**C语言词法分析程序的设计与实现**

**目录**

[实验内容及要求 2](#_Toc115703316)

[实验环境 2](#_Toc115703317)

[程序设计说明 3](#_Toc115703318)

[一、总体设计 3](#_Toc115703319)

[1.1、总体思路 3](#_Toc115703320)

[1.2、类设计 3](#_Toc115703321)

[1.2.1、Symbol类 3](#_Toc115703322)

[1.2.2、Scanner类 4](#_Toc115703323)

[1.3、流程设计 6](#_Toc115703324)

[二、主要模块设计 6](#_Toc115703325)

[2.1、字符缓冲区 6](#_Toc115703326)

[2.2、类型判断 7](#_Toc115703327)

[2.3、数字处理 7](#_Toc115703328)

[2.4、注释处理 9](#_Toc115703329)

[2.5、字符串处理 10](#_Toc115703330)

[2.6、标识符处理 11](#_Toc115703331)

[2.7、操作符/界限符处理 12](#_Toc115703332)

[2.8、错误处理 12](#_Toc115703333)

[三、运行结果及分析 14](#_Toc115703334)

[3.1、输入源程序 14](#_Toc115703335)

[3.2、执行结果 14](#_Toc115703336)

[3.2.1、控制台输出 14](#_Toc115703337)

[3.2.2、文件输出 15](#_Toc115703338)

[3.3、分析说明 17](#_Toc115703339)

[四、LEX实现 18](#_Toc115703340)

[4.1、正则表达式 18](#_Toc115703341)

[4.2、辅助函数 19](#_Toc115703342)

[4.3、错误警告 20](#_Toc115703343)

[4.4、运行结果 20](#_Toc115703344)

[4.4.1、输入程序 20](#_Toc115703345)

[4.4.2、运行结果 20](#_Toc115703346)

[4.5、对比分析 22](#_Toc115703347)

实验内容及要求

* 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。
* 可以识别并跳过源程序中的注释。
* 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。
* 检查源程序中存在的词法错误，并报告错误所在的位置。
* 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。

实验环境

|  |  |  |
| --- | --- | --- |
| **环境/工具** | **说明** | **版本/网址** |
| **Windows11** | **OS** | **220916** |
| **TDM-gcc** | **编译器** | **10.3.0** |
| **Flex** | **LEX** | **2.5.4** |
| **FSM simulator** | **NFA绘制** | **http://ivanzuzak.info/noam/webapps/fsm\_simulator/** |

程序设计说明

一、总体设计

1.1、总体思路

通过良好的高层封装实现对各单词符号的提取和处理，将各类记号的读取分为多个模块，降低耦合性。同时模仿**Java**语言中java.util.Scanner类的读取模式next()，调用nextSymbol()函数进行单词符号获取，实现流畅且清晰的单词符号处理。

1.2、类设计

1.2.1、Symbol类

* **类说明**

记录单词符号的**详细信息**，包括：

* + - 单词符号的**内容**
    - 单词符号的**记号类型**
    - 单词符号在原文件中的**行列位置**

单词符号的记号类型包括：

enum class Type

    {

        END,            // 结束符

        ERROR,          // 错误符

        KEYWORD,        // 关键字

        IDENTIFIER,     // 标识符

        NUMBER,         // 数字

        OPERATOR,       // 操作符

        LIMIT,          // 界限符

        STRING,         // 字符

        COMMENT,        // 注释

    };

* **主要函数**

void append(char ch) // 向字符数组中添加字符

char rollBack() // 从字符数组中回退一个字符

Type attr() // 获取记号类型（属性）

char operator[](int pos) // 重载[]运算符直接读取类字符数组内容

* **源代码**

**图形用户界面

低可信度描述已自动生成**

1.2.2、Scanner类

* **类说明**

读取输入的文件流，通过词法分析，以**记号**为单位获取源文件信息。

词法分析支持：

* + 标识符（英文）
  + 数字，包括十六进制、八进制、二进制前缀以及各类后缀
  + 操作符
  + 界限符
  + 字符串
  + 注释
* **主要函数**

