
西安建筑科技大学

# 实 验 报 告

课程名称：编译原理

实验名称：词法分析程序设计

院 （系）：信控学院

专业班级：计算机 2203

姓　　名：梁桐

学　　号：2209060322

指导教师：叶 娜

2025 年 3 月 26 日

# 实验一 词法分析程序设计

## 一、实验目的

理解词法分析器的任务和工作原理；掌握词法分析器的构建过程，并能够针对给定语言的词法规则，使用某种高级编程语言实现其词法分析器。

## 二、实验内容

1．给定表 1 所示的一个简单语言的单词符号表，其中，标识符是以字母开头、以字母和数字组成的任意符号串，常数为整数，即以数字组成的符号串。请完成以下任务：

（1）画出识别该语言词法单元的状态转换图；

（2）依据状态转换图，设计并编制词法分析程序，实现从输入源程序中，识别出各类单词，即关键字、标识符、常数、运算符、界符五大类，并输出各个单词的种别码和单词符号自身的值。

（3）设计恰当的测试用例对各类单词的识别进行测试。

表 1 某简单语言的词法规则

| 单词符号 | 种别码 | 单词符号 | 种别码 |
|---|---|---|---|
| void | 101 | >= | 207 |
| main | 102 | < | 208 |
| int | 103 | <= | 209 |
| char | 104 | == | 210 |
| if | 105 | <> | 211 |
| else | 106 | ++ | 212 |
| for | 107 | -- | 213 |
| while | 108 | ( | 301 |
| + | 201 | ) | 302 |
| - | 202 | { | 303 |
| * | 203 | } | 304 |
| / | 204 | ; | 305 |
| = | 205 | 标识符 | 400 |
| > | 206 | 常数 | 500 |

