# 说明文档

| 姓名 | 学号 | 班级 |
| --- | --- | --- |
| 凌国瀚 | 2018213344 | 2018211314 |

## 目录



## 题目要求

### 实验内容及要求

1. 可以识别出用 C 语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。
2. 可以识别并跳过源程序中的注释。
3. 可以统计源程序中的语句行数、各类单词的个数、以及字符总数，并输出统计结果。
4. 检查源程序中存在的词法错误，并报告错误所在的位置。
5. 对源程序中出现的错误进行适当的恢复，使词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中存在的所有词法错误。

### 实现方法要求

分别用以下两种方法实现

- 方法一：
    - 采用 C/C++ 作为实现语言，手工编写词法分析程序。
- 方法二：
    - 编写 LEX 源程序，利用 LEX 编译程序自动生成词法分析程序。

---

# 开发环境与运行环境

*代码编辑器*：Visual Studio Code 15.0 with Remote Process Explorer

*操作系统*：Windows 10 (version: 2004)

*运行环境*：Windows Subsystem of Linux (Ubuntu-20.04 LTS)

*编译器*：GCC-9 / G++-5 on Linux

*词法分析器*：flex on Linux

---

# 程序设计说明

## 文件说明

```
2018211314-凌国瀚-2018213344
├── CPP
|   ├── analysis
|   ├── analysis.cpp
|   ├── in.c
|   └── out.txt
├── LEX
|   ├── a.out
|   ├── lex_source.l
|   ├── lex.yy.c
|   ├── result.txt
|   └── test1.c
├── README.md
└── README.pdf
```

文件树如上所示，其中`CPP`文件夹存放 C++ 版本的词法分析程序，`LEX`文件夹存放 LEX 版本的词法分析程序。

`CPP`文件夹中，`analysis.cpp`为源代码，`analysis`为可执行文件，`in.c`是默认的待分析文本，`out.txt`是程序输出结果。

`LEX`文件夹中，`lex_source.l`为源代码，`lex.yy.c`为 flex 编译 LEX 文件后生成的中间 C 代码，`a.out`是可执行文件，`test1.c`是默认的待分析文本，`result.txt`是程序输出结果。

`README.md`为本说明文档，`README.pdf`是本文档的 pdf 版本，以便于在没有 Markdown 浏览条件时阅读本文档。

欲运行本程序，请参照运行说明。

## 运行说明

### C++ 版本

欲编译本程序并生成可执行文件，请在**源程序所在目录**的 Linux bash 中输入如下指令：

```
g++ analysis.cpp -o analysis
```

即可生成可执行文件`analysis`。

随后，直接使用以下指令即可运行可执行文件：

```
./analysis
```

### LEX 版本

欲编译本程序并生成可执行文件，请在**源程序所在目录**的 Linux bash 中依次输入如下指令：

```
flex lex_source.l
gcc lex.yy.c -ll
./a.out < test1.cpp
```

即可生成结果`result.txt`。

> **温馨提示**
>
> 1. 最后一行编译指令中，文件重定向可以更改，以选择`test1.cpp`以外的待分析文本。
> 2. 第二行指令中不要忘记链接 flex 库（-ll）。

## 程序简介

C++版本的程序源文件内容**每次一行**地读入缓冲 buffer 中，向前指针则在每次 buffer 更新后从 buffer 的头部开始扫描，根据扫描得到不同的字符调用相应的分析代码。缓冲 buffer 的设置保证了读入字符的正确性，同时，本程序还支持判断单行和多行注释，并跳过其内容；也支持错误检测和错误处理，支持检测的错误如下：

1. *Exponent has no digits*：指数部分没有数字，如`2.3e`、`4e+`等
2. *ID cannot start with digits*：记号不能以数字开头，如`123abc`、`666_bcd`等
3. *Missing terminating " character*：字符串未匹配到末尾的冒号
4. *Missing terminating ' character*：字符未匹配到末尾的冒号
5. *Invalid char value*：非法的字符，正确的字符格式为`'a'`、`'\n'`等，非法的字符格式为`'a1'`、`'b\t'`等

6. *Invalid number pattern (double dots)*：一个数字出现了两个逗号

## 数据结构

> 本部分仅讨论 C++ 版本程序

### 计数变量

```
int line = 0, column = 0, cnt_word = 0, cnt_char = 0;
```

计数变量用于记录当前指针所分析的字符在源文件中的行数和列数，也记录了源文件中的单词数量和字符数量。**请注意，被分析程序判断为非法/错误的单词将不参与计数。**

### IO 流

```
ifstream in_file_stream;
ofstream out_file_stream;
```

本程序利用 C++特有的 IO 流处理文件输入和输出。

### 缓冲区

```
string buffer, token;
```

利用 C++ 的`string`对象作为缓冲区的数据结构。

`buffer`是数据缓冲区，在程序中，`buffer`存储了当前扫描行数的一整行内容。它的主要功能是超前扫描若干个字符，其目的是为了得到某一个单词符号的确切性质。

