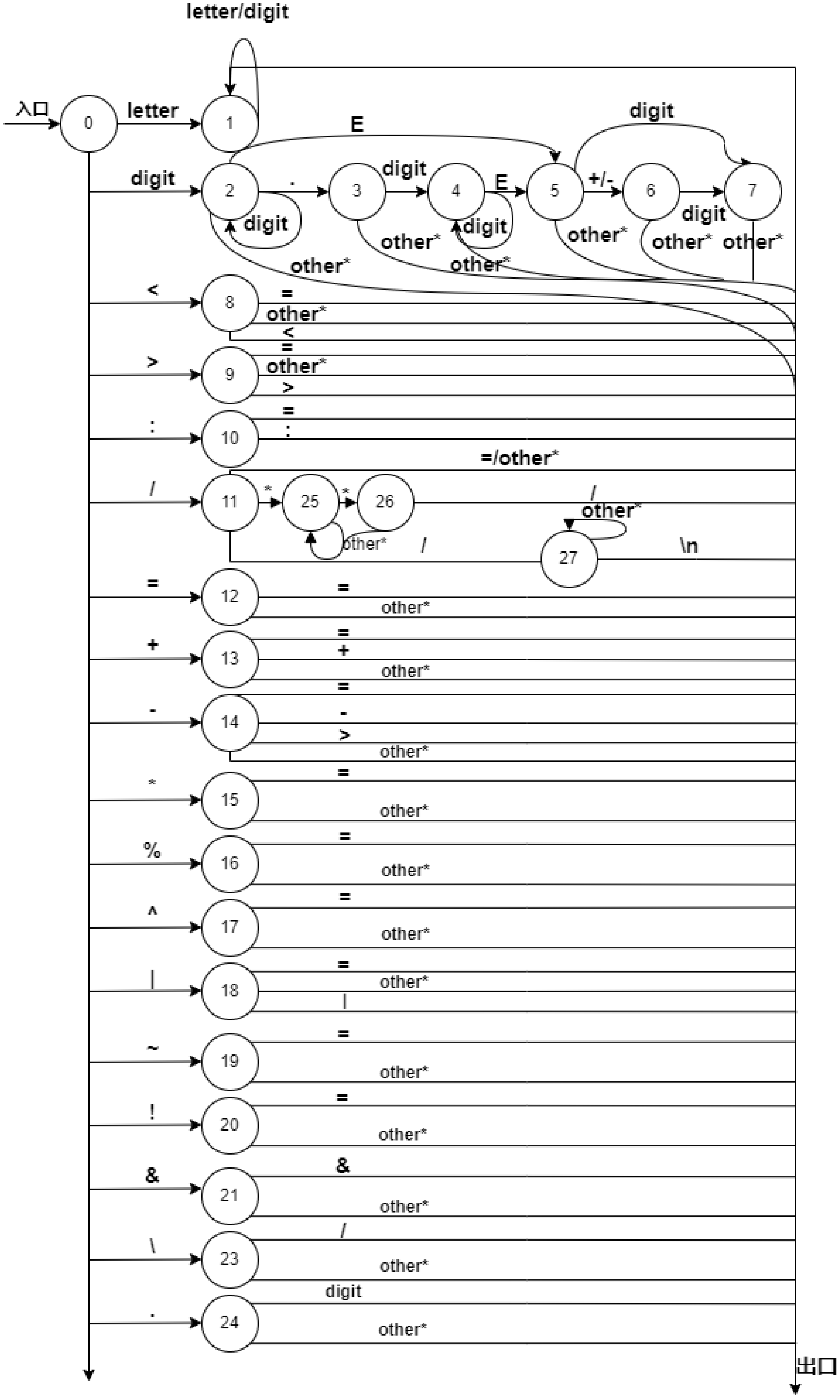
编译原理实验报告-词法分析程序

刘帅 2020212267

1 - c语言实现

1.1- 程序状态转换图


letter/digit
入口
0
letter
1
E
digit
digit
2
.
3
digit
4
E
5
+/-
6
7
digit
digit
digit
other*
digit
other*
other*
other*
other*
other*
other*
<
8
=
other*
<
>
9
=
other*
>
:
10
=
:
=/other*
/
11
*
25
*
26
/
other*
other*
/
27
\n
=
12
=
other*
+
13
=
+
other*
-
14
=
-
>
other*
*
15
=
other*
%
16
=
other*
^
17
=
other*
|
18
=
other*
|
~
19
=
other*
!
20
=
other*
&
21
&
other*
\
23
/
other*
.
24
digit
other*
出口