2．在线学习 LEX/FLEX 相关内容，并整理记录相关笔记。

推荐学习网址：https://www.educoder.net/shixuns/x452vfi6/challenges

## 三、实验设备

计算机、编程语言集成开发环境。

## 四．实验原理（或程序框图）及步骤

### 1、设计构思

设计实验方案：

根据实验指导书，明确实验需求，确定词法分析器的功能模块。

确定输入输出方式：支持文件输入与控制台输入两种模式。

增加错误检测功能：当扫描到无法识别的词素时，输出错误提示及行号，错误信息以红色显示。

模块划分：

输入处理模块：实现文件输入和控制台输入选择，负责读取源代码。

分词模块：逐行扫描源代码，根据字符类型拆分成一个个词素（Token 字符串）。

词法识别模块：针对每个词素，利用映射表及规则判断其类型（关键字、运算符、界符、标识符、数字常量）。

错误处理模块：检测无法识别的词素，并报告错误行号（错误信息以红色显示）。

输出模块：将每个 Token 的种别码和词素输出到控制台。

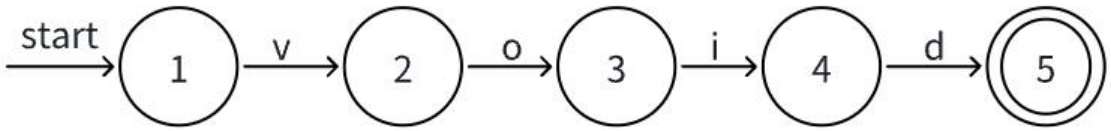
**2.正则表达式和状态转换图**

关键字的正则表达式与状态转换图

关键字：void

正则表达式：void

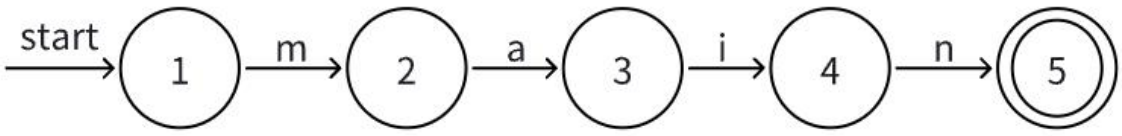
状态转换图：




关键字：main

正则表达式：main

状态转换图：

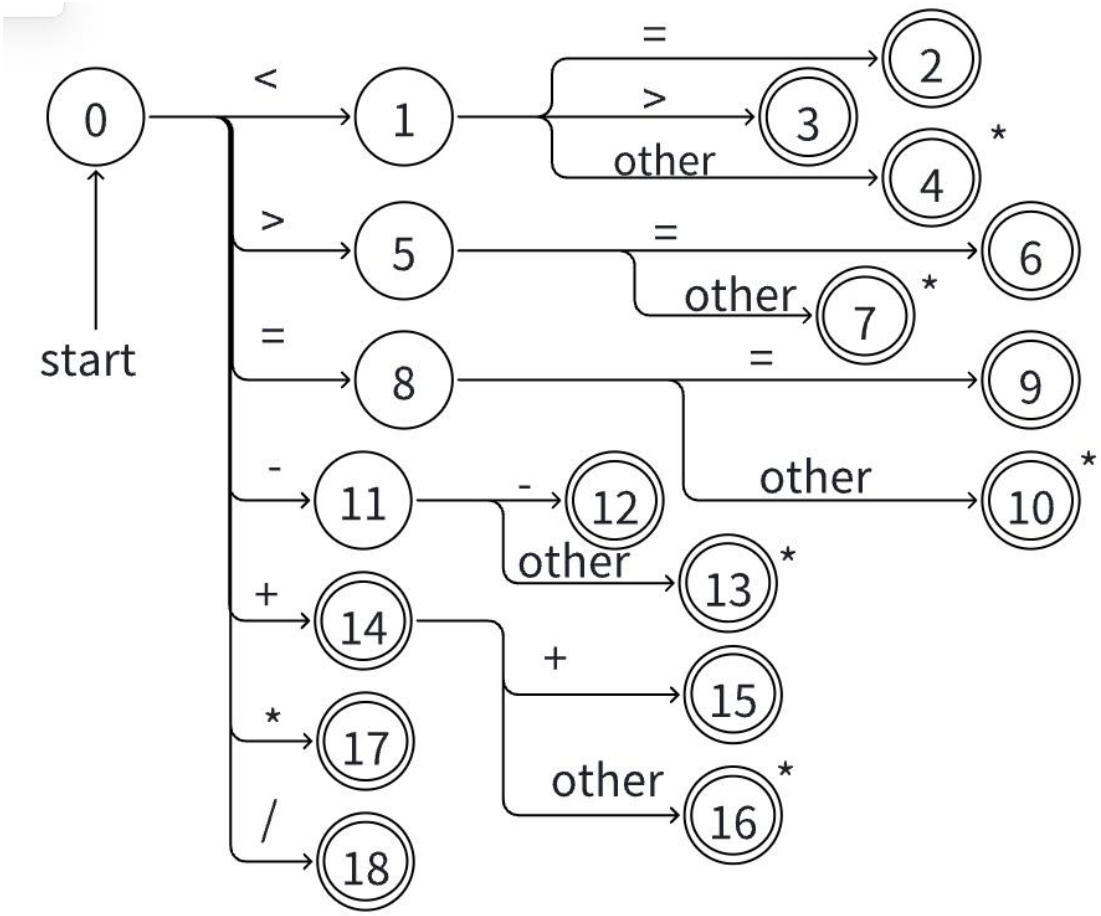



剩余的关键字以此类推。

运算符的正则表达式：

<|<=|<>|>|>=|=|==|+|++|-|--

运算符的状态转换图：




标识符的正则表达式：

letter(letter|dight)*

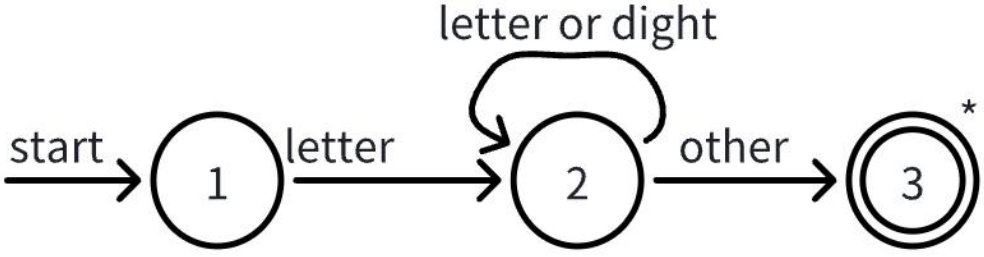
词法单元模式：

dight->[0-9]

dights->dight+

letter->[A-Za-z]|-

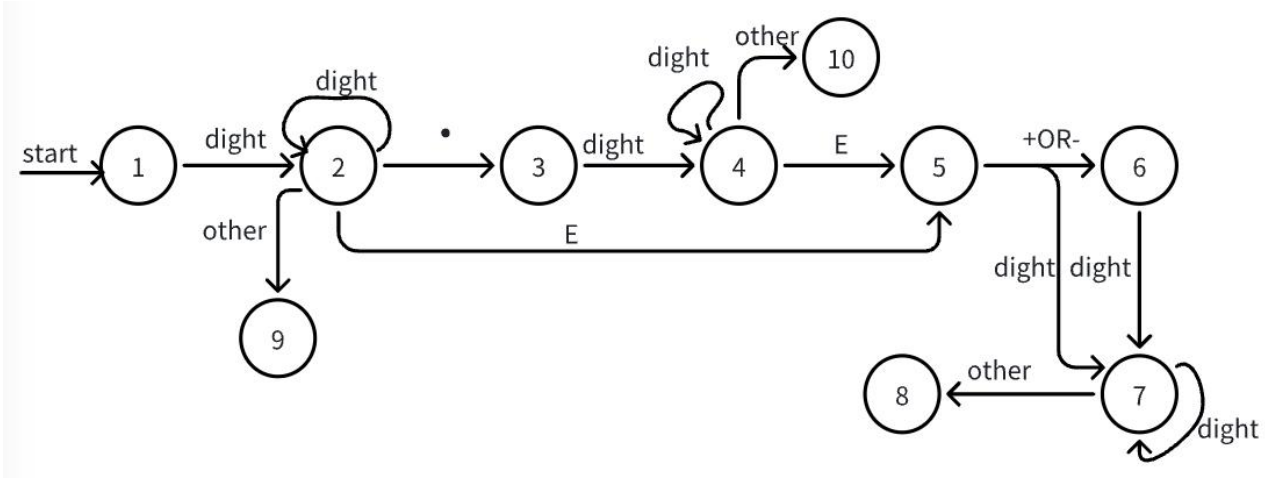
标识符的状态转换图：




常数的正则表达式：

dights(.dights)?(E[+-]?dights)?

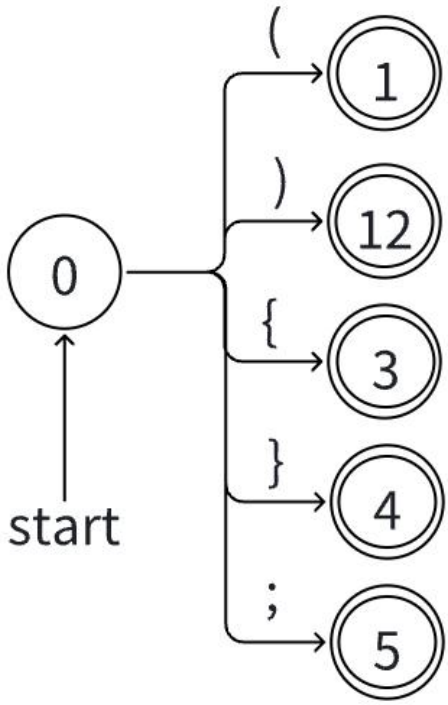
常数的状态转换图：




界符的正则表达式：

(|){|};