`token`是已经扫描成功的字符，当一个单词被认为扫描完毕，此时就应该输出`token`，因为它存储了一个单词的全部字符。

### 向前指针

```
string::iterator ptr_forward = buffer.end();
```

`ptr_forward`是向前指针，指示了当前即将分析的字符。它的数据类型是`string::iterator`，用以适应`buffer`的数据类型`string`。

### 关键字集合

```
string words[] = {"include", "define", "auto", "double", "int", "struct", "break",
"else", "long", "switch", "case", "enum", "register", "typedef", "char", "extern",
"return", "union", "const", "float", "short", "unsigned", "continue", "for",
"signed", "void", "default", "goto", "sizeof", "volatile", "do", "if", "static",
"while"};
set<string> keywords(words, words + 34);
```

利用 C++ 数据结构`set`，定义关键字集合。这样定义的原因在于，当分析器扫描到一个记号时，可以直接使用`set`的内置函数判断其是否为 C 语言关键字。

**分析容器**

```
vector<string> table;
map<string, int> counter_map;
```

`table`是一个`vector`容器，其记录了每一个记号（ID）的出现次数，这个容器的设计目的在于，当文本中多次出现同样模式的记号时，将会将它们归结为同一个记号而非多个记号。

`counter_map`是一个`map`容器，用于统计各类单词的个数。

## 函数结构

> 本部分仅讨论 C++ 版本程序

下方的注释解释了所有函数的接口及其用法。

```
/**
 * @brief 向符号表中插入符号。这个符号可能已经存在，也可能是新符号。
 * 若为新符号，插入符号表末尾。
 *
 * @return int 返回插入的符号在符号表的位置，若是新符号，则一定在符号表的末尾
 */
int table_insert()

/**
 * @brief 读取并测试实数，其中包含浮点数或带有E/e的指数。
 * 注意，本函数涉及状态转换。
 * 状态1为初始状态，读取实数；
 * 状态2为读取小数点后数字的状态；
 * 状态3为读取指数符号后数字的状态；
 * 状态4为读取指数符号后的缓冲态，以防止指数符号后出现+/-符号；
 * 状态5为读取指数符号后数字的状态。
 *
 * @param state_param 状态参数，输入1则开始判断实数
 * @return true 正确的数字格式
 * @return false 数字后紧接着记号字符，请输出错误的记号提示
 */
bool test_digits(int state_param)
```

```
/**
 * @brief 测试注释内容
 *
 * @return true 注释通过词法分析
 * @return false 读到错误或EOF
 */
bool test_comments()

/**
 * @brief 封装输出样式
 *
 * @param type_str 输出的记号类型
 */
void output_line(string type_str)

/**
 * @brief 在输出开头展示说明文本
 *
 */
void show_head_word()

/**
 * @brief 展示结果文本并关闭IO流
 *
 */
void show_result()
```

---

# 运行结果及分析

> 本部分仅讨论 C++ 版本程序

## 输入

测试的输入文件为其它部分均正确的 C 语言代码（详见in.c），在其 main 函数中设置几处词法错误如下：

```
int main()
{
    char *s = "12av";
    char a = 'a',a1 = 'r, b = 'a';  // 字符错误
    int 123abc; // 记号错误
    int c = 1e+3;
    int d = 2e, e = 1.23.1; // 指数错误，小数点错误
    printf("%c\n", '\n');
    char *str = "abcdefg;   // 字符串错误
}
```

## 运行结果及其说明

运行结果较长，此处将节选运行结果进行说明，详细内容请见out.txt。

> 若欲查看 LEX 版本的分析结果，请将欲分析代码复制至 LEX 文件夹的 test1.c 中，并根据运行说明所指示的步骤运行 LEX 版本代码。

```
            Specification

[ID-<number>]: 用户定义或额外导入的库函数中的记号
[keyword]: C语言保留字
[num]: 全体实数，支持指数表示
[comments]: 注释
[punct]: 标点符号
[char]: 字符
[string]: 字符串
[arith-op]: 算数运算符
[asgn-op]: 复合运算符
[ptr-op]: 指针运算符
[bit-op]: 位运算符
[logic-op]: 逻辑运算符
[relop-op]: 关系运算符

请注意：以下的Column对于多个字符的记号来说，指向的是其最后一个字符所在的列数
```