1.2 - 代码描述

1.2.1 调库&定义结构体&常量和全局变量

```
#include<iostream>
#include<stdio.h>
#include<vector>
#include<string>
#include<fstream>
using namespace std;
#define buffersize 2048
int state;//状态指示
char C;//存放当前读入的字符
string tempstr;//为当前读入内存的字符串构建缓冲区
char buffer[buffersize]; //将文件中读出的信息存入buffer
int forwardpoint = -1; //字符指针
int rows = 1;   //文件行数
int sum_char = 0; //文件总字数
struct token {
    string mark; //记号
    string name; //属性
    int count;  //出现次数
};

vector<string> keyword = {
"auto","break","case","char","const","continue","default","do","double","else","enum","extern","float","for","goto","if","inline","int","long","register","restrict","return","short","signed","sizeof","string","static","struct","switch","typedef","union","unsigned","void","volatile","while","_Alignas","_Alignof","_Atomic","_Bool","_Complex","_Generic","_Imaginary","_Noreturn","_Static_assert","_Thread_local" };
```

1.2.2 字符串操作相关操作

```
void get_char() {//从buffer中读入字符
    forwardpoint = (forwardpoint + 1) % buffersize;
    C = buffer[forwardpoint];
}

void cat() {//将字符C连接到nowstr字符串后面
    tempstr.push_back(C);
}

void retract() { //指针前移 等待回溯
    forwardpoint = (forwardpoint - 1) % buffersize;
}

void trace_back() {///回溯到上一个读入的字符
    retract();
    sum_char--;
    if (C == '\n') {
        rows--;
    }
}
```

1.2.3 判断类函数

```
bool is_Num(char c) {//判断是否读入的是数字
    if (c >= '0' && c <= '9')
        return true;
    return false;
}

bool is_letter(char c) {//判断读入的是否为字母
    if ((c >= 'a' && c <= 'z') || (c <= 'Z' && c >= 'A') || c == '_')
        return true;
    return false;
}

bool is_keyword() {//判断当前的tempstr是否在keyword列表中
    for (int i = 0; i < keyword.size(); i++) {
        if (tempstr == keyword[i]) {
            return true;
        }
    }
    return false;
}

bool iseven() {//判断“/”的个数，如果为偶数，说明是注释
    int num = 0;
    int i = tempstr.size() - 2;
    while (tempstr[i] == '\\') {
        num++;
        i--;
    }
    if (num % 2 == 0)
        return true;
    return false;
}
void error(int signal) {
    if (signal == 13)
        cout << "读入了无法识别的字符"<<endl;
    else if (signal == 3)
        cout << "小数点后没有数字"<<endl;
    else if (signal == 5) {
        cout << "指数后没有出现+-或数字"<<endl;
    }
    else if (signal == 6) {
        cout << "+-后没有出现数字"<<endl;
    }
}
```

1.2.4 将识别到的token加入到最终结果result中

```
void addToken(string a, string b, vector<token>& tempresult) {//将识别到的token放到result
    int i;
    for (i = 0; i < tempresult.size(); i++)
        if (tempresult[i].mark == a && tempresult[i].name == b) {
            tempresult[i].count++;
            break;
        }
    if (i == tempresult.size()) {
        token t;
        t.mark = a;
        t.name = b;
        t.count = 1;
        tempresult.push_back(t);
    }
}
```