界符的状态转换图：




**3.数据结构**

映射表（map）

关键字表：map<string, int> keywordMap 用于存储关键字及其对应种别码。

运算符表：map<string, int> opMap 用于存储运算符（含多字符）及其种别码。

界符表：map<char, int> delimiterMap 用于存储单字符界符及其种别码。

Token 结构体

用于存储每个词法单元的识别结果，包括种别码和词素字符串。

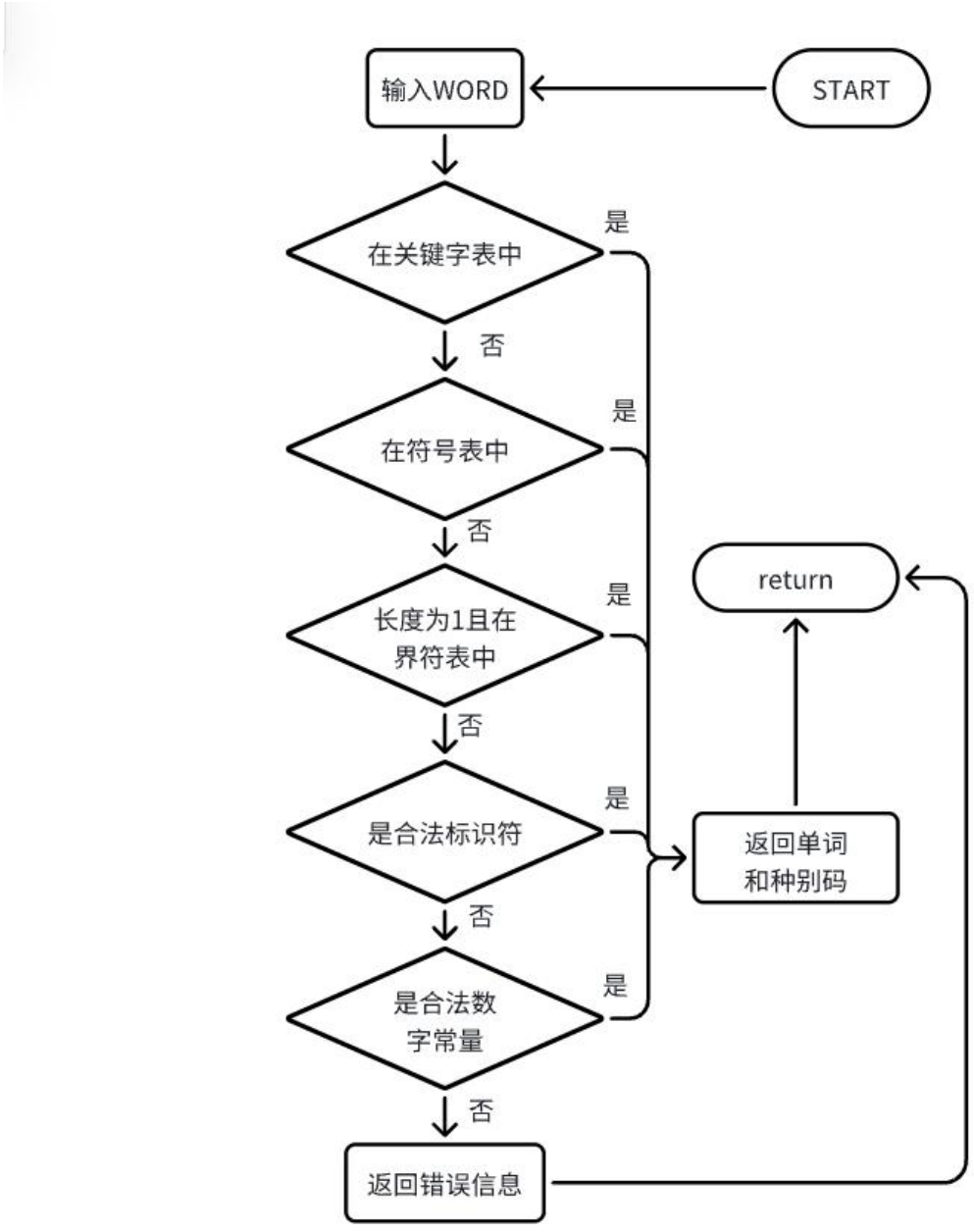
字符串向量

在分词模块中，使用 vector<string> 存储一行拆分出的各个词素。

辅助变量

如行号计数器、字符位置指针等。

**4.关键流程**




功能说明

接收一个单词（词素字符串），按照以下顺序进行判断并返回一个 Token：

关键字判断：如果 word 存在于 keywordMap 中，则填充 Token 的 code 与 lexeme 并返回。

运算符判断：检查 opMap 是否包含该字符串。

界符判断：如果 word 长度为 1，并且在 delimiterMap 中，返回对应 Token。

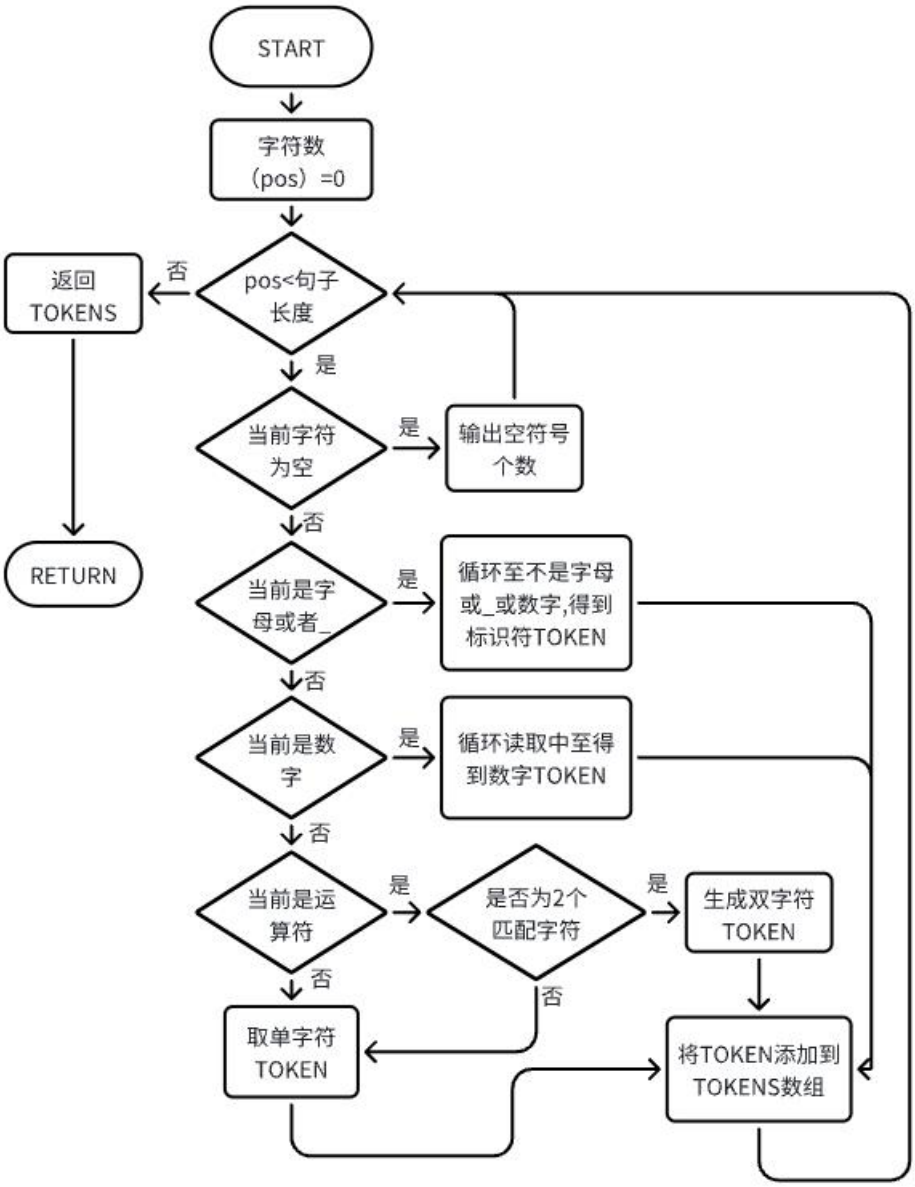
标识符判断：判断 word 是否以字母或下划线开头，并且后续字符只能是字母、数字或下划线（注意这里修改过，允许标识符包含数字）。

数字常量判断：检测是否为数字常量，支持整数、小数和科学计数法。

使用循环检查每个字符，记录小数点与指数符号出现次数，以及指数后是否允许符号。

如果以上都不匹配，则返回 code = 0，表示无法识别。

分词流程如下：




功能说明

对输入的整行代码进行逐字符扫描，切分成多个词素（token 字符串），并返回一个字符串向量。

过程：

跳过空白符：如果遇到空格或制表符，则统计数量并输出提示（并不放入 tokens 中）。

根据字符类型切分：

如果字符是字母或下划线，连续读取直到遇到非标识符字符，形成一个标识符。

如果是数字，则连续读取形成数字常量。

如果是运算符或界符，则先尝试读取两个字符（以支持 "=="、"<="、"++"、"--"），否则读取单个字符。

其他字符作为单独 token 处理。

**5.程序创新点**

（1）文件识别功能的增强

除了支持基本的控制台输入，软件增加了对 C 语言源代码文件的识别功能。

用户可以选择文件输入模式，软件将自动读取文件内容、逐行扫描并进行词法分析，同时对每一行的词素进行检测和错误提示（错误信息显示为红色，并包含行号）。

这一功能不仅提高了软件的实用性，也更贴近实际编译器的工作流程，为后续集成更多编译器前端模块奠定基础。

（2）错误检测与反馈

在词法识别过程中，软件能够自动检测无法识别的词素，并及时输出包含错误行号的红色提示信息。

这种实时错误反馈机制，帮助用户快速定位代码中的问题，有助于提高编程调试效率，并且在实验报告中体现出对错误处理的重视与改进。

（3）跨平台打包与便携性

软件经过打包处理后，实现了跨平台运行（例如在 Windows 和 Linux 系统上均可运行），增强了软件的便携性和实际应用价值。

用户无需额外安装复杂环境即可体验词法分析器的全部功能，这一点在实验成果展示中也具有较高的创新性。

（4）良好的用户交互设计

软件提供了直观的菜单选择界面，用户可以在文件输入和控制台输入之间自由切换，使得操作更简单、用户体验更佳。

同时，软件对空白符、制表符的处理也做了详细提示，增加了对输入流的透明性，方便用户理解词法分析器的工作原理。

## 五．程序源代码

**1.单个词素分析程序**

```
Token analyzeToken(const string& word) {
    Token token;
    // 1) 判断是否为关键字
    if (keywordMap.find(word) != keywordMap.end()) {
```

```
        token.code = keywordMap[word];
        token.lexeme = word;
        return token;
    }
    // 2) 判断是否为运算符
    if (opMap.find(word) != opMap.end()) {