out.txt的最上方显示了单词类型提示和注意事项。

```
----------------Result-------------------
Line:Column    Type      Token
<    1:2         punct     #            >
<    1:9         keyword   include      >
<    1:11        relop-op  <            >
<    1:15        ID-1      math         >
<    1:16        punct     .            >
<    1:17        ID-2      h            >
<    1:18        relop-op  >            >
<    2:2         punct     #            >
<    2:9         keyword   include      >
...
```

out.txt的主体部分如上所示，通过一个三元组，表示了一个单词的行列数、类型以及其内容。由于中文字符在 C 语言中的编码以及字符串和注释的长度可能过长等原因，Token 部分将不显示字符串和注释的内容。

```
...
< Error(97,15): Exponent has no digits >
< Error(97,15): ID cannot start with digits >
<   97:16        punct     ,            >
<   97:18        ID-35     e            >
```

```
<   97:20        asgn-op   =            >
< Error(97,23): Invalid number pattern (double dots) >
< Error(97,23): ID cannot start with digits >
< Error(97,26): Invalid number pattern (double dots) >
< Error(97,26): ID cannot start with digits >
<   97:27        num       1            >
<   97:28        punct     ;            >
<   98:11        ID-25     printf       >
<   98:12        punct     (            >
<   98:18        string    -            >
<   98:19        punct     ,            >
<   98:24        char      '\n'         >
<   98:25        punct     )            >
<   98:26        punct     ;            >
<   99:9         keyword   char         >
<   99:11        arith-op  *            >
<   99:14        ID-36     str          >
<   99:16        asgn-op   =            >
< Error(99,26): Missing terminating " character >
<  100:2         punct     }            >
```

在 main 函数的部分，可以看到词法分析器成功检测到了词法错误，并通过适当修整使得词法分析能够顺利进行。

```
---------------ID table------------------
1       math
2       h
3       stdio
4       stdlib
5       string
6       bool
7       false
8       true
9       MAXSIZE
10      ElementType
11      Position
12      LNode
13      List
14      Data
15      Last
16      MakeEmpty
17      L
18      malloc
...
```

在主体内容的下方，`out.txt`还记录了所有检测到的记录（ID），并展示了其序号和内容。

```
--------------Analysis-----------------
1       ID          137
2       arith-op    15
3       asgn-op     15
4       char        2
5       comments    19
6       keyword     42
7       logic-op    3
8       num         15
9       ptr-op      16
10      punct       130
11      relop-op    18
12      string      5
-----------------Total-----------------
Total lines: 100
Total words: 399
Total characters: 2431
```