1.2.5 词法分析函数

根据状态转换图的逻辑，对程序状态进行转移跳转处理

```
vector<token> analysis(ifstream& f) {
    vector<token> result;
    bool flag = false;
    f.read(buffer, buffersize - 1);
    if (f.gcount() < buffersize - 1) {
        buffer[f.gcount()] = EOF;
    }
    buffer[buffersize - 1] = EOF;
    state = 0;
    while (!flag) {
        get_char();
        if (C == '\n')
            rows++;
        if (C != EOF) {
            sum_char++;
        }
        if (C == EOF && forwardpoint != buffersize - 1)//如果buffer的空间大于txt的长度，则将所有内容能够成功读入内存
            flag = true;
        else if (C == EOF && forwardpoint == buffersize - 1) {//如果buffer空间不足，则将目前读入的最后一个字符当作EOF，防止溢出。
            f.read(buffer, buffersize - 1);
            if (f.gcount() < buffersize - 1) {
                buffer[f.gcount()] = EOF;
            }
            continue;
        }
        //将文件内容读入buffer

        switch (state) {
        case 0:
            if (is_letter(C)) {
                state = 1;
                cat();
            }
            else if (is_Num(C) && C != '0') {
                state = 2;
                cat();
            }
            else {
                switch (C) {
                case '<':state = 8; break;
                case '>':state = 9; break;
                case':':state = 10; break;
                case'?':addToken("分界符", "?", result); break;
                case'(':addToken("分界符", "(", result); break;
                case')':addToken("分界符", ")", result); break;
                case ',':addToken("分界符", ",", result); break;
                case ';': addToken("分界符", ";", result); break;
                case '{': addToken("分界符", "{", result); break;
                case '}': addToken("分界符", "}", result); break;
                case '[': addToken("分界符", "[", result); break;
                case ']': addToken("分界符", "]", result); break;
                case'/':state = 11; break;
                case '=': state = 12; break;
                case '+': state = 13; break;
                case '-': state = 14; break;
                case '*': state = 15; break;
                case '%': state = 16; break;//加减乘除运算
                case '^':state = 17; break;
                case '|': state = 18; break;
                case '~': state = 19; break;
                case '!': state = 20; break;
                case '&': state = 21; break;
                case '\'':state = 23; cat(); break;
                case '.': state = 24; break;
                case EOF:break;
                }
            }
            break;
        case 1:
            if (is_letter(C) || is_Num(C)) {//如果识别到的是数字或字母，则在状态循环
                cat();
                state = 1;
            }
            else {
                trace_back();
                state = 0;
                if (is_keyword()) {//识别到keyword
                    addToken("keyword", tempstr, result);
                }
                else//识别到非数字&非字母
                    addToken("id", tempstr, result);
                tempstr.clear();
            }
            break;
        case 2:
            if (is_Num(C)) {
                state = 2;
                cat();
            }
            else {//科学计数法或小数的表示
                switch (C) {
                case '.':cat(); state = 3; break;
                case 'E':cat(); state = 5; break;
                default://整数
                    trace_back();
                    state = 0;
                    addToken("整数", tempstr, result);
                    tempstr.clear();
                    break;
                }
            }
            break;
        case 3:
            if (is_Num(C)) {
                cat();
                state = 4;
            }
            else {//浮点数
                trace_back();
                tempstr.push_back('0');
                addToken("浮点数", tempstr, result);
                state = 0;
                tempstr.clear();
            }
            break;
        case 4:
            if (is_Num(C)) {
                cat();
                state = 4;
            }
            else if (C == 'E') {
                state = 5;
                cat();
            }
            else {
                trace_back();
                state = 0;
                addToken("浮点数", tempstr, result);
                tempstr.clear();
            }
            break;
        case 5:
            if (is_Num(C)) {
                state = 7;
                cat();
            }
            else if (C == '+' || C == '-') {
                cat();
                state = 6;
            }
            else {
                trace_back();
                cout << "第" << rows << "行出现错误"<<"       错误类型：";
                error(5);
                state = 0;
                tempstr.clear();
            }
            break;
        case 6:
            if (is_Num(C)) {
                cat();
                state = 7;
            }
            else {
                trace_back();
                cout << "第" << rows << "行出现错误" << "       错误类型：";

                error(6);
                state = 0;
                tempstr.clear();
            }
            break;
        case 7:
            if (is_Num(C)) {
                cat();
                state = 7;
            }
            else {
                trace_back();
                state = 0;
                addToken("指数", tempstr, result);
                tempstr.clear();
            }
            break;
        case 8:
            if (C == '=') {
                addToken("关系符", "<=", result);
                state = 0;
            }
            else if (C == '<') {
                addToken("位操作符", "<<", result);
                state = 0;
            }
            else {
                addToken("关系符", "<", result);
                trace_back();
                state = 0;
            }
            break;
        case 9:
            if (C == '='){
                addToken("关系符",">=",result);
                state = 0;
            }
            else if (C == '>') {
                addToken("位操作符", ">>", result);
                state = 0;
            }
            else {