        token.code = opMap[word];
        token.lexeme = word;
        return token;
    }
    // 3) 判断是否为界符（只考虑单字符界符）
    if (word.size() == 1 && delimiterMap.find(word[0]) != delimiterMap.end()) {
        token.code = delimiterMap[word[0]];
        token.lexeme = word;
        return token;
    }
    // 4) 判断是否为标识符
    // 标识符要求：首字符为字母或下划线，后续字符可以为字母、数字或下划线
    bool isIdentifier = true;
    if (!word.empty() && isLetterOrUnderscore(word[0])) {
        for (size_t i = 1; i < word.size(); i++) {
            if (!isLetterOrUnderscore(word[i]) && !isdigit(word[i])) {
                isIdentifier = false;
                break;
            }
        }
    }
    else {
        isIdentifier = false;
    }
    if (isIdentifier) {
        token.code = 400;
        token.lexeme = word;
        return token;
    }
    // 5) 判断是否为数字常量（支持整数、小数、科学计数法）
```

```
bool isNumber = true;
int dotCount = 0;
int eCount = 0;
bool afterE = false;
if (word.empty()) isNumber = false;
for (size_t i = 0; i < word.size(); i++) {
    char c = word[i];
    if (isdigit(c)) {
        // 数字正常
    }
    else if (isLetterOrUnderscore(c) && (c != 'e' && c != 'E')) {
        isNumber = false;
        break;
    }
    else if (c == '.') {
        dotCount++;
        if (dotCount > 1 || eCount > 0) {
            isNumber = false;
            break;
        }
    }
    else if (c == 'e' || c == 'E') {
        eCount++;
        if (eCount > 1 || i == 0) {
            isNumber = false;
            break;
        }
        afterE = true;
    }
    else if ((c == '+' || c == '-') && afterE) {
        afterE = false;
    }
    else {
        isNumber = false;
        break;
    }
```

```
    }
    if (isNumber) {
        token.code = 500;
        token.lexeme = word;
        return token;
    }
    // 6) 如果均不匹配，则无法识别
    token.code = 0;
    token.lexeme = word;
    return token;
}
```

