判断其类型，并将其加入分词表中。

```
    void analyze(){ //词法分析器主体部分
        while(ptr_i < input.length()){ //一个字符一个字符地读入输入文件
            next_state(input[ptr_i]);
            if(error){cout<<"编译出错"<<endl; break;}
            if(isFstate(state)){ //已到达终止状态
                Termination(state); //根据终止状态的类型判断该词的类型
            }
            ptr_i++;
        }
    }
```

3. 实验结果

3.1 实验文件

实验代码和结果文件已上传至github：https://github.com/xiangd3/LexAnalyzer

其中 *in.txt* 为输入C语言源程序，*out.txt* 为实验结果输出文件，LexAnalyzer.exe文件在cmake-build-debug目录下。

3.2 结果截图

输出格式为：(词语类型, 所在表中的位置) 词语。输出截图如下：

```
( KEY, 0)
int
( KEY, 1)
main
( P, 18)
(
( KEY, 2)
void
( P, 19)
)
( P, 16)
{
( KEY, 0)
int
( ID, 0)
a
```

4. 实验体会

这次实验是对于编译原理的初次尝试，在这次实验的过程中我学习掌握了对文本词语进行词性分析分类的技巧，也在实践代码的过程中加深了我对课上所学习的理论知识的理解和体会，巩固了对自动状态机的设计方法，这对于我以后的学习实验会有很大的帮助。

编译原理这门课乐趣无穷，希望我在以后的学习生活中能够不断进步，保持热情！

主要参考文献：

1.词法分析器设计与实现，https://blog.csdn.net/cike110120/article/details/26555257

2.词法分析器（C++实现）简单实现，https://www.cnblogs.com/gidear/p/14198340.html