                addToken("关系符", ">", result);
                trace_back();
                state = 0;
            }
            break;
        case 10:
            if (C == '=') {
                addToken("关系符", ":=", result);
                state = 0;
            }
            else {
                addToken("分界符", ":", result); break;
                trace_back();
                state = 0;
            }
        case 11:
            switch (C) {
            case'/':
                state = 27;
                break;
            case'*':
                state = 25;
                break;
            case'=':
                addToken("赋值运算符", "/=", result);
                state = 0;
                break;
            default:
                addToken("算数运算符", "/", result);
                trace_back();
                state = 0;
                break;
            }
            break;
        case 25:
            if (C == '*')
                state = 26;
            else {
                state = 25;
                cat();

            }
            break;
        case 26:
            if (C == '/') {
                trace_back();
                state = 0;
                addToken("字符串", tempstr, result);
                tempstr.clear();
            }
            else {
                state = 25;
                cat();
            }
            break;
        case 27:
            if (C == '\n')
                state = 0;
            else
```

```
                state = 27;
            break;
        case 12:
            if (c == '=') {
                addToken("关系符", "==", result);
                state = 0;
            }
            else {
                addToken("赋值运算符", "=", result);
                state = 0;
                trace_back();
            }
            break;
        case 13:
            if (c == '=') {
                addToken("赋值运算符", "+=", result);
                state = 0;
            }
            else if (c == '+') {
                addToken("算数运算符", "++", result);
            }
            else {
                addToken("算数运算符", "+", result);
                state = 0;
                trace_back();
            }
            break;
        case 14:
            if (c == '=') {
                addToken("赋值运算符", "-=", result);
                state = 0;
            }
            else if (c == '-') {
                addToken("算术运算符", "--", result);
                state = 0;
            }
            else if (c == '>') {
                addToken("特殊操作符", "->", result);
            }
            else {
                addToken("算术运算符", "-", result);
                state = 0;
                trace_back();
            }
            break;
        case 15:
            if (c == '=') {
                addToken("赋值运算符", "*=", result);
                state = 0;
            }
            else {
                addToken("算术运算符", "*", result);
                state = 0;
                trace_back();
            }
            break;
        case 16:
            if (c == '=') {
                addToken("赋值运算符", "%=", result);
                state = 0;
            }
            else {
                addToken("算术运算符", "%", result);
                state = 0;
                trace_back();
            }
            break;
        case 17:
            if (c == '=') {
                addToken("赋值运算符", "^=", result);
                state = 0;
            }
            else {
                addToken("位运算符", "^", result);
                state = 0;
                trace_back();
            }
            break;
        case 18:
            if (c == '=') {
                addToken("赋值运算符", "|=", result);
                state = 0;
            }
            else if (c == '|') {
                addToken("逻辑运算符", "||", result);
                state = 0;
            }
            else {
                addToken("位运算符","|", result);
                state = 0;
                trace_back();
            }
            break;
        case 19:
            if (c == '=') {
                addToken("赋值运算符", "~=", result);
                state = 0;
            }
            else {
                addToken("位运算符", "~", result);
                state = 0;
                trace_back();
            }
            break;
        case 20:
            if (c == '=') {
                addToken("关系符", "!=", result);
                state = 0;
            }
            else {
                addToken("逻辑运算符", "!", result);
                state = 0;
                trace_back();
            }
            break;
        case 21:
            if (c == '&') {
                addToken("逻辑运算符", "&&", result);
                state = 0;
            }
            else {
                addToken("特殊操作符", "&", result);
                state=0;
                trace_back();
            }
            break;
        case 22://由于读入时中文会显示乱码 将此状态隐去
            if (c == '"') {
                cat();
                if (iseven()) {
                    addToken("字符串", tempstr, result);
                    tempstr.clear();
                    state = 0;
                }
                else state = 22;
            }
            else {
                cat();
                state = 22;
            }
            break;
        case 23:
            if (c == '\'') {
                cat();
                if (iseven()) {
                    addToken("字符", tempstr, result);
                    tempstr.clear();
                    state = 0;
                }
                else state = 23;
            }
            else {
                cat();
                state = 23;
            }
            break;
        case 24:
            if (isdigit(c)) {
                tempstr.push_back('0');
                tempstr.push_back('.');
                cat();
                state = 4;
            }
            else {
                addToken("特殊操作符", ".", result);
                trace_back();
                state = 0;
            }
            break;
        default:
            break;
        }
    }
    return result;
}
```