char \_nextChar();           // 从缓冲区中读取字符

void \_rollBack();           // 缓冲区指针回退一个字符

static bool \_isXXX(char); // 判断字符类型

void \_procXXX ();           // 各类记号处理

void \_reportError(ErrorType e);     // 报告错误信息

bool open(const char\* file\_name);   // 打开文件

Symbol nextSymbol();                 // 读取下一记号

* **源代码**

****

****

1.3、流程设计

**图示

描述已自动生成**

二、主要模块设计

2.1、字符缓冲区

**2.1.1、设计模式**

参考教材中的双段缓冲区进行设计，采用单指针，通过操作记号类Symbol的**append(char)** 和**rollBack()** 方法完成**字符同步**。

单指针方式在回退操作位于边界附近时容易触发二次文件读取，因此设计有**回退计数器**，只有在回退计数器为0时才进行文件读取。

**2.1.2、源代码**

****

2.2、类型判断

参考设计流程图，在读取到第一个字符时进行**类型初判**：

文本

描述已自动生成

2.3、数字处理

**2.3.1、设计模式**

数字处理模块支持C语言前缀和后缀，主要分为三部分处理：**前缀处理，中间数处理，后缀处理**。

**2.3.2、有限状态自动机**

图示为前缀和中间数处理的有限状态自动机，由于大小限制，**对于需要回退的非接收结果未标出。**

**图示, 示意图

描述已自动生成**

图示为后缀处理的有限状态自动机，为上图后缀处理的细化。其中的**float/int**状态是在数字处理阶段定义，目的是**支持词法分析阶段的后缀错误检测**。同样的，由于大小限制，**对于需要回退的非接收结果未标出**。

**图示

描述已自动生成**

**2.3.3、部分源代码**

通过该自动机设计C++源代码，本次多采用if-else结构辅以部分标志位来完成。

文本

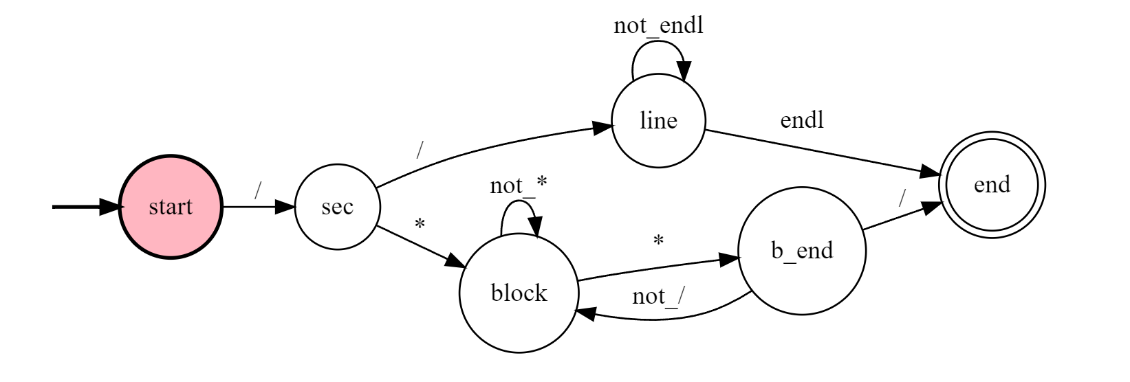
描述已自动生成

2.4、注释处理

**2.4.1、设计模式**

注释处理基于类型判断模块的除号判断，通过后续字符判断其是注释还是操作符。

**2.4.2、有限状态自动机**

****

**2.4.3、部分源代码**

对于状态机图中block状态和b\_end状态的转移，在C++源代码中通过end标志位进行处理。

**文本

描述已自动生成**

2.5、字符串处理

**2.5.1、设计模式**

字符串处理基于类型判断模块的引号判断，支持转义换行以及单引号内字符个数超限警告。在实际开发过程中，应当考虑Unicode字符以及各类不同编码字符的处理，尤其是对于有字符数量限制的单引号字符。不过考虑到本程序设计的重点，本次实验未实现Unicode字符的解析。