在out.txt最下方，给出了词法分析的综合结果，这里记录了所有源文件中各种类型词语的数量，同时也记录了源文件的总行数和词语、字符数量。

---

## 源程序

C++

```cpp
#include <algorithm>
#include <cctype>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <set>
#include <string>
#include <vector>

using namespace std;

int line = 0, column = 0, cnt_word = 0, cnt_char = 0;

ifstream in_file_stream;
ofstream out_file_stream;

string in_file_str, out_file_str, buffer, token;
string::iterator ptr_forward = buffer.end(); // 向前指针
string words[] = {"include", "define", "auto", "double", "int", "struct", "break",
"else", "long", "switch", "case", "enum", "register", "typedef", "char", "extern",
"return", "union", "const", "float", "short", "unsigned", "continue", "for",
"signed", "void", "default", "goto", "sizeof", "volatile", "do", "if", "static",
```

```
"while"};
vector<string> table;
map<string, int> counter_map;

/**
 * @brief 向符号表中插入符号。这个符号可能已经存在，也可能是新符号。
 * 若为新符号，插入符号表末尾。
 *
 * @return int 返回插入的符号在符号表的位置，若是新符号，则一定在符号表的末尾
 */
int table_insert()
{
    vector<string>::iterator it = find(table.begin(), table.end(), token);

    if (it == table.end())
    {
        table.push_back(token);
        it = find(table.begin(), table.end(), token);
    }

    return distance(table.begin(), it) + 1;
}

/**
 * @brief 读取并测试实数，其中包含浮点数或带有E/e的指数。
 * 注意，本函数涉及状态转换。
 * 状态1为初始状态，读取实数；
 * 状态2为读取小数点后数字的状态；
 * 状态3为读取指数符号后数字的状态；
 * 状态4为读取指数符号后的缓冲态，以防止指数符号后出现+/-符号；
 * 状态5为读取指数符号后数字的状态。
 *
 * @param state_param 状态参数，输入1则开始判断实数
 * @return true 发生了错误
 * @return false 未发生错误
 */
bool test_digits(int state_param)
{
    int state = state_param;
    bool not_end_boolen = true, is_error = false;

    while ((ptr_forward != buffer.end()) && not_end_boolen)
    {
        switch (state)
        {
        case 1:
            // 数字部分
            if (*ptr_forward == '.')
            {
                token.append(1, *ptr_forward++);
                column++;
                state = 2;
            }
            else if (*ptr_forward == 'E' || *ptr_forward == 'e')
```

```
            {
                token.append(1, *ptr_forward++);
                column++;
                state = 3;
            }
            else if (isdigit(*ptr_forward))
            {
                token.append(1, *ptr_forward++);
                column++;
                state = 1;
            }
            else
                not_end_boolen = false; // 识别结束

            break;

        case 2:
            // 读取到小数点
            if (token.find('.') != string::npos)
            {
                out_file_stream << "< Error(" << line << "," << column << "):
Invalid number pattern (double dots) >" << endl;
                is_error = true;
                not_end_boolen = false;
            }
            else if (*ptr_forward == 'E' || *ptr_forward == 'e')
            {
                token.append(1, *ptr_forward++);
                column++;
                state = 3;
            }
            else if (isdigit(*ptr_forward))
            {
                token.append(1, *ptr_forward++);
                column++;
                state = 1;
            }
            else
                not_end_boolen = false;

            break;

        case 3:
            // 读取到指数符号
            if (*ptr_forward == '+' || *ptr_forward == '-')
            {
                token.append(1, *ptr_forward++);
                column++;
                state = 4;
            }
            else if (isdigit(*ptr_forward))
            {
                token.append(1, *ptr_forward++);
                column++;
```

```
                state = 5;
            }
            else
            {
                out_file_stream << "< Error(" << line << "," << column << "):
Exponent has no digits >" << endl;
                is_error = true;
                not_end_boolen = false;
            }

            break;

        case 4:
            // +/-号后必须有一个数字
            if (isdigit(*ptr_forward))
            {
                token.append(1, *ptr_forward++);
                column++;
                state = 5;
            }
            else
            {
                out_file_stream << "< Error(" << line << "," << column << "):
Exponent has no digits >" << endl;
                is_error = true;
                not_end_boolen = false;
            }

            break;

        case 5:
            // 仅数字部分
            if (isdigit(*ptr_forward))
            {
                token.append(1, *ptr_forward++);
                column++;
                state = 5;
            }
            else
                not_end_boolen = false;

            break;

        default:
            // 实际上代码不可能运行到此处
            out_file_stream << "< Error(" << line << "," << column << "): Function
test_digits() Error! >" << endl;
            not_end_boolen = false;
            is_error = true;
            break;
        }
    }

    if ((*ptr_forward >= 'a' && *ptr_forward <= 'z') || (*ptr_forward >= 'A' &&
```

```
*ptr_forward <= 'Z') || *ptr_forward == '_')
        return true;

    return is_error;
}

/**
 * @brief 测试注释内容
 *
 * @return true 注释通过词法分析
 * @return false 读到错误或EOF
 */
bool test_comments()
{
    while (true)
    {
        while ((ptr_forward != buffer.end()) && (*ptr_forward != '*'))
        {
            token.append(1, *ptr_forward++);
            column++;
        }

        if (ptr_forward == buffer.end())
        {
            if (getline(in_file_stream, buffer, '\n'))
            {
                // 一行注释结束，继续下一行
                line++;
                ptr_forward = buffer.begin();
                column = 1;
                continue;
            }
            else
                return false;
        }
        else
        {
            // 读到注释结束符的星号
            token.append(1, *ptr_forward++);
            column++;

            if (ptr_forward == buffer.end()) // 若 '*/' 被换行分开，则继续读取下一行
                continue;
            else if (*ptr_forward == '/')
            {
                // 注释结束
                token.append(1, *ptr_forward++);
                column++;
                return true;
            }
            else // 星号后是其他字符，仍继续读取注释
                continue;
        }
    }
```

```
}

/**
 * @brief 封装输出样式
 *
 * @param type_str 输出的记号类型
 */
void output_line(string type_str)
{
    string type_str_finder = type_str;
    if (type_str_finder.substr(0, 2) == "ID")
        type_str_finder = "ID";

    map<string, int>::iterator iter = counter_map.find(type_str_finder);