说明：该函数依次判断输入的字符串是否为关键字、运算符、界符、标识符或数字常量。如果都不匹配，则返回 code=0。详见实验原理的词素分析流程图。

**2.分词函数代码（将一行分割为多个单词）**

```
vector<string> tokenizeLine(const string& line) {
    vector<string> tokens;
    size_t pos = 0;
    while (pos < line.size()) {
        // 如果遇到空白符或制表符，统计并输出提示，然后跳过
        if (line[pos] == ' ' || line[pos] == '\t') {
            int count = 0;
            while (pos < line.size() && (line[pos] == ' ' || line[pos] == '\t')) {
                count++;
                pos++;
            }
            cout << "[跳过 " << count << " 个空白符]" << endl;
            continue;
        }
        // 根据当前字符决定分词规则
        char cur = line[pos];
        string tokenStr = "";
        // 如果是字母或下划线，则扫描标识符（允许后续含数字）
        if (isLetterOrUnderscore(cur)) {
            while (pos < line.size() && (isLetterOrUnderscore(line[pos]) || isdigit(line[pos]))) {
                tokenStr.push_back(line[pos]);
```

```
            pos++;
        }
    }
    // 如果是数字，则扫描数字（包括小数及科学计数法部分）
    else if (isdigit(cur)) {
        while (pos < line.size() && (isdigit(line[pos]) || line[pos] == '.' ||
            line[pos] == 'e' || line[pos] == 'E' ||
            ((line[pos] == '+' || line[pos] == '-') && pos > 0 &&
                (line[pos - 1] == 'e' || line[pos - 1] == 'E'))))
        {
            tokenStr.push_back(line[pos]);
            pos++;
        }
    }
    // 如果是可能的运算符或界符，则先尝试两字符匹配（如 "==", "<=", "++", "--"）
    else if (isOpDelimChar(cur)) {
        if (pos + 1 < line.size()) {
            string twoChars;
            twoChars.push_back(line[pos]);
            twoChars.push_back(line[pos + 1]);
            if (opMap.find(twoChars) != opMap.end()) {
                tokenStr = twoChars;
                pos += 2;
            }
            else {
                tokenStr.push_back(cur);
                pos++;
            }
        }
        else {
            tokenStr.push_back(cur);
            pos++;
        }
    }
    // 否则将当前字符作为单独的 token
    else {
```

```
            tokenStr.push_back(cur);
            pos++;
        }
        tokens.push_back(tokenStr);
    }
    return tokens;
}
```

说明：本函数逐字符扫描输入行，遇到连续空白符时输出提示（不将空白符加入分词结果），根据字符类型（字母、数字、运算符或界符）将一行文本拆分成若干个词素，并返回一个字符串向量。详见实验原理的分词程序流程图。

**3.主函数代码（控制台函数）**

```
int main() {
    while (true) {
        cout << "请选择输入模式：\n1. 文件输入\n2. 控制台输入\n3. 退出程序" << endl;
        int mode = 0;
        cin >> mode;
        cin.ignore(); // 清除换行符

        if (mode == 1) {
            // 文件输入模式
            cout << "请输入包含 C 语言代码的.c 文件名（例如 code.c）:" << endl;
            string filename;
            getline(cin, filename);

            ifstream infile(filename);
            if (!infile.is_open()) {
                cerr << RED_TEXT << "无法打开文件 " << filename << RESET_TEXT <<
endl;
                continue; // 返回模式选择界面
            }

            cout << "读取文件内容并进行词法分析：" << endl;
            string line;
            int lineNum = 0;
            while (getline(infile, line)) {
                lineNum++;
```

```
                processLine(line, lineNum);
            }
            infile.close();
            cout << "文件词法分析完成！" << endl;
        }
        else if (mode == 2) {
            // 控制台输入模式
            cout << "请输入包含单词的句子进行词法分析（输入 'quit' 退出该模式）:" <<
endl;
            int lineNum = 0;
            while (true) {
                lineNum++;
                cout << "> ";
                string input;
                getline(cin, input);
                if (input == "quit")
                    break;
                processLine(input, lineNum);
            }
        }
        else if (mode == 3) {
            cout << "程序退出！" << endl;
            break;
        }
        else {
            cout << RED_TEXT << "输入模式错误，请重新选择！" << RESET_TEXT << endl;
        }
    }
    return 0;
}
```