1.2.6 输出结果

```
void output(vector<token> result) {
    int count_keyword = 0;
    int count_id = 0;
    int count_int = 0;
    int count_float = 0;
    int count_exponent = 0;
    int count_relationalOperator = 0;
    int count_logicOperator = 0;
    int count_bitOperator = 0;
    int count_assignOperator = 0;
    int count_specialOperator = 0;
    int count_arithmeticOperator = 0;
    int count_string = 0;
    int count_char = 0;
    int count_delimeter = 0;
    cout << "记号                " << "属性" << "                          出现次
数" << endl;
    for (int i = 0; i < result.size(); i++) {
        int j = 30 - result[i].name.size();
        for (int k = 0; k < j; k++) {
            result[i].name.push_back(' ');
        }
    }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "keyword") {
            cout << result[i].mark <<"   "<<result[i].name << result[i].count <<
endl;
            count_keyword++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "id") {
            cout << result[i].mark <<"   "<<result[i].name << result[i].count <<
endl;
            count_id++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "整数") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_int++;
        }
    for (int i = 0; i < result.size(); i++)
```

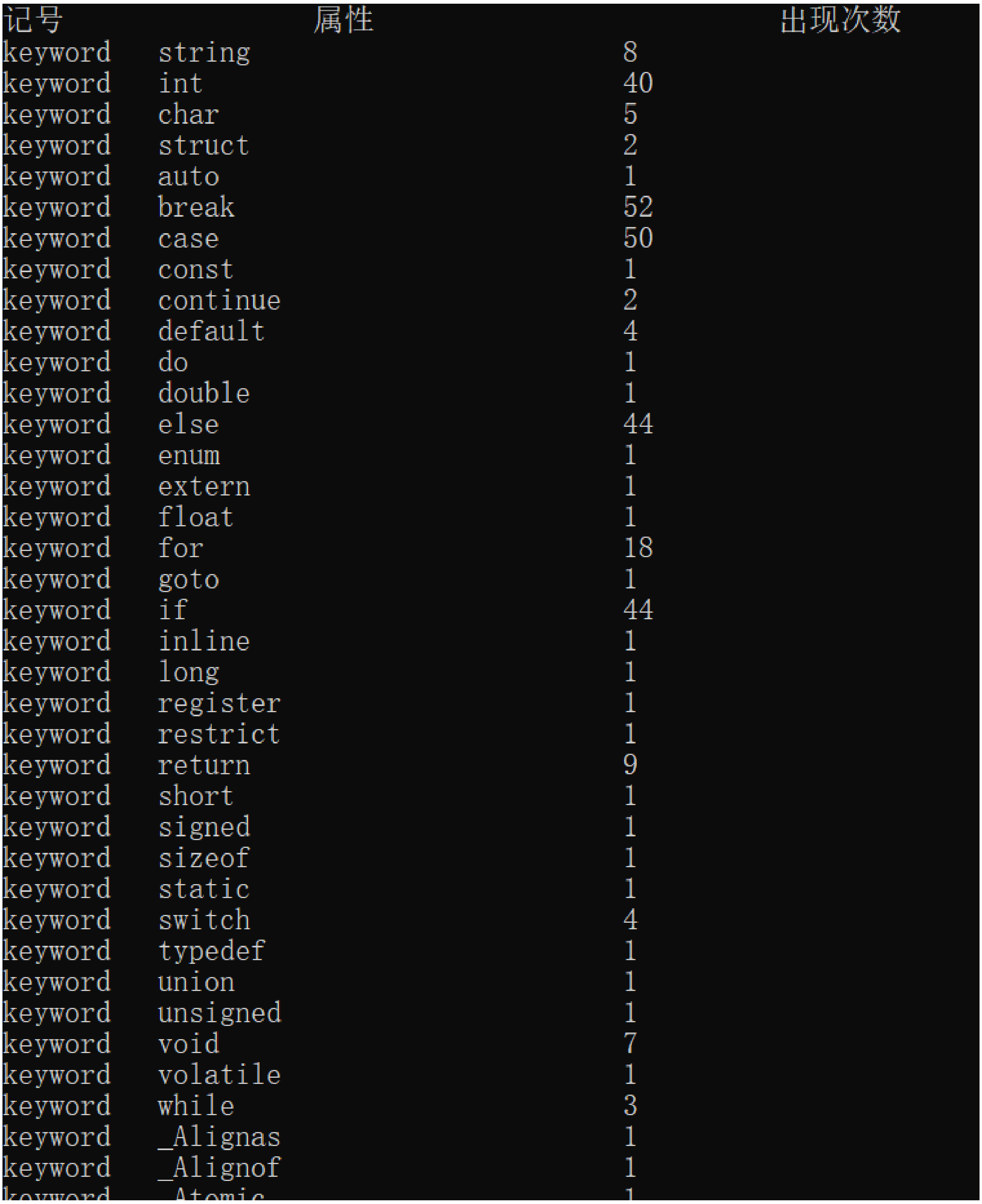
```
        if (result[i].mark == "浮点数") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_float++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "指数") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_exponent++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "关系符") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_relationalOperator++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "逻辑运算符") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_logicOperator++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "位操作符") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_bitOperator++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "赋值运算符") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_assignOperator++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "特殊操作符") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_specialOperator++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "算术运算符") {
            cout << result[i].mark << "   " << result[i].name << result[i].count
<< endl;
            count_arithmeticOperator++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "字符串") {
            cout << result[i].mark<<"   " << result[i].name << result[i].count
<< endl;
            count_string++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "字符") {
            cout << result[i].mark << result[i].name << result[i].count << endl;
            count_char++;
        }
    for (int i = 0; i < result.size(); i++)
        if (result[i].mark == "分界符") {
            cout << result[i].mark << result[i].name << result[i].count << endl;
            count_delimeter++;
        }
    if (count_keyword > 0)
        cout << "keywords字数:  " << count_keyword << endl;
    if (count_id > 0)
        cout << "id字数:" << count_id << endl;
    if (count_int > 0)
        cout << "整数个数:" << count_int << endl;
    if (count_float > 0)
        cout << "浮点数个数:" << count_float << endl;
    if (count_exponent > 0)
        cout << "指数个数:" << count_exponent << endl;
    if (count_relationalOperator > 0)
        cout << "关系符个数:" << count_relationalOperator << endl;
    if (count_logicOperator > 0)
        cout << "逻辑运算符个数:" << count_logicOperator << endl;
    if (count_bitOperator > 0)
        cout << "位操作符个数:" << count_bitOperator << endl;
    if (count_assignOperator > 0)
        cout << "赋值运算符个数:" << count_assignOperator << endl;
    if (count_specialOperator > 0)
        cout << "特殊操作符个数:" << count_specialOperator << endl;
    if (count_arithmeticOperator > 0)
        cout << "算术运算符个数:" << count_arithmeticOperator << endl;
    if (count_string > 0)
        cout << "字符串个数:" << count_string << endl;
    if (count_char > 0)
        cout << "字符个数:" << count_char << endl;
    if (count_delimeter > 0)
        cout << "分界符个数:" << count_delimeter << endl;

    cout << "行数:" << rows << endl;
    cout << "字数:" << sum_char << endl;
}
//输出结果
```