**2.5.2、有限状态自动机**

下图中mark为前后匹配的双引号或单引号，esp为转义字符’\’

**图示

描述已自动生成**

**2.5.3、源代码**

对于状态机图中comment状态和escape状态的转移，即多个转义符的识别，在C++源代码中通过escape标志位进行处理。

**文本

中度可信度描述已自动生成**

2.6、标识符处理

**2.6.1、设计模式**

标识符处理基于C语言标识符定义要求：必须由字母（区分大小写）、数字、下划线组成，且标识符的第一个字符不可以是数字。该版本设计中未支持UTF-8模式，即未支持中文标识符。

在标识符成功识别后，会通过扫描函数判断该标识符是否为C语言关键字。

在实际开发过程中，应当考虑Unicode字符以及各类不同编码字符的处理，不过考虑到本程序设计的重点，本次实验未实现Unicode字符的解析。

**2.6.2、有限状态自动机**

**图示

描述已自动生成**

**2.6.3、源代码**

**文本

描述已自动生成**

2.7、操作符/界限符处理

**2.7.1、设计模式**

由于长度较为固定，操作符/界限符的识别采用单段if-else模式，辅以rollBack()回退以支持不同长度的字符识别。

由于仅要求输出单词与记号，因此该设计中未区分不同操作符。

**2.7.2、部分源代码**

图片包含 文本

描述已自动生成

2.8、错误处理

**2.8.1、设计模式**

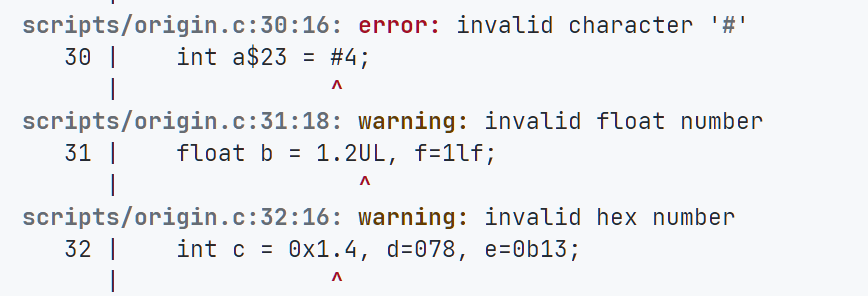
**在前序记号处理的过程中，如果遇到错误，则会让字符缓冲区指针回退，同时立即进行错误处理并报告错误到控制台。后续将从回退的位置继续重新开始进行单词分析。**

参考**gcc/g++**的错误警告格式，在Scanner类中设计有行缓冲区，用于标明字符缓冲区指针所在行，即同时也可以**标明错误所在行**。支持部分词法分析阶段可以发现的错误：