    if (iter != counter_map.end())
        counter_map[type_str_finder]++;
    else
        counter_map[type_str_finder] = 1;

    if (type_str == "comments" || type_str == "string")
    {
        out_file_stream << "<"
                        << right << setw(5) << line << ":"
                        << left << setw(10) << column
                        << left << setw(10) << type_str
                        << left << setw(13) << "-"
                        << ">" << endl;
    }
    else
    {
        out_file_stream << "<"
                        << right << setw(5) << line << ":"
                        << left << setw(10) << column
                        << left << setw(10) << type_str
                        << left << setw(13) << token << ">" << endl;
    }
}

/**
 * @brief 在输出开头展示说明文本
 *
 */
void show_head_word()
{
    out_file_stream << right << setw(25) << "Specification" << endl
                    << endl;
    out_file_stream << "[ID-<number>]: 用户定义或额外导入的库函数中的记号" << endl;
    out_file_stream << "[keyword]: C语言保留字" << endl;
    out_file_stream << "[num]: 全体实数，支持指数表示" << endl;
    out_file_stream << "[comments]: 注释" << endl;
    out_file_stream << "[punct]: 标点符号" << endl;
    out_file_stream << "[char]: 字符" << endl;
    out_file_stream << "[string]: 字符串" << endl;
    out_file_stream << "[arith-op]: 算数运算符" << endl;
```

```
    out_file_stream << "[asgn-op]: 复合运算符" << endl;
    out_file_stream << "[ptr-op]: 指针运算符" << endl;
    out_file_stream << "[bit-op]: 位运算符" << endl;
    out_file_stream << "[logic-op]: 逻辑运算符" << endl;
    out_file_stream << "[relop-op]: 关系运算符" << endl
                    << endl;
    out_file_stream << "请注意：以下的Column对于多个字符的记号来说，指向的是其最后一个
字符所在的列数" << endl
                    << endl;
    out_file_stream << "----------------Result-------------------" << endl;
    out_file_stream << right << setw(6) << "Line"
                    << ":"
                    << left << setw(10) << "Column"
                    << left << setw(10) << "Type"
                    << left << setw(13) << "Token" << endl;
}

/**
 * @brief 展示结果文本并关闭IO流
 *
 */
void show_result()
{
    int i = 1;

    out_file_stream << "---------------ID table-----------------" << endl;

    for (vector<string>::iterator it = table.begin(); it != table.end(); it++)
        out_file_stream << left << setw(5) << i++ << "\t" << *it << endl;

    i = 1;

    out_file_stream << "---------------Analysis-----------------" << endl;

    for (map<string, int>::iterator it = counter_map.begin(); it !=
counter_map.end(); it++)
        out_file_stream << left << setw(5) << i++ << "\t" << left << setw(10) <<
it->first << "\t" << it->second << endl;

    cout << "分析完成，请到目标文件" << out_file_str << "查看输出结果！ " << endl;
    out_file_stream << "-----------------Total-----------------" << endl;
    out_file_stream << "Total lines: " << line << endl;
    out_file_stream << "Total words: " << cnt_word << endl;
    out_file_stream << "Total characters: " << cnt_char << endl;

    in_file_stream.close();
    out_file_stream.close();
}

int main()
{
    set<string> keywords(words, words + 34);
    char C;
    cout << "请输入源文件名称（回车则默认使用in.c）： " << endl;
```

```
    getline(cin, in_file_str);

    if (in_file_str == "")
        in_file_str = "in.c";

    in_file_stream.open(in_file_str.c_str());

    if (!in_file_stream)
    {
        cout << "无法打开源文件! " << endl;
        return -1;
    }

    cout << "请输入目标文件名称（回车则默认使用out.txt）: " << endl;
    getline(cin, out_file_str);

    if (out_file_str == "")
        out_file_str = "out.txt";

    out_file_stream.open(out_file_str.c_str());

    if (!out_file_stream)
    {
        cout << "无法创建目标文件! " << endl;
        return -1;
    }

    show_head_word();

    while (true)
    {
        // 前进指针读到缓存末尾
        if (ptr_forward == buffer.end())
        {
            // 读到行末
            if (getline(in_file_stream, buffer, '\n'))
            {
                line++;              // 读取新行
                cnt_char += column; // 每次换行前加上当前行的字符数
                ptr_forward = buffer.begin();
                column = 1;
            }
            else // 读到EOF则输出结果
            {
                show_result();
                return 0;
            }
        }

        while ((ptr_forward != buffer.end()) && isspace(*ptr_forward))
        {
            // 跳过空字符
            ptr_forward++;
            column++;
```

```
        }

        if (ptr_forward != buffer.end())
        {
            // 未读到缓冲区末尾
            token = "";
            C = *ptr_forward;

            if ((C >= 'a' && C <= 'z') || (C >= 'A' && C <= 'Z') || C == '_')
            {
                token.append(1, C);
                ptr_forward++;
                column++;

                while ((ptr_forward != buffer.end()) && (isalnum(*ptr_forward) ||
*ptr_forward == '_'))
                {
                    // 合法标识符可包含下划线
                    token.append(1, *ptr_forward++);
                    column++;
                }

                if (keywords.count(token) == 0)
                    output_line("ID-" + to_string(table_insert()));
                else
                    output_line("keyword");

                cnt_word++; // 单词数加一
            }
            else if (C >= '0' && C <= '9')
            {
                token.append(1, C);
                ptr_forward++;
                column++;
                if (!test_digits(1)) // 读取无符号实数剩余部分
                {
                    output_line("num");
                    cnt_word++;
                }
                else
                    out_file_stream << "< Error(" << line << "," << column << "):
ID cannot start with digits >" << endl;
            }
            else
            {
                switch (C)
                {
                case '+':
                    token.append(1, C);
                    ptr_forward++;
                    column++;

                    if (ptr_forward == buffer.end())
                        output_line("arith-op");
```

```
                    else
                    {
                        if (*ptr_forward == '+')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("arith-op");
                        }
                        else if (*ptr_forward == '=')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("asgn-op");
                        }
                        else
                            output_line("arith-op");