说明：

主函数提供三种输入模式：文件输入、控制台输入和退出。

文件输入模式下，提示输入文件名，逐行读取文件并调用 processLine 处理，同时统计行号；

控制台输入模式下，用户可逐行输入源代码（输入 "quit" 退出该模式）；

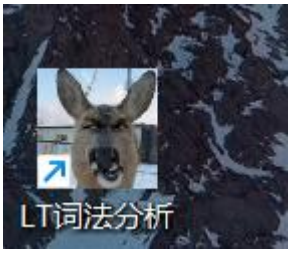
如果用户输入的模式错误，则以红色显示错误信息。*

## 六．实验数据、结果分析

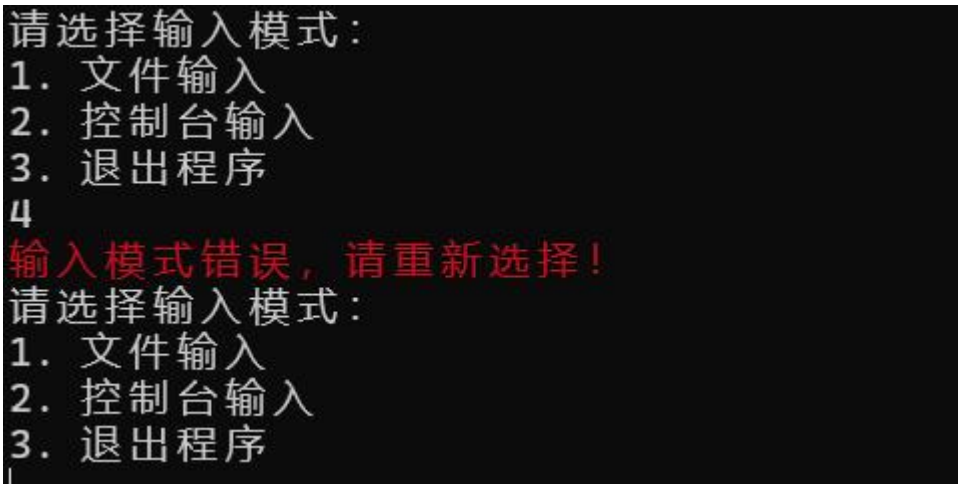
1.程序安装

打开以下任意一个安装包选择安装路径可正常安装程序。


选择安装路径成功安装后桌面会有软件快捷方式




2.打开程序测试输入




若输入错误则会报错并提醒重新输入。

2.选择控制台输入并输入测试用例，正常跳过空白符并分析词素。

```
请选择输入模式:
1. 文件输入
2. 控制台输入
3. 退出程序
2
请输入包含单词的句子进行词法分析（输入 'quit' 退出该模式）:
> void main(){string _iijjkk123=2+1;}
[跳过 1 个空白符]
[跳过 1 个空白符]
种别码: 101, 单词: void
种别码: 102, 单词: main
种别码: 302, 单词: (
种别码: 303, 单词: )
种别码: 304, 单词: {
种别码: 400, 单词: string
种别码: 400, 单词: _iijjkk123
种别码: 205, 单词: =
种别码: 500, 单词: 2
种别码: 201, 单词: +
种别码: 500, 单词: 1
种别码: 301, 单词: ;
种别码: 305, 单词: }
>
```

3.控制台输入无种别码词素，生成红色报错提示。

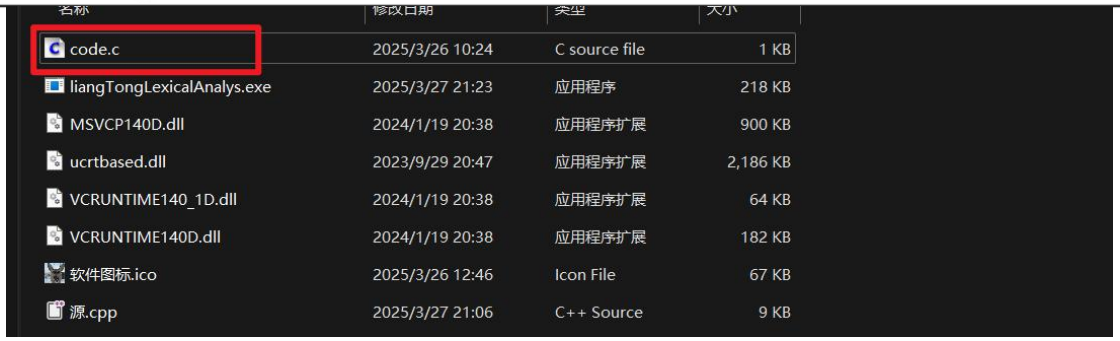
```
> "the dog == 2"
[跳过 1 个空白符]
[跳过 1 个空白符]
[跳过 1 个空白符]
[错误] 第 2 行: 无法识别的词素 '"'
种别码: 400, 单词: the
种别码: 400, 单词: dog
种别码: 206, 单词: ==
种别码: 500, 单词: 2
[错误] 第 2 行: 无法识别的词素 '"'
```

输入 quit 结束程序。

4.测试文件输入，输入 1 选择文件输入模式

输入文件名，测试文件保存在软件目录下，是一个冒泡排序 C 语言程序。

```
请选择输入模式:
1. 文件输入
2. 控制台输入
3. 退出程序
1
请输入包含C语言代码的.c文件名（例如 code.c）:
code.c
```



| 名称 | 修改日期 | 类型 | 大小 |
|---|---|---|---|
| code.c | 2025/3/26 10:24 | C source file | 1 KB |
| liangTongLexicalAnalys.exe | 2025/3/27 21:23 | 应用程序 | 218 KB |
| MSVCP140D.dll | 2024/1/19 20:38 | 应用程序扩展 | 900 KB |
| ucrtbased.dll | 2023/9/29 20:47 | 应用程序扩展 | 2,186 KB |
| VCRUNTIME140_1D.dll | 2024/1/19 20:38 | 应用程序扩展 | 64 KB |
| VCRUNTIME140D.dll | 2024/1/19 20:38 | 应用程序扩展 | 182 KB |
| 软件图标.ico | 2025/3/26 12:46 | Icon File | 67 KB |
| 源.cpp | 2025/3/27 21:06 | C++ Source | 9 KB |

```c
void Bubble_sort(int arr[], int size)
{
    int j,i,tem;
    for (i = 0; i < size-1;i ++)
        int count = 0;
        for (j = 0; j < size-1 - i; j++)
        {
            if (arr[j] > arr[j+1])
            {
                tem = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tem;
                count = 1;

            }
        }
        if (count == 0)
                break;
```

按回车后程序正常分析词素和种别码，对于没有种别码的单词会进行红色字符报错，并提示错误在哪一行

```
1
请输入包含C语言代码的.c文件名（例如 code.c）：
code.c
读取文件内容并进行词法分析：
[跳过 1 个空白符]
[跳过 1 个空白符]
[跳过 1 个空白符]
[跳过 1 个空白符]
种别码：101，单词：void
种别码：400，单词：Bubble_sort
种别码：302，单词：(
种别码：103，单词：int
种别码：400，单词：arr
种别码：306，单词：[
种别码：307，单词：]
[错误] 第 1 行：无法识别的词素 ','
种别码：103，单词：int
种别码：400，单词：size
种别码：303，单词：)

种别码：304，单词：{
[跳过 1 个空白符]
[跳过 1 个空白符]
```