* + 非法字符
  + 字符串/注释结尾缺失
  + 字符常量超限
  + 浮点常量无效
  + 十六进制错误
  + 八进制错误
  + 二进制错误

对于**浮点常量无效、十六进制/八进制/二进制错误，分析器仅给出警告信号**，不计入错误统计。

**2.8.2、显示效果**

****

**2.8.3、部分源代码**



三、运行结果及分析

3.1、输入源程序

输入源程序包括**各类标识符、操作符识别测试**以及各类**错误测试**，具体如下：

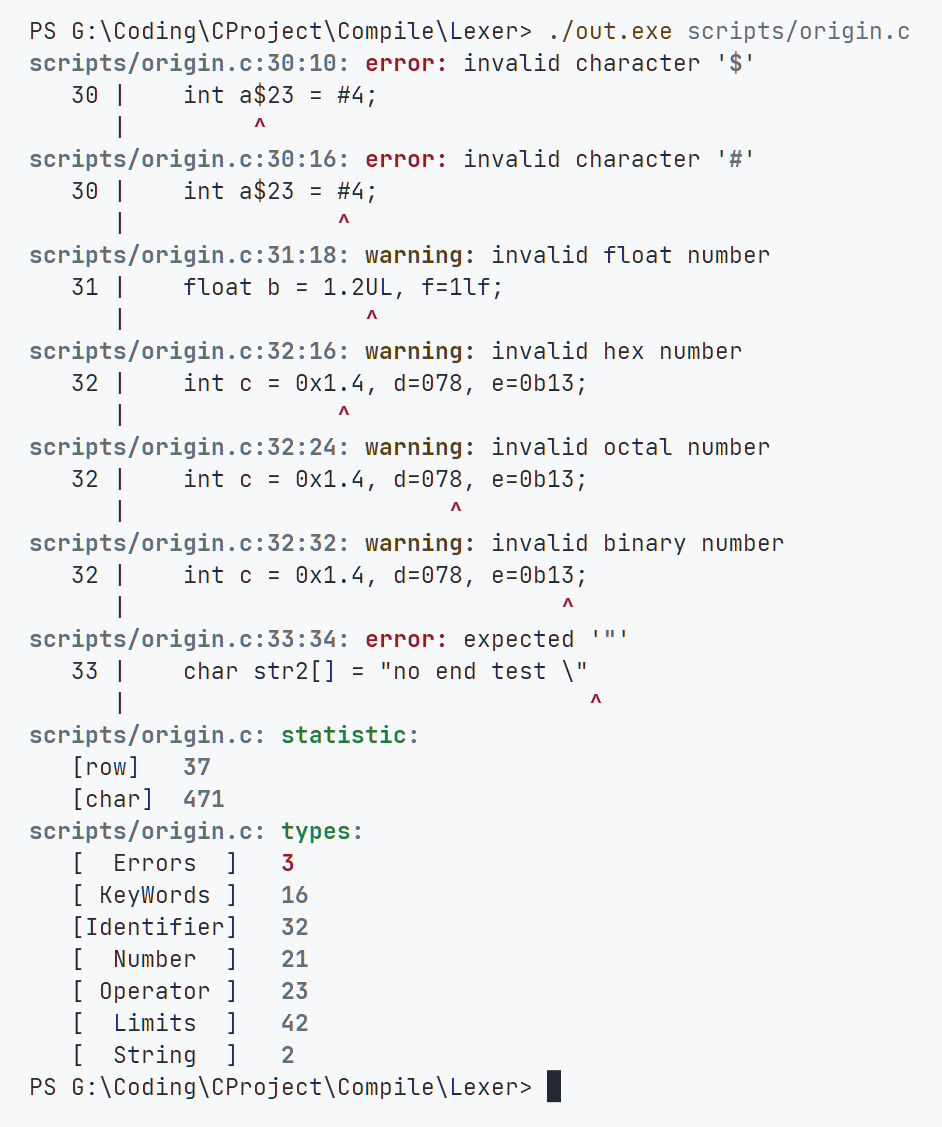
**文本

描述已自动生成**

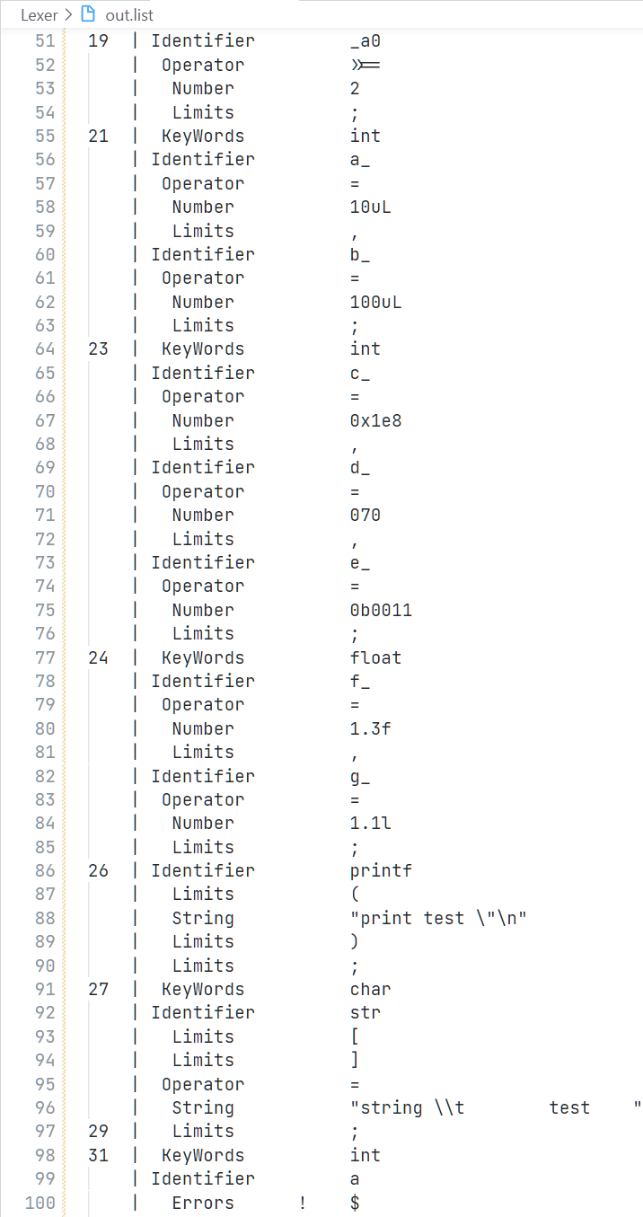
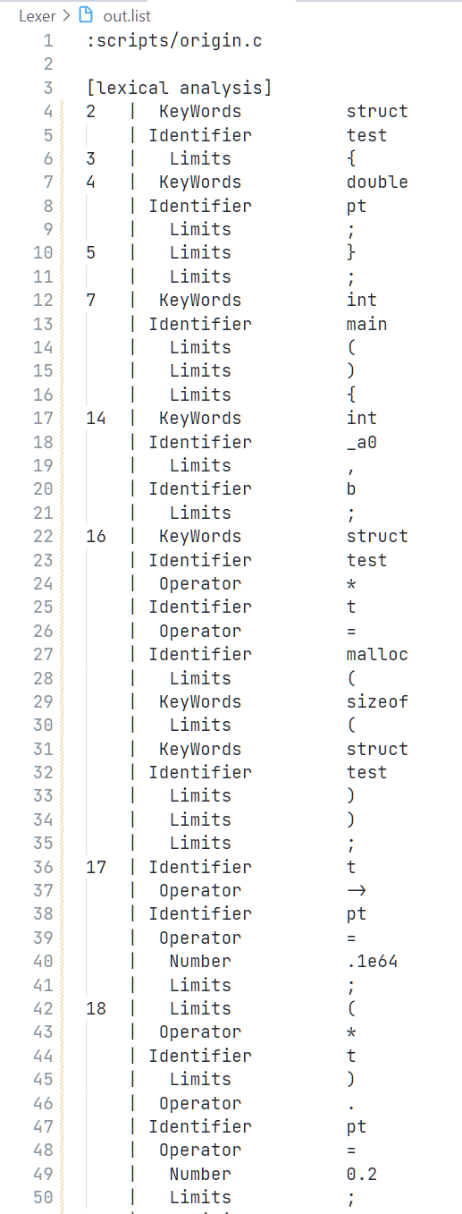
3.2、执行结果