                    }

                    cnt_word++;
                    break;

                case '-':
                    token.append(1, C);
                    ptr_forward++;
                    column++;

                    if (ptr_forward == buffer.end())
                        output_line("arith-op");
                    else
                    {
                        if (*ptr_forward == '-')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("arith-op");
                        }
                        else if (*ptr_forward == '=')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("asgn-op");
                        }
                        else if (*ptr_forward == '>')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("ptr-op");
                        }
                        else
                            output_line("arith-op");
                    }

                    cnt_word++;
                    break;
```

```
            case '*':
                token.append(1, C);
                ptr_forward++;
                column++;

                if (ptr_forward == buffer.end())
                    output_line("arith-op");
                else
                {
                    if (*ptr_forward == '=')
                    {
                        token.append(1, *ptr_forward++);
                        column++;
                        output_line("asgn-op");
                    }
                    else
                        output_line("arith-op");
                }

                cnt_word++;
                break;

            case '/':
                token.append(1, C);
                ptr_forward++;
                column++;

                if (ptr_forward == buffer.end())
                {
                    output_line("arith-op");
                    cnt_word++;
                }
                else
                {
                    if (*ptr_forward == '=')
                    {
                        // 除法复合赋值
                        token.append(1, *ptr_forward++);
                        column++;
                        output_line("asgn-op");
                        cnt_word++;
                    }
                    else if (*ptr_forward == '/')
                    {
                        // 单行注释，读到行末
                        token.append(1, *ptr_forward++);
                        column++;

                        while (ptr_forward != buffer.end())
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                        }
```

```
                    output_line("comments");
                }
                else if (*ptr_forward == '*')
                {
                    // 多行注释可以换行
                    token.append(1, *ptr_forward++);
                    column++;
                    int ret = test_comments();

                    if (ret)
                        output_line("comments");
                    else
                    {
                        show_result();
                        return 0;
                    }
                }
                else
                {
                    // 除号后是其他字符，为单个除号
                    output_line("arith-op");
                    cnt_word++;
                }
            }

            break;

        case '%':
            token.append(1, C);
            ptr_forward++;
            column++;

            if (ptr_forward == buffer.end())
                output_line("arith-op");
            else
            {
                if (*ptr_forward == '=')
                {
                    token.append(1, *ptr_forward++);
                    column++;
                    output_line("asgn-op");
                }
                else
                    output_line("arith-op");
            }

            cnt_word++;
            break;

        case '&':
            token.append(1, C);
            ptr_forward++;
            column++;
```

```
                    if (ptr_forward == buffer.end())
                        output_line("bit-op");
                    else
                    {
                        if (*ptr_forward == '=')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("asgn-op");
                        }
                        else if (*ptr_forward == '&')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("logic-op");
                        }
                        else
                            output_line("bit-op");
                    }

                    cnt_word++;
                    break;

                case '|':
                    token.append(1, C);
                    ptr_forward++;
                    column++;

                    if (ptr_forward == buffer.end())
                        output_line("bit-op");
                    else
                    {
                        if (*ptr_forward == '=')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("asgn-op");
                        }
                        else if (*ptr_forward == '|')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("logic-op");
                        }
                        else
                            output_line("bit-op");
                    }

                    cnt_word++;
                    break;

                case '^':
                    token.append(1, C);
                    ptr_forward++;
```

```
                column++;

                if (ptr_forward == buffer.end())
                    output_line("bit-op");
                else
                {
                    if (*ptr_forward == '=')
                    {
                        token.append(1, *ptr_forward++);
                        column++;
                        output_line("asgn-op");
                    }
                    else
                        output_line("bit-op");
                }

                cnt_word++;
                break;

            case '~':
                token.append(1, C);
                ptr_forward++;
                column++;
                output_line("bit-op");
                cnt_word++;
                break;

            case '<':
                token.append(1, C);
                ptr_forward++;
                column++;

                if (ptr_forward == buffer.end())
                    output_line("relop-op");
                else
                {