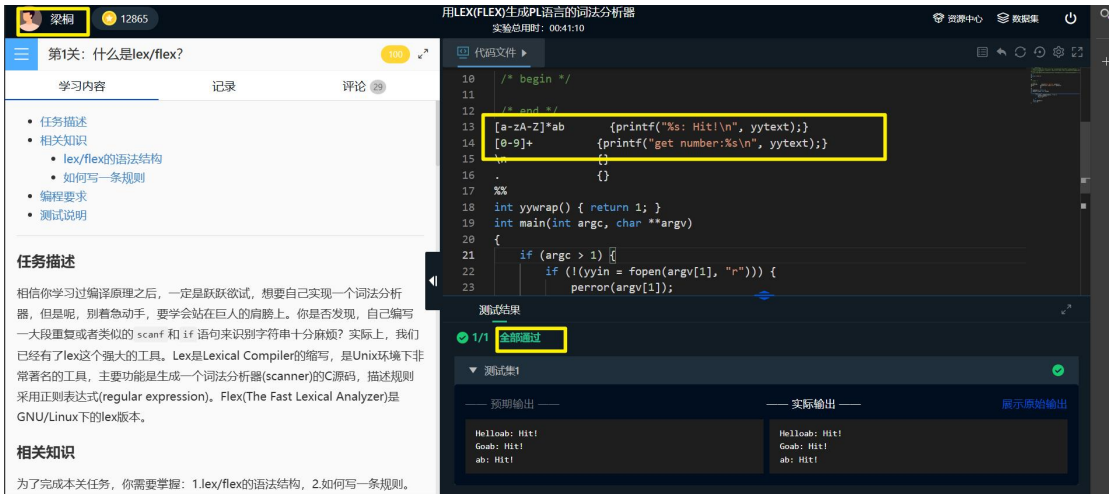
```
种别码：400，单词：i
种别码：211，单词：++
种别码：303，单词：)
[跳过 1 个空白符]
种别码：304，单词：{
[跳过 2 个空白符]
[跳过 1 个空白符]
种别码：400，单词：scanf
种别码：302，单词：(
[错误] 第 33 行：无法识别的词素 '"'
[错误] 第 33 行：无法识别的词素 '%'
种别码：400，单词：d
[错误] 第 33 行：无法识别的词素 '"'
[错误] 第 33 行：无法识别的词素 ','
[错误] 第 33 行：无法识别的词素 '&'
种别码：400，单词：arr
种别码：306，单词：[
种别码：400，单词：i
```

5. 输入 3 退出程序，打开任务管理器可观察程序停止运行。

## 七. 学习记录

### 1.头歌平台学习记录

头歌平台实验 1




测试用例全部通过

代码结构详解

头文件包含：

使用 %{ ... %} 块包含了 <stdio.h>，以便使用 printf 函数输出结果。

词法规则区 (%% 内)：

自定义代码区（begin 和 end 之间）： 此部分预留给用户扩展代码，目前为空，但可以添加额外的初始化或者辅助函数。

规则 [a-zA-Z]*ab:

匹配规则：[a-zA-Z]* 表示任意个字母（大写或小写）的序列，后接固定的字符串 "ab"（要求必须是小写）。

动作：匹配到的字符串存储在全局变量 yytext 中，然后调用 printf("%s: Hit!\n", yytext); 输出匹配信息。

规则 [0-9]+:

匹配一串连续的数字，并输出 "get number:" 后跟数字序列。

规则 \n 和 .：

用于处理换行符和其他所有未匹配的字符，保证输入中的不合法部分不会干扰整体的扫描。

辅助函数：

int yywrap() { return 1; }：表示当输入结束后，词法分析器自动结束扫描。

主函数 main:

检查命令行参数，如果提供了文件名，则打开该文件作为输入。

使用 while (yylex()) 循环调用词法分析器 yylex()，直到文件结束。

核心功能与注意点

核心功能：

合法输入识别： 代码核心在于 [a-zA-Z]*ab 规则，这条规则确保只有以小写字母“ab”结尾的字符串才会输出 “Hit!”。

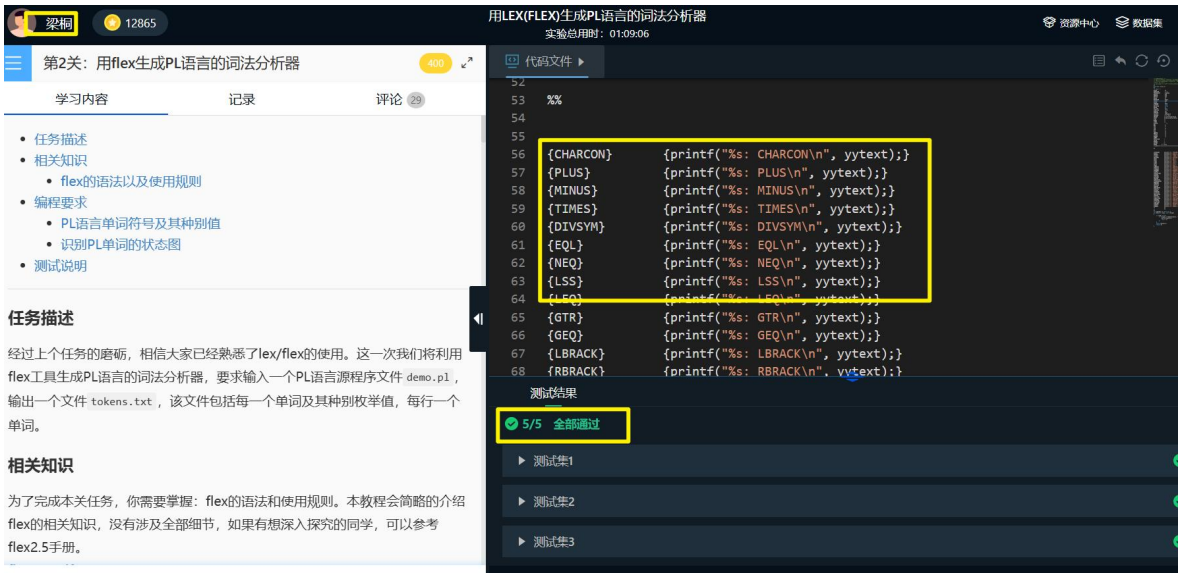
数字处理： 除了字符串，还实现了对纯数字的识别。

注意点：

匹配优先级： 规则顺序决定了匹配优先级。此处首先匹配以 "ab" 结尾的字符串，其次匹配数字，最后通过 . 捕获其他不符合规则的字符。

扩展空间： begin 和 end 之间的区域可以用于添加额外代码，例如状态控制或者更多匹配规则。

头歌平台实验 2




测试用例全部通过。

下面是一份对实验 2 PL 词法分析器代码的总结笔记（不超过 400 字）：

该代码使用 LEX/FLEX 构建了一个 PL 语言的词法分析器。程序分为两大部分：定义 token 和匹配规则。首先在“begin”与“end”之间，用宏定义形式声明了 PL 支持的各种关键字和符号，如 OFSYM、ARRAYSYM、PROGRAMSYM、运算符（PLUS、MINUS、TIMES、DIVSYM 等）、比较符号（EQL、NEQ、LSS、LEQ、GTR、GEQ）、标点符号（LBRACK、RBRACK、LPAREN、RPAREN、COMMA、SEMICOLON、COLON、PERIOD）、赋值符号 BECOME，以及整型常量 INTCON、字符常量 CHARCON 和标识符 IDENT。最后还定义了一个 ERROR 规则用于捕获非法字符。

在“%%”部分，每个 token 对应一个动作，即当规则匹配时调用 printf 输出 token 种别和匹配文本。匹配顺序由上至下，确保优先匹配关键字和多字符符号，防止与单字符符号冲突。辅助规则\n 和.分别处理换行符和其他未匹配字符，保证词法分析的连贯性。最后，通过 yywrap()

函数与 main 函数整合，实现对输入文件的扫描与词法分析。

此代码能识别完整的 PL 源程序，并在遇到非法字符时给出错误提示，适用于平台提供的五个测试集。

**2.程序中遇到的困难与解决思路**

在开发这个词法分析器的过程中，遇到了一些挑战和挫折，下面列举几个主要的问题及其解决方案：

分词规则的设计

困难：最开始设计分词时，如何将连续的字符正确拆分成单个词素（如将 "a=2-1;" 分成 "a", "=", "2", "-", "1", ";"），确保每种类型（标识符、数字、运算符、界符）都能被正确识别，是个较为复杂的问题。

解决方案：经过查阅相关编译原理资料，我们采用了逐字符扫描的方式，依次判断字符类型，并对连续的空白符进行统计提示。为此设计了 tokenizeLine 函数，通过精细的条件判断（例如对字母、数字、运算符、界符的不同处理）来实现正确分词。通过多次调试与测试（使用控制台和文件输入模式分别验证），不断修正了判断条件，最终获得了较为准确的分词效果。

标识符和数字常量识别的模糊性

困难：在 C 语言中，标识符的规则允许数字出现在非首字符位置，而数字常量的格式可能和标识符混淆（例如 "123abc" 既不符合标识符也不是合法数字）。

解决方案：通过细致阅读教材和实验指导书，我们修改了 analyzeToken 函数中对标识符和数字的判断逻辑，允许标识符包含数字但要求首字符必须是字母或下划线。同时，对数字常量的处理增加了对小数点和科学计数法的判断，并设置了合适的条件，确保两者区分明确。

错误处理与反馈设计

困难：在实验要求中，错误提示要求必须显示行号，并且错误信息需要以红色显示。如何在控制台中实现彩色输出一开始没有明确思路。

解决方案：通过搜索 ANSI 终端颜色代码，我们了解到使用 \033[31m 可以将文本显示为红色，而 \033[0m 可以重置颜色。于是我们在输出错误信息的地方增加了这两个宏定义（RED_TEXT 和 RESET_TEXT），确保在检测到无法识别的词素时，能够正确输出红色错误提示，并且显示错误的行号。

文件输入和控制台输入模式的整合

困难：最初只实现了控制台输入模式，但在实验中需要支持文件输入，并且在文件扫描结束后还能选择其他模式。如何在设计上使程序既能处理文件，又能保持良好的用户交互体验，是一个设计挑战。

解决方案：我们在 main 函数中设计了一个循环菜单，让用户可以选择文件输入、控制台输入或退出程序。通过在文件输入模式中逐行读取代码、传递行号到 processLine 函数，并在结束后返回主菜单，从而实现了良好的模式切换。多次测试后，确保各个模式均能正确运行并且互不干扰。

调试和测试

困难：由于词法分析器涉及多种情况（包括边界条件、错误输入等），在实际运行中发现部分边界情况无法正确识别，如连续空格、含有非法字符的词素等。

解决方案：针对这些问题，我们设计了一系列测试用例（包括正确输入和错误输入），在文件和控制台两种模式下反复测试，通过调试输出逐步查找问题所在，并及时修改代码逻辑，最终使得所有测试用例都能达到预期效果。

## 八. 实验小结

通过本次实验，我深入理解了词法分析器的核心原理，掌握了正规表达式、状态转换图和模块化设计等关键知识。在设计与实现过程中，我将输入处理、分词、词法识别和错误检测模块化，保证了代码结构清晰且便于扩展。调试过程中，通过反复查阅资料和测试，我解决了标识符与数字常量识别模糊、连续空白符处理等问题，显著提升了调试和测试能力。软件不仅支持控制台输入，还增加了文件输入功能，并实现了行号跟踪与错误提示（以红色显示），改善了用户体验。此次实验不仅巩固了理论知识，更使我对实际编译器前端设计有了初步认识，为今后更复杂的编译器设计打下坚实基础。