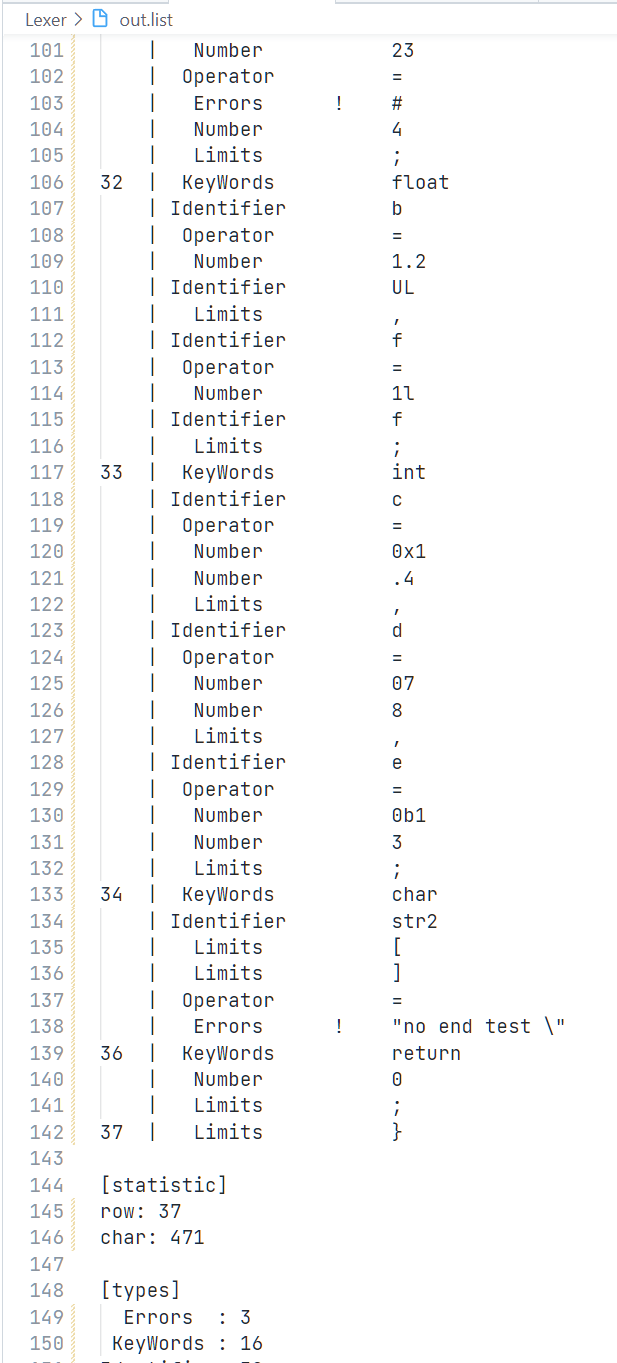
将源文件以参数形式传递给词法分析程序处理，分析结果将同时在控制台和文件中输出。其中控制台主要用于显示错误信息和统计信息；文件中包括各记号的详细信息和统计信息

3.2.1、控制台输出

****

3.2.2、文件输出

****

**背景图案

中度可信度描述已自动生成**

3.3、分析说明

本词法分析程序将单词记号分为：

* + - 关键字 [KeyWords] **(int, while, if …)**
    - 标识符 [Identifier] **(a0\_, b, c …)**
    - 数字 [Number] **(10ll, .1e64, 1.2f …)**
    - 操作符 [Operator] **(+=, >>=, ->, …)**
    - 界限符 [Limits] **({, [, (, ;, …)**
    - 字符串 [String] **(“xxx”, ‘x’)**

对于**单词符号的分析**，可以看到，词法分析的输出文件中能够标识每行的各类单词符号，并在末尾记录总行数、总字符数，各记号个数。

对于**错误报告**，可以看到词法分析程序能够进行错误恢复，并准确标明**错误的所在位置和类型**。同时在输出末尾标明**错误的总个数**

四、LEX实现

4.1、正则表达式

图形用户界面, 文本, 应用程序

描述已自动生成

根据LEX程序的要求，结合实验中的词法分析程序，设计了基于正则表达式的LEXER。

关键匹配项说明：

* **前序定义**

提前将出**现频率高**的字符定义为宏，包括0-9数字、界限符号、操作符号、关键字以及后缀。

NUM         [0-9]

LETTER      [A-Za-z]

LIMIT       [\{\}\[\(\)\[\],;]

OP          (<|>|=|\+|-|\\*|\/|\%|&|\|\!|\^|\.)|(>=|<=|==|!=|\+\+|--|&&|\|\||\+=|-=|\\*=|\/=|%=|&=|\|=|\^=|>>|<<|->)|(>>=|<<=)

I\_SUFFIX    (u|U|l|L|ll|LL|ul|uL|UL|lu|Lu|LU|ull|uLL|ULL|llu|LLu|LLU)