                    if (*ptr_forward == '=')
                    {
                        token.append(1, *ptr_forward++);
                        column++;
                        output_line("relop-op");
                    }
                    else
                        output_line("relop-op");
                }

                cnt_word++;
                break;

            case '=':
                token.append(1, C);
                ptr_forward++;
                column++;
```

```
                    if (ptr_forward == buffer.end())
                        output_line("asgn-op");
                    else
                    {
                        if (*ptr_forward == '=')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("relop-op");
                        }
                        else
                            output_line("asgn-op");
                    }

                    ++cnt_word;
                    break;

                case '>':
                    token.append(1, C);
                    ptr_forward++;
                    column++;

                    if (ptr_forward == buffer.end())
                        output_line("relop-op");
                    else
                    {
                        if (*ptr_forward == '=')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("relop-op");
                        }
                        else
                            output_line("relop-op");
                    }

                    cnt_word++;
                    break;

                case '!':
                    token.append(1, C);
                    ptr_forward++;
                    column++;

                    if (ptr_forward == buffer.end())
                        output_line("punct");
                    else
                    {
                        if (*ptr_forward == '=')
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("relop-op");
                        }
```

```
                        else
                            output_line("punct");
                    }

                    cnt_word++;
                    break;

                case '\"':
                    token.append(1, C);
                    ptr_forward++;
                    column++;

                    while (true)
                    {
                        while ((ptr_forward != buffer.end()) && (*ptr_forward !=
'\"'))
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                        }

                        if (ptr_forward == buffer.end())
                        {
                            out_file_stream << "< Error(" << line << "," << column
<< "): Missing terminating \" character >" << endl;
                            break;
                        }
                        else if (*(ptr_forward - 1) == '\\')
                        {
                            // 跳过转义符\"
                            token.append(1, *ptr_forward++);
                            column++;
                            continue;
                        }
                        else
                        {
                            token.append(1, *ptr_forward++);
                            column++;
                            output_line("string");
                            break;
                        }
                    }

                    cnt_word++;
                    break;

                case '\'':
                    token.append(1, C);
                    ptr_forward++;
                    column++;

                    while ((ptr_forward != buffer.end()) && (*ptr_forward !=
'\''))
                    {
```

```
                        token.append(1, *ptr_forward++);
                        column++;
                    }

                    if (ptr_forward == buffer.end())
                        out_file_stream << "< Error(" << line << "," << column <<
"): Missing terminating \' character >" << endl;
                    else if ((*(ptr_forward - 2) == '\\' && token.size() == 3) ||
(*(ptr_forward - 2) != '\\' && token.size() == 2))
                    {
                        // char类型只能是一个字符，或是两个字符的转义符
                        token.append(1, *ptr_forward++);
                        column++;
                        output_line("char");
                        cnt_word++;
                    }
                    else
                    {
                        ptr_forward++;
                        column++;
                        out_file_stream << "< Error(" << line << "," << column <<
"): Invalid char value >" << endl;
                    }

                    break;

                case '.':
                    token.append(1, C);
                    ptr_forward++;
                    column++;

                    if ((ptr_forward != buffer.end()) && isdigit(*ptr_forward))
                    {
                        // 小数点
                        token.append(1, *ptr_forward++);
                        column++;
                        if (!test_digits(2)) // 读取无符号实数剩余部分
                        {
                            output_line("num");
                            cnt_word++;
                        }
                        else
                            out_file_stream << "< Error(" << line << "," << column
<< "): ID cannot start with digits >" << endl;
                        break;
                    }

                    output_line("punct");
                    cnt_word++;
                    break;

                case '#':
                case '{':
                case '}':
```

```
                case '[':
                case ']':
                case '(':
                case ')':
                case '?':
                case ':':
                case ',':
                case ';':
                case '\\':
                    token.append(1, C);
                    ptr_forward++;
                    column++;
                    output_line("punct");
                    cnt_word++;
                    break;

                default:
                    ptr_forward++;
                    column++;
                    out_file_stream << "< Error(" << line << "," << column << "):
Invalid character >" << endl;
                    break;
                } // end of switch
            }     // end of else
        }         // end of if
    }             // end of while

    return 0;
}
```