1.2.7 主函数

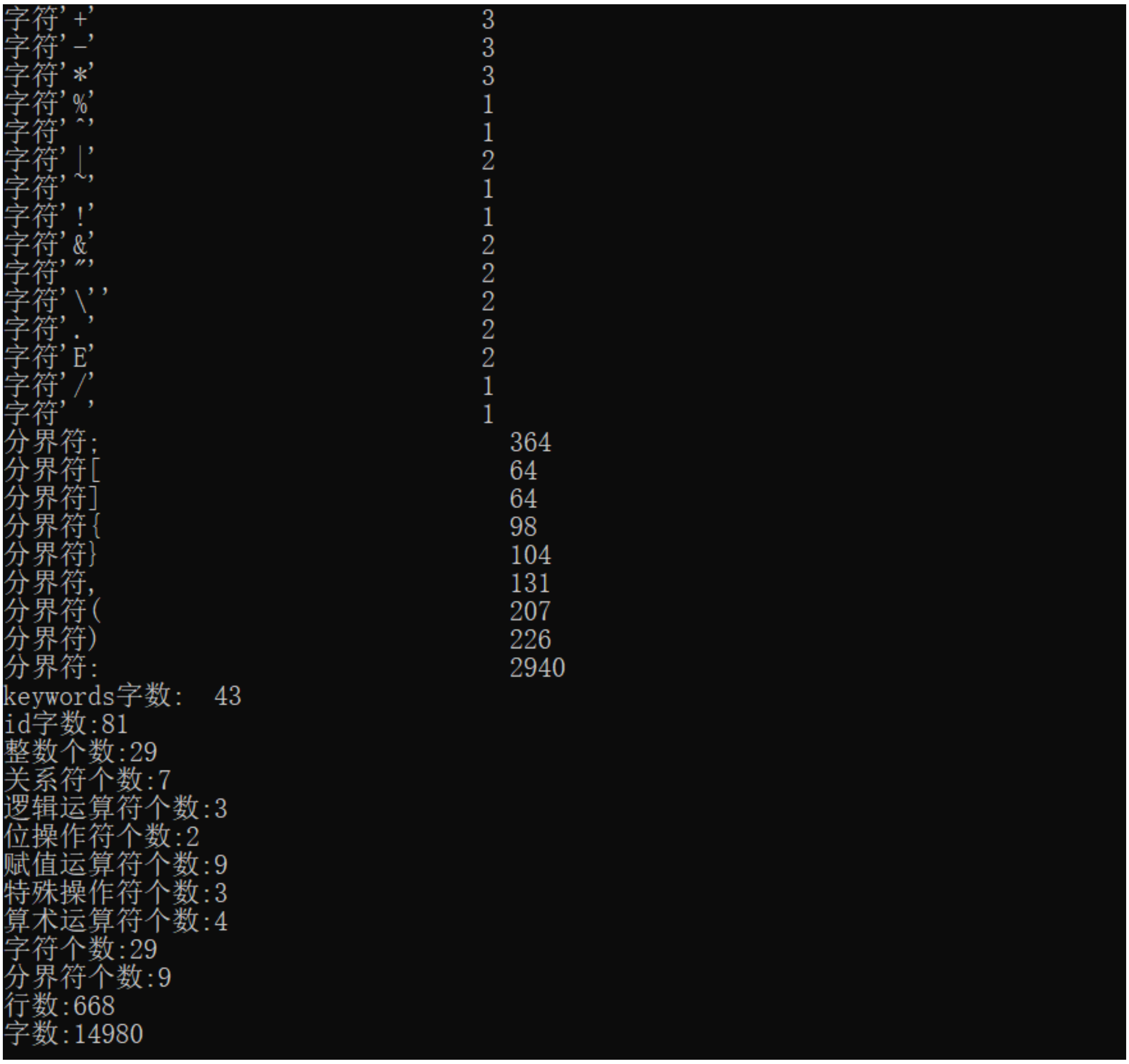
```
int main(void) {
    ifstream fs;
    fs.open("test.txt", ios::in);
    if (fs.is_open() == false)
        exit(0);
    vector<token> result = analysis(fs);
    fs.close();
    output(result);
}
```