F\_SUFFIX    (f|F|l|L)

KEY         (auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|goto|if|int|long|register|return|short|signed|sizeof|static|struct|switch|union|unsigned|void|volatile|while)

* + **数字**

在该项匹配中，分为十进制整数、十进制浮点、十六进制、八进制、二进制数分别构造正则表达式，在统计环节**统一统计为数字类别**：

hex         (0x|0X)[0-9A-Fa-f]+

octal       0[0-7]+

binary      (0b|0B)[0-1]+

integer     0|([1-9]+{NUM}\*(([Ee][+-]?){NUM}+)?{I\_SUFFIX}?)

float       {NUM}\*\.{NUM}+(([Ee][+-]?){NUM}+)?{F\_SUFFIX}?

* + **字符串**

为了实现实验中的**字符串闭合缺失**的错误警告，在定义字符串的正则表达式的同时还另外定义了缺失闭合型字符串。

string      (\'\\?{LETTER}?\')|(\"([^\n\\\"]\*(\\.|\\+\n)?)\*\")

string\_err  \"([^\n\\\"]\*(\\.)?)\*\n

如此，便可以在字符串闭合失败时进行错误警告，同时不会将字符串行的内容重新进行其他类型的匹配。

图片包含 文本

描述已自动生成

图片包含 徽标

描述已自动生成

* + 注释

注释区分了单行和多行注释，同时在正则表达式中考虑了**多个\*号以及转义字符**的影响。

comment     (\/\/[^\n]\*)|(\/\\*([^\\*]\*(\\*+[^\/]+)\*)\*\\*\/)

4.2、辅助函数

为实现**字符统计、行数统计以及记号统计**功能，在LEX源程序中添加了各类计数器以及辅助函数。

%{/\*2022-10-01\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

int chars = 0,rows = 1,errors = 0;

int keywords = 0,ids = 0,nums = 0,operators = 0,limits = 0;

int strings = 0,comment = 0;

int find(const char\* input,char c){ // 字符查找

    int count = 0;

    for(int i = 0; i < strlen(input); i++)

        if(input[i] == '\n')    count++;

    return count;

}

int length(const char\* input){      // 字符长度计算(去除空格、换行)

    int count = 0;

    for(int i = 0;;i++){

        if(input[i] == 0)   break;

        if(input[i] != '\n' && input[i] != ' ' && input[i] != '\t')

            count ++;

    }

    return count;

}

#define ADDROW() {rows += find(yytext,'\n');};  // 行数统计

#define ADD(X) {X++;chars += length(yytext);};  // 字符统计

%}

4.3、错误警告

在所有预置匹配项均未成功时，将在控制台输出错误信息。

.  {printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* error: %d:%s\n",rows,yytext);ADD(errors);rows+=find(yytext,'\n');}

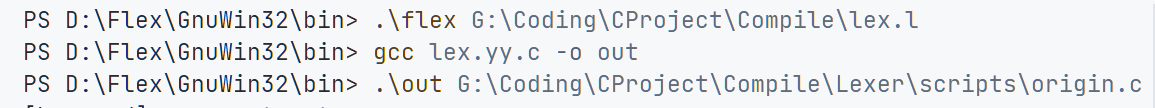
4.4、运行结果

4.4.1、输入程序

输入程序与测试C++实现的词法分析程序相同。

4.4.2、运行结果

编译生成：

****

运行结果：

**图形用户界面

描述已自动生成图形用户界面

中度可信度描述已自动生成**

**图片包含 图形用户界面

描述已自动生成图形用户界面

中度可信度描述已自动生成**

4.5、对比分析

在控制台输出结果中可以清楚的看到，LEX程序能够准确识别各类记号，同时能够准确的统计各类记号的数量和错误。

统计情况对比：

表格

低可信度描述已自动生成表格

中度可信度描述已自动生成

能够看到，C++实现的词法分析程序和LEX实现的词法分析器具有相同的识别结果，这也一定程度上验证了两类程序的正确性。