LEX

```
%{
#include <stdio.h>
#include <string.h>

int wordCount = 0;
int charCount = 0;
int columnCount = 0;
int lineCount = 1;
int IDCount = 0;
int stringCount = 0;
FILE *fp;


%}

delim        ([' ']|[\t])+
ANY          [.]
WARP         [\n]
```

```
NOTWARP      [^\n]
letter       [A-Za-z]
digit        [0-9]
NOTSTR       [^\"]
NOTCHAR      [^\']
ESC          [\\]
NOTESC       [^\\]
KEYWORD
include|define|char|short|int|unsigned|long|float|double|struct|union|void|enum|si
gned|const|volatile|typedef|auto|register|static|extern|break|case|continue|defaul
t|do|else|for|goto|if|return|switch|while|sizeof
STR          \"{NOTWARP}*\"
CHAR         ({delim}?\'{NOTESC}?\')|({delim}?\'{ESC}{letter}?\')
PUNC         \,|\;
DOT          \.
RELOP        \=\=|\<|\>|\<\=|\>\=
EQUAL        \=
OPERATOR     \+|\-|\*|\\|\&|\||\^|\?|\%|\+\+|\-\-|\+\=|\-\=|\*\=|\/\=
BRACKET      \(|\)|\{|\}|\[|\]
ID           ({letter}|\_)({letter}|{digit}|\_)*
NUM          {digit}+(\.{digit}+)?(E(\+|\-)?digit+)?
MACRO        \#
INCLUDELIB   \<({letter}|{digit}|{DOT})*\>
LINECOMMENT  \/\/{NOTWARP}*
error1       {digit}+{letter}+({letter}|{digit})*
error2       (\"{NOTSTR}*)
error3       @|~|`
error4       (({delim}?\'{NOTESC}{NOTCHAR}))|(({delim}?\'{ESC}{NOTCHAR}{NOTCHAR}))
%x           IN_COMMENT

%%
{delim}        {
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{KEYWORD}      {
    fprintf(fp,"KEYWORD(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
    wordCount++;
    }
{ID}           {
    fprintf(fp,"ID%d(%d, %d): %s\n",IDCount,lineCount,columnCount,yytext);
    IDCount++;
    columnCount+=yyleng;
    charCount+=yyleng;
    wordCount++;
    }
{NUM}          {
    fprintf(fp,"NUM(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{LINECOMMENT}  {
```

```
    fprintf(fp,"LINENOTE(%d, %d): %s\n",lineCount,columnCount,yytext);
    }
{STR}           {
    fprintf(fp,"STRING%d(%d, %d): %s\n",stringCount,lineCount,columnCount,yytext);
    stringCount++;
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{CHAR}          {
    fprintf(fp,"CHAR(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{PUNC}          {
    fprintf(fp,"PUNCTUATION(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount++;
    charCount++;
    }
{DOT}           {
    fprintf(fp,"DOT(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount++;
    charCount++;
    }
{MACRO}         {
    fprintf(fp,"MACRO(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount++;
    charCount++;
    }
{OPERATOR}      {
    fprintf(fp,"OPERATOR(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount++;
    charCount++;
    }
{RELOP}         {
    fprintf(fp,"RELOP(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{EQUAL}         {
    fprintf(fp,"EQUAL(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount++;
    charCount++;
    }
{BRACKET}       {
    fprintf(fp,"BRACKET(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{INCLUDELIB}    {
    fprintf(fp,"INCLUDE(%d, %d): %s\n",lineCount,columnCount,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{error1}        {
```

```
    fprintf(fp,"ERROR: Line %d, Column %d: Identity cannot start with number
\"%s\"\n",lineCount,columnCount+1,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{error2}       {
    fprintf(fp,"ERROR:Line %d, Column %d: String cannot match the left puncuation
\n",lineCount,columnCount+1);
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{error3}       {
    fprintf(fp,"ERROR: Line %d, Column %d: Invalid symbol
\"%s\"\n",lineCount,columnCount+1,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
    }
{error4}       {
    fprintf(fp,"ERROR: Line %d, Column %d: Invalid char pattern
\"%s\"\n",lineCount,columnCount+1,yytext);
    columnCount+=yyleng;
    charCount+=yyleng;
}

<INITIAL>{
"/*"           BEGIN(IN_COMMENT);yymore();
}
<IN_COMMENT>{
"*/"           fprintf(fp,"ANNODATE(%d, %d):
%s\n",lineCount,columnCount,yytext);BEGIN(INITIAL);
[^*\n]+        yymore();
"*"            yymore();
\n             lineCount++;yymore();
}


{WARP}         {
    lineCount++;
    columnCount=0;
    }
.              {
    fprintf(fp,"UNKNOW(%d, %d): %s\n",lineCount,columnCount,yytext);
    charCount++;
    columnCount++;
    }

%%

int main()
{
    fp = fopen("result.txt", "w");
    yylex();
    printf("\nFinished.\nResult:\n");
    printf("Chars: %d \n",charCount);
```

```
    printf("Lines: %d \n",lineCount);
    printf("Words: %d \n",wordCount);
    fclose(fp);
    return 0;
}

int yyWARP()
{
    return 1;
}
```