1.3 输出结果

我们选用的测试样例即词法分析程序的代码，由于程序体量较大，故展示部分截图。尽管我们设置了注释内容，然而在txt向c++内存中读入中文时会出现编码错误，因此取消了识别注释的状态。



```
记号               属性                              出现次数
keyword   string                  8
keyword   int                     40
keyword   char                    5
keyword   struct                  2
keyword   auto                    1
keyword   break                   52
keyword   case                    50
keyword   const                   1
keyword   continue                2
keyword   default                 4
keyword   do                      1
keyword   double                  1
keyword   else                    44
keyword   enum                    1
keyword   extern                  1
keyword   float                   1
keyword   for                     18
keyword   goto                    1
keyword   if                      44
keyword   inline                  1
keyword   long                    1
keyword   register                1
keyword   restrict                1
keyword   return                  9
keyword   short                   1
keyword   signed                  1
keyword   sizeof                  1
keyword   static                  1
keyword   switch                  4
keyword   typedef                 1
keyword   union                   1
keyword   unsigned                1
keyword   void                    7
keyword   volatile                1
keyword   while                   3
keyword   _Alignas                1
keyword   _Alignof                1
keyword   _Atomic                 1
```



```
字符’+’                      3
字符’-’                      3
字符’*’                      3
字符’%’                      1
字符’^’                      1
字符’|’                      2
字符’~’                      1
字符’!’                      1
字符’&’                      2
字符’”’                      2
字符’\’’                     2
字符’.’                      2
字符’E’                      2
字符’/’                      1
字符’ ’                      1
分界符;                        364
分界符[                        64
分界符]                        64
分界符{                        98
分界符}                        104
分界符,                        131
分界符(                        207
分界符)                        226
分界符:                        2940
keywords字数:  43
id字数:81
整数个数:29
关系符个数:7
逻辑运算符个数:3
位操作符个数:2
赋值运算符个数:9
特殊操作符个数:3
算术运算符个数:4
字符个数:29
分界符个数:9
行数:668
字数:14980
```

2 - lex实现

2.1 lex的第一部分

第一小部分以符号%{和%}包裹，里面为以C语法写的一些定义和声明：例如，文件包含，宏定义，常数定义，全局变量及外部变量定义，函数声明等。这一部分被Lex翻译器处理后会全部拷贝到文件lex.yy.c中

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    int count = 0;
%}
```

2.2 设定词法规则

定义whitespace针对换行符、转义字符、回车进行识别

opterator识别+ − ∗/ ><等关系字符

reservedword 保留字 即系统内置的keyword

delimiter 识别定界符，如()//等

constant识别数字（0-9的正闭包）

identifier识别用户定义关键字

```
delim [" "\n\t\r]
whitespace {delim}+
operator \+|-|\*|\/|:=|>=|<=|#|=|<<|>>|\+\+|\<|\>|\{|\}
reservedWord int|include|main|return|using|if|namespace|cout|cin|std|iostream
delimiter [,\.;\(\)\"]
constant ([0-9])+
identfier [A-Za-z]([A-Za-z][0-9])*
```

2.3 正规定义和状态定义

```
%%
{reservedWord} {count++;printf("%d\t(keyword,%s)\n",count,yytext);}
\"[^\"]*\" {count++;printf("%d\t(count,%s)\n",count,yytext);}
{operator} { count++;printf("%d\t(operator,%s)\n",count,yytext); }
{delimiter} {count++;printf("%d\t(delimiter,%s)\n",count,yytext);}
{constant} {count++;printf("%d\t(constant,%s)\n",count,yytext);}
{identfier} {count++;printf("%d\t(id,%s)\n",count,yytext);}
{whitespace} { /* do    nothing*/ }
%%
```

2.4 执行函数

yylex对文本进行扫描

若yywrap()返回0，则继续扫描
若返回1，则返回报告文件结尾的0标记。
由于词法分析器总会调用yywrap，因此辅助函数中最好提供yywrap，
如果不提供，则在用C编译器编译lex.yy.c时，需要链接相应的库，库中会给出标准的yywrap函数（标准函数返回1）

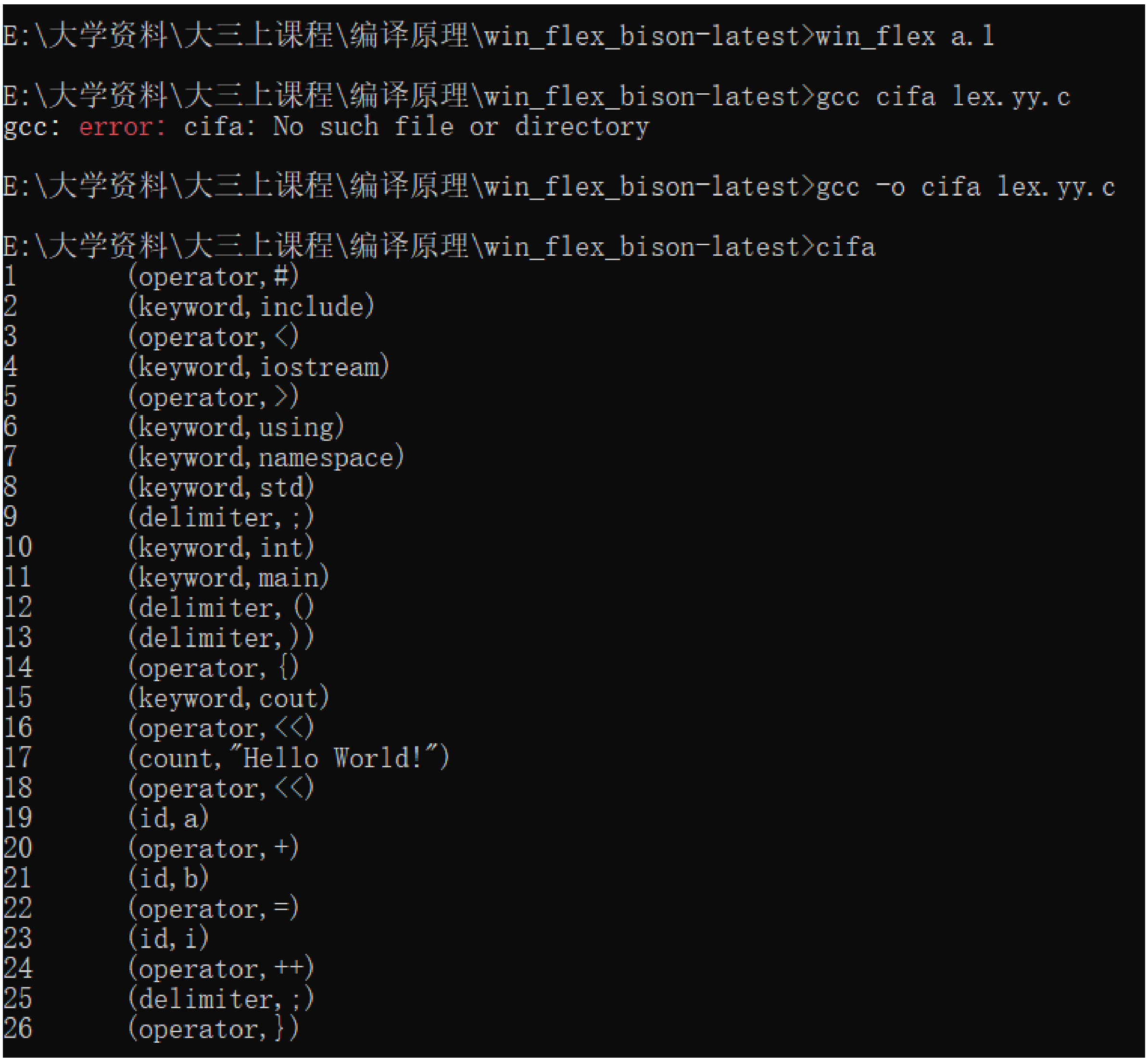
```
int main()
{
    yyin = fopen("input.txt","r");
        yylex();
    fclose(yyin);
}
int yywrap()
{
    return 1;
}
```

2.5 执行样例与结果

以最简单的c++程序作为样例进行执行

```
#include<iostream>
using namespace std;
int main(){
    cout<<"Hello World!"<<a + b = i++;
}
```

执行结果为



```
E:\大学资料\大三上课程\编译原理\win_flex_bison-latest>win_flex a.l

E:\大学资料\大三上课程\编译原理\win_flex_bison-latest>gcc cifa lex.yy.c
gcc: error: cifa: No such file or directory

E:\大学资料\大三上课程\编译原理\win_flex_bison-latest>gcc -o cifa lex.yy.c

E:\大学资料\大三上课程\编译原理\win_flex_bison-latest>cifa
1       (operator,#)
2       (keyword,include)
3       (operator,<)
4       (keyword,iostream)
5       (operator,>)
6       (keyword,using)
7       (keyword,namespace)
8       (keyword,std)
9       (delimiter,;)
10      (keyword,int)
11      (keyword,main)
12      (delimiter,()
13      (delimiter,))
14      (operator,{)
15      (keyword,cout)
16      (operator,<<)
17      (count,"Hello World!")
18      (operator,<<)
19      (id,a)
20      (operator,+)
21      (id,b)
22      (operator,=)
23      (id,i)
24      (operator,++)
25      (delimiter,;)
26      (operator,})
```

3 - 实验总结

通过编写词法分析让我对于词法识别的自动机有了更深入的了解，在设计状态转移图的时候，我借鉴了课本的部分内容，然而书上有些情况并没有考虑周全，例如对于位运算符操作、幂等等复制运算，我针对部分情况进行了补充。同时熟悉了文件读写的操作。词法分析作为语法分析的子程序，是文法分析中较为重要的环节。本次实验将词法分析独立出来，体现了程序高内聚解耦合的优点，采用模块化的设计加强了词法分析的可移植性。除此以外，通过学习lex词法分析语言并进行词法分析使我接触了一门新的语言，并在其中体会到了前人的智慧，通过将语法程序更好的封装从而增强模块化的特性。在本次实验过程中，我对自动机、词法分析的相关知识有了更深一步的理解和运用，增强了自己的代码能力和自动机设计能力。